HTML ET CSS MASTERCLASS

Le cours complet éd. 2020

PIERRE GIRAUD
pierre-giraud.com

Sommaire

Préambule

De quoi traite ce cours ?

Bienvenue à tous dans ce cours complet traitant de deux langages web incontournables : le HTML et CSS.

Le HTML et le CSS sont à la fois des langages qu'il convient de maitriser si on envisage une carrière dans le web et qui sont relativement simples à comprendre. Cela en fait donc des langages parfaits pour des débutants en programmation.

Nous nous intéresserons dans ce cours aux dernières versions stables de ces langages, à savoir le HTML5 / HTML Living standard et le CSS3 / CSS4.

Quels sont les objectifs du cours et à qui s'adresse-t-il ?

Les objectifs de ce cours sont avant tout que vous compreniez bien la place et le rôle de ces deux langages puis que vous appreniez à les utiliser. Pour cela, nous allons explorer les grandes fonctionnalités du HTML et du CSS en commençant avec des choses simples et en allant progressivement vers plus de complexité.

Ce cours s'adresse ainsi aux débutants puisqu'il est très progressif et que chaque notion est expliquée et illustrée aussi bien qu'aux développeurs possédant déjà un bagage informatique et voulant aller plus loin dans leur maitrise du HTML et du CSS.

Notez bien que ce cours n'est pas une liste de définition ou un simple empilage de connaissances. Au contraire, l'idée est que vous compreniez l'intérêt de chaque fonctionnalité ainsi que quand et comment les utiliser et que vous compreniez notamment comment les différents éléments de langages interagissent entre eux.

A la fin de ce cours, vous devriez ainsi avoir acquis une certaine autonomie, être capable de créer des structures relativement complexes et devriez pouvoir résoudre par vous-même la plupart des problématiques liées aux langages HTML et CSS.

Philosophie du cours (méthodologie / pédagogie)

Le domaine de la programmation informatique est un domaine en constante évolution et qui évolue surtout de plus en plus vite. Il est donc essentiel qu'un développeur possède ou acquière des facultés d'adaptation et c'est la raison pour laquelle ce cours a pour but de vous rendre autonome.

Pour servir cet objectif, les différentes notions abordées dans ce cours sont illustrées par de nombreux exemples et exercices. Je vous conseille fortement de passer du temps sur chaque exemple et chaque exercice et de ne pas simplement les survoler car c'est comme cela que vous apprendrez le mieux.

En effet, en informatique comme dans beaucoup d'autres domaine, la simple lecture théorique n'est souvent pas suffisante pour maitriser complètement un langage. La meilleure façon d'apprendre reste de pratiquer et de se confronter aux difficultés pour acquérir des mécanismes de résolution des problèmes.

Ensuite, une fois ce cours terminé, pensez à rester curieux et à vous tenir régulièrement au courant des avancées des langages et surtout continuez à pratiquer régulièrement.

Plan du cours

Ce cours contient 18 sections qui s'enchainent dans un ordre logique et cohérent. Nous allons commencer par étudier des concepts fondamentaux du HTML et du CSS pour démarrer sur de bonnes bases.

Ensuite, nous alternerons entre l'apprentissage de telle fonctionnalité en HTML et de telle autre en CSS.

A partir d'un certain point du cours, nous allons commencer à réutiliser des choses vues avant pour construire des projets de plus en plus complexes. Je vous conseille donc vivement de suivre ce cours dans l'ordre donné.

PARTIE I

Introduction au cours

Introduction au HTML et au CSS

Le but de ce premier chapitre est de vous convaincre rapidement de l'importance et de la place particulière du HTML et du CSS parmi l'ensemble des langages informatiques.

Nous allons également nous intéresser à la question « pourquoi apprendre à coder » et passer en revue les alternatives existantes à l'apprentissage de la programmation en soulignant les points forts et faibles de chacune d'entre elles.

HTML et CSS : deux langages incontournables

Il existe aujourd'hui des dizaines et des dizaines de langages informatiques et de programmation différents : HTML, CSS, JavaScript, PHP, Python, Ruby on Rails, C, C#, C++, Java, etc. pour ne citer qu'eux.

Certains de ces langages ont des possibilités et des rôles similaires, ce qui signifie qu'ils vont être (dans une certaine mesure) interchangeables : on va pouvoir utiliser un langage ou l'autre pour effectuer une même opération selon notre sensibilité personnelle ou selon l'environnement dans lequel on se trouve.

D'autres langages, en revanche, vont être beaucoup plus exclusifs ou ne pas avoir de concurrent et on va donc obligatoirement devoir passer par eux pour effectuer certaines opérations. Cela va être le cas du HTML et du CSS.

En effet, le HTML et le CSS sont deux véritables standards en informatique qui n'ont à l'heure actuelle aucun concurrent comme cela va pouvoir être le cas pour le langage PHP par exemple (pour lequel il existe des alternatives comme Ruby on Rails ou Python entre autres).

De plus, les langages HTML et CSS vont se trouver à la base de tout projet web car ils ont un rôle qui les rend incontournables : les navigateurs (Google Chrome, Safari, etc.) sont des programmes qui ont été construits pour pouvoir lire du code HTML au départ et qui ne peuvent aujourd'hui lire que du code HTML, CSS et JavaScript et nous allons donc nous appuyer sur ces langages (et sur le HTML en particulier) pour pouvoir afficher nos pages.

En bref : quel que soit votre projet web (blog, site e-commerce, application mobile, etc.), vous devrez forcément utiliser du HTML et du CSS.

Pour être un peu plus précis et pour anticiper un peu sur la suite de ce cours, le HTML est un langage de structure : il permet d'indiquer au navigateur que tel contenu est un titre, que tel autre est un simple texte, que cet objet est une image, que celui-ci est une liste, etc. Le navigateur, qui « comprend » le HTML, va se baser sur ces indications pour afficher les contenus.

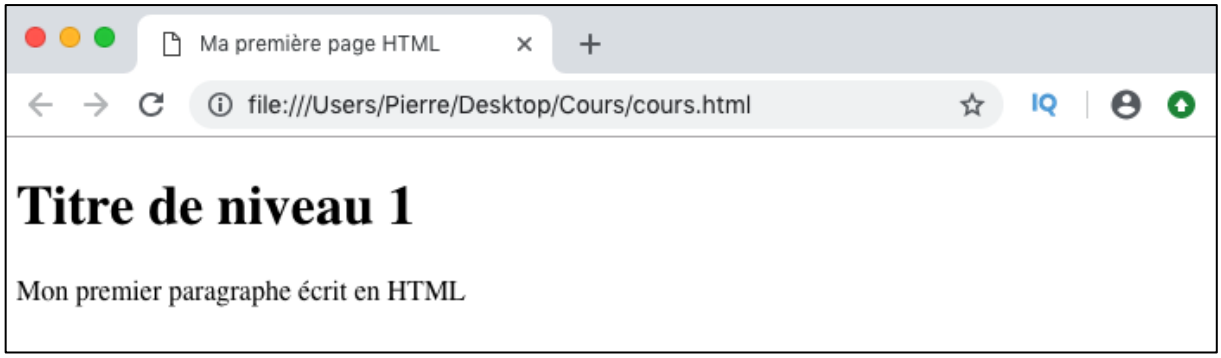
Voici un premier exemple de code HTML :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset = "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Titre de niveau 1</h1>
        <p>Mon premier paragraphe écrit en HTML</p>
    </body>
</html>
```

Je n'ai pas l'intention de vous expliquer précisément ce que représente chaque élément dans ce code pour le moment car ceci n'aurait aucun intérêt. Pas d'inquiétude, nous aurons le temps de voir tout cela par la suite.

L'idée ici est simplement de commencer à vous familiariser avec la syntaxe du HTML et de voir comment le navigateur va traiter ce code. On va ici se concentrer sur deux lignes : la ligne contenant l'élément HTML `h1` qui représente un titre de niveau 1 et celle contenant l'élément `p` qui représente un paragraphe.

Observons comment le navigateur va traiter ces contenus en ouvrant notre fichier dans Google Chrome par exemple :

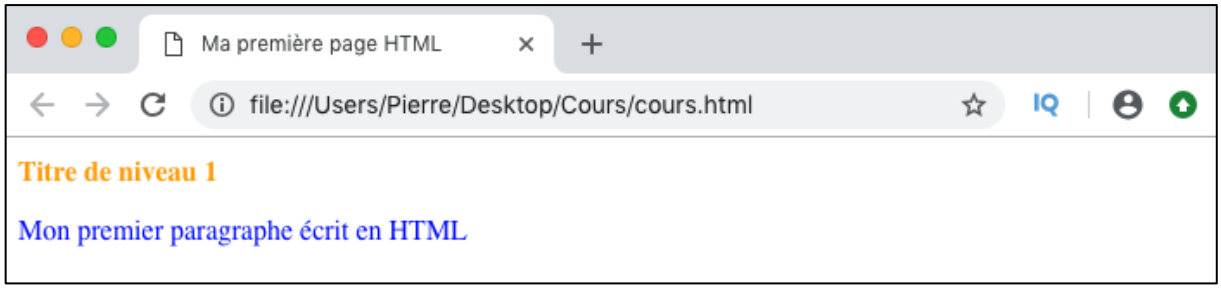


Comme vous pouvez le voir, seul le contenu textuel placé entre `<h1>` et `</h1>` et entre `<p>` et `</p>` est affiché à l'utilisateur. On voit bien que le navigateur comprend que ces deux contenus ne sont pas équivalents puisqu'il les traite de manière différente en les affichant différemment.

Ajoutons maintenant un peu de CSS à la page. Le CSS est un langage de styles : il permet de modifier l'apparence ou le rendu visuel de nos contenus HTML et donc de nos pages web.

```css
h1{
    font-size: 16px;
    color: orange;
}

p{
    font-size: 16px;
    color: blue;
}
```

Ce code CSS indique au navigateur que les titres de niveau 1 devront avoir une taille de 16px (pixels) et être de couleur orange. On indique également que nos paragraphes doivent avoir une taille de 16px et une couleur bleue. Voici le résultat :



Une nouvelle fois, n'essayez pas de comprendre ces codes immédiatement : nous allons le faire dans les prochains chapitres. Le but ici n'est que de vous montrer à quoi le « code » en général va pouvoir ressembler.

Quelles alternatives à l'apprentissage des langages informatiques ?

De nombreuses personnes veulent lancer leur site internet sans forcément apprendre à coder. Est-ce une bonne idée ? Quels sont les risques ? Quelles alternatives existent pour créer un site web sans avoir à le coder ? Quels sont leurs points forts et points faibles ?
Il existe selon moi trois alternatives à l'apprentissage de la programmation qu'on va pouvoir considérer lorsqu'on a un projet web mais qu'on ne veut pas forcément devenir développeur :

- Le recours à un développeur / une agence externe ;
- L'utilisation d'un CMS ;
- L'utilisation d'un éditeur WYSIWIG.

Je vais tenter d'expliquer les points forts et faibles de chacune de ces alternatives et vous montrer en quoi la connaissance ou à minima la compréhension du fonctionnement des langages informatiques reste quasi indispensable quelle que soit l'option choisie.

Solution n°1 : faire appel à un développeur externe

La première solution, qui est certainement la plus évidente pour créer un site internet lorsqu'on ne veut pas le coder soi-même, est de faire appel à un développeur externe ou même à une agence de développement.

Cette première alternative présente à la fois des avantages et des inconvénients. Tout d'abord, parlons du prix. Ici, il va être très variable selon le site que vous allez vouloir créer bien sûr mais également selon le prestataire choisi.

En effet, si vous faites appel à une agence, le coût sera très souvent beaucoup plus élevé que si vous faites appel à un freelance (d'autant plus si vous passez par une plateforme d'annonces où ils sont en concurrence).

Sans rentrer dans le débat sur la qualité, il me semble quand même honnête de dire qu'aujourd'hui une personne seule ne peut pas tout (bien) faire : design, intégration, etc.

En plus de cela, deux autres questions d'importance se posent lorsque vous faites appel à un prestataire externe. La première est : comment bien expliquer ce que vous voulez créer comme site si vous ne savez pas comment le code fonctionne ?

Un client qui n'a aucune notion technique ne va pas pouvoir définir clairement ce qu'il veut. En effet, comprenez bien que le développement est une langue différente tout comme peut l'être la mécanique. Lorsque vous n'exprimez pas précisément votre besoin avec ses spécificités techniques, vous laissez le soin au développeur de les imaginer, ce qui entraine souvent des incompréhensions entre le client et le prestataire développeur.

Imaginez que vous faisiez appel à des ouvriers pour construire une maison mais sans passer par un architecte : vous n'allez pas pouvoir décrire précisément ce que vous voulez. Vous allez décrire votre besoin avec vos mots comme par exemple : « je veux une terrasse, 4 chambres, une cheminée », etc.

De votre point de vue vous « avez tout dit ». Cependant, du point de vue des ouvriers, l'essentiel est manquant : quelles matières utiliser ? quel type d'évacuation des eaux installer ? où faire passer les câbles électriques ? quel type de toiture choisir ?

Cela va donc naturellement mener à un client mécontent et qui va demander des modifications. Souvent, il va demander ce qui lui semble être de « petites modifications » mais les modifications demandées ne vont pouvoir être effectuées qu'en changeant en profondeur le site.

Si vous passez par une grosse agence, il est ici possible qu'il y ait une personne dont le rôle est de transcrire vos besoins en termes techniques. Cependant, ce genre d'agences est bien souvent hors budget pour le commun des mortels.

Enfin, reste la question de la maintenabilité du site : une fois le site livré, comment le faire évoluer ? Comment faire si des bugs apparaissent avec le temps et l'évolution des langages informatiques ? Ici, ne comptez pas trop sur votre prestataire de base pour vous servir d'aide en continu (à moins que vous ne le payiez à nouveau à chaque fois !).

Attention : je n'essaie pas de vous convaincre de ne pas passer par une agence ici. Ce que je veux vous montrer est que passer par un prestataire ne dispense pas de connaitre certaines bases en informatique : si vous connaissez ces bases et si vous comprenez comment fonctionne un site web vous pourrez vous exprimer beaucoup plus clairement et exprimer votre besoin dans un langage qui sera compréhensible pas les développeurs. Vous aurez ainsi beaucoup plus de chances d'avoir un site conforme à vos attentes et vous économiserez sur la facture finale (car moins d'allers-retours et d'incertitude).

Solution n°2 : Utiliser un CMS

Un CMS (pour « Content Management System » ou « Système de Gestion de Contenu ») désigne un ensemble d'applications / de logiciels qui vont permettre de mettre en place et de gérer un site internet.

Un CMS permet donc de posséder un site internet sans se soucier de comment fonctionne le code. Vous pouvez ici imaginer les différentes commandes d'une voiture : lorsque vous achetez une voiture, vous n'avez pas besoin de savoir comment la voiture fonctionne en soi ou de qui se cache sous le capot ; vous n'avez qu'à utiliser les commandes à votre disposition : pédales, levier de vitesse, volant, etc.

Parmi les CMS les plus connus en France on peut notamment nommer WordPress qui permet aujourd'hui de créer plus ou moins n'importe quel type de site ou PrestaShop pour créer un site e-commerce.

Les CMS sont généralement conçus sur le même modèle d'architecture modulable : les utilisateurs vont installer le CMS de base et vont ensuite pouvoir ajouter différents modules ou plugins pour personnaliser leur site.

Utiliser un CMS semble donc parfait pour quelqu'un qui ne sait pas coder à priori. La réalité est toutefois différente. Ici, vous devez bien comprendre que les CMS sont souvent très complexes et que leurs plus grands avantages (la simplicité d'installation et la modularité) vont être également souvent leurs plus grandes faiblesses.

En effet, comme les CMS sont créés de façon à pouvoir être utilisés par tous, la plupart d'entre eux sont d'une complexité rare (du point de vue du code) et vont souvent posséder des tonnes de fonctionnalités qui ne vont pas vous être utiles afin de couvrir le plus de besoins possibles et ainsi satisfaire le plus d'utilisateurs.

De plus, une autre contrepartie de cette architecture commune et modulable est qu'il va être très compliqué d'effectuer des modifications sur la structure de votre site à posteriori sans tout casser à moins d'avoir de sérieuses connaissances techniques sur le CMS en question.

Enfin, il reste le problème de l'incompatibilité possible entre différents modules que vous pourriez installer après avoir créé votre site.

L'usage des CMS reste cependant un bon compromis pour une personne lançant un site sans prétention ou pour quelqu'un qui n'a pas un besoin très spécifique mais, pour maitriser complètement son CMS et pour faire évoluer son site en toute sérénité, il faut finalement être un excellent développeur et un utilisateur expérimenté du CMS en question.

Solution n°3 : la création avec un éditeur WYSIWIG

Les éditeurs WYSIWIG (« What You See Is What You Get » ou en français « Ce que vous voyez est ce que vous obtenez ») sont des éditeurs qui vont créer eux même le code à partir d'un modèle que vous allez créer.

Un éditeur WYSIWIG va se présenter de la façon suivante : une palette d'outils sur le côté et une page dans laquelle vous allez pouvoir utiliser ces outils. Vous allez ainsi par exemple pouvoir insérer un rectangle dans votre page puis placer du texte dedans puis changer la taille de ce texte et etc.

L'éditeur va ensuite se charger de transformer ce que vous avez créé (« ce que vous voyez ») en code.

Le gros souci avec ces éditeurs est qu'ils ne possèdent pas la même logique qu'un humain et qu'ils ne peuvent pas penser un projet ou une page dans sa globalité. Ainsi, le code créé va très souvent être de très mauvaise qualité et cela va impacter le référencement de votre site entre autres.

Je ne parlerai pas plus de ces éditeurs qui ne constituent pas une alternative valide à mes yeux par rapport aux autres solutions proposées.

En résumé : apprendre à coder, ce n'est pas que pour les développeurs !

Je pense vous avoir démontré l'intérêt de maitriser au moins les bases du développement dans la partie précédente si vous avez un projet lié au web : économie d'argent, plus d'efficacité, création d'un site conforme à vos attentes, etc.

En effet, apprendre à coder ou tout au moins connaitre les bases en développement c'est avant tout mettre un pied dans le monde des développeurs pour pouvoir les comprendre et pouvoir vous faire comprendre.

De plus, connaitre le rôle des différents langages et comprendre comment fonctionne votre site permet de pouvoir le gérer de manière beaucoup plus efficace. En effet, je suis et reste persuadé qu'on ne peut pas travailler sereinement lorsqu'on ne comprend pas son outil de travail ou la structure dans laquelle on travaille.

C'est évident et pourtant la plupart des gens essayent de se persuader et de persuader les autres du contraire. Pourquoi ? Car tout le monde utilise internet aujourd'hui et la majorité des gens ne veulent pas faire l'effort de comprendre comment tout cela fonctionne.

Ce n'est pas un problème lorsque vous êtes un simple utilisateur, mais ça le devient lorsque vous devez gérer un site internet ou même lorsque vous travaillez dans un domaine lié au digital.

Deviendriez-vous plombier sur un coup de tête ? Non, car vous n'y connaissez rien en plomberie. C'est exactement pareil sur le web.

L'immense majorité des échecs liés au web proviennent du fait que des personnes se lancent dans l'aventure sans la moindre connaissance de leur environnement.

N'oubliez pas qu'il est essentiel pour qu'un commerce fonctionne -et aujourd'hui plus que jamais- d'avoir une compréhension de son propre business, de son architecture et de son infrastructure.

Si vous faîtes l'effort d'acquérir ces connaissances, je vous garantis que vous avez d'ores-et-déjà battu 99% de vos concurrents.

Convaincu de l'utilité d'apprendre à coder ? Dans ce cas, passons à la suite ! Car je suis certain que vous êtes impatients de découvrir ce que signifient les initiales « HTML » et « CSS » !

Définition et utilisation du HTML et du CSS

Nous allons dans cette leçon poser une première définition du HTML et du CSS et comprendre plus précisément quel va être leur rôle dans la création d'un site Internet.

On va ainsi expliquer ce qu'on peut faire et ce qu'on ne peut pas faire avec chacun de ces deux langages.

Le HTML : langage de balisage

Le HTML est un langage qui a été créé en 1991. Les sigles « HTML » sont l'abréviation de « HyperText Markup Language » ou « langage de balisage hypertexte » en français.
Le HTML est donc un langage de balisage, c'est-à-dire un langage qui va nous permettre de définir les différents contenus d'une page.

Pourquoi est-il si important de définir les différents contenus d'une page ? Pour répondre à cette question, il va avant tout falloir que vous compreniez ce qu'est un site internet et ce qu'il se passe lorsque vous essayez d'accéder à un site internet.

Ici, je vais essayer de rester le plus simple possible. Tout d'abord, qu'est-ce qu'un site internet ? Un site internet est un ensemble de fichiers de code et de médias (images, vidéos, etc.) liés entre eux et disponibles à tout moment via le web.

Pour que les différentes ressources constituant un site internet soient toujours accessibles, on les héberge généralement sur un serveur d'un hébergeur professionnel comme OVH par exemple.

Un serveur est un ordinateur plus ou moins comme les nôtres et qui stocke les différentes ressources d'un ou de plusieurs site internet. Les seules différences entre un serveur et nos ordinateurs est qu'un serveur est généralement beaucoup plus puissant que nos ordinateurs, qu'il est (normalement) toujours connecté à Internet et qu'il va posséder quelques logiciels supplémentaires lui permettant d'exécuter certains codes.

Notez ici qu'on appelle ces ordinateurs des « serveurs » car leur seul et unique rôle va être de « servir » (au sens de « fournir ») des pages dès que quelqu'un va le leur demander.

Comment accède-t-on à une page d'un site internet ? Pour répondre à cette deuxième question, je vais m'aider d'un exemple. Imaginons que l'on souhaite accéder à la page d'accueil de mon site. Pour cela, on va taper www.pierre-giraud.com dans la barre de notre navigateur.

Lorsqu'on demande à accéder à une page d'un site internet, nous sommes ce qu'on appelle des « clients ».

Notre navigateur va contacter le serveur sur lequel est hébergée la page à laquelle on souhaite accéder et lui demander de renvoyer les différents éléments de la page : la page sous forme de code HTML et potentiellement les différents médias intégrés dans la page

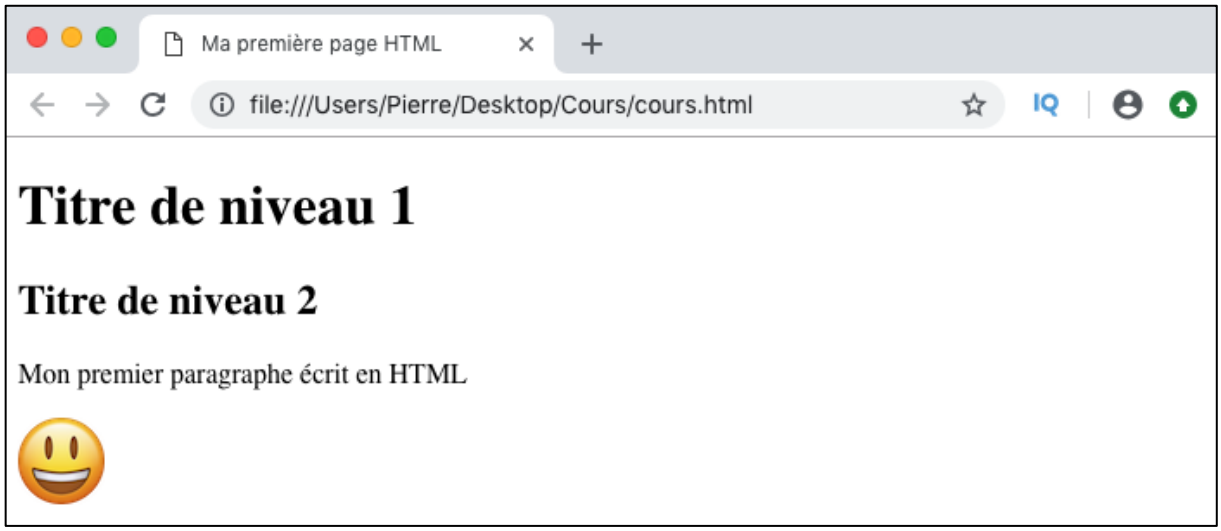
(ce qu'il se passe est en pratique bien évidemment plus complexe mais encore une fois je simplifie volontairement ici).

Le navigateur va donc recevoir ces différents éléments et les afficher. Cependant, comment fait-il pour savoir ce qu'il doit afficher ? Il va bien évidemment utiliser le code HTML. En effet, le navigateur comprend bien les différentes balises HTML (le HTML utilise ce qu'on appelle des « balises » pour définir les contenus) et va donc « comprendre » de quoi est constituée notre page et ce qu'il doit afficher.

Le rôle du HTML est donc crucial puisqu'il va être notre langage privilégié pour indiquer aux navigateurs ce quoi est constituée chaque page et ce qu'ils doivent afficher. Grâce au HTML, on va par exemple pourvoir indiquer que tel contenu est un texte qui n'est qu'un paragraphe, que tel autre contenu est un texte qui est un titre de niveau 1 dans notre page, que tel autre contenu est une liste, un lien, etc.

En plus de cela, le HTML va également nous permettre d'insérer différents médias (images, vidéos, etc.) dans nos pages web en indiquant au navigateur « à cette place-là dans ma page, je veux que s'affiche cette image ». Notez que dans ce cas précis, pour que le navigateur affiche la bonne image, on va lui fournir l'adresse de l'image dans le code HTML.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset = "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Titre de niveau 1</h1>
        <h2>Titre de niveau 2</h2>
        <p>Mon premier paragraphe écrit en HTML</p>

        <img src="smile.png" alt="Un emoji content">
    </body>
</html>
```



Le CSS : langage de styles

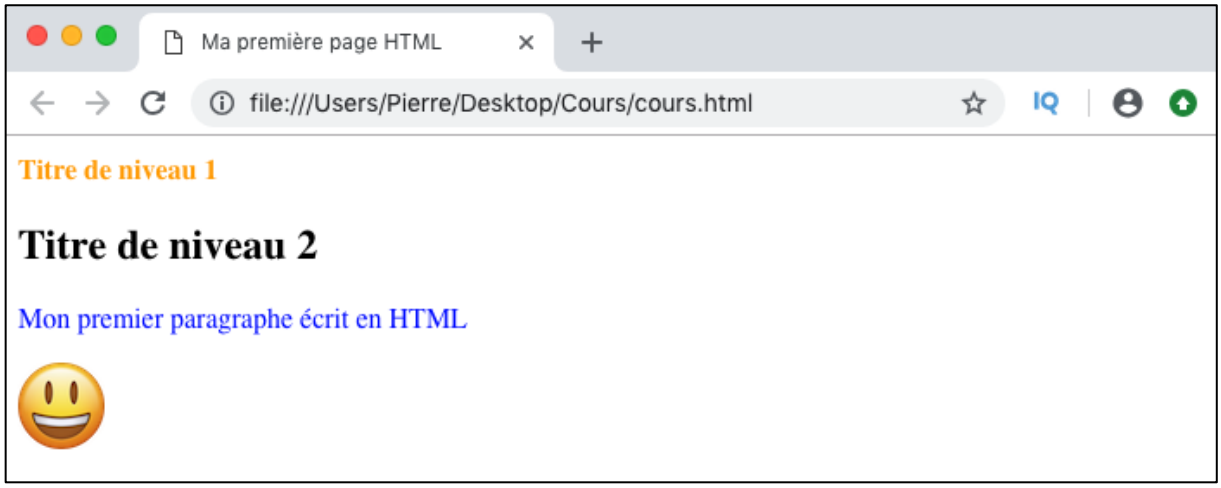
Le CSS a été créé en 1996, soit 5 ans après le HTML. Les sigles « CSS » sont l'abréviation de « Cascading StyleSheets » ou « feuilles de styles en cascade » en français.

Le CSS vient résoudre un problème bien différent du HTML : en effet, le HTML sert à définir les différents éléments d'une page, à leur donner du sens. Le CSS, lui, va servir à mettre en forme les différents contenus définis par le HTML en leur appliquant des styles.

Le HTML va donc créer la structure des pages tandis que le CSS va nous permettre de modifier l'apparence des contenus de la page. On va ainsi par exemple pouvoir définir la taille, la couleur ou l'alignement de certains contenus HTML et notamment en l'occurrence de certains textes dans notre page.

Pour styliser le contenu lié à un élément HTML en CSS, nous allons pouvoir le cibler en nous basant sur l'élément HTML en question ou en utilisant d'autres procédés que nous verrons plus tard dans ce cours.

```css
h1{
    font-size: 16px;
    color: orange;
}

p{
    font-size: 16px;
    color: blue;
}
```

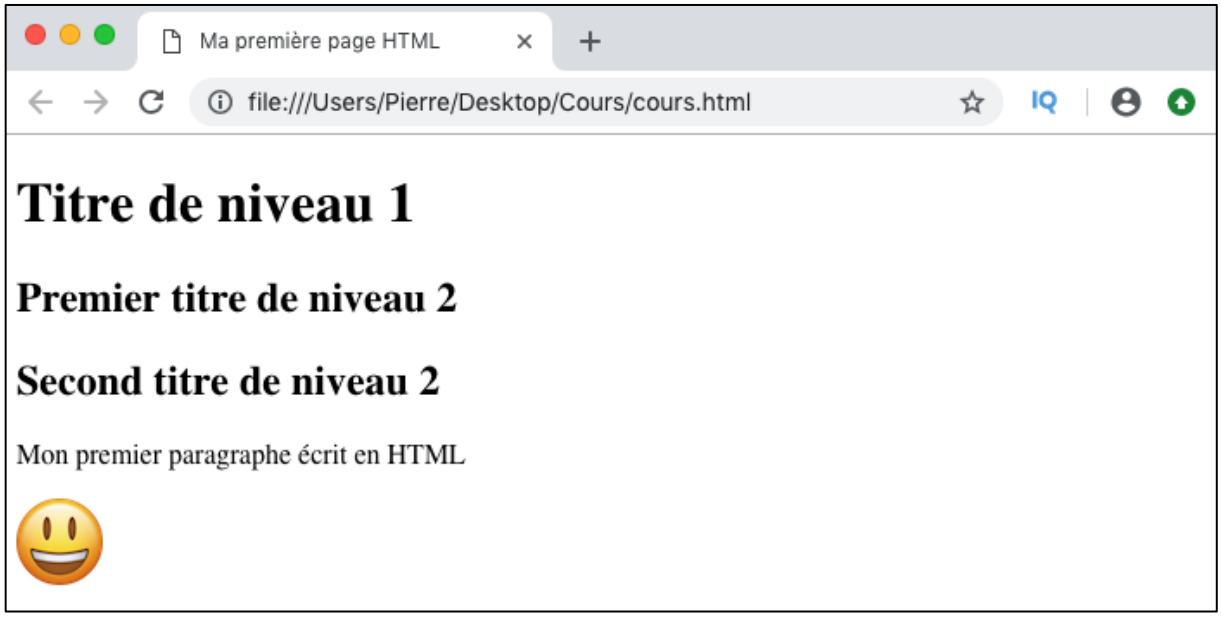


A retenir : n'utilisez pas le HTML pour mettre en forme vos contenus !

Voilà une chose que je vais vous répéter encore et encore au fil de ces premiers chapitres : vous ne devez jamais utiliser le HTML pour faire le travail du CSS.

En effet, si vous affichez la page en HTML créée ci-dessus dans votre navigateur sans y ajouter de CSS, vous pouvez remarquer que le contenu qui a été déclaré comme étant un titre s'affiche en grand et en gras, tandis que la taille du texte de notre paragraphe est plus petite et la police moins grasse.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset = "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Titre de niveau 1</h1>
        <h2>Premier titre de niveau 2</h2>
        <h2>Second titre de niveau 2</h2>
        <p>Mon premier paragraphe écrit en HTML</p>

        <img src="smile.png" alt="Un emoji content">
    </body>
</html>
```



Certains débutants en déduisent donc « pour mettre un texte en grand et en gras, il suffit d'utiliser une balise représentant un titre en HTML ». Sortez-vous cela immédiatement de la tête !

Le HTML est un langage qui a été créé pour structurer des pages et pour donner du sens au contenu. Le principe même du HTML est d'indiquer aux navigateurs que tel contenu est un texte qui doit être considéré comme un titre tandis que tel autre contenu est un texte qui doit être considéré comme un simple paragraphe et etc.

Si votre navigateur affiche par défaut les contenus textuels que vous avez déclaré comme des titres en HTML en grand et en gras et à l'inverse les paragraphes en plus petit c'est

justement parce que chaque navigateur possède sa propre feuille de styles (c'est-à-dire son propre code CSS) qui sera utilisé si aucun code CSS n'a été précisé de notre côté. Je répète ici car cela me semble très important : chaque navigateur a une feuille de style dans laquelle sont définis des styles pour chaque contenu HTML qui seront appliqués par défaut si aucun autre code CSS n'est fourni. Cela permet d'apporter une mise en forme minimale au contenu dans les cas rares où les styles CSS ne seraient pas définis ou si notre code CSS ne peut pas être chargé.

En fournissant nos styles CSS pour les différents contenus de notre page, nous allons donc pouvoir définir l'apparence de ceux-ci puisque les styles que nous allons fournir vont être considérés comme prioritaires par rapport à ceux du navigateur.

```css
h2:first-of-type{
    font-size: 16px;
    color: orange;
}
```

N'essayez pas forcément de comprendre comment fonctionne le code ci-dessus, ce n'est pas le but ici. La seule chose que vous devez comprendre est qu'on applique ici des styles CSS à notre premier contenu HTML placé entre les balises `<h2>` et `</h2>` (qui servent à définir un titre de niveau 2) mais pas au deuxième.

Pour notre premier titre de niveau 2, on indique qu'on veut une taille de texte de 16px et une couleur de texte orange et c'est donc ce que va afficher le navigateur. Comme on n'indique aucun style pour le deuxième titre de niveau 2, le navigateur va charger ses styles par défaut.

Note : Chaque version de chaque navigateur possède sa propre feuille de style, et c'est la raison pour laquelle une même page peut être rendue différemment d'un navigateur à un autre ou même d'une version d'un navigateur à une autre version de ce même navigateur ! Cependant, aujourd'hui, un véritable effort d'harmonisation a été fait et la plupart des navigateurs utilisent plus ou moins les mêmes styles par défaut.

En bref, retenez que vous ne devez jamais utiliser le HTML pour modifier l'apparence d'un contenu car cela est le rôle du CSS. Si vous faussez le tout en déclarant par exemple des titres qui n'en sont pas, vous pervertissez le rôle du HTML et cela va impacter fortement la qualité de votre page web, sa validité et votre référencement global.

Histoire et évolution de l'informatique et du web

L'objectif de cette leçon est de souligner la rapide évolution du domaine informatique et des langages web. Cela permet notamment de comprendre pourquoi en informatique il ne faut jamais rien prendre pour acquis et continuer à se documenter et à suivre l'évolution du domaine.

Nous allons également répondre aux questions suivantes : qui crée les langages web et décide de leur évolution ? Comment cette évolution est-elle intégrée dans le web ?

L'informatique, un domaine en constante et rapide évolution

L'informatique est un domaine qui évolue vite et dont les normes ne sont pas encore tout à fait fixées, à la différence de matières comme les mathématiques par exemple. En effet, il y a 20 ans de cela, le web était bien différent d'aujourd'hui et il sera certainement à nouveau très différent dans quelques années.

En effet, l'évolution des technologies et des composants physiques toujours plus performants soutiennent cette perpétuelle évolution.

Pour rappel, jusqu'au début des années 2000, les données étaient stockées sur des disquettes dont la capacité de stockage n'excédait pas les 1440 Ko c'est-à-dire le poids d'une image de qualité moyenne aujourd'hui et nous avions des modems de connexion avec une capacité de transmission des données de 64 Ko/s.

Avec le succès et la démocratisation des ordinateurs et d'internet un peu partout, cependant, les technologies n'ont cessé d'évoluer de plus en plus vite et les composants physiques sont devenus de plus en plus performants entre les années 2000 et 2010.

Or, avec plus de puissance dans nos ordinateurs et des connexions beaucoup plus rapides, de nouvelles opportunités se sont ouvertes dans le monde de l'informatique et du web en particulier.

Les langages informatiques, qui sont au centre du web, ont donc bien évidemment évolué en même temps que le champ des possibilités s'est ouvert, en s'enrichissant et en multipliant leurs fonctionnalités.

L'augmentation de la vitesse de connexion a entre autres permis de créer des sites web beaucoup plus complexes et d'intégrer de plus en plus de médias, comme de l'audio et de la vidéo ou des images de plus en plus lourdes, ce qui était inconcevable avant car ces médias auraient mis des jours à charger.

L'augmentation des performances des différents composants physiques a permis de réaliser des calculs toujours plus complexes côté serveur et de multiplier également les capacités de stockage de ceux-ci.

Langages de programmation : création et évolution

Pour bien comprendre comment sont créés et comment évoluent les différents langages de programmation, il faut avant tout savoir qu'il existe deux types de langages :

- Les langages dits « Open Source » : ce sont des langages qu'on va pouvoir utiliser sans licence et n'importe qui va (généralement) pouvoir participer à leur développement ;
- Les langages créés et gérés par des entreprises privées comme le langage Java qui appartenait à l'origine à la société Sun qui a ensuite été rachetée par la société Oracle.

Concernant les langages Open Source, comme je viens de le dire, n'importe qui peut en théorie contribuer à leur évolution en testant ou en fournissant des notes sur les sites officiels des groupes qui gèrent les différents langages.

En effet, bien qu'Open Source, des groupes de travail différents se sont constitués autour des différents langages. Deux groupes vont particulièrement nous intéresser dans le cadre du HTML et du CSS : le W3C ou « World Wide Web Consortium » et le WHATWG ou « Web Hypertext Application Technology Working Group ».

Pour vous donner un peu de contexte sans entrer dans la polémique, le W3C était à l'origine le groupe de référence chargé de définir et de veiller au développement des langages HTML et CSS.

Cependant, en 1998, le W3C décide d'abandonner l'évolution du HTML pour se concentrer sur un autre langage qui devait lui succéder : le XHTML.

Certaines personnes, mécontentes de l'inertie du W3C et du temps que prenait le développement de nouveaux standards et qui voyaient du potentiel dans le HTML ont alors décidé de continuer son évolution en formant un groupe nommé le WHATWG.

Face à l'échec du XHTML, en 2006, le W3C revient sur ses pas et exprime son intérêt pour travailler avec le WHATWG sur le HTML. Ce travail en commun a fonctionné jusqu'en 2011, date à laquelle l'écart entre les objectifs et les méthodes des deux groupes est devenu trop grand.

En effet, le WHATWG milite pour un « Living Standard » pour le HTML, c'est-à-dire des améliorations constantes et immédiatement intégrées tandis que le W3C préfère mettre à jour le HTML dès que beaucoup d'avancées ont été faites et après s'être assuré que chaque nouveau composant fonctionne parfaitement.

Ce qu'il faut bien comprendre ici est que le W3C est un groupe composé de plus de 300 grandes entreprises qui font du lobbysme et donc que ce groupe subit de grandes pressions. A l'inverse, le WHATWG est le résultat de l'effort commun de trois « entreprises du web » : Opera, Mozilla et Apple, ce qui le rend beaucoup plus flexible.

Aujourd'hui, on se retrouve donc dans une situation conflictuelle où le WHATWG avance beaucoup plus vite que le W3C et accuse le W3C de voler les différents standards créés par le WHATWG pour les porter à leur crédit.

Dans ce cours, je me baserai sur les recommandations du W3C puisque ce sont celles communément admises et puisqu'elles reprennent (c'est un fait) la majorité du travail du WHATWG, avec un peu de retard certes mais ce « retard » laisse le temps aux navigateurs d'intégrer les changements ce qui est finalement plutôt une bonne chose.

Le W3C définit trois états d'avancement pour ses différents documents qui vont chacun traiter d'un aspect d'un langage :

- Le statut de recommandation, ce qui veut dire que le document sert de standard et que le support pour les éléments ou pratiques qu'il définit doit être intégré par les navigateurs ;
- Le statut de candidat à la recommandation, ce qui signifie que les points abordés dans le document sont à priori fixés et attendent une dernière validation ;
- Le statut « travail en cours », ce qui signifie que le document possède encore des zones d'ombre à mettre au clair, etc.

Notez qu'il est bien évidemment également très important pour un développeur de se tenir au courant des nouveautés pour d'une part exploiter toutes les possibilités des langages et d'autre part savoir lorsque certains codes ou pratiques sont dépréciés, c'est-à-dire lorsque le support d'un certain élément de langage est abandonné pour pouvoir mettre à jour les parties du code d'un site les utilisant et éviter les bugs sur le site en question.

Concernant le langage CSS, c'est beaucoup plus simple puisque dans ce cas le W3C est le seul groupe qui s'occupe de son évolution et cela n'est contesté par personne.
Pour résumer, voici les groupes de travail faisant autorité sur les langages Open Source les plus populaires :

- HTML : W3C / WHATWG ;
- CSS : W3C ;
- JavaScript : ECMA ;
- PHP : PHP Group ;
- Python : Python Software Foundation.

Les versions actuelles du HTML et CSS

Jusqu'à récemment, les évolutions des langages HTML et CSS étaient « brutales » pour l'utilisateur final puisque le travail d'amélioration était effectué sur l'ensemble du langage avant d'être donné comme recommandation au public.

Ainsi, on est passé du HTML version 1 en 1991 au HTML version 2 en 1994 au HTML3 et etc. jusqu'au HTML5 en novembre 2014.

Concernant le CSS, nous sommes passés de la version 1 en 1996 au CSS2 en 1998 puis finalement au CSS2.1 en 2011.

La durée entre le CSS2 et le CSS2.1 s'explique par les nombreux allers-retours qui ont été faits entre les statuts de « candidat à la recommandation » et de « travail en cours » suite aux rejets consécutifs du W3C de valider la nouvelle version du langage comme recommandation.

Cela a permis de montrer les limites de ce système de travail sur l'ensemble du langage : il était même devenu quasiment impossible de procéder comme cela suite à la multiplication des fonctionnalités de chaque langage et le travail était mal organisé.

Sous la pression du WHATWG et pour pouvoir les concurrencer, le W3C a donc commencé à organiser le travail sur chaque langage par « modules ». Chaque module va concerner un aspect précis d'un langage et son développement va pouvoir avancer plus ou moins indépendamment des autres modules.

Par exemple, pour le CSS, nous avons un module de travail sur la couleur, un autre sur la police d'écriture, un autre sur la mise en page, etc.

Cela nous a ainsi permit de commencer à travailler sur l'évolution de chaque fonctionnalité d'un langage de manière indépendante.

Dès qu'un module est suffisamment avancé, on l'envoie comme candidat au statut de recommandation et il peut alors être validé comme nouveau standard (ou pas).

Dans ce cours, j'utiliserai toujours les derniers modules reconnus comme « recommandation » par le W3C. Certains possèdent l'étiquette « HTML5 », d'autres « HTML Living Standard » ou « CSS3 » ou « CSS4 ». Parfois, j'aborderai également certaines fonctionnalités qui ne sont que candidates à la recommandation mais qui vont très probablement obtenir le statut de recommandation dans un futur proche. Je vous l'indiquerai dans ce cas.

Local, préproduction et production

Dans cette leçon, nous allons distinguer le travail en local du travail en préproduction et en production et comprendre les enjeux, impacts et intérêts de chacune de ces différentes méthodes de travail.

Que signifie travailler en « local » ? Travailler en « production » ?

Lorsqu'on travaille sur un projet en informatique il existe deux façons de travailler : nous allons pouvoir travailler soit en local, soit en production.

Travailler en local signifie travailler sur et avec des fichiers enregistrés sur notre propre ordinateur. Comme toutes les ressources sont situées sur notre ordinateur, nous seuls allons pouvoir voir les résultats des différents codes que l'on crée.

Travailler en production, au contraire, signifie travailler sur des fichiers qui sont stockées sur un serveur web, c'est-à-dire modifier des ressources auxquels les utilisateurs peuvent accéder via internet.

Quand travailler en local ou en production ?

Imaginons que nous devions intervenir sur un site internet déjà existant et visité par des utilisateurs tous les jours.

En intervenant directement sur les fichiers sur le serveur, c'est-à-dire en production, nos modifications vont être immédiatement visibles par les utilisateurs.

Cela peut avoir deux impacts évidents : tout d'abord, selon les modifications effectuées, certains utilisateurs peuvent voir leur expérience sur le site altérée : imaginez un site internet qui est modifié en même temps que vous le visitez ! De plus, si jamais nous faisons une erreur sur notre code, c'est tout le site internet qui risque d'être vulnérable ou même inaccessible.

Pour ces raisons, un bon développeur préfèrera toujours tant que possible effectuer les modifications ou développer les nouvelles fonctionnalités en local, afin de limiter l'impact sur le site live.

Du local à la préprod, de la préprod à la prod, de la prod au local

Comment travailler en local sur un site internet déjà « en ligne », c'est-à-dire accessible par des visiteurs partout dans le monde ?

C’est finalement très simple : nous allons effectuer une copie du site, c’est-à-dire une copie de tous les fichiers et média constituant le site sur nos ordinateurs puis travailler sur cette copie.

Une fois les modifications effectuées et testées en local, nous allons les déployer sur serveur à un moment de fréquentation minimum ou allons placer le site en maintenance durant le temps de l’implémentation des modifications selon le degré d’importance de celles-ci.

Cette façon de travailler est la plus sûre et la meilleure dans le cas de site personnels ou de petits sites. Cependant, dans le cas de sites déjà importants et sur lesquels de nombreuses personnes travaillent, il va être très compliqué de travailler comme cela puisque le site risque de changer entre le moment de la copie et le moment où nos modifications sont terminées car d’autres développeurs auront eux-mêmes implémenté de nouvelles fonctionnalités.

Dans ce cas-là, l’entreprise aura tout intérêt à mettre en place ce qu’on appelle une pré-production ou « préprod ». L’idée ici va être de copier l’ensemble du site sur un autre serveur test qui ne sera accessible que pour certaines personnes (pour toutes les personnes de l’entreprise par exemple).

Cette solution, plus compliquée à mettre en place lorsqu’on est seul, permet de travailler sereinement à plusieurs sur plusieurs nouvelles fonctionnalités en parallèle et permet également de tester « en conditions réelles » l’impact des différentes modifications sur le site de base et sur les autres.

Pour ce cours, je pars du principe que vous n’avez pas encore de site et que vous travaillez seul. Nous travaillerons donc en local c’est-à-dire chacun avec des fichiers stockés sur nos propres ordinateurs.

Choix et installation de l'éditeur de texte

Nous en avons désormais fini avec la théorie et les définitions, et les chapitres qui vont suivre vont être l'occasion pour vous de commencer à pratiquer en codant en HTML et en CSS.

Mais avant tout, il va nous falloir mettre en place notre environnement de travail, c'est-à-dire réunir les différents éléments qui vont nous permettre de pouvoir coder en HTML et en CSS.

Pour coder en HTML et en CSS, cela va être très simple puisque nous n'aurons besoin que d'un éditeur de texte.

Qu'est-ce qu'un éditeur de texte ?

Un éditeur de texte est un programme qui va nous permettre d'écrire des lignes de code et de simplifier l'écriture de ce code en embarquant une panoplie de fonctionnalités utiles comme l'auto-complétion de certaines commandes de code et etc.

Un éditeur de texte ne doit pas être confondu avec un outil de traitement de texte comme Word. L'éditeur de texte a véritablement vocation à créer des pages de code dans n'importe quel langage comme le HTML, le CSS, le JavaScript ou même encore le PHP en utilisant du texte brut tandis qu'un outil de traitement de texte comme Word permet de créer du texte « enrichi » c'est-à-dire du texte mis en forme (choix de l'alignement, de la police, du poids de celle-ci, etc.).

Aujourd'hui, chaque système d'exploitation reconnu (Windows, Mac, Linux) embarque son ou ses propres éditeurs de texte nativement, ce qui signifie que votre ordinateur dispose déjà certainement d'un éditeur de texte préinstallé vous permettant de créer des pages de code en HTML, CSS, etc.

Comment bien choisir son éditeur de texte ?

Il existe aujourd'hui des milliers d'éditeurs disponibles sur le web.

Certains fonctionnent exclusivement sous certains environnements (Windows, Mac Os, etc.) tandis que d'autres vont être cross-plateformes (fonctionner sous plusieurs environnements). Certains éditeurs de texte vont être gratuits tandis que d'autres vont être payants ou disponibles sous forme d'abonnement. Le plus connu des éditeurs de texte est certainement Notepad++, le fameux éditeur fonctionnant avec Windows.

Un éditeur va donc nous permettre de pouvoir écrire des pages de code et de les enregistrer au bon format (c'est-à-dire avec la bonne extension). Un bon éditeur de texte est selon moi un éditeur qui va proposer différentes options pour vous aider à coder. Parmi ces options, on peut notamment citer :

- La mise en couleur des différents éléments d'un langage informatique pour pouvoir les distinguer (un bon éditeur de texte fera apparaitre les éléments HTML d'une couleur différente des attributs, etc.) ;
- L'affichage d'indications lorsque l'on fait une faute de syntaxe dans le code avec une explication du problème en question ;
- L'auto-complétion des balises (un bon éditeur écrira automatiquement la balise fermante d'un élément HTML par exemple dès que vous aurez écrit la balise ouvrante) ;
- Des bibliothèques (notamment jQuery) intégrées ou téléchargeables et installables en 1 clic ;
- Une indentation automatique et pertinente.

Aux vues de ce que nous allons réaliser durant ce cours, et pour un débutant, la plupart des éditeurs vont se valoir et vont parfaitement convenir. Je vous conseille donc simplement d'en trouver un compatible avec votre système et qui a déjà fait ses preuves. Personnellement, j'utiliserai pour ce cours la version gratuite du logiciel Komodo, c'est-à-dire Komodo Edit qui est multiplateformes (il fonctionne aussi bien sous Windows que Mac ou encore Ubuntu).

Vous pouvez télécharger la version gratuite de cet éditeur de texte à l'adresse suivante : https://www.activestate.com/products/komodo-ide/downloads/edit/ (***connexion Internet requise***) et commencer à coder immédiatement en HTML, CSS, etc. Faites bien attention à télécharger Komodo Edit et non pas Komodo IDE qui est la version payante du logiciel.

Une fois l'éditeur de votre choix installé, n'hésitez pas à l'ouvrir pour vous familiariser avec son affichage et découvrir les nombreuses options qu'il propose.

Un bon éditeur de texte devrait avoir un fond foncé lorsque vous créez une nouvelle page. Si ce n'est pas le cas, je vous conseille fortement de changer la couleur de fond (vous devrez certainement également changer la couleur de votre code afin d'y voir) pour coder avec un plus grand confort visuel.

Voici une liste d'autres éditeurs reconnus si jamais vous voulez davantage de choix :

- Microsoft Visual Code : Microsoft Visual Code est le dernier éditeur à la mode parmi les développeurs et ceci pour de bonnes raisons. Gratuit, il dispose de toutes les fonctionnalités qu'un développeur peut attendre et d'une très bonne ergonomie ;
- Atom : Atom est doté d'une excellente ergonomie qui facilite grandement la prise en main et l'approche du code pour les nouveaux développeurs. Cet éditeur de texte dispose de toutes les fonctions qu'on attend d'un bon éditeur : bibliothèques intégrées, auto-complétion des balises, etc.
- Notepad++ : Certainement l'éditeur de texte le plus connu de tous les temps, Notepad++ est également l'un des plus anciens. Il a passé le test du temps et a su s'adapter au fur et à mesure en ajoutant des fonctionnalités régulièrement comme l'auto-complétion des balises, le surlignage des erreurs de syntaxe dans le code etc. Le seul bémol selon moi reste son interface qui est à peaufiner.
- Brackets : Brackets est un éditeur très particulier puisqu'il est tourné uniquement vers les langages de développement front-end (c'est-à-dire HTML, CSS et JavaScript). Cependant, il dispose d'une excellente ergonomie (UI / UX) et d'un support extensif pour les langages supportés.

Logiciel éditeur de texte contre éditeur en ligne

Si vous vous intéressez à la programmation, vous connaissez peut-être les sites codepen.io ou jsbin.com ? Ces deux sites permettent d'écrire du code HTML, CSS ou JavaScript et de voir le résultat immédiatement.

En cela, ils servent le même rôle qu'un éditeur de texte HTML mais sont encore plus pratiques, notamment lorsque vous voulez tester rapidement un bout de code ou pour des démonstrations de cours.

Cependant, retenez bien qu'ils sont aussi limités car il y a plusieurs choses que vous ne pourrez pas faire en termes de développement avec ces sites. Parmi celles-ci, on notera que vous ne pourrez par exemple pas exécuter de code PHP ou un quelconque code utilisant un langage dit server-side ni créer plusieurs pages et les lier entre elles (comme c'est le cas lorsque l'on doit créer un site) ou du moins pas gratuitement.

Cette solution n'est donc pas satisfaisante si vous souhaitez véritablement vous lancer dans le développement et c'est la raison pour laquelle tous les développeurs utilisent un éditeur de texte.

Je vous conseille donc durant ce cours d'utiliser un maximum votre éditeur pour bien vous familiariser avec celui-ci et pour assimiler les différentes syntaxes des langages que l'on va étudier.

PARTIE II

Les bases du HTML

Éléments, balises et attributs HTML

Le HTML est un langage de balisage, c'est-à-dire un langage qui va servir à définir chaque contenu dans une page. Mais comment fait-on en pratique pour indiquer que tel contenu est un titre, tel autre est un lien, etc. ? C'est ce que nous allons voir dans cette leçon.

Les éléments HTML

Le langage HTML tout entier repose sur l'utilisation d'éléments. Si vous comprenez bien ce qu'est un élément, vous comprenez le HTML.

Les éléments HTML vont nous permettre de créer la structure d'une page HTML, de définir chaque contenu de notre page et également de passer certaines informations utiles au navigateur pour afficher la page (comme le type d'encodage utilisé par exemple pour que celui-ci affiche bien nos accents français).

Dans une page, nous allons utiliser les éléments en HTML pour marquer du contenu, c'est-à-dire pour lui donner du sens aux yeux des navigateurs et des moteurs de recherche. Selon l'élément utilisé, un navigateur va reconnaître le contenu comme étant de telle ou telle nature.

Ainsi, on va utiliser des éléments pour définir un paragraphe ou un titre par exemple, ou encore pour insérer une image ou une vidéo dans un document.

L'élément `p`, par exemple, sert à définir un paragraphe, tandis que l'élément `a` va nous permettre de créer un lien, etc.

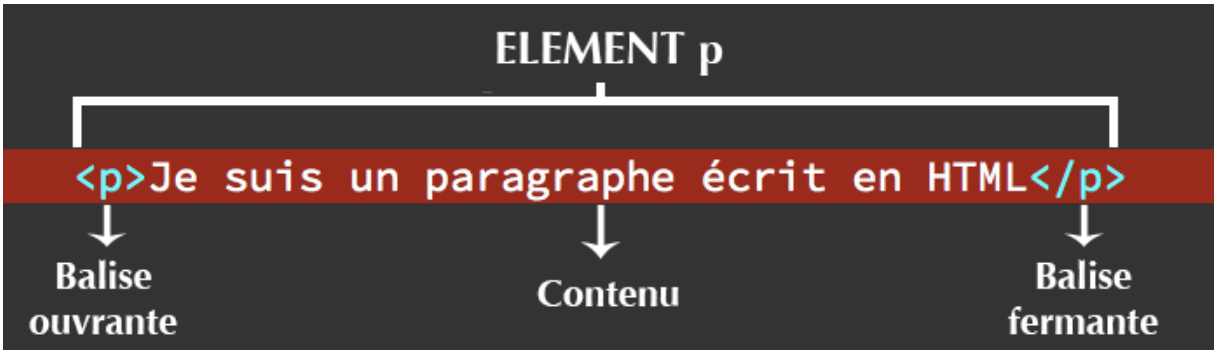
Aujourd'hui, il existe plus de 120 éléments HTML différents aux rôles très variés et qui font la richesse de ce langage. Nous allons étudier et nous servir d'une grande partie d'entre eux dans ce cours.

Les balises HTML

Un élément HTML peut être soit constitué d'une paire de balises (ouvrante et fermante) et d'un contenu, soit d'une balise unique qu'on dit alors « orpheline ».

L'élément `p` (qui sert à définir un paragraphe) est par exemple constitué d'une balise ouvrante, d'une balise fermante et d'un contenu textuel entre les balises. L'idée ici est que le texte contenu entre les deux balises va être le texte considéré par le navigateur comme étant un paragraphe.

Voici comment on va écrire cela :

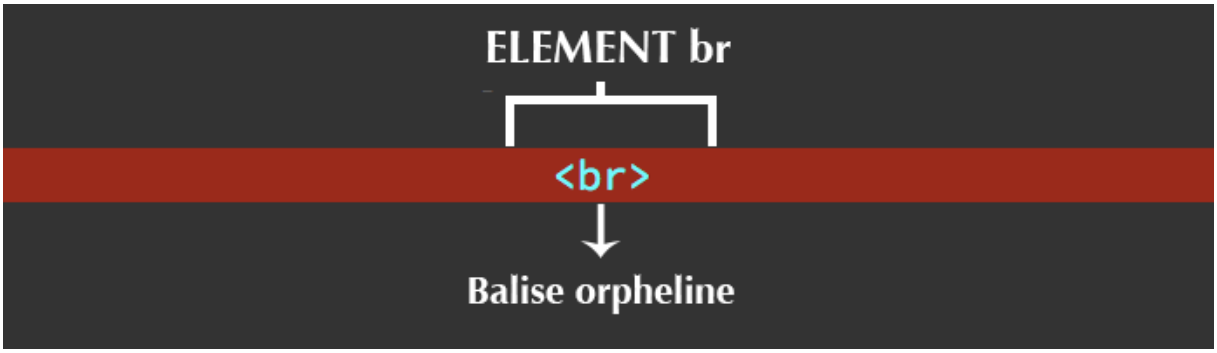


Comme vous pouvez le constater, la balise ouvrante de l'élément est constituée d'un chevron ouvrant <, du nom de l'élément en question et d'un chevron fermant >.

Notez bien ici la différence entre la balise ouvrante et la balise fermante de notre élément p : la balise fermante contient un slash avant le nom de l'élément.

Vous pouvez déjà retenir cette syntaxe qui sera toujours la même en HTML.

Certains éléments en HTML ne vont être constitués que d'une balise qu'on appelle alors orpheline. Cela va être le cas pour certains éléments qui ne possèdent pas de contenu textuel comme l'élément br par exemple qui sert simplement à créer un retour à la ligne en HTML et qui va s'écrire comme ceci :



Notez ici qu'il est possible que vous rencontriez un jour une syntaxe un peu différente pour les balises orphelines utilisant un slash après le nom de l'élément comme ceci : <br/>.

Cette syntaxe est une syntaxe qui était acceptée il y a quelques années mais qui est aujourd'hui dépréciée en HTML. Elle provient en fait d'un autre langage qui est le XML. Il est déconseillé de l'utiliser aujourd'hui en HTML puisqu'elle risque de ne plus être reconnue par les navigateurs dans le futur.

Par ailleurs, notez que certains développeurs ont tendance, par abus de langage, à confondre les termes « élément » et « balise » en utilisant le mot « balise » pour désigner un élément. Vous verrez peut-être cela écrit dans d'autres cours.

C'est toutefois un abus de langage et je pense qu'il est préférable pour une bonne compréhension d'appeler chaque objet d'un langage par son vrai nom. Je vous conseille donc de retenir plutôt ce que j'ai expliqué plus haut.

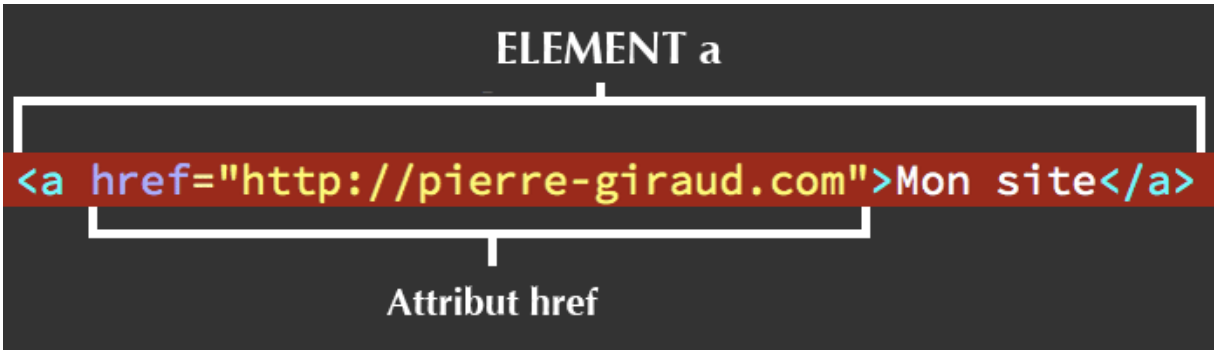
Les attributs HTML

Finalement, les éléments vont également pouvoir contenir des attributs qu'on va alors placer au sein de la balise ouvrante de ceux-ci. Pour certains éléments, les attributs vont être facultatifs tandis que pour d'autres ils vont être obligatoires pour garantir le bon fonctionnement du code HTML.

Les attributs vont en effet venir compléter les éléments en les définissant plus précisément ou en apportant des informations supplémentaires sur le comportement d'un élément.
Un attribut contient toujours une valeur, qu'on peut cependant parfois omettre dans le cas des attributs ne possédant qu'une seule valeur (car la valeur est alors considérée comme évidente).

Prenons ici l'exemple de l'élément `a` qui est l'abréviation de « anchor » ou « ancre » en français. Cet élément va principalement nous servir à créer des liens vers d'autres pages. Pour le faire fonctionner correctement, nous allons devoir lui ajouter un attribut `href` pour « hypertexte reference » ou « référence hypertexte » en français.

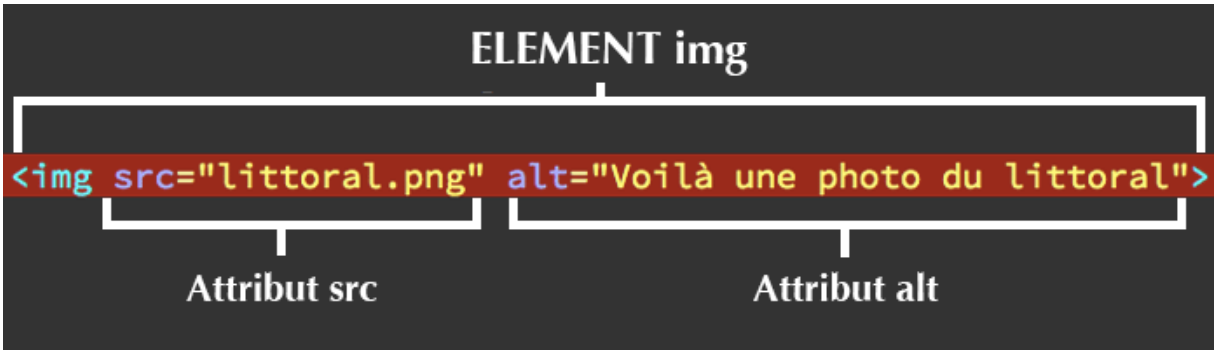
En effet, c'est l'attribut `href` qui va nous permettre de préciser la cible du lien, c'est-à-dire la page de destination du lien en lui passant l'adresse de la page en question en valeur.



Un autre exemple : l'élément `img`, servant à insérer une image dans une page HTML, va lui nécessiter deux attributs qui sont les attributs `src` et `alt`.

L'attribut `src` (pour « source ») va prendre comme valeur l'adresse de l'image tandis que l'attribut `alt` (pour « alternative ») va nous permettre de renseigner une description textuelle de l'image qui sera affichée dans les cas où l'image ne serait pas disponible pour une raison ou une autre : image introuvable, impossible à charger, etc.

L'attribut `alt` va également se révéler indispensable pour rendre notre site accessible aux non-voyants ou aux mal voyants et pour leur fournir une bonne expérience de navigation puisqu'ils sont généralement équipés de lecteur spéciaux qui vont pouvoir lire la valeur de l'attribut `alt` et donc leur permettre de se faire une représentation du contenu de l'image.



Notez au passage que l'élément `img` n'est constitué que d'une seule balise orpheline, tout comme l'élément `br` vu précédemment. On place dans ce cas les attributs au sein de la balise orpheline.

Structure minimale d’une page HTML valide

Le W3C, en définissant des standards de développement, a véritablement simplifié la tâche aux développeurs.

En effet aujourd’hui, tous les navigateurs sérieux suivent les standards proposés par le W3C, ce qui n’était pas forcément le cas dans le passé.

Cela nous permet donc d’être plus ou moins certain qu’un même code va produire le même résultat quel que soit le navigateur utilisé par les visiteurs qui consultent la page.

Cependant, cela n’est possible que parce que les développeurs eux-mêmes suivent certains schémas de construction de leur page qui sont anticipables et attendus par les navigateurs.

Le schéma de base d’une page HTML va donc toujours être le même. C’est ce que nous appellerons la « structure minimale d’une page HTML valide ».

Cette nouvelle leçon va être consacrée à la création d’une première page HTML dite « valide », c’est-à-dire qui respecte les standards, et va nous permettre de comprendre le rôle de chaque élément de cette structure minimale.

La structure minimale d’une page HTML

Pour qu’une page HTML soit déclarée valide, elle doit obligatoirement comporter certains éléments et suivre un schéma précis.

Vos pages HTML devraient toujours être valides pour les raisons évoquées ci-dessus. En effet, une page non valide ne sera pas comprise par le navigateur qui va alors potentiellement mal l’afficher voire dans certains cas ne pas l’afficher du tout.

De plus, une page non valide sera également mal analysée par les moteurs de recherche et ces mêmes moteurs de recherche risquent donc de ne pas la valoriser, c’est-à-dire de ne pas la proposer aux utilisateurs recherchant une information que votre page contient.

En d’autres termes, posséder des pages non valides risque d’impacter négativement le référencement de ces pages et de votre site en général dans les moteurs de recherche.

Voici ci-dessous le code minimum pour créer une page HTML valide. Nous allons dans la suite de ce chapitre expliquer le rôle de chaque élément et attribut.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>

    </body>
</html>
```

Le DOCTYPE

Tout d'abord, nous devrons toujours démarrer une page HTML en précisant le `doctype` de notre document. Comme son nom l'indique, le `doctype` sert à indiquer le type du document.

Faites bien attention à l'écriture du `doctype` : vous pouvez remarquer que la balise représentant le `doctype` commence par un point d'exclamation. Ceci est un cas unique.
Dans la balise de l'élément `doctype`, on va préciser le langage utilisé, à savoir le HTML dans notre cas.

Il est possible que vous trouviez encore sur certains sites une déclaration du `doctype` possédant une syntaxe différente et plus complexe comme celle-ci : `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`.

Cette écriture correspond à une ancienne syntaxe qui était utilisée jusqu'à la version 4 du HTML.

L'élément HTML

Après avoir renseigné le type du document, nous devons utiliser un élément `html`. Cet élément est composé d'une paire de balises ouvrante `<html>` et fermante `</html>`.

L'élément `html` va représenter notre page en soi. On va insérer tout le contenu de notre page (et donc les autres éléments) à l'intérieur de celui-ci.

Les éléments head et body

A l'intérieur de l'élément `html`, nous devons à nouveau indiquer obligatoirement deux éléments qui sont les éléments `head` et `body` et qui vont avoir des rôles très différents.

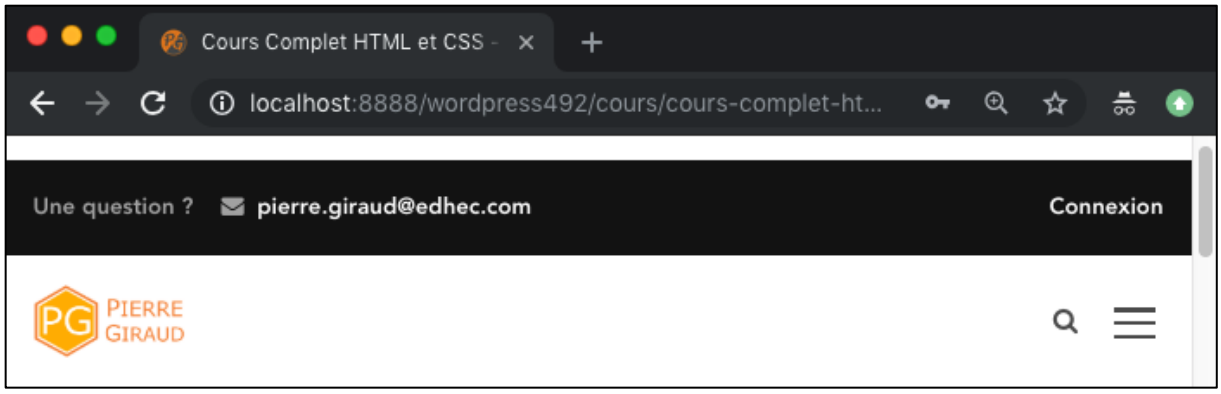
L'élément `head` est un élément d'en-tête. Il va contenir des éléments qui vont servir à fournir des informations sur la page au navigateur, comme le titre de la page ou encore le type d'encodage utilisé pour que celui-ci puisse afficher les caractères de texte correctement.

L'élément `body` va lui contenir tous les éléments définissant les contenus « visibles » de la page, c'est-à-dire les contenus à destination de l'utilisateur et notamment les différents textes présents dans la page, les images, etc.

Les éléments title et meta

Au sein de l'élément `head`, nous allons devoir à minima indiquer deux éléments qui vont permettre de donner des informations essentielles sur la page au navigateur : les éléments `title` et `meta`.

L'élément `title` va nous permettre d'indiquer le titre de la page en soi, qui ne doit pas être confondu avec les différents textes définis comme des titres dans la page. Ce titre de page est le texte visible sur le haut des onglets de votre navigateur par exemple :



L'élément `meta` sert lui à transmettre des meta informations sur la page au navigateur. Cet élément possède de nombreux attributs différents. Le type d'informations transmis va dépendre de l'attribut que l'on va préciser.

Ici, ce qui nous intéresse va être de préciser le type d'encodage utilisé dans nos pages. Cela va permettre aux navigateurs d'afficher correctement nos différents textes avec les accents, les cédilles, etc.

Pour faire cela, nous allons utiliser l'attribut `charset` pour « characters set » c'est-à-dire « ensemble de caractères » de l'élément `meta` et lui fournir la valeur `utf-8`.

La valeur `utf-8` est de loin la valeur la plus utilisée sur le web et est la valeur de référence pour tous les alphabets latins. Cela va permettre à chacun de nos caractères de s'afficher correctement dans le navigateur.

Note : Lorsque vous travaillez en local vous devrez, avec certains éditeurs, renseigner également l'encodage de la page dans l'éditeur afin que le contenu de celle-ci s'affiche bien et notamment si vous utilisez un outil de prévisualisation de page fourni par votre éditeur.

Voilà tout pour la structure minimale d'une page HTML valide, qui représente toujours la première étape de création d'une vraie page web. Pour le moment, notre page ne possède pas de contenu visible. Nous allons en ajouter progressivement au cours des prochaines leçons.

Un mot sur l'imbrication des balises et des éléments

Un autre concept qu'il vous faut comprendre absolument pour coder en HTML est la façon dont les éléments HTML s'imbriquent les uns dans les autres.

Vous l'avez probablement remarqué : ci-dessus, nous avons placé des éléments HTML entre les balises ouvrantes et fermantes d'autres éléments (par exemple, les éléments `title` et `meta` ont été placés à l'intérieur de l'élément `head`).

On appelle cela l'imbrication. L'imbrication est l'une des fonctionnalités du HTML qui fait toute sa force (nous découvrirons réellement pourquoi plus tard, avec l'étude du CSS).

Cependant, comme toujours, on ne peut pas imbriquer des éléments HTML n'importe comment et il faut suivre des règles précises.

Ainsi, nous n'avons absolument pas le droit de « croiser » les balises des éléments ou, pour le dire plus clairement : le premier élément déclaré doit toujours être le dernier refermé, tandis que le dernier ouvert doit toujours être le premier fermé.

Par exemple, vous voyez que notre premier élément déclaré est l'élément `html`, qui contient les éléments `head` et `body`. L'élément `html` devra donc être le dernier élément refermé.

Faites bien attention à distinguer les éléments contenus dans d'autres et ceux qui sont au même niveau hiérarchique. Par exemple, ici, `title` et `meta` sont deux éléments « enfants » de l'élément `head` (car ils sont contenus à l'intérieur de celui-ci), mais sont deux « frères » : aucun des deux ne contient l'autre.

Le schéma ci-dessous présente toutes les situations valides d'imbrication d'éléments :

```
<balise ouvrante élément A>

        <balise ouvrante élément B>
        </balise fermante élément B>

        <balise ouvrante élément C>
        </balise fermante élément C>

        <balise orpheline élément D>

</balise fermante élément A>
```

Pour savoir quand un élément doit être placé dans un autre, il faut penser en termes de « dépendance » : les informations transmises par l'élément dépendent-elle d'un autre ou sont-elles totalement indépendantes ? Ce concept peut être complexe à se représenter lorsqu'on début à peine, mais ne vous inquiétez pas en pratiquant un peu tout cela va très vite devenir évident.

Enregistrement et affichage d'une page HTML

Nous avons défini la structure minimale d'une page HTML valide lors de la leçon précédente et l'importance de produire un code valide. Il est maintenant temps de voir comment enregistrer notre fichier pour pouvoir afficher le résultat de notre page dans un navigateur.

L'enregistrement de notre fichier

Maintenant que nous avons établi les bases, nous allons pouvoir commencer à coder ensemble.

Pour cela, commencez par ouvrir votre éditeur de texte puis créez un nouveau fichier vide. Par défaut, votre fichier devrait être un fichier au format `.txt`, c'est-à-dire un simple fichier texte.

La plupart des éditeurs de texte proposent aujourd'hui des fonctionnalités nous aidant à coder plus rapidement et à créer un meilleur code.

Parmi ces fonctionnalités, on peut notamment citer l'auto-complétion des balises HTML qui correspond à l'écriture automatique de la balise fermante d'un élément dès la déclaration de la balise ouvrante (pour un élément constitué d'une paire de balises) ou encore la vérification automatique de la validité du code écrit.

Pour que ces fonctionnalités s'activent, il va cependant falloir que notre éditeur de texte sache quel langage de code est utilisé dans notre page.

Pour le lui indiquer, nous avons deux façons de procéder : on peut soit déjà enregistrer notre fichier avec la bonne extension c'est-à-dire ici `.html`, soit préciser le langage utilisé dans l'un des onglets de l'éditeur (si vous cherchez bien, vous devriez voir quelque part une liste de langages informatiques proposés).

Ici, nous allons utiliser la méthode la plus simple qui va être d'enregistrer notre fichier. Pour cela, allez dans l'onglet « fichier » puis « enregistrer sous » ou utilisez le raccourci CMD+SHIFT+S (pour Mac) ou CTRL+SHIFT+S (pour Windows).

Pour la suite de ce cours, je vous propose de créer un dossier qu'on va appeler « cours » sur votre bureau. Nous allons enregistrer nos différents fichiers dans ce dossier.

Il y a quelques règles à respecter lorsqu'on enregistre un fichier de code : le nom du fichier ne doit pas contenir d'espace ni de caractères spéciaux (pas de caractères accentués ou de ponctuation ni de sigles) et doit commencer par une lettre. Ici, on peut enregistrer notre fichier sous le nom `cours.html` par exemple pour faire très simple.

Dès que votre fichier est enregistré au bon format, nous allons retourner dans notre éditeur et écrire la structure minimale d'une page HTML. Notre éditeur sait maintenant qu'on écrit en HTML et va donc certainement utiliser une palette de couleurs, l'auto-complétion, etc. pour nous aider à coder.

Ensuite, nous allons ajouter un grand titre dans notre page et un paragraphe en utilisant respectivement les éléments `h1` et `p`.

Dès que votre page est prête, nous allons l'enregistrer à nouveau pour ensuite l'afficher dans notre navigateur.

Afficher le résultat d'un code HTML dans le navigateur

Pour afficher notre fichier dans le navigateur, nous allons aller le chercher dans notre dossier puis effectuer un clic droit dessus et choisir « ouvrir avec… » + le navigateur de votre choix.

Voici le résultat que vous devriez avoir :

Une petite astuce ici : si jamais c'est votre éditeur qui est le programme choisi par défaut pour ouvrir votre fichier de code HTML et non pas votre navigateur, je vous invite à définir le navigateur comme programme par défaut. Ainsi, vous n'aurez plus qu'à double cliquer sur le fichier en question pour qu'il s'ouvre automatiquement dans le navigateur.

L’indentation et les commentaires en HTML

Avant d’aller plus loin dans notre apprentissage du HTML et du CSS, il me semble intéressant de déjà vous parler de certaines pratiques de développeurs qui sont appelées des « bonnes pratiques » afin que vous puissiez rapidement les assimiler et qu’elles deviennent rapidement des automatismes.

Les bonnes pratiques sont là pour nous aider à coder plus proprement et plus lisiblement, donc pour nous faciliter la vie, pas le contraire !

Nous allons dans cette partie parler d’indentation et de commentaires HTML.

Retours à la ligne et indentation

Indenter correspond à créer des retraits en début de ligne dans votre éditeur de façon cohérente et logique.

Vous l’avez certainement remarqué dans les codes précédents : j’accentue plus au moins le retrait à gauche de chacune de mes lignes de code.

Indenter va nous permettre d’avoir un code plus propre et plus lisible, donc plus compréhensible. Indenter permet également de plus facilement détecter les erreurs potentielles dans un code.

Le but de l’indentation est de nous permettre de discerner plus facilement les différents éléments ou parties de code. Regardez plutôt les deux exemples suivants.

Sans indentation :

```
<!DOCTYPE html>
<html>
<head>
<title>Ma première page HTML</title>
<meta charset="utf-8">
</head>
<body>
<h1>Je viens d'écrire un titre en HTML !</h1>
<p>Et voilà mon premier paragraphe :)</p>
</body>
</html>
```

Avec indentation :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Je viens d'écrire un titre en HTML !</h1>
        <p>Et voilà mon premier paragraphe :)</p>
    </body>
</html>
```

J'espère que vous serez d'accord pour dire que le second code est plus clair que le premier.

Concernant l'indentation, il n'y a pas de règle absolue, notamment par rapport à la taille du retrait de chaque ligne et différents éditeurs peuvent d'ailleurs posséder différentes règles par défaut sur ce sujet.

Pour ma part, je me contente de retourner à la ligne et d'effectuer une nouvelle tabulation à chaque fois que j'ouvre un nouvel élément dans un autre, ce qui correspond sur mon éditeur à un retrait de 4 espaces.

Par exemple, ci-dessus, vous pouvez voir que j'ai utilisé une tabulation avant d'écrire la balise ouvrante de mon élément `head` qui est contenu dans mon élément `html`.

J'ai ensuite effectué une nouvelle tabulation à l'intérieur de l'élément `head` pour écrire mon élément `title`.

En revanche, je n'ai pas créé de nouvelle tabulation pour l'élément `meta` étant donné que cet élément n'est pas contenu dans `title` (ce n'est pas un élément enfant de l'élément `title`) mais est au même niveau hiérarchique que lui (on parle d'élément frère).

Vous pouvez observer la même chose avec les éléments `head` et `body` par rapport à l'élément `html` : j'ai décalé `head` et `body` avec le même écart par rapport à `html` car ces deux éléments sont frères (aucun des deux ne contient / n'est contenu dans l'autre).

Au final, indenter permet de créer une hiérarchie dans le code et l'un des buts principaux de l'indentation en HTML va être de comprendre très rapidement quels éléments sont imbriqués dans quels autres dans notre page pour avoir une meilleure vue d'ensemble de notre code et faciliter sa compréhension.

Note : Les retours à la ligne et l'indentation créés dans l'éditeur n'affectent pas le résultat final dans le navigateur. Nous aurons l'occasion de reparler de cela un peu plus tard dans ce cours.

Définition et utilité des commentaires en HTML

Les commentaires vont être des lignes de texte que l'on va écrire au milieu de notre code, afin de donner des indications sur ce que fait le code en question.

Les commentaires ne seront pas affichés par le navigateur lorsque celui-ci va afficher la page : ils ne vont servir qu'aux développeurs créant ou lisant le code.

Les commentaires vont pouvoir être très utiles dans trois situations :

1. Dans le cas d'un gros / long projet, afin de bien se rappeler soi-même pourquoi nous avons écrit tel ou tel code, ou encore pour se repérer dans le code ;
2. Si l'on souhaite distribuer son code, ou si l'on travaille à plusieurs, cela fait beaucoup plus professionnel et permet aux autres développeurs de comprendre beaucoup plus rapidement et facilement le code distribué ;
3. Pour « neutraliser » une certaine partie d'un code sans toutefois le supprimer. Il suffit en effet de placer toute la partie du code en question en commentaire afin que celui-ci soit ignoré par le navigateur.

Syntaxe et écriture des commentaires en HTML

Les commentaires peuvent être mono-ligne (c'est-à-dire écrits sur une seule ligne dans l'éditeur) ou multi-lignes (écrits sur plusieurs lignes dans l'éditeur).

A la différence de nombreux autres langages informatiques utilisant également des commentaires, la syntaxe utilisée pour créer un commentaire mono-ligne va être la même que celle pour créer un commentaire multi-lignes en HTML.

Voici comment s'écrit un commentaire en HTML :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <!--Deux titres h1 et h2-->
        <h1>Mon titre principal</h1>
        <h2>Je suis un titre important</h2>

        <!--Deux paragraphes différents-->
        <p>Voici mon premier paragraphe.</p>
        <p>Et en voilà un second !</p>
    </body>
</html>
```

Les commentaires en HTML vont prendre la forme d'une balise orpheline très particulière, avec un chevron ouvrant suivi d'un point d'exclamation suivi de deux tirets au début, du commentaire en soi puis à nouveau de deux tirets et d'un chevron fermant (sans point d'exclamation cette fois, attention !).

Même si vous ne voyez pas forcément l'intérêt de commenter dès maintenant, je vous garantis que c'est souvent ce qui sépare un bon développeur d'un développeur moyen. Faites donc l'effort d'intégrer cette bonne pratique dès à présent, cela vous sera bénéfique dans le futur.

Mise en garde : ne placez pas d'informations sensibles en commentaire !

Vous vous rappelez lorsque je vous ai dit que vos commentaires n'étaient pas affichés aux yeux de vos visiteurs ?

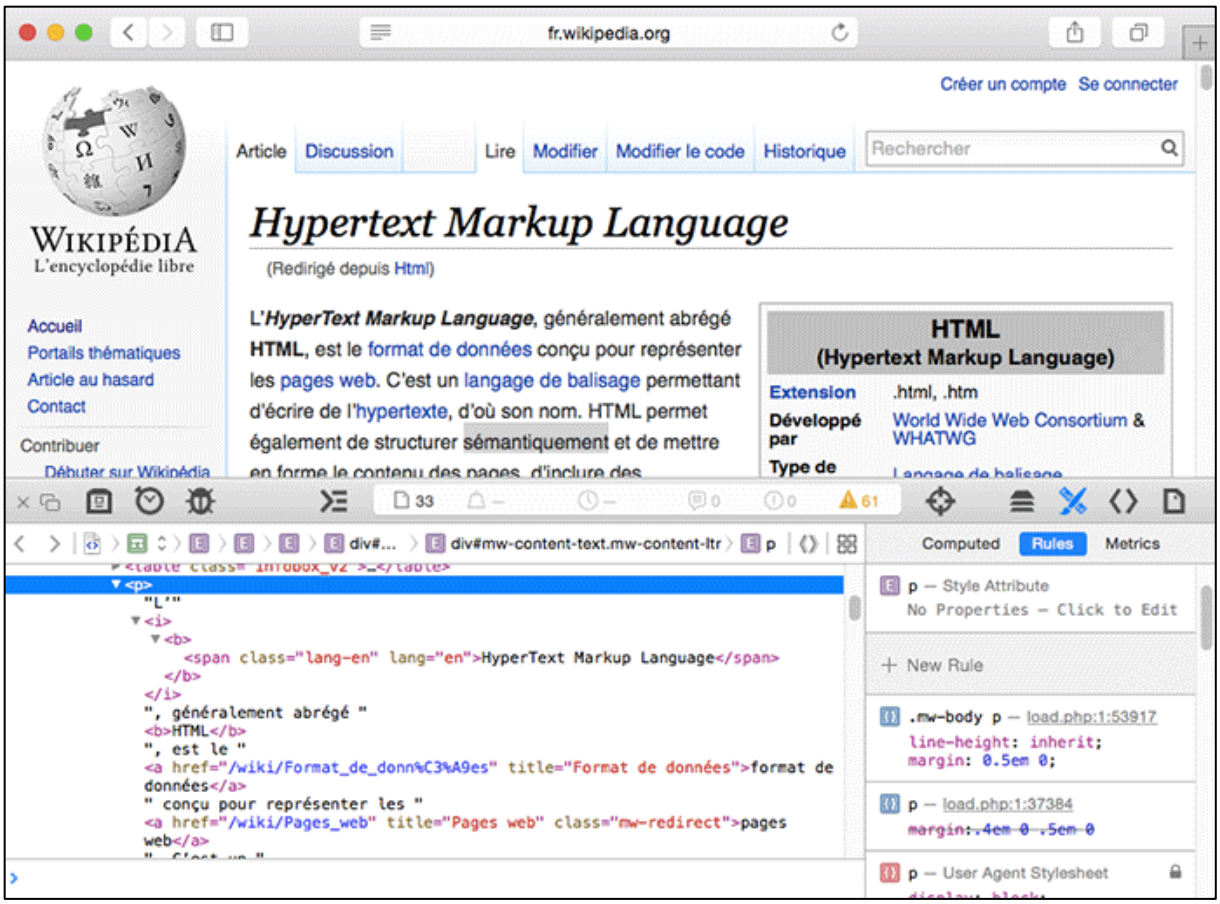
C'est vrai, mais faîtes bien attention, car cela ne veut pas dire qu'il est impossible pour un visiteur de voir ce que vous avez écrit en commentaire.

En effet, tous vos visiteurs peuvent voir à tout moment, s'ils le souhaitent, non seulement le contenu de vos commentaires mais aussi l'intégralité de votre code HTML.

Pour cela, il suffit simplement d'activer les outils pour développeurs dont dispose chaque navigateur puis de faire un clic droit sur la page et « d'inspecter l'élément ».

Cela est valable pour n'importe quel site. Regardez par exemple avec le site Wikipédia. Une fois les outils pour développeurs activés, il suffit d'un clic droit…

…et le contenu HTML de la page est affiché dans une console !



Faîtes donc bien attention à ne jamais écrire d'informations sensibles au sein de vos commentaires, comme des codes ou mots de passe par exemple.

Titres et paragraphes en HTML

Dans cette leçon et dans les suivantes nous allons étudier quelques-uns des éléments HTML les plus utiles et les plus utilisés. Cela va également nous permettre de bien nous familiariser avec la syntaxe du HTML grâce à des exemples d'application.

Dans cette leçon, nous allons en particulier nous concentrer sur la définition de titres et de paragraphes en HTML.

Pourquoi différencier titres et paragraphes en HTML ?

Une nouvelle fois, rappelez-vous que le HTML est un langage de balisage.

Le seul et unique rôle du HTML est de nous permettre de créer un document structuré en différenciant d'un point de vue sémantique les différents éléments d'une page.

Utiliser les bons éléments HTML pour définir précisément les différents contenus de nos pages va permettre aux navigateurs (et aux moteurs de recherche) de comprendre de quoi et comment est composée notre page et donc de l'afficher et de la référencer au mieux.

Or, les titres font partie des éléments auxquels les moteurs de recherche, entre autres, vont apporter une grande importance pour comprendre le sujet de notre page. En effet, Google par exemple va se servir des contenus définis comme titres pour comprendre de quoi notre page traite en y accordant plus d'importance qu'aux contenus définis comme des paragraphes.

Définition de titres en HTML

Il existe six niveaux hiérarchiques de titres ("heading" en anglais) définis par les éléments `h1`, `h2`, `h3`, `h4`, `h5` et `h6` qui vont nous permettre d'organiser le contenu dans nos pages.

Bon à savoir : « h » signifie « heading », soit l'équivalent du mot « titre » en français. Les éléments en HTML portent souvent l'initiale de ce qu'ils représentent, en anglais.

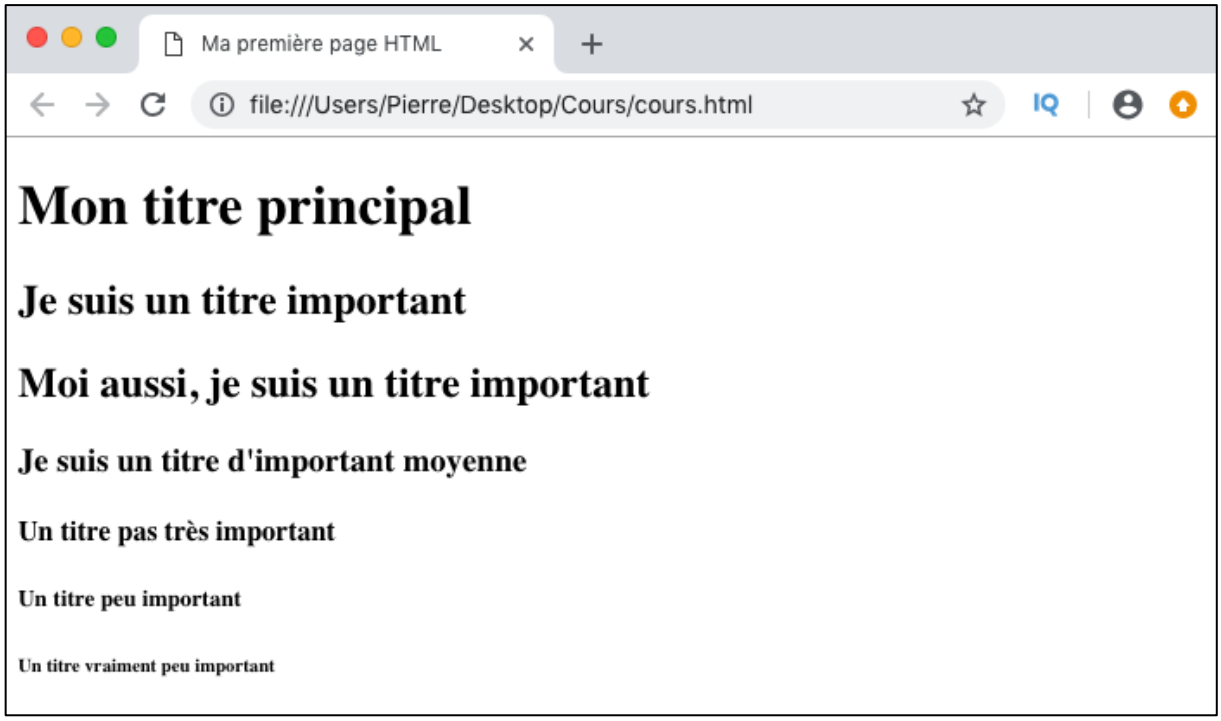
L'élément `h1` représente un titre principal dans notre page ou dans une section de page et, à ce titre, nous n'allons pouvoir utiliser qu'un seul élément `h1` par page (ou par section de page, nous reviendrons sur ce concept plus tard).

Ici, je tiens d'ores-et-déjà à attirer votre attention sur un point : si vous déclarez plusieurs titres `h1` dans votre page, ceux-ci s'afficheront bien : il n'y aura aucun blocage au niveau de l'éditeur ou du navigateur. Cependant, rappelez-vous une nouvelle fois que le HTML est un langage de sémantique et que sa bonne utilisation repose donc sur des règles et un ensemble de bonnes pratiques.

Dans le cas présent, utiliser plusieurs `h1` serait aberrant car le `h1` est censé représenter le titre principal de la page.

En revanche, nous allons pouvoir utiliser autant de titres de niveau `h2`, `h3` etc. que l'on souhaite dans notre page. Théoriquement, si nos pages sont bien construites, nous ne devrions que rarement dépasser le niveau de titre `h3`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <!--Un titre très important-->
        <h1>Mon titre principal</h1>

        <!--Un titre important-->
        <h2>Je suis un titre important</h2>

        <!--Un autre titre important-->
        <h2>Moi aussi, je suis un titre important</h2>

        <!--Un titre un peu moins important-->
        <h3>Je suis un titre d'important moyenne</h3>

        <!--Etc, etc.-->
        <h4>Un titre pas très important</h4>
        <h5>Un titre peu important</h5>
        <h6>Un titre vraiment peu important</h6>
    </body>
</html>
```



Comme vous pouvez le voir, le navigateur comprend bien qu'il s'agit de titres d'importances diverses et les traite donc différemment par défaut, en leur attribuant une mise en forme différente (très grand et très gras pour un titre de niveau `h1`, puis de plus en plus petit jusqu'à `h6`).

Encore une fois, n'utilisez pas un élément de type `h*` pour écrire un texte en grand et en gras ! Utilisez le pour définir un titre dans votre page. Nous nous chargerons de la mise en forme du contenu plus tard, grâce au CSS.

Note : Il convient de ne pas confondre les éléments `h*` et l'élément `title` : l'élément `title` sert à donner un titre A notre page tandis que les éléments `h*` servent à définir des titres DANS notre page, c'est-à-dire à hiérarchiser et à organiser le contenu de notre page.

Définition de paragraphes en HTML

Pour créer des paragraphes en HTML, nous allons utiliser l'élément `p`.

On peut créer autant de paragraphes que l'on souhaite dans une page. A chaque nouveau paragraphe, il faut utiliser un nouvel élément `p`.

Pour chaque nouveau paragraphe, un retour à la ligne va être créé automatiquement et affiché par votre navigateur (exactement comme c'était le cas avec les titres).

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <!--Deux titres h1 et h2-->
        <h1>Mon titre principal</h1>
        <h2>Je suis un titre important</h2>

        <!--Deux paragraphes différents-->
        <p>Voici mon premier paragraphe.</p>
        <p>Et en voilà un second !</p>
    </body>
</html>
```


Ma première page HTML
file:///Users/Pierre/Desktop/Cours/cours.html
Mon titre principal
Je suis un titre important
Voici mon premier paragraphe.
Et en voilà un second !

Espaces et retours à la ligne en HTML

Dans cette nouvelle leçon, nous allons voir comment déclarer des espaces et des retours à la ligne dans notre code HTML qui vont être conservés dans le rendu de la page fait par le navigateur.

Les espaces et les retours à la ligne dans le code ne sont pas rendus

Les plus curieux d'entre vous auront, je suppose, déjà fait le test : vous pouvez ajouter autant d'espaces que vous le voulez au sein d'un paragraphe ou d'un titre ou effectuer des retours à la ligne dans votre code, ceux-ci ne seront jamais affichés visuellement dans votre navigateur.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <!--Les espaces ne seront pas affichés-->
        <h1>Mon titre                    principal</h1>
        <h2>Je suis       un       titre important</h2>

        <p>Voici
            mon
            premier
            paragraphe.</p>
        <p>Et en voilà un second !</p>
    </body>
</html>
```

En effet, pour effectuer des retours à la ligne ou marquer des espaces en HTML, nous allons à nouveau devoir utiliser des éléments.
Pensez bien à nouveau que le navigateur va simplement interpréter votre code pour afficher un résultat et pour cela il va se baser sur les éléments HTML fournis.

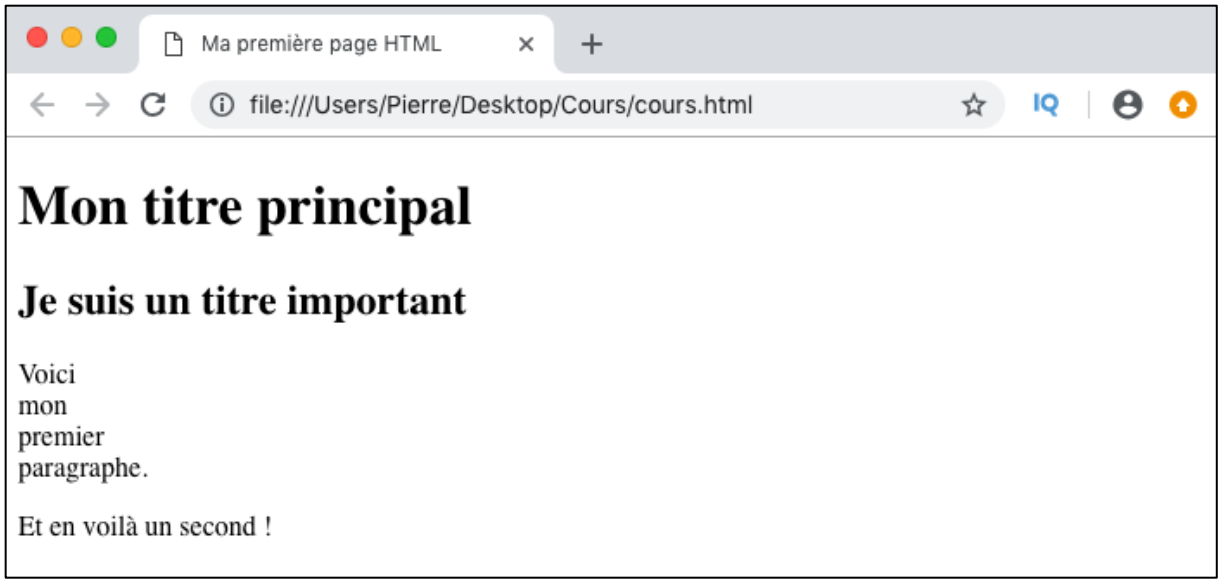
Les retours à la ligne en HTML

Pour effectuer un retour à la ligne en HTML nous allons devoir utiliser l'élément `br` qui est représenté par une unique balise orpheline.

Le nom de l'élément `br` est l'abréviation de « break », l'équivalent de « casser » en anglais (dans le sens de « casser une ligne »).

On peut utiliser autant d'éléments `br` que l'on souhaite au sein d'un titre ou d'un paragraphe par exemple.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <!--Les espaces ne seront pas affichés-->
        <h1>Mon titre                    principal</h1>
        <h2>Je suis       un       titre important</h2>

        <p>Voici <br>mon <br>premier <br>paragraphe.</p>
        <p>Et en voilà un second !</p>
    </body>
</html>
```



Les changements de thématique en HTML

Parfois, au sein d'un article par exemple, vous aborderez des thématiques différentes que vous voudrez séparer.

Dans ce cas-là, il peut être intéressant d'utiliser l'élément `hr` plutôt que `br`. L'élément `hr` a justement été créé pour définir un retour à la ligne avec changement de thématique. Tout comme `br`, cet élément est représenté par une balise orpheline.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <!--Les espaces ne seront pas affichés-->
        <h1>Mon titre                    principal</h1>
        <h2>Je suis       un       titre important</h2>

        <!--hr marque un changement de thématique-->
        <p>HTML : le HTML est un langage utilisé pour marquer
                  sémantiquement un contenu.</p>
        <hr>
        <p>CSS : le CSS sert à mettre en forme des éléments en
                 leur appliquant des styles.</p>
    </body>
</html>
```

La gestion des espaces en HTML

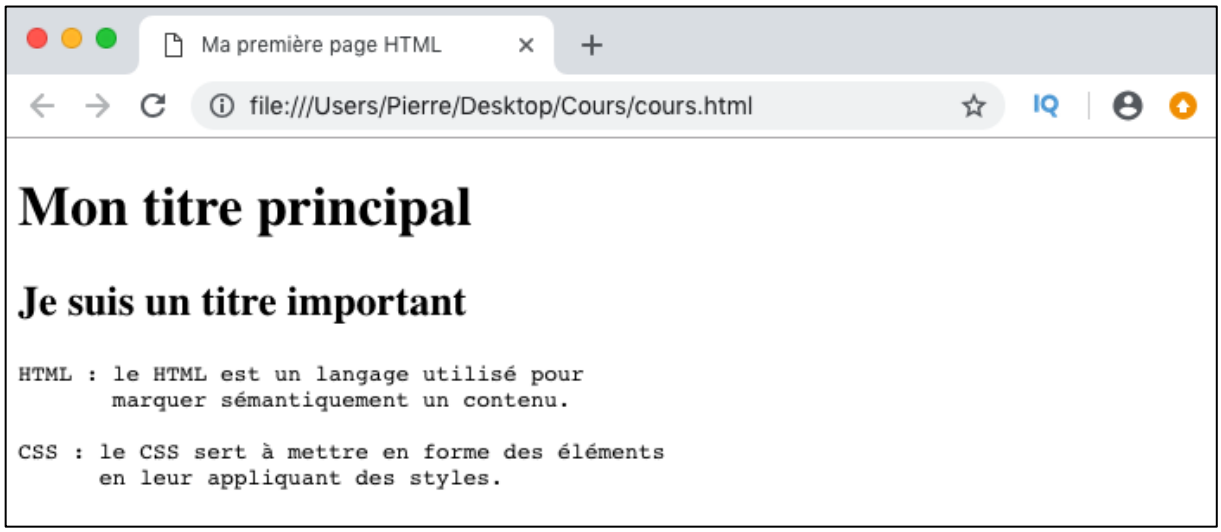
Il n'existe pas à proprement parler d'élément permettant de définir les espaces en HTML. Cependant, des solutions existent pour ajouter des espaces au sein de nos textes qui vont être conservés dans le rendu visuel. On va ainsi pouvoir :

- Utiliser l'élément HTML de pré formatage `pre` ;
- Utiliser des entités HTML ;
- Utiliser les marges CSS (dont nous parlerons dans le chapitre qui leur est consacré).

L'élément HTML pre

L'élément `pre` sert à pré formater un texte. Cela signifie que tout le contenu qui se trouve à l'intérieur de cet élément va conserver la mise en forme que nous allons lui donner lors du rendu fait par le navigateur.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <!--Les espaces ne seront pas affichés-->
        <h1>Mon titre                    principal</h1>
        <h2>Je suis       un       titre important</h2>

        <!--L'élément pre conserve les espaces et les retours à la ligne-->
        <pre>
HTML : le HTML est un langage utilisé pour
       marquer sémantiquement un contenu.

CSS : le CSS sert à mettre en forme des éléments
      en leur appliquant des styles.
        </pre>
    </body>
</html>
```



Attention ici : le contenu va s'afficher exactement de la même façon que dans votre éditeur par rapport à la page complète de code. C'est la raison pour laquelle j'ai enlevé l'indentation ici.

Pour des raisons de sémantique, on essaiera tant que possible de se passer de cet élément HTML qui ne possède pas de sens en soi mais qui sert justement à conserver des mises en forme… Ce qui est plutôt le rôle du CSS normalement. On préfèrera tant que possible utiliser le CSS pour ce genre d'opérations de mise en forme.

Les entités HTML

Une entité HTML est une suite de caractère qui est utilisée pour afficher un caractère réservé ou un caractère invisible (comme un espace) en HTML.

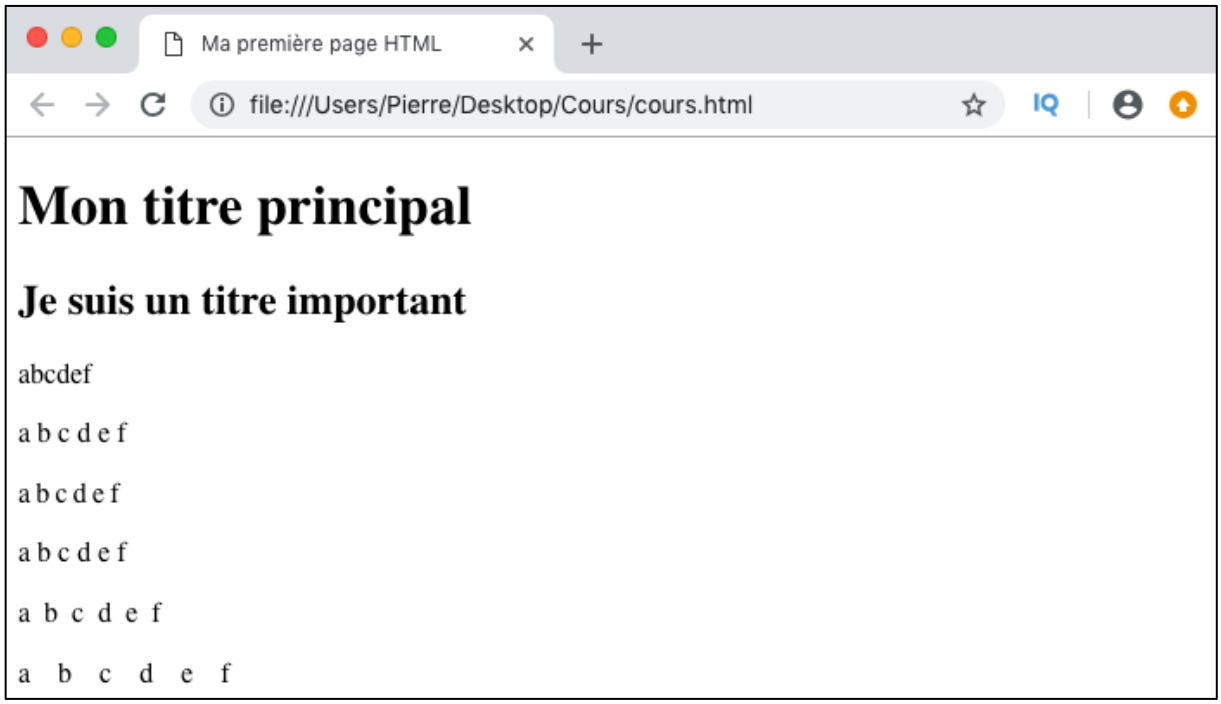
Qu'est-ce qu'un caractère réservé ? C'est un caractère qui possède déjà une signification particulière en HTML. Par exemple, imaginions que l'on souhaite afficher le caractère `<` dans un texte.

On ne va pas pouvoir mentionner ce caractère tel quel dans notre éditeur car le navigateur va interpréter cela comme l'ouverture d'une balise d'un élément. Il va donc falloir indiquer au navigateur qu'on souhaite afficher le caractère `<` en tant que tel et non pas ouvrir une balise.

Pour cela, il va falloir échapper le sens de ce caractère, et c'est ce à quoi vont nous servir les entités HTML. Nous aurons l'occasion de reparler de ces entités HTML plus tard dans cette partie. Pour le moment, concentrons-nous sur celles qui nous intéressent à savoir celles qui vont nous permettre d'ajouter des espaces.

- L'entité HTML ` ` (« non breaking space ») va nous permettre d'ajouter une espace simple dit espace « insécable » ;
- L'entité HTML ` ` (« en space ») va nous permettre de créer une espace double ;
- L'entité HTML ` ` (« em space ») va nous permettre de créer une espace quadruple ;
- L'entité HTML ` ` (« thin space ») va nous permettre de créer un espace très fin (demi-espace).

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <!--Les espaces ne seront pas affichés-->
        <h1>Mon titre                    principal</h1>
        <h2>Je suis       un       titre important</h2>

        <p>abcdef</p>
        <p>a b c d e f</p>
        <p>a b c d e f</p>
        <p>a b c d e f</p>
        <p>a b c d e f</p>
        <p>a b c d e f</p>
    </body>
</html>
```



Comme vous pouvez le voir, les espaces sont bien créés. Ici, je n'ai utilisé qu'une entité entre chaque caractère mais rien ne vous empêche d'en utiliser plusieurs d'affilée.

Notez cependant que l'utilisation des entités HTML à cette fin devrait toujours être une solution de dépannage et qu'on préférera généralement laisser toutes les questions de mise en page au CSS tant que possible.

Définir le niveau d'importance des contenus textuels en HTML

Dans cette leçon, nous allons voir comment indiquer aux navigateurs et surtout aux moteurs de recherche quelles parties de texte sont plus importantes que d'autres et doivent être considérées en priorité.

La problématique des niveaux d'importance des textes

Pour comprendre la problématique des niveaux d'importance des textes, il faut avant tout que vous ayez des notions en optimisation du référencement (SEO) ou du moins que vous compreniez les enjeux liés au SEO.

Lorsqu'on crée un site, en général, on veut que ce site soit visité. Pour qu'il soit visité, l'un des meilleurs moyens est que les pages de notre site ressortent parmi les premiers résultats lorsqu'un utilisateur fait une recherche sur un moteur de recherche (comme Google) sur un sujet abordé dans nos pages.

Les techniques d'optimisation que l'on va pouvoir mettre en place pour faire en sorte que nos pages soient les mieux classées possibles dans Google (et dans les autres moteurs de recherche) par rapport à certains mots clefs sont regroupées sous le terme « SEO ».

Le contenu d'une page est aujourd'hui le critère SEO le plus important pour Google puisque le but de Google est de fournir la réponse la plus pertinente pour chaque requête de ses utilisateurs.

En ce sens, il va donc falloir porter une attention toute particulière à la manière dont on va rédiger chaque page si on veut que celles-ci soient optimisées. Ainsi, il va avant tout falloir cibler un mot clef principal pour chaque page et écrire à chaque fois du contenu pertinent par rapport à ce mot clef.

En écrivant nos textes, souvent, certaines parties de contenu vont être plus importantes que d'autres et on va donc vouloir que les moteurs de recherche les remarquent en priorité et les traitent avec plus d'importance que le reste de nos textes.

On va pouvoir faire cela grâce à différents éléments HTML qui vont nous servir à indiquer des niveaux d'importance relatifs pour certaines parties de notre contenu.

Indiquer qu'un texte est très important avec l'élément strong

L'élément HTML `strong` va être utilisé pour signifier qu'un contenu est très important et doit être considéré comme tel par les moteurs de recherche (et les navigateurs).

En résultat, le navigateur affichera par défaut le contenu à l'intérieur de l'élément `strong` en gras. Encore une fois, n'utilisez pas `strong` pour mettre un texte en gras !

Utilisez strong pour marquer un texte qui vous semble très important. Nous utiliserons le CSS pour gérer le poids d'un texte.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Le langage HTML</h1>

        <!--Strong sert à indiquer qu'un contenu est très important-->
        <p>HTML signifie HyperText Markup Language. Ce langage est
        utilisé pour <strong>marquer sémantiquement un contenu</strong>.</p>
    </body>
</html>
```

Mettre un texte en emphase avec l'élément em

L'élément HTML em (pour emphasis ; « emphase » ou « accentuation » en français) va être utilisé pour mettre des termes en emphase.

On va pouvoir utiliser cet élément pour souligner un contraste par exemple ou une définition. Concrètement, les moteurs de recherche vont accorder une importance moins grande aux textes dans les éléments em qu'à ceux dans strong mais une importance plus grande à ces textes qu'à des simples paragraphes.

Le résultat visuel par défaut de l'emphase est la mise en italique du texte contenu dans l'élément.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Le langage HTML</h1>

        <!--Strong sert à indiquer qu'un contenu est très important,
        em indique une mise en emphase-->
        <p>HTML signifie <em>HyperText Markup Language</em>. Ce langage est
        utilisé pour <strong>marquer sémantiquement un contenu</strong>.</p>
    </body>
</html>
```

Mettre un contenu pertinent en relief avec l'élément mark

L'élément mark va être utilisé pour mettre en relief certains textes qui vont être pertinents dans un certain contexte. Le texte mis en relief n'est pas forcément important en soi par rapport au sujet de l'article mais va être pertinent pour un certain utilisateur dans un certain contexte.

Il est difficile d'illustrer l'intérêt de cet élément pour le moment car celui-ci va souvent être utilisé de manière dynamique et avec des langages dynamiques donc comme du JavaScript.

Imaginons par exemple que votre site possède un champ de recherche. Lorsque l'utilisateur recherche un terme dans une page, ce terme en particulier va donc être important pour lui et on va donc vouloir le mettre en relief par rapport aux autres.

L'élément de marquage à utiliser dans cette situation-là est l'élément mark dont le rôle est encore une fois d'indiquer la pertinence d'un certain texte en fonction d'un contexte.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Le langage HTML</h1>

        <!--Strong sert à indiquer qu'un contenu est très important,
        em indique une mise en emphase,
        mark permet de mettre en relief un contenu pertinent-->
        <p><mark>HTML</mark> signifie <em>HyperText Markup Language</em>.
        Ce langage est utilisé pour <strong>marquer sémantiquement
        un contenu</strong>.</p>
    </body>
</html>
```

De manière pratique, toutefois, l'élément mark est relativement peu utilisé.

Créer des listes en HTML

Dans cette nouvelle leçon, nous allons voir à quoi servent les listes, découvrir les différents types de listes en HTML et apprendre à en créer.

Qu'est-ce qu'une liste HTML ? A quoi servent les listes ?

Le HTML est un langage de sémantique : son rôle est de donner du sens aux différents contenus d'une page afin que ceux-ci soient correctement reconnus et afficher et que les navigateurs et les moteurs de recherche « comprennent » nos pages.

Les listes HTML répondent tout à fait à cet objectif puisqu'elles permettent d'ordonner du contenu. Ce contenu peut être hiérarchisé ou non.

Une liste en HTML est composée de différents éléments de listes. Visuellement une liste va ressembler à ceci :

- Premier élément de ma liste,
- Deuxième élément de ma liste,
- Troisième élément de ma liste.

Les listes vont ainsi nous permettre de lister plusieurs éléments en les groupant sous un dénominateur commun qu'est la liste en soi. Les navigateurs et les moteurs de recherche vont donc comprendre qu'il y a une relation entre les différents éléments de liste.

Les listes vont donc déjà être très utiles pour apporter de la clarté et de l'ordre à nos documents. En plus de cela, nous allons également utiliser des listes HTML pour créer des menus de navigation (nous verrons comment plus tard dans ce cours).

Il existe deux grands types de listes en HTML : les listes ordonnées et les listes non-ordonnées. Il existe également un troisième type de liste un peu particulier et moins utilisé : les listes de définitions.

Les listes non ordonnées

Les listes non-ordonnées vont être utiles pour lister des éléments sans hiérarchie ni ordre logique.

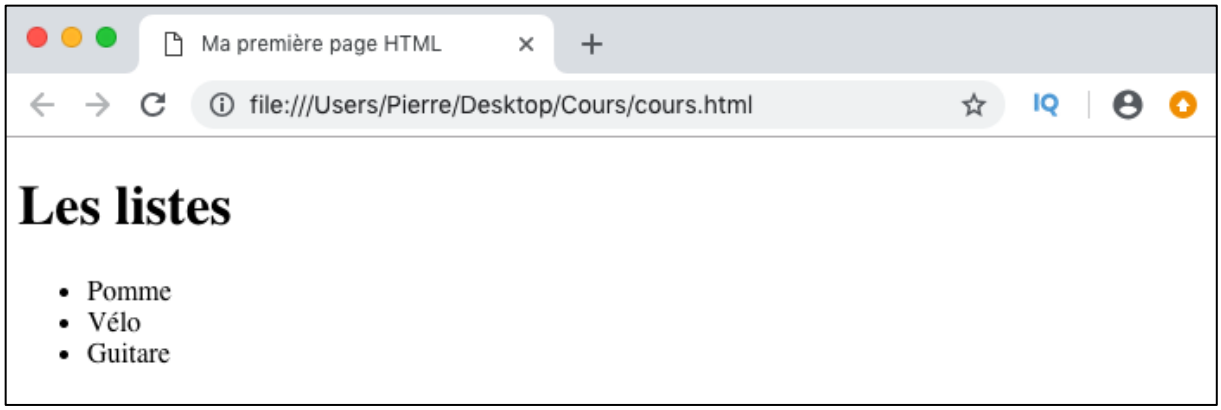
Par exemple, si je souhaite lister les mots « pomme », « vélo » et « guitare », sans plus de contexte, j'utiliserai une liste non-ordonnée.

En effet, on ne peut pas dégager de notion d'ordre, de hiérarchie ou de subordination entre ces trois termes (du moins pas sans un contexte précis).

Pour créer une liste non-ordonnée, nous allons avoir besoin d'un élément `ul` (pour « unordered list », ou « liste non-ordonnée » en français) qui va représenter la liste en soi

ainsi que d'un élément li (« list item » ou « élément de liste ») pour chaque nouvel élément de liste.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Les listes</h1>

        <!--Un exemple de liste non-ordonnée-->
        <ul>
            <li>Pomme</li>
            <li>Vélo</li>
            <li>Guitare</li>
        </ul>
    </body>
</html>
```



Comme vous pouvez le remarquer, on va placer les éléments li à l'intérieur de l'élément de liste ul (à partir de ce moment, bien indenter son code commence à être important pour ne pas se perdre). C'est tout à fait logique puisque les éléments de liste « appartiennent » à une liste en particulier.

Visuellement, des puces (les points noirs) apparaissent automatiquement devant chaque élément d'une liste non-ordonnée par défaut. Nous allons pouvoir changer ce comportement et personnaliser l'apparence de nos listes en CSS grâce notamment à la propriété list-style-type que nous étudierions plus tard dans ce cours.

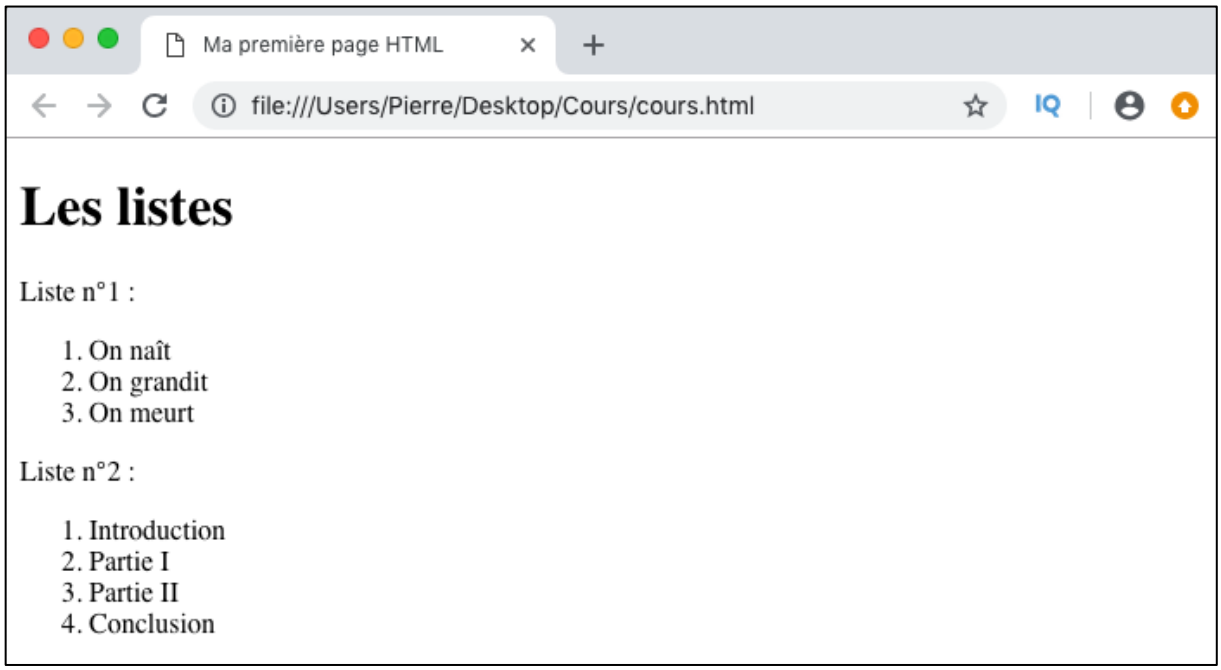
Les listes ordonnées

Au contraire des listes non-ordonnées, nous allons utiliser les listes ordonnées lorsqu'il y aura une notion d'ordre ou de progression logique ou encore de hiérarchie entre les éléments de notre liste.

Par exemple, si l'on souhaite lister les étapes naturelles de la vie, ou lorsque l'on crée un sommaire, on utilisera généralement des listes ordonnées.

Pour créer une liste ordonnée, nous allons cette fois-ci utiliser l'élément `ol` (pour « ordered list » ou « liste ordonnée ») pour définir la liste en soi et à nouveau des éléments `li` pour chaque élément de la liste.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Les listes</h1>

        <!--Deux exemples de listes ordonnées-->
        <p>Liste n°1 :</p>
        <ol>
            <li>On naît</li>
            <li>On grandit</li>
            <li>On meurt</li>
        </ol>

        <p>Liste n°2 :</p>
        <ol>
            <li>Introduction</li>
            <li>Partie I</li>
            <li>Partie II</li>
            <li>Conclusion</li>
        </ol>
    </body>
</html>
```



Comme vous le voyez, ce sont cette fois-ci des numéros qui sont affichés devant chaque élément de la liste par défaut.

Encore une fois, nous allons pouvoir changer ce comportement et afficher différents styles de puces avec la propriété CSS `list-style-type`.

Les attributs HTML des listes ordonnées

En plus des propriétés CSS, on va pouvoir utiliser certains attributs HTML avec nos listes ordonnées pour modifier leur présentation.

Ici, vous devez absolument comprendre que ces attributs ne devraient pas exister à priori puisque le HTML ne devrait pas se soucier de la présentation mais devrait laisser l'apparence au CSS.

Cependant, les langages sont en constante évolution ce qui signifie que leur rôle n'est pas encore tout à fait fixé et que certains langages ne sont pas capables de faire certaines choses aujourd'hui. Ce sont les raisons principales pour l'existence de ces attributs que je vais vous présenter par souci d'exhaustivité.

Retenez simplement que vous devriez toujours privilégier l'utilisation du langage CSS pour la mise en forme des contenus HTML tant que vous le pouvez.

Commençons avec l'attribut `type` qui va nous permettre de changer l'apparence des puces d'une liste ordonnée.

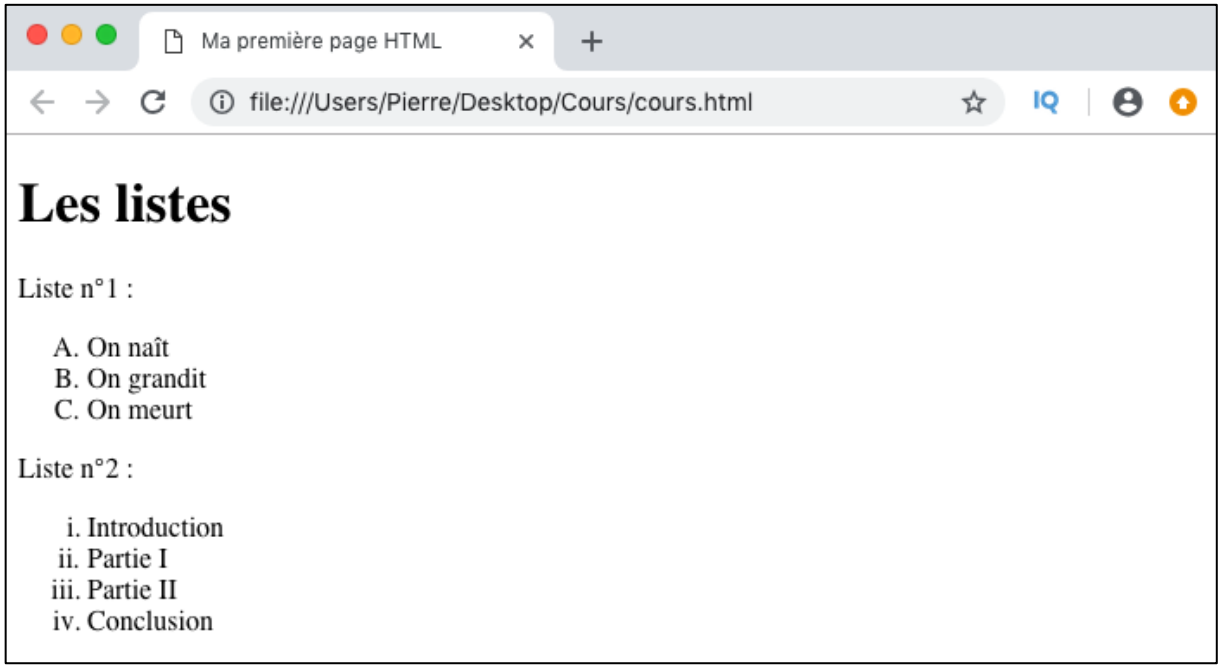
Cet attribut existait autrefois également pour les listes non ordonnées mais a été déprécié pour celles-ci. Il est possible qu'il soit également déprécié dans le futur pour les listes ordonnées puisque le HTML ne devrait pas se charger de la mise en forme.

Pour cette raison, je vous déconseille de l'utiliser et vous conseille de préférer l'utilisation du CSS. Cependant, je vous présente tout de même les valeurs qu'on va pouvoir lui fournir par souci d'exhaustivité :

- « 1 » : valeur par défaut. Des chiffres apparaitront devant chaque élément de la liste ;
- « I » : Des chiffres romains majuscules apparaitront devant chaque élément de la liste ;
- « i » : Des chiffres romains minuscules apparaitront devant chaque élément de la liste ;
- « A » : Des lettres majuscules apparaitront devant chaque élément de la liste ;
- « a » : Des lettres minuscules apparaitront devant chaque élément de la liste.

Voici également un exemple d'utilisation de cet attribut :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Les listes</h1>

        <!--Deux exemples de listes ordonnées-->
        <p>Liste n°1 :</p>
        <ol type="A">
            <li>On naît</li>
            <li>On grandit</li>
            <li>On meurt</li>
        </ol>

        <p>Liste n°2 :</p>
        <ol type="i">
            <li>Introduction</li>
            <li>Partie I</li>
            <li>Partie II</li>
            <li>Conclusion</li>
        </ol>
    </body>
</html>
```



Notez que par défaut le premier élément d'une liste ordonnée va avoir comme puce le chiffre « 1 » ou la première lettre de l'alphabet « a ». On va pouvoir changer ce comportement et faire démarrer notre liste ordonnée à partir d'un point choisi grâce aux attributs HTML `start` ou `value`.

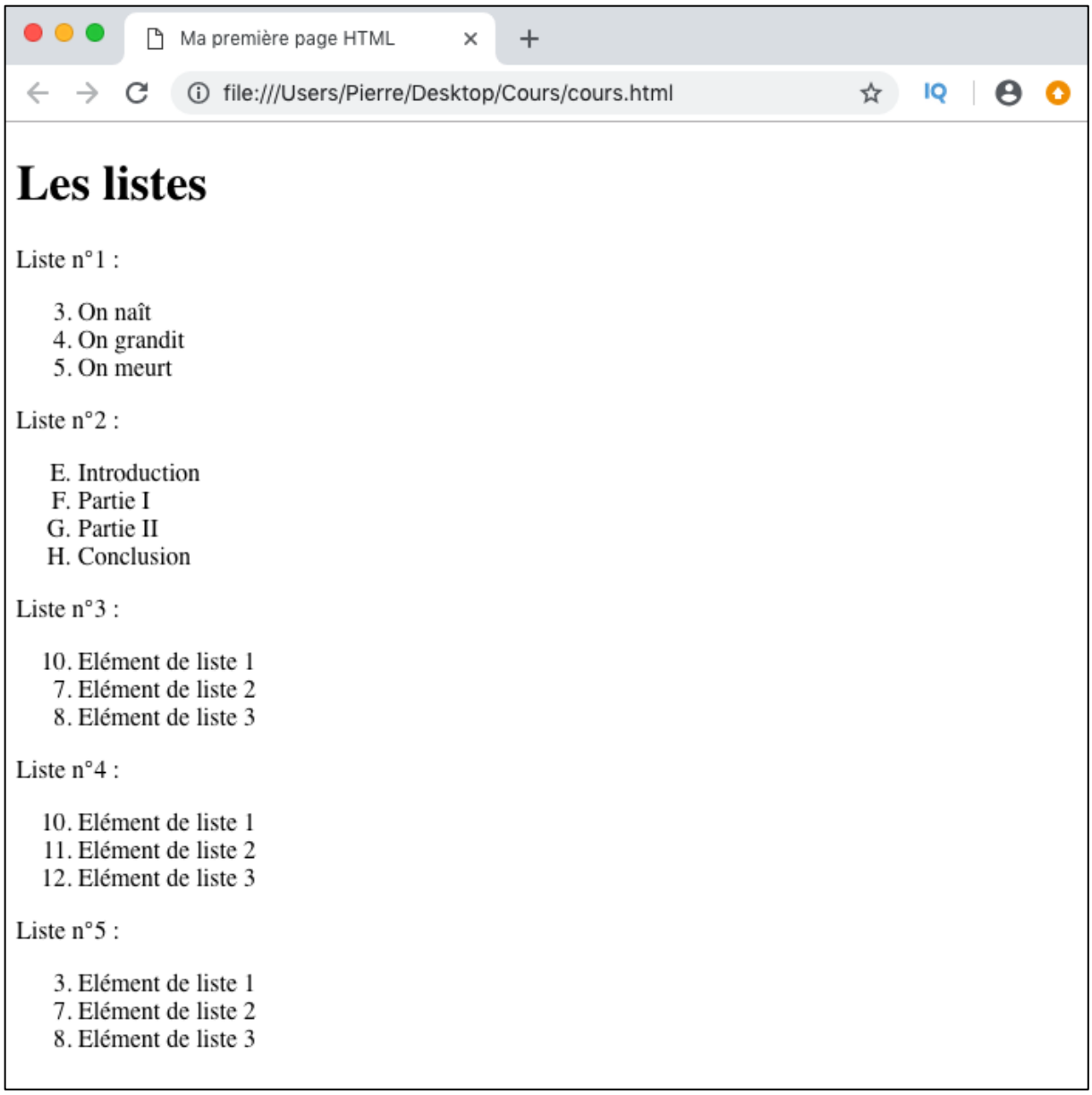
L’attribut start va nous permettre de choisir un point de départ pour notre liste ordonnée. On va donc le placer dans la balise ouvrante de l’élément représentant la liste ol.

L’attribut value va lui en revanche nous permettre de choisir la valeur de chaque puce d’éléments de liste. On va pouvoir ajouter un attribut value pour chaque élément li. Dans le cas où certains éléments li ne possèderaient pas d’attribut value, la valeur de leur puce va s’incrémenter normalement par rapport à la valeur de la puce de l’élément précédent (c’est-à-dire ajouter 1 par rapport à la puce précédente).

Notez que dans le cas où un attribut start est précisé pour la liste et un attribut value est précisé pour le premier élément de liste, l’attribut value, plus précis, va avoir la priorité et imposer sa valeur.

Voici quelques exemples d’utilisation de ces attributs :

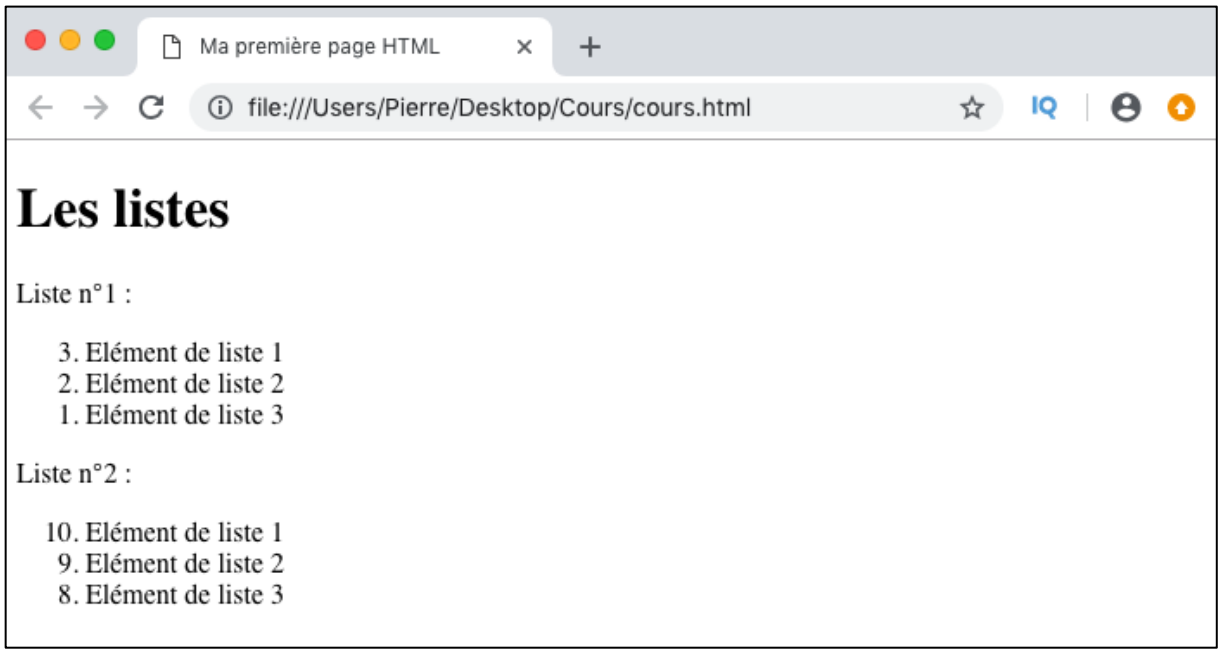
```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Les listes</h1>

        <!--Deux exemples de listes ordonnées-->
        <p>Liste n°1 :</p>
        <ol start="3">
            <li>On naît</li>
            <li>On grandit</li>
            <li>On meurt</li>
        </ol>

        <p>Liste n°2 :</p>
        <ol type="A" start="5">
            <li>Introduction</li>
            <li>Partie I</li>
            <li>Partie II</li>
            <li>Conclusion</li>
        </ol>

        <p>Liste n°3 :</p>
        <ol start="10">
            <li>Elément de liste 1</li>
            <li value="7">Elément de liste 2</li>
            <li>Elément de liste 3</li>
        </ol>

        <p>Liste n°4 :</p>
        <ol>
            <li value="10">Elément de liste 1</li>
            <li>Elément de liste 2</li>
            <li>Elément de liste 3</li>
        </ol>

        <p>Liste n°5 :</p>
        <ol start="5">
            <li value="3">Elément de liste 1</li>
            <li value="7">Elément de liste 2</li>
            <li>Elément de liste 3</li>
        </ol>
    </body>
</html>
```



Finalement, nous allons pouvoir inverser le compte des puces des éléments de liste ordonnées grâce à l'attribut `reversed`. Le premier élément de la liste aura alors la puce avec la valeur la plus élevée, puis on enlèvera un par nouvel élément jusqu'à arriver à « 1 » ou « a » pour le dernier élément de liste par défaut.

L'attribut `reversed` ne possède qu'une valeur qui est `reversed` (identique au nom de l'attribut, comme pour tous les attributs qui ne possèdent qu'une valeur en HTML). Comme `reversed` ne possède qu'une valeur, la valeur est dite évidente et peut être omise.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Les listes</h1>

        <p>Liste n°1 :</p>
        <ol reversed>
            <li>Elément de liste 1</li>
            <li>Elément de liste 2</li>
            <li>Elément de liste 3</li>
        </ol>

        <p>Liste n°2 :</p>
        <ol start="10" reversed>
            <li>Elément de liste 1</li>
            <li>Elément de liste 2</li>
            <li>Elément de liste 3</li>
        </ol>
    </body>
</html>
```



Les listes de définitions

Les listes de définitions, encore appelées « listes de descriptions » vont nous permettre de lister des termes et d'ajouter des définitions ou descriptions pour chacun de ces termes.

Pour créer une liste de définitions, nous allons cette fois-ci utiliser l'élément dl signifiant « description list » ou « liste de description / définition » en français pour définir la liste en

soi, puis des éléments `dt`(description term) pour chaque élément à décrire et enfin l'élément `dd` pour la définition / description en soi.

Pensez bien lorsque vous créez une liste de définitions à toujours placer le terme à définir avant sa définition, c'est-à-dire l'élément `dt` avant l'élément `dd`. Cela est normalement assez intuitif.

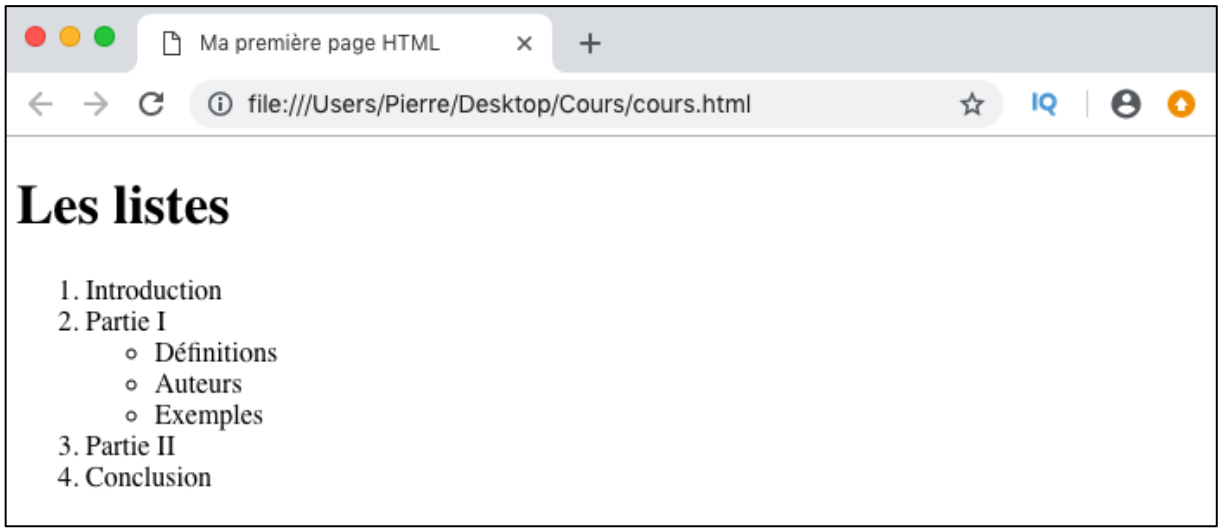
```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Les listes</h1>

        <!--Une liste de définition-->
        <dl>
            <dt>HTML</dt>
            <dd>HTML signifie HyperText Markup Language.</dd>
        </dl>
    </body>
</html>
```

L'imbrication de listes

Finalement, sachez qu'il est tout-à-fait possible d'imbriquer une liste dans une autre en suivant quelques règles simples.

Pour imbriquer une liste dans une autre, il suffit de définir une nouvelle liste à l'intérieur de l'un des éléments d'une autre liste, juste avant la balise fermante de cet élément.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Les listes</h1>

        <!--Listes imbriquées-->
        <ol>
            <li>Introduction</li>
            <li>Partie I
            <!--On imbrique une liste non-ordonnée dans une liste ordonnée-->
                <ul>
                    <li>Définitions</li>
                    <li>Auteurs</li>
                    <li>Exemples</li>
                </ul>
            </li>
            <li>Partie II</li>
            <li>Conclusion</li>
        </ol>
    </body>
</html>
```



Comme vous pouvez le voir, il devient ici très important de bien indenter son code afin de ne pas se perdre au milieu de nos listes !

Notez que l'on peut imbriquer autant de listes que l'on souhaite les unes dans les autres. Cependant, pour des raisons évidentes de lisibilité, il est conseillé de ne pas créer plus de niveaux de listes que ce qui est strictement nécessaire pour servir vos besoins.

Créer des liens en HTML

Les liens hypertextes sont l'une des fondations du HTML qui est, rappelons-le, un langage de marquage hypertexte justement.

Dans cette nouvelle leçon, nous allons découvrir et expliquer à quoi correspond un lien en HTML et allons apprendre à créer différents « types » de liens en HTML, que ce soit des liens ramenant à un autre endroit d'une même page (liens ancres), des liens menant vers d'autres pages d'un même site (liens internes) ou des liens envoyant l'utilisateur vers un autre site (liens externes).

Définition d'un lien HTML

Les liens en HTML vont nous servir à créer des ponts entre différentes pages d'un même site ou de sites différents. Le principe d'un lien est le suivant : en cliquant sur une ancre (qui est la partie visible par l'utilisateur d'un lien et qui peut être un texte comme une image), nos utilisateurs vont être redirigés vers une page cible.

Il existe deux types principaux de liens hypertextes en HTML :

- Les liens internes qui vont servir à naviguer d'une page à l'autre dans un même site ;
- Les liens externes qui vont envoyer les utilisateurs vers une page d'un autre site.

Ces deux types de liens vont être l'objet d'enjeux et d'implications différents en termes d'optimisation du référencement (SEO) et vont pouvoir être créés de façons différentes en HTML.

Nous allons également pouvoir utiliser les liens en HTML pour naviguer au sein d'une même page et renvoyer à un endroit précis de celle-ci. Cela est utile pour fournir des accès ou des repères rapides à nos utilisateurs dans le cas d'un page très longue.

Nous appelons ce type de liens faits au sein d'une même page des liens « ancres » tout simplement (même si ce terme peut porter à confusion car le terme « ancre » est aussi utilisé pour définir la partie visible et cliquable d'un lien).

Dans chacun de ces cas, nous allons devoir utiliser l'élément HTML `a` accompagné de son attribut `href` pour créer un lien en HTML.

Notez que les liens sont également un aspect fondamental d'une stratégie d'optimisation de référencement (SEO), que ce soit dans le cas de liens internes (pour effectuer ce qu'on appelle un maillage interne et donner plus de valeur à certaines de nos pages aux yeux de Google) ou de liens externes qui sont un des critères principaux de classement pour Google. Il est donc très important de bien les comprendre et les maitriser !

Création de liens : l'élément a et son attribut href

Quel que soit le type de liens que l'on souhaite créer en HTML (liens internes, des liens externes, ou des liens vers un autre endroit d'une même page), nous utiliserons toujours l'élément `a` qui est l'abréviation de « anchor » ou « ancre » en français accompagné de son attribut `href` pour « hypertext reference » ou « référence hypertexte » en français.

L'élément HTML `a` est composé d'une paire de balises (balises ouvrante et fermante) et d'un contenu entre les balises que l'on va appeler "ancre". Ce contenu peut être un texte, une image, etc. et sera la partie visible et cliquable du lien pour les utilisateurs.

L'attribut `href` va nous servir à indiquer la cible du lien, c'est-à-dire l'endroit où l'utilisateur doit être envoyé après avoir cliqué sur le lien. Nous allons indiquer cette cible en valeur de l'attribut `href`.

D'un point de vue du code, la seule chose qui va changer pour créer un lien ancre plutôt qu'interne ou interne plutôt qu'externe va être la façon dont on va construire la valeur qu'on va passer à l'attribut `href`.

Créer des liens externes en HTML

Un lien externe est un lien qui part d'une page d'un site et ramène les utilisateurs vers une autre page d'un autre site.

Pour créer un lien vers une page d'un autre site en HTML (ou lien externe), il va falloir indiquer l'adresse complète de la page (c'est-à-dire son URL) en valeur de l'attribut `href` de notre lien.

Imaginons par exemple que nous voulions créer un lien vers la page d'accueil de Wikipédia. L'URL de cette page est la suivante : https://www.wikipedia.org/.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= 'utf-8'>
    </head>
    <body>
        <p>Pour créer des liens en HTML, on utilise l'élément a et son attribut href.</p>
        <p>Ex : lien vers <a href="https://www.wikipedia.org/">la home de Wikipedia</a></p>
    </body>
</html>
```

Comme vous le voyez, on utilise bien un élément `a` pour créer un lien externe en HTML. On place notre attribut `href` au sein de la balise ouvrante de cet élément. Ici, on précise en valeur l'adresse (URL) de la page d'accueil de Wikipédia.

Entre les deux balises, nous plaçons la partie visible et cliquable (l'ancre) de notre lien. Ici, cela correspond au texte « la home de Wikipédia ». Les utilisateurs vont pouvoir cliquer sur ce texte pour être renvoyés vers la page d'accueil de Wikipedia.

Note : Ici, nous avons choisi de placer un texte comme ancre de notre lien, mais rien ne nous empêche de placer une image à la place afin de créer une image cliquable.
Vous pouvez remarquer que le navigateur applique automatiquement des styles à la partie cliquable de nos liens :

- le texte (notre ancre) est de couleur différente (bleu avant de cliquer puis volet une fois le lien visité) ;
- le texte est souligné ;
- le curseur de notre souris change de forme lorsqu'on passe sur le lien.

Nous allons bien évidemment pouvoir changer ces comportements en appliquant nos propres styles CSS à nos éléments `a`. Nous verrons comment faire cela plus tard dans ce cours.

Bon à savoir : Lorsque l'attribut `href` prend une URL complète en valeur, on parle de valeur absolue (car celle-ci est fixe et ne dépend de rien). Les liens externes utilisent toujours des valeurs absolues. On parle de valeur absolue en opposition aux valeurs dites relatives, qui sont dans ce contexte des valeurs qui vont indiquer l'emplacement de la page cible relativement à la page source du lien.

Créer des liens internes en HTML

Le deuxième grand type de liens que l'on va pouvoir créer en HTML correspond aux liens internes, c'est-à-dire à des liens renvoyant vers d'autres pages d'un même site.

Explication du fonctionnement des liens internes

Nous allons avoir plusieurs solutions pour créer des liens internes. Tout d'abord, nous allons tout simplement pouvoir faire exactement comme pour les liens externes et indiquer l'adresse complète (adresse absolue) de la page cible du lien en valeur de l'attribut `href`

Cependant, si cette première manière de faire semble tout à fait fonctionner et être la plus simple à priori, ce n'est pas du tout la plus optimisée d'un point de vue de l'utilisation des ressources et elle peut même s'avérer problématique selon comment votre site est construit (notamment si vos URL sont construites dynamiquement).

Pour ces raisons, nous préciserons généralement plutôt une valeur de type relatif en valeur de l'attribut `href` de nos liens internes en HTML.

On dit que la valeur est relative car on va devoir indiquer l'adresse de la page de destination relativement à l'adresse de la page de départ (c'est-à-dire celle à partir de laquelle on fait notre lien).

Comment savoir quelle valeur relative utiliser ? Pour bien comprendre comment fonctionnent les valeurs relatives, vous devez avant tout savoir ce qu'est un site web.

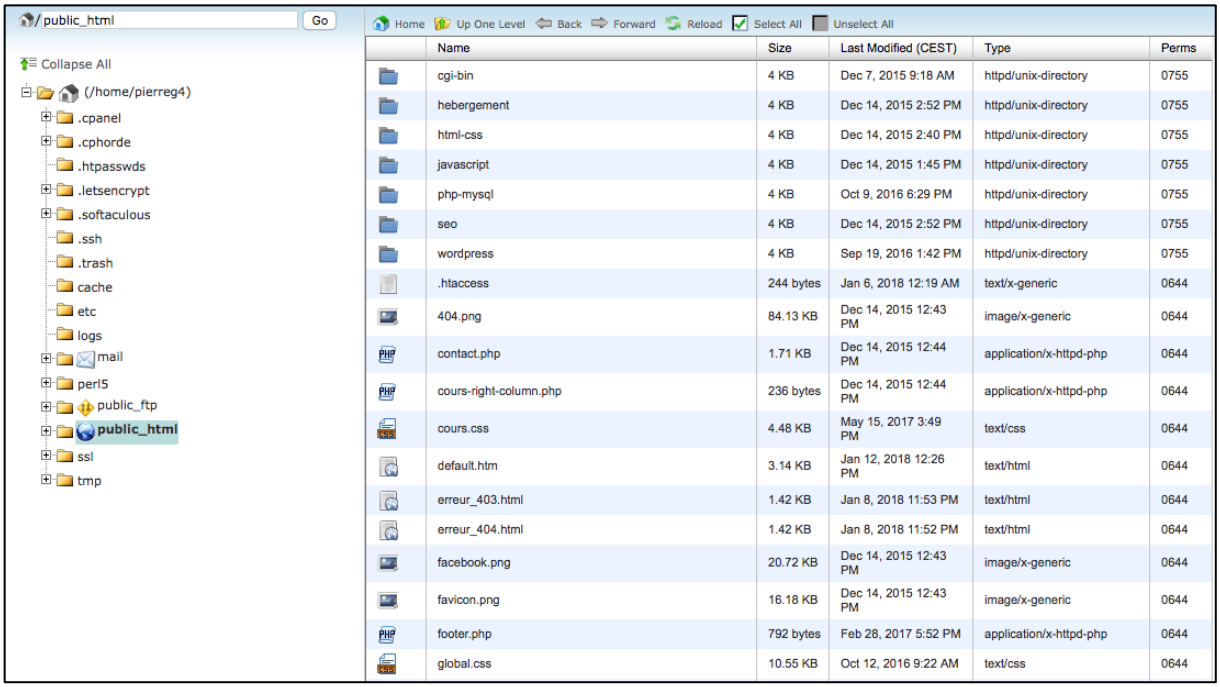
Un site web n'est qu'un ensemble de fichiers et de ressources (pages de code de différents types et fichiers médias comme des images, etc.) liés entre eux.

Toutes ces ressources vont être hébergées (stockées) sur un ordinateur très puissant et constamment connecté à Internet qu'on appelle serveur, dans un dossier principal qu'on va appeler la racine d'un site.

Pour pouvoir stocker nos différentes ressources sur un serveur et faire en sorte que notre site soit tout le temps accessible via Internet, nous passons généralement par un hébergeur qui va nous louer un serveur ou une partie d'un de ses serveurs.

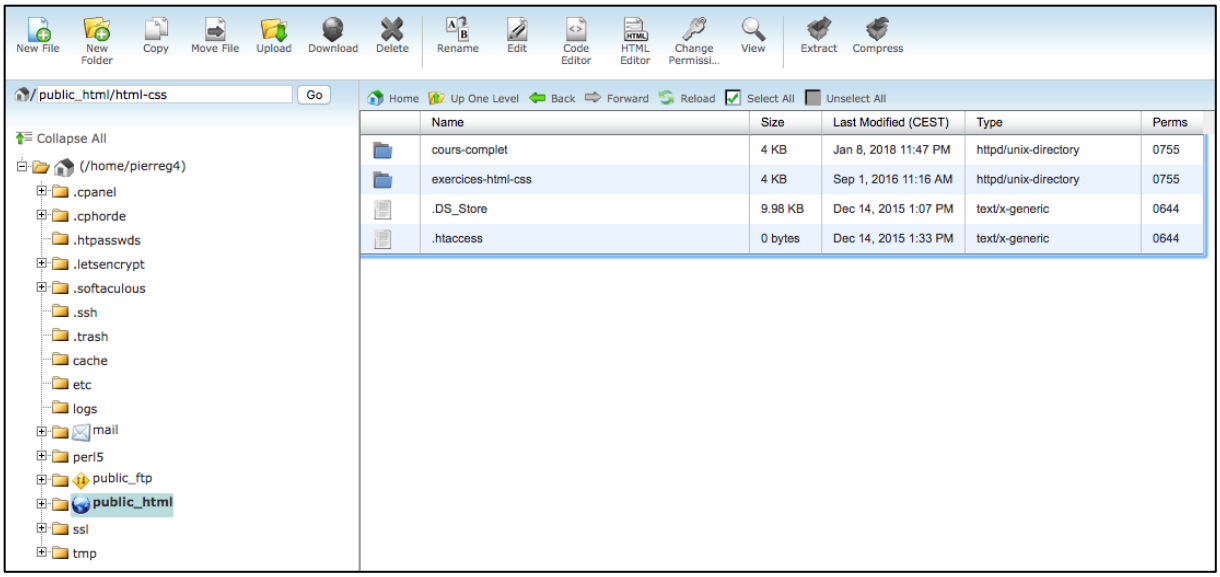
Dans ce dossier racine, il va très souvent y avoir d'autres dossiers (des sous-dossiers donc), des fichiers, des images, etc.

Par exemple, voici une première image de la racine de l'ancienne version de mon site (à laquelle j'accède depuis l'interface d'administration de mon hébergeur) :

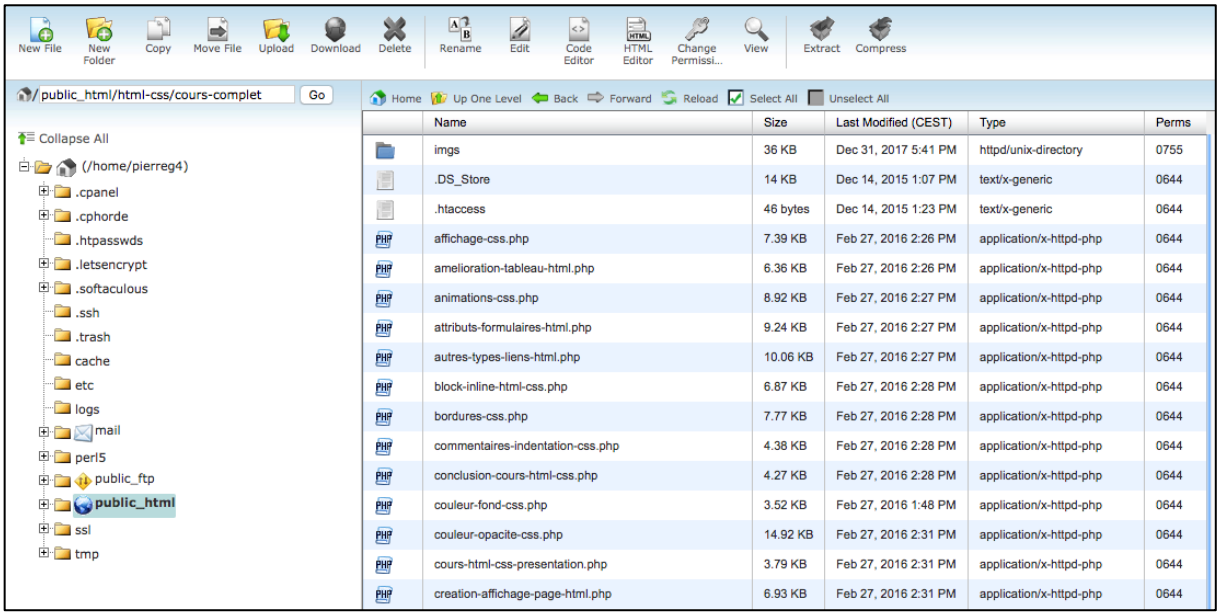


Name	Size	Last Modified (CEST)	Type	Perms
cgi-bin	4 KB	Dec 7, 2015 9:18 AM	httpd/unix-directory	0755
hebergement	4 KB	Dec 14, 2015 2:52 PM	httpd/unix-directory	0755
html-css	4 KB	Dec 14, 2015 2:40 PM	httpd/unix-directory	0755
javascript	4 KB	Dec 14, 2015 1:45 PM	httpd/unix-directory	0755
php-mysql	4 KB	Oct 9, 2016 6:29 PM	httpd/unix-directory	0755
seo	4 KB	Dec 14, 2015 2:52 PM	httpd/unix-directory	0755
wordpress	4 KB	Sep 19, 2016 1:42 PM	httpd/unix-directory	0755
.htaccess	244 bytes	Jan 6, 2018 12:19 AM	text/x-generic	0644
404.png	84.13 KB	Dec 14, 2015 12:43 PM	image/x-generic	0644
contact.php	1.71 KB	Dec 14, 2015 12:44 PM	application/x-httpd-php	0644
cours-right-column.php	236 bytes	Dec 14, 2015 12:44 PM	application/x-httpd-php	0644
cours.css	4.48 KB	May 15, 2017 3:49 PM	text/css	0644
default.htm	3.14 KB	Jan 12, 2018 12:26 PM	text/html	0644
erreur_403.html	1.42 KB	Jan 8, 2018 11:53 PM	text/html	0644
erreur_404.html	1.42 KB	Jan 8, 2018 11:52 PM	text/html	0644
facebook.png	20.72 KB	Dec 14, 2015 12:43 PM	image/x-generic	0644
favicon.png	16.18 KB	Dec 14, 2015 12:43 PM	image/x-generic	0644
footer.php	792 bytes	Feb 28, 2017 5:52 PM	application/x-httpd-php	0644
global.css	10.55 KB	Oct 12, 2016 9:22 AM	text/css	0644

Vous pouvez voir différents fichiers et ressources et des dossiers. En cliquant sur le dossier « html-css » par exemple, on va trouver de nouveaux fichiers et dossiers :



Puis en explorant le contenu du dossier « cours-complet » :



Pour lier ces différents fichiers et ressources entre eux (et donc offrir entre autres à nos utilisateurs la possibilité de naviguer de l'un à l'autre), nous allons utiliser des valeurs relatives, c'est-à-dire que nous allons préciser l'emplacement de la ressource que l'on souhaite utiliser (ou vers laquelle on souhaite renvoyer) relativement à l'emplacement de la page / ressource qui souhaite l'utiliser (la page de départ).

C'est comme cela que nous allons procéder dans le cas précis de la création de liens internes : nous allons préciser en valeur de l'attribut `href` l'emplacement de la page cible (page de destination) par rapport à l'emplacement sur le serveur de la page source.

Trois cas vont alors se présenter à nous :

1. Le cas où les deux pages (source et destination) se trouvent dans un même dossier ;

2. Le cas où la page de destination se trouve dans un sous dossier par rapport à la page source ;
3. Le cas où la page de destination se trouve dans un dossier parent par rapport à la page source.

Pour chacun de ces cas, nous allons construire la valeur de notre attribut `href` de manière différente :

- Si les deux pages se situent dans le même dossier, alors on pourra se contenter de préciser le nom de la page cible/de destination (avec son extension) en valeur de l'attribut `href` ;
- Si la page cible se situe dans un sous dossier par rapport à la page à partir de laquelle on fait un lien, alors on précisera le nom du sous dossier suivi d'un slash suivi du nom de la page cible en valeur de l'attribut `href` ;
- Si la page cible se situe dans un dossier parent par rapport à la page de départ, alors il faudra indiquer deux points suivis d'un slash suivi du nom de la page de destination en valeur de l'attribut `href`.

Exemples pratiques de création de liens internes

Pour illustrer cela, je vous propose de créer un premier dossier sur le bureau de votre ordinateur. Nous imaginerons que ce dossier correspond à la racine d'un site Internet et nous pouvons par exemple l'appeler `racine`.

Dans ce dossier, vous allez créer une première page HTML que vous pourrez appeler `home.html` et un second dossier qu'on peut appeler `cours` par exemple.

Dans ce sous dossier `cours`, je vous propose de créer deux pages `presentation.html` et `livres.html` par exemple ainsi qu'un troisième dossier qu'on pourra appeler `code`.

Dans ce dernier dossier `code`, nous créerons finalement une dernière page qu'on appellera liens.html.

L'idée ici va être de faire des liens entre ces différentes pages. Encore une fois, vous pouvez imaginer que tout ce qui se trouve au sein de votre dossier `racine` est l'équivalent de la structure d'un site web.

A partir de notre page `home.html`, on veut créer trois liens :

- Un vers la page `presentation.html` située dans le sous dossier `cours` ;
- Un vers la page `livres.html` située dans le sous dossier `cours` ;
- Un vers la page `liens.html` située dans le sous-sous dossier `code` ;

Voici comment on va s'y prendre :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Home</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <p>Cette page se nomme "home.html" et se situe
        dans le dossier "racine" (niveau 0)</p>

        <p>Accédez directement à :</p>
        <ul>
            <li>La page <a href="cours/presentation.html">presentation.html</a></li>
            <li>La page <a href="cours/livres.html">livres.html</a></li>
            <li>La page <a href="cours/code/liens.html">liens.html</a></li>
        </ul>
    </body>
</html>
```

Ici, on voit bien que pour créer un lien vers une page située dans un sous dossier, il faut indiquer le nom du sous dossier suivi d'un slash suivi du nom de la page en valeur de l'attribut `href`.

Vous pouvez également remarquer que si la page cible du lien se situe dans un sous-sous dossier, alors il faudra préciser le nom du sous dossier suivi d'un slash suivi du nom du sous dossier puis à nouveau d'un slash et finalement le nom de la page de destination en valeur de l'attribut `href`.

Bon à savoir : Il faudra indiquer le nom de tous les sous dossiers traversés pour atteindre la page de destination à partir de la page de départ en valeur de l'attribut `href`.

On va également créer trois liens à partir de notre page `presentation.html` :

- Un vers la page `home.html` située dans le dossier parent `racine` ;
- Un vers la page `livres.html` située dans le même dossier `cours` ;
- Un vers la page `liens.html` située dans le sous dossier `code`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Présentation</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <p>Cette page se nomme "presentation.html" et se
        situe dans le dossier "cours" (niveau 1)</p>

        <p>Accédez directement à :</p>
        <ul>
            <li>La page <a href="../home.html">home.html</a></li>
            <li>La page <a href="livres.html">livres.html</a></li>
            <li>La page <a href="code/liens.html">liens.html</a></li>
        </ul>
    </body>
</html>
```

Ici, pas de surprise :

- On utilise la notation « ../home.html » pour créer un lien vers la page « home.html » située dans un dossier parent ;
- La page « livres.html » est située dans le même dossier que la page « presentation.html », on se contente donc de préciser son nom en valeur de l'attribut `href` ;
- La page « liens.html » est située dans le sous dossier « code », on utilise donc la notation « code/liens.html ».

Finalement, on va à nouveau vouloir créer trois liens à partir de la page `liens.html` vers :

- La page `home.html` située dans le dossier parent `racine` ;
- La page `presentation.html` située dans le dossier parent `cours` ;
- La page `livres.html` située dans le dossier parent `cours`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Liens</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <p>Cette page se nomme "liens.html" et se
        situe dans le dossier "code" (niveau 2)</p>

        <p>Accédez directement à :</p>
        <ul>
            <li>La page <a href="../../home.html">home.html</a></li>
            <li>La page <a href="../livres.html">livres.html</a></li>
            <li>La page <a href="../presentation.html">presentation.html</a>
        </ul>
    </body>
</html>
```

On va donc utiliser ici la notation ../ pour signifier que les pages de destination se trouvent dans des dossiers parents par rapport à notre page de départ. Notez qu'on précise deux fois ../ pour créer un lien vers notre page « home.html » tout simplement car cette page est située dans un dossier deux niveaux au-dessus de la page « liens.html ».

Bon à savoir : On indiquera autant de fois ../ en valeur de l'attribut href qu'il faudra traverser de dossiers pour atteindre la page de destination à partir de la page de départ.

A noter : Il est possible que vous deviez un jour faire un lien vers une page qui se situe dans un autre dossier qui n'est pas un ascendant (parent) ou un descendant (enfant) direct de la page de départ. Dans ce cas-là, vous devrez utiliser un mélange de ../ et de noms de dossiers en valeur de l'attribut href.

Pour illustrer cela, on peut créer un nouveau dossier seo dans le dossier cours et placer une page qu'on va appeler referencement.html dedans. L'idée va ici de faire un lien de la page liens.html vers referencement.html et inversement. Voici comment on va s'y prendre :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Liens</title>
        <meta charset='utf-8'>
    </head>
    <body>
        <p>Cette page se nomme "liens.html" et se
        situe dans le dossier "code" (niveau 2)</p>

        <p>Accédez directement à :</p>
        <ul>
            <li>La page <a href="../../home.html">home.html</a></li>
            <li>La page <a href="../livres.html">livres.html</a></li>
            <li>La page <a href="../presentation.html">presentation.html</a>
            <li>La page <a href="../seo/referencement.html">referencement.html</a></li>
        </ul>
    </body>
</html>
```

Ici, on indique qu'il faut commencer par remonter dans le dossier parent cours puis ensuite aller dans le dossier seo pour accéder à la page referencement.html à partir de la page liens.html. N'hésitez pas à deviner le code pour aller dans l'autre sens, ça vous fera un bon exercice !

Ouvrir un lien dans un nouvel onglet grâce à l'attribut target

Jusqu'à présent, lorsque nous cliquions sur les liens créés en HTML, nous étions immédiatement redirigés vers la nouvelle page et notre page cible s'ouvrait dans le même emplacement (le même cadre ou « frame ») que le lien de départ.

Souvent, si vous avez un site, vous voudrez généralement que le lien s'ouvre dans un nouvel onglet afin que le visiteur ne perde pas la page de votre site. On va pouvoir faire cela avec l'attribut target.

L'attribut target va nous permettre de choisir où doit s'ouvrir notre page de destination. En pratique, nous utiliserons très souvent la valeur _blank qui spécifie que la nouvelle page doit s'ouvrir dans un nouvel onglet.

Voici un exemple pratique avec un lien externe menant vers la page d'accueil de Wikipédia et s'ouvrant dans un nouvel onglet :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= 'utf-8'>
        <style>
            a{
                color: black;
                text-decoration:none;
                cursor: auto;
            }
        </style>
    </head>
    <body>
        <p>Un lien vers la page d'accueil
        de <a href="https://www.wikipedia.org/" target="_blank">Wikipédia</a></p>
    </body>
</html>
```

Pour information, voici les différentes valeurs que peut prendre l'attribut HTML target afin de modifier le comportement de nos liens HTML :

Valeur de target	**Comportement**
_self	Valeur par défaut : la page cible s'ouvre dans le même emplacement (cadre ou « frame ») que là où l'utilisateur a cliqué
_blank	La page cible s'ouvre dans un nouvel onglet ou dans une nouvelle fenêtre
_parent	La page cible s'ouvre dans la cadre (frame) de niveau immédiatement supérieur par rapport à l'emplacement du lien
_top	La page cible s'ouvre dans la fenêtre hôte (par-dessus le frameset)
Nom du cadre (frame)	Ouverture de la page cible dans le cadre portant le nom cité (en valeur de l'attribut name)

Créer des liens vers une autre partie d'une même page en HTML

Dans certains cas, lorsque vous construisez une page très longue, il va pouvoir être intéressant de proposer un sommaire avec des liens cliquables qui vont transporter l'utilisateur directement à un endroit précis de la page.

Nous allons également pouvoir créer ce type de liens (parfois appelé liens « ancre ») avec notre élément HTML `a` ainsi qu'avec un attribut `id` qui va nous servir à identifier l'endroit de la page où l'utilisateur doit être renvoyé.

L'attribut `id` sert, comme son nom l'indique, à identifier un élément HTML en particulier dans une page pour ensuite pouvoir le cibler précisément (pour pouvoir lui appliquer des styles CSS entre autres).

Nous étudierons le fonctionnement de l'attribut `id` en détail plus tard dans ce cours. Pour le moment, retenez simplement que chaque `id` dans une page doit posséder une valeur unique puisqu'encore une fois un `id` sert à identifier précisément un élément en particulier.

Pour créer un lien de type « ancre », nous allons devoir procéder en deux étapes : tout d'abord, nous allons attribuer un attribut `id` aux différents éléments de notre page au niveau desquels on souhaite renvoyer nos visiteurs.

Ensuite, nous allons créer nos liens de renvoi en soi. Pour cela, nous allons passer à nos différents attributs `href` les valeurs correspondantes aux noms de mes `id` précédés d'un dièse (symbole #).

Voyons immédiatement comment créer ce type de lien en pratique. Pour bien illustrer leur fonctionnement, il va falloir que le contenu de notre page soit plus grand que la hauteur de la fenêtre pour faire apparaitre une barre de défilement (scrolling). Nous avons différents moyens de faire cela : créer une page avec beaucoup de texte, réduire la largeur d'affichage du texte en CSS pour qu'il prenne plus de place en hauteur, etc.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        Accès direct à :
        <ul>
          <li><a href="#liens">Création de liens en HTML</a> ;</li>
          <li><a href="#ext">Liens externes</a> ;</li>
          <li><a href="#int">Liens internes</a> ;</li>
          <li><a href="#tget">Attribut target</a> ;</li>
          <li><a href="#ancre">Liens "ancre"</a>.</li>
        </ul>

        <h2 id="liens">Création de liens en HTML</h2>
        <p class="long">Imaginez beaucoup de contenu</p><br><br><br><br>
        <h2 id="ext">Créer des liens externes en HTML</h2>
        <p class="long">Imaginez beaucoup de contenu</p><br><br><br><br>
        <h2 id="int">Créer des liens internes en HTML</h2>
        <p class="long">Imaginez beaucoup de contenu</p><br><br><br><br>
        <h2 id="tget">L'attribut target</h2>
        <p class="long">Imaginez beaucoup de contenu</p><br><br><br><br>
        <h2 id="ancre">Créer des liens ancres</h2>
        <p class="long">Imaginez beaucoup de contenu</p><br><br><br><br>
        <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
        <p class="long">Imaginez beaucoup de contenu</p><br><br><br><br>
    </body>
</html>
```

Dans l'exemple ci-dessus, j'ai créé une liste de type sommaire au début de ma page car celle-ci est très longue et j'ai accolé un `id` à chacun de mes titres `h2`.

Chaque élément de mon sommaire va correspondre à une partie vers laquelle je souhaite envoyer mes visiteurs. Je vais donc placer un élément de lien HTML `a` dans chacun des éléments de liste.

Ici, je vous invite à regarder attentivement les valeurs données à mes différents attributs `href` : vous remarquez que les différentes valeurs correspondent aux noms de mes `id` précédés d'un dièse (symbole #), symbole qui sert justement à cibler un `id` en CSS.

Lorsqu'on clique sur un élément du sommaire, nous sommes renvoyés directement à la partie de la page correspondante.

Utiliser une image en ancre de lien HTML

Nous n'avons pas encore étudié les images et je ne veux pas vous donner trop d'informations d'un coup afin que vous puissiez vous concentrer sur le sujet actuellement traité.

Cependant, sachez qu'il va être tout à fait possible d'utiliser une image à la place d'un texte comme ancre ou contenu cliquable pour un lien. Nous verrons comment faire cela dans le chapitre réservé à l'insertion d'images en HTML.

Envoi de mails et téléchargement de fichiers

L'élément `a` en HTML ne va pas uniquement être très utile pour créer des liens entre différentes pages ou de ramener à différents endroits d'une même page mais va également nous permettre de lier des ressources à nos pages.

Dans ce nouveau chapitre, nous allons étudier deux autres utilisations courantes de cet élément :

- Utiliser l'élément `a` pour permettre aux utilisateurs de nous envoyer un mail ;
- Utiliser l'élément `a` pour permettre aux utilisateurs de télécharger un fichier.

Utiliser l'élément a pour permettre l'envoi d'un mail

On peut utiliser l'élément `a` pour transmettre notre adresse mail à nos utilisateurs et leur permettre de nous envoyer simplement un mail.

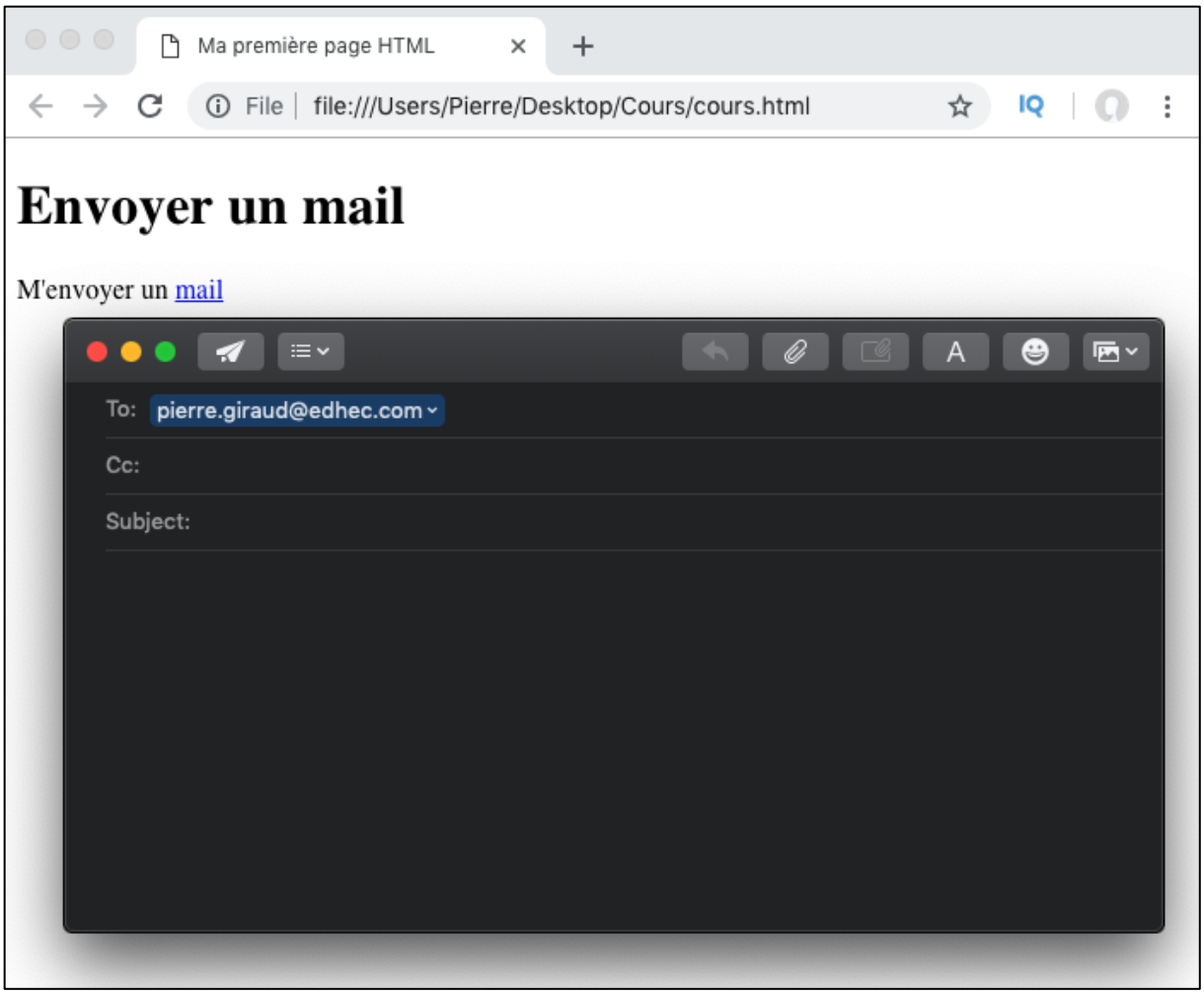
Pour permettre l'envoi d'un mail en HTML, on va placer indiquer en valeur de l'attribut `href` de notre élément `a` la valeur `mailto :` suivie de notre adresse email.

Lorsqu'un visiteur va cliquer sur notre lien, sa messagerie par défaut va automatiquement s'ouvrir. Par exemple, si vous avez un Mac, ce sera certainement l'application « Mail » qui va s'ouvrir. De plus, le champ destinataire sera automatiquement rempli avec notre adresse email.

Note : si vous travaillez sous Windows, il est possible que rien ne se passe si vous n'avez configuré aucune messagerie par défaut.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Envoyer un mail</h1>
        <p>M'envoyer un <a href="mailto:pierre.giraud@edhec.com">mail</a></p>
    </body>
</html>
```

Dès qu'un visiteur clique sur le texte de notre lien, sa messagerie par défaut s'ouvre s'il en a configuré une
:



Utiliser l'élément a pour permettre le téléchargement d'un fichier

Vous pouvez encore utiliser l'élément `a` pour permettre à vos visiteurs de télécharger certains types de fichiers, comme des fichiers PDF ou Word par exemple.

Pour la plupart des fichiers, il va simplement suffire d'indiquer leur adresse (relative ou leur URL complète) en valeur de l'attribut `href`. Lorsqu'un utilisateur va cliquer sur le lien, le fichier lié va s'ouvrir dans le navigateur et l'utilisateur n'aura alors plus qu'à faire un clic droit sur le fichier ou utiliser les options de son navigateur pour l'enregistrer.

Cette première solution fonctionne mais demande à ce que l'utilisateur fasse lui-même la démarche de télécharger le fichier. On va également pouvoir « forcer » le téléchargement d'un fichier en ajoutant un attribut `download` dans l'élément `a` tout en indiquant l'adresse du fichier en question en valeur de l'attribut `href`.

L'attribut `download` peut prendre en valeur (facultative) le nom sous lequel on souhaite que les utilisateurs téléchargent le fichier.

Dès que l'utilisateur va cliquer sur le lien, l'attribut `download` va faire que le le téléchargement du fichier lié va se lancer automatiquement. Attention cependant, cet

attribut n’a pendant très longtemps pas été supporté par certains navigateurs majeurs dont Safari. Certaines anciennes versions de Safari encore présentes aujourd’hui peuvent donc ne pas le reconnaitre.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ma première page HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Cours HTML</h1>
        <!--On imagine qu'on possède une image nommée "logo.png" située dans le
        même dossier que notre page HTML-->
        <p>Mon <a href="logo.png" download="pg-logo">logo</a> à télécharger</p>
    </body>
</html>
```

Note : Vous n’allez pas pouvoir faire télécharger n’importe quel type de fichiers à vos visiteurs. En effet, les navigateurs récents vont bloquer le téléchargement de certains fichiers dont les extensions auront été jugées comme « potentiellement dangereuses ». De plus, certaines autres extensions de fichier vont être directement interprétées par le navigateur de vos visiteurs. Cela va être le cas si vous essayez de faire télécharger la source d’un fichier `.html` à vos visiteurs par exemple : le navigateur va ouvrir le fichier et interpréter son code comme pour n’importe quelle autre page et n’afficher donc que le résultat visuel lié au code HTML du fichier. Ici, une astuce simple consiste à compresser les fichiers que vous voulez faire télécharger en `.zip` par exemple.

Compatibilité, support et validation du code

Nous allons terminer cette première partie relative aux notions de base en HTML avec un mot sur la problématique complexe de la compatibilité du code entre les différents navigateurs ainsi que sur l'importance d'avoir toujours un code valide et optimisé.

La comptabilité intra-navigateur, inter-navigateurs et relatives aux différents appareils du code

Le problème de compatibilité et de consistance du code n'est pas nouveau : il y a quelques années seulement de cela, le web était beaucoup plus décousu qu'aujourd'hui et les règles n'étaient pas encore fixées.

Jusqu'au début des années 2000, Microsoft n'avait pas de concurrent sérieux et utilisait de sa puissance économique afin que son navigateur Internet Explorer (aujourd'hui appelé Edge) soit installé par défaut sur toutes les machines.

Cela permettait à Microsoft de s'offrir des libertés et notamment celle de développer de nouveaux standards de code ou d'implémenter des codes d'une façon différentes des autres.

Ensuite, dans les années 2000, de nouveaux acteurs sérieux ont fait apparition et ont commencé à se livrer une guerre économique. A cette époque, il était courant que différents navigateurs implémentent de manière totalement différente certains codes et utilisent leurs propres normes.

Ainsi, certains éléments ou attributs HTML par exemple n'étaient pas supportés par certains navigateurs et on devait donc créer des codes différents afin que chaque navigateur affiche le résultat voulu.

Depuis quelques années cependant, et grâce à l'impulsion du W3C qui n'a cessé de pousser des standards de développement, on assiste à une homogénéisation et une uniformisation de la prise en charge des codes sur la plupart des navigateurs sérieux. Cela est une excellente nouvelle pour nous, développeurs.

Cependant, tout n'est pas encore parfait comme on a pu le voir avec la prise en charge de l'attribut `download` par exemple dans les leçons précédentes.

Aujourd'hui, cependant, il nous faut faire face à de nouveaux défis tout aussi importants qui sont des défis liés aux différents appareils utilisés par nos visiteurs et à l'ergonomie.

En effet, les langages informatiques ont évolué au cours de ces dernières années pour proposer toujours davantage de fonctionnalités pour répondre notamment aux technologies émergentes et aux nouveaux besoins des utilisateurs.

L'apparition de l'internet mobile et sur tablette a notamment considérablement complexifié les questions liées à l'ergonomie des sites web puisqu'il était hors de question d'afficher

autant d'information ou d'avoir des pages aussi lourdes sur mobile que sur un ordinateur de bureau classique par exemple.

Il faudra donc toujours garder ces questions en tête lorsque nous développerons nos propres projets et bien réfléchir en amont pour produire le résultat le plus conforme à nos attentes possibles.

Les entités HTML

Le HTML possède des caractères réservés. Par exemple, vous ne pouvez pas écrire les signes « < » et « > » tels quels dans vos pages web, tout simplement car le navigateur pensera que vous venez d'ouvrir une balise d'élément.

Pour remédier à ce problème, nous allons devoir « échapper » ces caractères réservés en utilisant ce qu'on appelle des entités HTML. Les entités vont être des suites de caractères représentant un caractère réservé en HTML.

Voici quelques-unes des entités les plus courantes et leur signification :

Nom de l'entité	**Résultat visuel**
&lt;	< (chevron ouvrant)
&gt;	> (chevron fermant)
&amp;	& (esperluette)
	(espace insécable)

Nous connaissons déjà certaines de ces entités comme l'entité ` ` (pour « non-breaking space ») qui sert à créer une espace insécable en HTML.

Voyons immédiatement un exemple d'utilisation de ces entités :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <p>L'élément HTML break s'écrit comme ceci : &lt;br&gt;</p>
        <p>De     grandes    espaces</p>
    </body>
</html>
```

Tester la validité de son code

Vous devez toujours vous efforcer d'écrire un code valide. Cela évitera des bugs potentiels et votre site sera au final mieux référencé sur les moteurs de recherche.

Pour vérifier la validité d'un code HTML ou CSS, le w3c (World Wide Web Consortium), c'est-à-dire l'organisme qui gère l'évolution des langages lus par les navigateurs comme le HTML et le CSS entre autres, a mis à disposition des validateurs de code qui sont gratuits.

Vous pouvez trouver les validateurs HTML et CSS aux adresses suivantes :

- HTML : https://validator.w3.org/
- CSS : https://jigsaw.w3.org/css-validator/

PARTIE III

Les bases du CSS

Sélecteurs et propriétés CSS

Le CSS est un langage qui a été inventé pour styliser les contenus de nos pages en leur appliquant des styles.

Dans cette nouvelle partie, nous allons passer en revue les notions de base du CSS en comprenant notamment les grands principes de fonctionnement de ce langage et en apprenant à cibler des contenus de manière précise pour pouvoir leur appliquer des styles.

Les sélecteurs CSS : définition

Pour pouvoir appliquer un style à un contenu, il va déjà falloir le cibler, c'est-à-dire trouver un moyen d'indiquer qu'on souhaite appliquer tel style à un contenu en particulier.

Pour cela, nous allons utiliser des sélecteurs. Les sélecteurs sont l'un des éléments fondamentaux du CSS.

De manière très schématique et très simplifiée, nous allons utiliser nos sélecteurs en CSS pour cibler des contenus HTML et leur appliquer des styles.

Il existe différents types de sélecteurs en CSS : certains sélecteurs vont s'appuyer sur le nom des éléments, comme le sélecteur CSS `p` par exemple qui va servir à cibler tous les éléments `p` d'une page. Ce type de sélecteurs est appelé « sélecteur d'éléments » tout simplement car ils vont être identiques aux éléments HTML sélectionnés ou encore « sélecteurs simples ».

D'autres sélecteurs, en revanche, vont être plus complexes et nous permettre de sélectionner un élément HTML en particulier ou un jeu d'éléments HTML en fonction de leurs attributs ou même de leur état : on va ainsi pouvoir appliquer des styles à un élément uniquement lorsque la souris de l'utilisateur passe dessus par exemple.

Nous allons apprendre à utiliser la majorité des sélecteurs CSS et notamment les plus courants et les plus utiles dans la suite de ce cours.

Les propriétés CSS : définition

Les propriétés vont nous permettre de choisir quel(s) aspect(s) (ou "styles") d'un élément HTML on souhaite modifier.

Par exemple, nous allons pouvoir modifier la couleur d'un texte et lui appliquer la couleur que l'on souhaite grâce à la propriété `color` (« couleur », en français).

Une propriété va être accompagnée d'une ou plusieurs valeurs qui vont définir le comportement de cette propriété.

Par exemple, la propriété `color` peut prendre le nom d'une couleur (en anglais). Si l'on

donne la valeur `red`(rouge) à notre propriété `color`, les textes au sein des éléments HTML auxquels on applique cette propriété s'afficheront en rouge.

Les déclarations CSS : premier exemple pratique

Prenons immédiatement un premier exemple ensemble en expliquant bien à quoi correspond chaque élément du code afin d'illustrer ce que nous venons de dire et de bien voir comment le CSS fonctionne.

Je vous demande pour le moment de ne pas vous soucier des questions pratiques concernant la liaison entre les codes HTML et CSS mais simplement de vous concentrer sur le code CSS présenté.

```css
p{
  color: blue;
  border: 2px solid orange;
  padding: 5px;
}
```

Détaillons le code CSS ci-dessus. Ici, nous utilisons le sélecteur CSS simple `p` pour cibler tous les paragraphes de nos pages HTML. Ensuite, nous ouvrons une paire d'accolades. Entre ces accolades, nous allons préciser les différents styles que l'on souhaite appliquer à nos éléments `p`.

En l'occurrence, on définit une couleur, bordure et une marge interne personnalisées pour tous nos paragraphes grâce aux propriétés CSS `color`, `border` et `padding`.

Le texte de nos paragraphes va donc s'afficher en bleu et nos paragraphes auront des bordures solides oranges de 2px d'épaisseur et des marges internes de 5px.

Le couple « propriété : valeur » est appelée « déclaration » en CSS. Chaque déclaration doit se terminer par un point-virgule.

On va pouvoir écrire autant de déclarations que l'on souhaite à l'intérieur du couple d'accolades qui suit un sélecteur en CSS et ainsi pouvoir définir le comportement de plusieurs propriétés facilement.

cours.html (~/Desktop)

cours.html × cours.css ×

(root) > Users > Pierre > Desktop > cours.html > Ln: 16 Col: 1

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="utf-8">
5          <title>Cours HTML et CSS</title>
6          <link rel="stylesheet" href="cours.css">
7      </head>
8      <body>
9          <h1>Un titre</h1>
10         <p>Un premier paragraphe avec des styles CSS ajoutés</p>
11         <p>Un autre paragraphe</p>
12     </body>
13 </html>
```

cours.css (~/Desktop)

cours.html × cours.css ×

(root) > Users > Pierre > Desktop > cours.css > Ln: 7 Col: 1

```
1  p{
2    color: blue;
3    border: 2px solid orange;
4    padding: 5px;
5  }
```

Cours HTML et CSS ×

File | /Users/Pierre/Desktop/cours.html#ancre

Un titre

Un premier paragraphe avec des styles CSS ajoutés

Un autre paragraphe

Où écrire le code CSS ?

Avant d'étudier les mécanismes du CSS en soi, il convient de comprendre où placer le code CSS afin qu'il s'applique bien à un fichier HTML.

En effet, lorsqu'on code, pensez bien que rien n'est jamais « magique » et qu'au contraire tout le code qu'on va pouvoir écrire repose sur des règles et des mécanismes. Comprendre ces règles et ces mécanismes et notamment comment différents langages de programmation vont pouvoir fonctionner ensemble est certainement l'une des choses les plus complexes lorsqu'on est débutant.

Pour cela, je pense qu'il ne faut pas essayer de tout comprendre tout de suite : c'est tout à fait normal s'il y a des mécanismes dont vous ne comprenez pas tous les rouages immédiatement. Avec un peu de temps, de la pratique et de nouvelles connaissances sur la programmation les choses pas claires au début devraient devenir de plus en plus évidentes.

Dans le cas présent, nous avons notre code HTML d'un côté et nous aimerions lui appliquer des styles en CSS. Cependant, il va falloir d'une manière ou d'une autre « lier » notre code CSS à notre code HTML afin que les éléments de nos pages HTML tiennent bien compte des styles qu'on a voulu leur appliquer en CSS.

Pour faire cela, nous allons pouvoir écrire le code CSS à trois endroits différents. Chaque méthode va présenter des avantages et des inconvénients selon une situation donnée et c'est le sujet que nous allons aborder dans cette leçon.

Méthode n°1 : écrire le CSS au sein du fichier HTML, dans un élément style

La première façon d'écrire du code CSS va être à l'intérieur même de notre page HTML, au sein d'un élément `style`.

En plaçant le CSS de cette façon, le code CSS ne s'appliquera qu'aux éléments de la page HTML dans laquelle il a été écrit.

Cette première méthode d'écriture du CSS n'est pas recommandée, pour des raisons de maintenance et d'organisation du code en général. Cependant, elle peut s'avérer utile pour modifier rapidement les styles d'une page HTML ou si vous n'avez pas facilement accès aux fichiers de style de votre site.

Nous voyons donc cette première méthode à titre d'exemple, afin que vous sachiez l'identifier si un jour vous voyez du code CSS écrit de cette façon dans un fichier et que vous puissiez l'utiliser si vous n'avez pas d'autre choix.

Nous allons devoir ici placer notre élément `style` au sein de l'élément `head` de notre fichier HTML. Voici comment on va écrire cela en pratique :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Où écrire le CSS ?</title>
        <meta charset= "utf-8">
        <style>
            body{
                background-color: orange;
            }
            p{
                color: blue;
                font-size: 16px;
            }
        </style>
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>
        <p>Un paragraphe</p>
        <p>Un deuxième paragraphe</p>
    </body>
</html>
```

Ici, nous créons un fichier HTML tout à fait classique contenant un titre `h1` et deux paragraphes.

Nous voulons ensuite rajouter des styles à notre page. Pour cela, nous plaçons un élément `style` dans l'élément `head` de notre page. Nous allons déclarer nos styles CSS au sein de cet élément.

Dans mon code CSS, je commence par cibler l'élément `body` avec le sélecteur élément du même nom et je définis une couleur de fond (`background-color`) orange pour cet élément. Comme l'élément `body` représente toute la partie visible de ma page, le fond de la page entière sera orange.

Ensuite, je définis également une couleur bleue pour le texte de mes paragraphes ainsi qu'une taille de police d'écriture de 16px.

Voici le résultat obtenu :

Méthode n°2 : déclarer le CSS au sein du fichier HTML, dans des attributs style

Nous pouvons également écrire notre code CSS au sein d'attributs `style` qu'on va ajouter à l'intérieur de la balise ouvrante des éléments HTML pour lesquels on souhaite modifier les styles.

Nous allons passer en valeurs des attributs `style` des déclarations CSS pour modifier certains styles précis de l'élément HTML en question. En effet, en utilisant cette méthode, les styles déclarés dans un attribut `style` ne vont s'appliquer qu'à l'élément dans lequel ils sont écrits, et c'est la raison pour laquelle nous n'allons pas avoir besoin de préciser de sélecteur ici.

Attention à ne pas confondre les attributs `style` qu'on va devoir placer au sein de la balise ouvrante de chaque élément dont on souhaite modifier les styles avec l'élément `style` qu'on va placer dans l'élément `head` de nos fichiers HTML.

Dans l'exemple ci-dessous, on applique à nouveau une couleur de fond orange à notre élément `body` ainsi qu'une couleur bleue et une taille de 20px au texte de notre premier paragraphe uniquement :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Où écrire le CSS ?</title>
        <meta charset= "utf-8">
    </head>

    <body style="background-color: orange;">
        <h1>Un titre de niveau 1</h1>
        <p style="color: blue; font-size: 20px;">Un paragraphe</p>
        <p>Un deuxième paragraphe</p>
    </body>
</html>
```

Cette deuxième méthode d'écriture du CSS, bien qu'elle puisse sembler pratique à priori puisqu'elle permet de n'appliquer des styles qu'à un élément en particulier plutôt qu'à tous les éléments d'un même type n'est également pas recommandée et est à éviter tant que possible pour des raisons de maintenabilité et de performance du code.

En effet, déclarer nos styles comme cela n'est vraiment pas efficient puisque cela va demander énormément d'écriture et également énormément de temps de réécriture le jour où l'on souhaite modifier des styles.

Pas d'inquiétude : nous allons apprendre à cibler précisément un élément ou un groupe d'éléments en particulier pour leur appliquer des styles personnalisés plus tard dans ce cours.

Méthode n°3 : écrire le CSS dans un fichier séparé

Finalement, nous pouvons écrire notre code CSS dans un fichier séparé portant l'extension « .css ». C'est la méthode recommandée, qui sera utilisée autant que possible.

Cette méthode comporte de nombreux avantages, notamment une meilleure maintenabilité du code grâce à la séparation des différents langages, ainsi qu'une meilleure lisibilité.

Cependant, le plus gros avantage de cette méthode est qu'on va pouvoir appliquer des styles à plusieurs pages HTML en même temps, d'un seul coup.

En effet, en utilisant l'une des deux premières méthodes, nous aurions été obligés de réécrire tout notre code CSS pour chaque page HTML (ou même pour chaque élément !) composant notre site puisque les codes CSS étaient déclarés dans une page ou dans un élément spécifique et ne pouvaient donc s'appliquer qu'à la page ou qu'à l'élément dans lesquels ils étaient déclarés.

De plus, en cas de modification, il aurait également fallu modifier chacune de nos pages à la main, ce qui n'est pas viable pour un site de taille moyenne qui va être composé de quelques centaines de pages.

En déclarant notre code CSS dans un fichier séparé, au contraire, nous allons pouvoir utiliser le code de ce fichier CSS dans autant de fichiers HTML qu'on le souhaite, en indiquant aux différents fichiers HTML qu'ils doivent appliquer les styles contenus dans ce fichier CSS. Ainsi, lorsque nous voudrons modifier par exemple la couleur de tous les paragraphes de nos pages HTML nous n'aurons qu'à modifier la déclaration relative dans le fichier CSS.

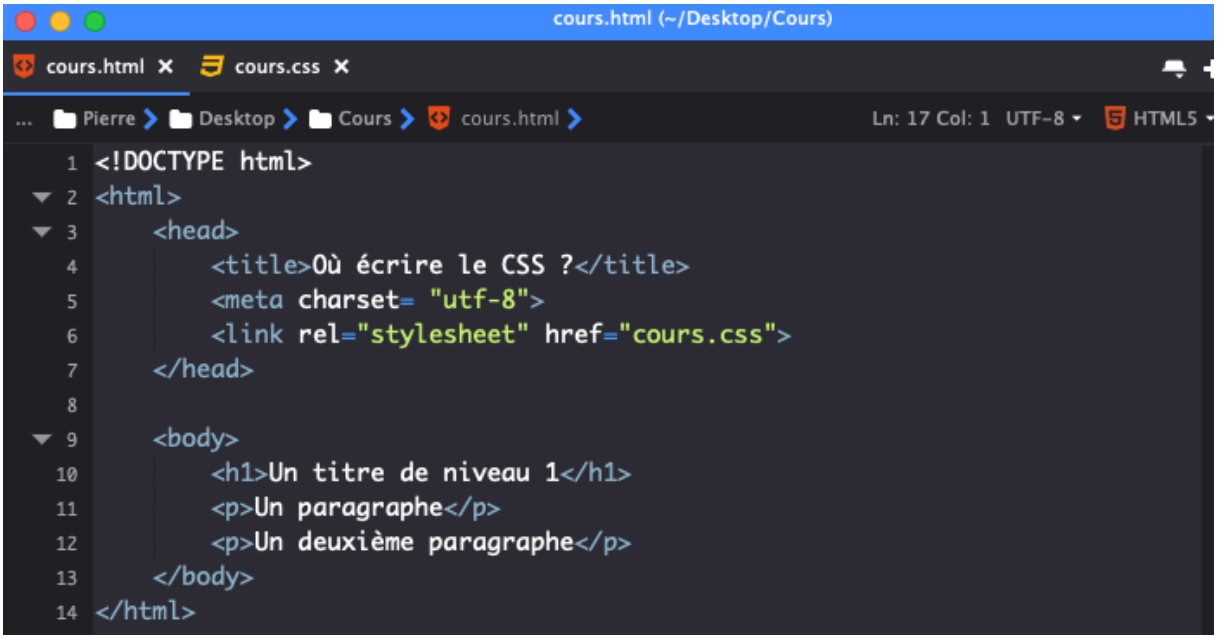
Voyons immédiatement comment mettre cela en place en pratique. Pour cela, nous allons commencer par créer un nouveau fichier dans notre éditeur qu'on va appeler `cours.css`. Nous allons enregistrer ce fichier et le placer dans le même dossier que notre page HTML pour plus de simplicité.

Nous travaillons donc dorénavant avec deux fichiers : un fichier appelé `cours.html` et un fichier `cours.css`.

Il va donc maintenant falloir « lier » notre fichier HTML à notre fichier CSS pour indiquer au navigateur qu'il doit appliquer les styles contenus dans le fichier `cours.css` à notre fichier `cours.html`.

Pour cela, nous allons utiliser un nouvel élément HTML : l'élément `link` (« lien », en français). On va placer l'élément `link` au sein de l'élément `head` de notre fichier HTML. Cet élément se présente sous la forme d'une balise orpheline et va avoir besoin de deux attributs pour fonctionner correctement :

- Un attribut `rel` qui va nous servir à préciser le type de ressource que l'on souhaite lier à notre fichier HTML. Dans notre cas, nous indiquerons la valeur `stylesheet` pour « feuille de style » ;
- Un attribut `href` qui va indiquer l'adresse relative de la ressource que l'on souhaite lier par rapport à l'emplacement de notre fichier HTML. Ici, comme nous avons enregistré nos deux fichiers dans le même dossier, il suffira d'indiquer le nom de notre fichier CSS en valeur de `href`.



Nos deux fichiers sont maintenant liés et les styles déclarés dans notre fichier CSS vont bien être appliqués aux éléments de notre page HTML.


Où écrire le CSS ?
File | file:///Users/Pierre/Desktop/Cours/cours.html
Un titre de niveau 1
Un paragraphe
Un deuxième paragraphe

Commentaires et indentation en CSS

Les commentaires ne sont pas des éléments spécifiques au langage HTML. En réalité, la grande majorité des langages de programmation permettent aux développeurs de commenter leur code via des syntaxes différentes propres à chaque langage car commenter est reconnu comme une bonne pratique en programmation et se révèle souvent indispensable ou à minima très utile.

Dans cette nouvelle leçon, nous allons donc voir comment commenter en CSS et également discuter de l'indentation en CSS.

Commenter le code CSS

Tout comme nous avons vu qu'on pouvait écrire des commentaires en HTML, nous allons également pouvoir commenter notre code CSS.

Les commentaires n'influent une nouvelle fois en rien sur le code et ne sont pas visibles par les utilisateurs.

Commenter le code CSS n'est pas une option : cela va très vite devenir indispensable car vous allez vous rendre compte que les fichiers CSS s'allongent très vite.

Il est donc essentiel de bien organiser et de bien commenter son code CSS afin de ne pas faire d'erreur en appliquant par exemple deux styles différents à un même élément.

Le CSS, tout comme le HTML et à la différence d'autres langages de développement ne possède qu'une seule syntaxe qui va nous permettre de créer à la fois des commentaires mono-ligne et multi-lignes.

Cette syntaxe est la suivante : `/*Un commentaire CSS*/`. Regardez plutôt l'exemple ci-dessous :

```
/*Un premier commentaire CSS*/
body{
    background-color: orange;
}

/*Un deuxième commentaire
 *sur plusieurs
 *lignes*/
p{
    /*color: blue;*/
    font-size: 20px;
}
```

Dans l'exemple ci-dessus, notez que les étoiles en début de ligne pour mon commentaire multi-lignes ne sont absolument pas nécessaires (à part pour la première ligne, évidemment) : ce n'est que de la décoration afin de bien voir que l'on commente.

Vous pouvez également remarquer une utilisation intéressante des commentaires et qui est très commune en CSS : le fait de commenter une déclaration CSS.

En effet, vous voudrez parfois supprimer momentanément une déclaration CSS, pour effectuer des tests par exemple. Plutôt que de l'effacer complètement, vous pouvez la commenter.

Ainsi, la déclaration CSS ne sera plus prise en compte. Vous n'aurez ensuite plus qu'à enlever le commentaire pour la « réactiver ».

Indenter en CSS

Indenter en CSS est également très important afin de conserver le plus de clarté possible dans son code et de paraître professionnel si un jour vous devez le distribuer.

En termes de règles, nous indenterons en général d'une tabulation les différentes déclarations concernant un sélecteur donné.

Pour plus de lisibilité, nous retournerons également à la ligne après chaque déclaration. Notez que cela augmentera de façon très minime le temps d'exécution du code et donc le temps d'affichage de la page.

Cependant, en phase de développement tout au moins, il est essentiel de conserver un code aéré et propre. Vous pourrez toujours le compresser par la suite ; de nombreux outils existent sur le web pour cela.

Sélecteurs CSS simples et combinateurs

Le CSS va nous permettre de mettre en forme nos contenus HTML en appliquant des styles aux différents éléments. Cependant, pour appliquer un style particulier à un ou plusieurs éléments HTML en CSS, il va avant tout falloir les cibler, c'est-à-dire indiquer avec précision à quels éléments doivent s'appliquer les styles créés en CSS.

Le but de cette leçon est d'apprendre à se servir de quelques sélecteurs CSS « simples » et de comprendre leur limitation. Nous allons également en profiter pour définir plus précisément les différents types de sélecteurs CSS et directement voir comment combiner différents sélecteurs simples pour en créer des plus complexes.

Les sélecteurs CSS éléments ou sélecteurs « simples »

Il existe de nombreux types de sélecteurs CSS et autant de moyens de cibler des contenus HTML en CSS.

La manière la plus simple de cibler un type d'éléments HTML en CSS est néanmoins d'utiliser des sélecteurs éléments ou sélecteurs « simples ». Ces sélecteurs sont appelés « sélecteurs éléments » tout simplement car ils reprennent le nom des éléments HTML qu'ils sélectionnent.

Par exemple, le sélecteur CSS `p` va cibler tous les éléments `p` (c'est-à-dire tous les paragraphes) d'une page HTML.

De même, le sélecteur CSS `h1` va nous permettre d'appliquer des styles à notre titre `h1`, le sélecteur `a` va nous permettre de mettre en forme nos liens, etc.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>
        <p>Un paragraphe contenant un <a href="http://wikipedia.org">lien</a></p>
        <p>Un deuxième paragraphe</p>
    </body>
</html>
```

```css
/*Notre titre h1 va s'afficher en rouge*/
h1{
    color: red;
}

/*Nos paragraphes seront bleus*/
p{
    color: blue;
}

/*Le texte de nos liens sera vert et gras*/
a{
    color: green;
    font-weight: bold;
}
```

Comprendre les limitations des sélecteurs CSS simples

L'utilisation de sélecteurs simples doit être favorisée tant que possible car ces sélecteurs consomment moins de ressources que des sélecteurs plus complexes et car ils sont plus clairs.

Ceci étant dit, nous n'allons bien souvent pas pouvoir nous contenter de n'utiliser que des sélecteurs simples car ceux-ci vont considérablement limiter nos options de ciblage et car ils ne vont pas nous permettre d'exploiter toute la puissance du CSS.

En effet, en utilisant uniquement les sélecteurs éléments, nous allons être obligés d'appliquer les mêmes styles à tous les éléments d'un même type ce qui n'est pas très flexible.

Comment appliquer des styles à un élément en particulier ou à plusieurs éléments différents choisis ? Pour faire cela, nous allons devoir utiliser des sélecteurs complexes.

Introduction aux sélecteurs CSS complexes et combinateurs

Toute la puissance du CSS réside dans les options que nous offre ce langage pour cibler précisément un contenu HTML dans une page.

En effet, en plus des sélecteurs simples, le CSS met à notre disposition une panoplie de sélecteurs que l'on va pouvoir utiliser pour cibler des contenus de manière très précise :

- On va pouvoir utiliser des sélecteurs CSS combinateurs qui vont être en fait la combinaison de plusieurs sélecteurs simples à l'aide de caractères spéciaux à la signification précise ;
- On va pouvoir cibler des contenus HTML selon le fait qu'ils possèdent un certain attribut ou même selon la valeur d'un attribut ;
- On va pouvoir utiliser les pseudo classes qui vont nous permettre d'appliquer des styles à des éléments en fonction de leur état, c'est-à-dire en fonction des actions d'un utilisateur (contenu cliqué, coché, visité, etc.), de la place de l'élément dans le document, etc. ;
- On va pouvoir utiliser les pseudo éléments qui vont nous permettre de n'appliquer des styles qu'à certaines parties des éléments.

Nous allons apprendre à faire tout cela au cours de ce cours. Pour le moment, toutefois, nous allons nous contenter de présenter et d'apprendre à manipuler les différents caractères « combinateurs » qui vont nous permettre de combiner des sélecteurs CSS simples afin d'en créer des plus complexes.

Sélectionner tous les éléments avec le sélecteur CSS universel ou sélecteur étoile (*)

Le sélecteur CSS étoile `*` ne nous permet pas à proprement parler de combiner différents sélecteurs simples entre eux mais permet de sélectionner tous les éléments HTML d'une page d'un coup ; c'est pourquoi il est également appelé sélecteur CSS universel.

Ce sélecteur va donc nous permettre d'appliquer les mêmes styles à tous les éléments d'une page. Cela peut être très utile pour par exemple définir une police par défaut ou effectuer un reset des marges de tous les éléments pour ensuite les positionner plus précisément.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>
        <p>Un premier paragraphe</p>
        <p>Un autre paragraphe</p>
        <ul>
            <li>Elément de liste 1</li>
            <li>Elément de liste 2</li>
        </ul>
    </body>
</html>
```

```css
*{
    color: blue;
}
```

Appliquer des styles à plusieurs éléments avec le caractère virgule (,)

Pour appliquer un même style à deux types éléments différents sans avoir à recopier le style deux fois en CSS, nous allons simplement pouvoir séparer nos deux sélecteurs par une virgule. Les styles CSS déclarés juste après s'appliqueront ainsi aux deux éléments ou groupes d'éléments sélectionnés.

Bien évidemment, rien ne nous empêche d'appliquer un même style à 3, 4, … éléments ou groupes d'éléments. Le tout est de séparer les différents sélecteurs par des virgules.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>
        <p>Un premier paragraphe</p>
        <p>Un autre paragraphe</p>
        <ul>
            <li>Elément de liste 1</li>
            <li>Elément de liste 2</li>
        </ul>
    </body>
</html>
```

```css
h1, p{
    color: blue;
}
```

Utiliser plusieurs sélecteurs CSS à la suite

En mentionnant plusieurs sélecteurs à la suite en CSS, nous allons pouvoir appliquer des styles à certains éléments contenus dans d'autres éléments.

Par exemple, utiliser le sélecteur `p a` en CSS va nous permettre d'appliquer des styles à tous les éléments `a` contenus dans des éléments `p` et seulement aux éléments `a` contenus dans des éléments `p`

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>
        <p>Un premier paragraphe</p>
        <!--Note : Utiliser href="#" permet de créer un lien HTML valide mais
        qui ne mène nulle part (qui est "vide"), ce qui est parfois utile-->
        <p>Un autre paragraphe avec un <a href="#">lien</a></p>
        <ul>
            <li><a href="#">Elément de liste (lien) 1</a></li>
            <li>Elément de liste 2</li>
        </ul>
    </body>
</html>
```

```css
p a{
    background-color: yellow;
}
```

Appliquer des styles aux enfants directs d'un autre élément

Nous allons également pouvoir cibler uniquement un élément ou un groupe d'éléments enfants directs d'un autre élément en utilisant le signe de supériorité stricte ou le caractère chevron fermant `>`.

Un élément est un enfant direct ou « descendant direct » d'un autre élément s'il est directement contenu dans celui-ci.

Par exemple, nous allons pouvoir appliquer des styles à tous les liens (éléments `a`) qui sont des enfants directs de l'élément `body` et uniquement à ceux-ci :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>
        <!--Note : Utiliser href="#" permet de créer un lien HTML valide mais
        qui ne mène nulle part (qui est "vide"), ce qui est parfois utile-->
        <a href="#">Un lien</a>
        <p>Un premier paragraphe</p>
        <p>Un autre paragraphe avec un <a href="#">lien</a></p>
        <ul>
            <li><a href="#">Elément de liste (lien) 1</a></li>
            <li>Elément de liste 2</li>
        </ul>
    </body>
</html>
```

```css
body> a{
    background-color: yellow;
}
```

Sélectionner l'élément suivant directement un autre élément en CSS

Le CSS va nous permettre de cibler encore plus précisément un élément en ciblant l'élément suivant directement un autre élément grâce au caractère `+`.

Par exemple, on va pouvoir cibler les éléments `a` (liens) suivant directement un élément `p` :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>
        <!--Note : Utiliser href="#" permet de créer un lien HTML valide mais
        qui ne mène nulle part (qui est "vide"), ce qui est parfois utile-->
        <a href="#">Un lien</a>
        <p>Un premier paragraphe</p>
        <a href="#">Un autre lien</a>
        <p>Un autre paragraphe avec un <a href="#">lien</a></p>
        <ul>
            <li><a href="#">Elément de liste (lien) 1</a></li>
            <li>Elément de liste 2</li>
        </ul>
    </body>
</html>
```

```css
/*Applique une couleur de fond jaune à tout élément a
 *suivant directement un élément p*/
p + a{
    background-color: yellow;
}
```

Sélectionner tous les éléments suivant un autre élément en CSS

Le caractère `~` va nous être plus permissif que le caractère `+` en nous permettant cette fois-ci de sélectionner tous les éléments déclarés après un autre élément en HTML de même niveau (c'est-à-dire possédant le même parent direct).

Par exemple, on va pouvoir cibler tous les éléments `a` placés après un élément `p` et qui sont de même niveau :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>
        <!--Note : Utiliser href="#" permet de créer un lien HTML valide mais
        qui ne mène nulle part (qui est "vide"), ce qui est parfois utile-->
        <a href="#">Un lien</a>
        <p>Un premier paragraphe</p>
        <a href="#">Un autre lien</a>
        <p>Un autre paragraphe avec un <a href="#">lien</a></p>
        <ul>
            <li><a href="#">Elément de liste (lien) 1</a></li>
            <li>Elément de liste 2</li>
        </ul>
        <a href="#">Un dernier lien</a>
    </body>
</html>
```

```css
/*Applique une couleur de fond jaune à tout élément a
 *suivant un élément p et de même niveau*/
p ~ a{
    background-color: yellow;
}
```

Ici, nos styles CSS ne s'appliquent pas à aux deux liens dans notre paragraphe et dans notre élément de liste car ceux-ci ne sont pas au même niveau que les différents paragraphes de notre page : ils sont inclus dans d'autres éléments et ne possèdent donc pas le même parent qu'un autre paragraphe pouvant les précéder.

Résumé des caractères spéciaux permettant de combiner des sélecteurs et signification

Vous pourrez trouver ci-dessous un résumé des différents caractères vus dans cette partie et qui vont nous permettre de combiner des sélecteurs CSS simples afin d'en créer des plus complexes :

Sélecteur CSS	Signification
`*`	Sélectionne tous les éléments
`E, F`	Sélectionne tous les éléments de type E et de type F
`E F`	Sélectionne tous les éléments F à l'intérieur des éléments E
`E > F`	Sélectionne les éléments F enfants directs des éléments E
`E + F`	Sélectionne tout élément F placé directement après un élément E
`E~F`	Sélectionne tout élément F placé après un élément E dans la page

Les attributs HTML class et id et les sélecteurs CSS associés

Cette leçon est consacrée à la découverte et à l'utilisation des attributs HTML `class` et `id`. Nous allons pouvoir ajouter ces deux attributs à n'importe quel élément HTML.

Ces deux attributs sont particuliers en HTML puisqu'ils n'ont pas été créés pour préciser le fonctionnement d'un élément HTML en particulier (ce qui est normalement le rôle de tout attribut HTML) mais vont être principalement utilisés pour cibler certains éléments HTML et leur appliquer des styles en CSS.

Présentation des attributs HTML class et id et cas d'utilisation

Les attributs HTML `class` et `id` sont des attributs dits globaux car on va pouvoir les ajouter dans la balise ouvrante de n'importe quel élément HTML.

Ces deux attributs vont être principalement utilisés dans un but de mise en forme : ils vont nous servir à appliquer des styles CSS aux éléments qui vont les contenir.

En effet, à la différence d'un attribut `href` par exemple, les attributs `class` et `id` ne vont pas servir à préciser le fonctionnement d'un élément HTML mais vont simplement être très utiles pour cibler un élément précisément.

Nous allons effectivement très facilement pouvoir nous resservir de ces deux attributs en CSS grâce aux sélecteurs associés `.class` et `#id`.

Premier exemple d'utilisation des attributs HTML class et id et des sélecteurs CSS associés

Voyons immédiatement de manière pratique comment vont fonctionner les attributs `class` et `id` et comment on va pouvoir les utiliser en CSS pour cibler et appliquer des styles particuliers à des éléments choisis.

Pour cela, nous allons créer une page HTML et allons placer des attributs `class` et `id` dans différents éléments.

Nous allons déjà devoir renseigner une valeur pour chaque attribut `class` et `id`. Les valeurs indiquées pour les attributs ne doivent contenir ni caractères spéciaux ni espaces et commencer par une lettre. En pratique, on essaiera d'attribuer des valeurs qui font sens à nos différents attributs.

Notez déjà que chaque `id` doit avoir une valeur propre ou unique dans une page. En revanche, on va pouvoir attribuer la même valeur à plusieurs attributs `class` différents.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1 id="orange">Un titre de niveau 1</h1>
        <p class="bleu">Un premier paragraphe</p>
        <p class="bleu">Un autre paragraphe avec un <a href="#">lien</a></p>
        <ul>
            <li>Elément de liste 1</li>
            <li class="vert">Elément de liste 2</li>
        </ul>
        <p id="gros">Un troisième paragraphe</p>
    </body>
</html>
```

```css
#orange{
    color: orange;
}

#gros{
    font-size: 24px;
}

.bleu{
    color: blue;
}

.vert{
    color: green;
}
```

Ici, on ajoute un attribut `id="orange"` dans la balise ouvrante de notre élément `h1` et un attribut `id="gros"` à notre dernier paragraphe. On ajoute également un même attribut `class="bleu"` à nos deux premiers paragraphes et un attribut `class="vert"` à un élément de notre liste.

Ensuite, on va lier des styles CSS à ces différents `id` et `class` en utilisant les sélecteurs CSS associés.

Pour cibler un `id` en particulier en CSS, on utilisera le symbole dièse (`#`) suivi de la valeur de l'`id` auquel on souhaite lier des styles.

Pour cibler une `class` en particulier en CSS, on utilisera le symbole point (`.`) suivi de la valeur de la `class` à laquelle on souhaite lier des styles.

Class vs id : Quelles différences et quel attribut utiliser ?

Il existe une différence notable entre les deux attributs `class` et `id` : chaque `id` doit avoir une valeur unique dans une même page tandis que plusieurs attributs `class` peuvent partager la même valeur.

Cela fait que l'attribut `id` est beaucoup plus spécifique que l'attribut `class` et que ces deux attributs vont avoir des rôles et buts différents notamment pour la mise en forme CSS.

Utilisation des classes en HTML et en CSS

Ainsi, nous utiliserons généralement des attributs `class` pour définir des styles généraux et communs à plusieurs éléments dans une même page. Comme nous pouvons donner une même `class` à plusieurs éléments, ils hériteront tous des mêmes styles sauf en cas de conflit (c'est-à-dire dans le cas où le comportement d'une même propriété a déjà été défini en CSS) bien évidemment.

Toute la puissance des attributs `class` et du sélecteur CSS associé va résider dans le fait qu'on va tout à fait pouvoir définir des styles CSS généraux liés à des sélecteurs `.class` avant même de commencer à écrire notre code HTML. Nous n'aurons ensuite plus qu'à fournir les attributs `class` à nos éléments HTML lors de la création de la page.

L'idée va être la suivante : créer des styles CSS et les attacher à des sélecteurs `.class` à priori puis attribuer les attributs `class` relatifs à certains éléments HTML choisis afin que les styles CSS correspondants leur soient appliqués.

Cette façon de procéder peut sembler étrange et "à l'envers" pour les personnes non expertes. Cependant, je vous garantis que c'est une très bonne façon de faire qui peut faire gagner énormément de temps pour un gros projet. C'est par ailleurs toute l'idée derrière l'utilisation de la librairie Bootstrap par exemple.

Utilisation des id en HTML et en CSS

En revanche, comme chaque `id` doit être unique dans une page, nous utiliserons ce sélecteur pour appliquer des styles très précis et pour être sûr de ne cibler qu'un élément HTML en CSS.

C'est la raison pour laquelle on se sert des attributs `id` pour créer des liens de type ancre par exemple. En effet, nous sommes sûrs de lever toute ambiguïté sur la sélection avec un `id` car encore une fois celui-ci doit être unique.

Les sélecteurs CSS `.class` et `#id` ne possèdent donc pas le même degré de précision et ainsi n'ont pas le même ordre de priorité dans les styles attribués aux éléments en cas de conflit. Je vous rappelle ici qu'en cas de conflit sur un style en CSS ce sont les styles du sélecteur le plus précis qui seront appliqués.

L'ordre de priorité d'application des styles en CSS est le suivant (du plus prioritaire ou moins prioritaire) :

1. Les styles liés à un sélecteur `#id` ;
2. Les styles liés à un sélecteur `.class`.
3. Les styles liés à un sélecteur élément ;

Plus d'exemples d'utilisation des attributs class et id en HTML et des sélecteurs CSS associés

Avant tout, retenez que les valeurs indiquées pour les attributs `class` et `id` ne doivent contenir ni caractères spéciaux ni espaces et commencer par une lettre. Idéalement, nous essaierons d'utiliser des noms qui font du sens pour nos attributs `class` et `id`.

On pourra par exemple utiliser des noms relatifs aux propriétés CSS définis avec les sélecteurs associés. Faites bien attention cependant à ne pas utiliser des noms protégés c'est-à-dire des noms déjà utilisés par le HTML et qui ont déjà une signification spéciale.

Attribuer un attribut class et un attribut id à un élément HTML

On peut tout à fait fournir plusieurs attributs à un élément HTML et notamment un attribut `class` et un attribut `id` à un élément.

Reprenons l'exemple précédent en ajoutant un attribut `class` à notre dernier paragraphe pour illustrer cela :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1 id="orange">Un titre de niveau 1</h1>
        <p class="bleu">Un premier paragraphe</p>
        <p class="bleu">Un autre paragraphe avec un <a href="#">lien</a></p>
        <ul>
            <li>Elément de liste 1</li>
            <li class="vert">Elément de liste 2</li>
        </ul>
        <p class="bleu" id="gros">Un troisième paragraphe</p>
    </body>
</html>
```

```css
#orange{
    color: orange;
}

#gros{
    font-size: 24px;
}

.bleu{
    color: blue;
}

.vert{
    color: green;
}
```

Ici, le dernier paragraphe de notre page possède à la fois un attribut `class="bleu"` et un `id="gros"`. Les styles CSS liés à ces deux attributs donc être appliqués à l'élément.

Ici, nos deux attributs `class="bleu"` et `id="gros"` nous servent à appliquer des propriétés CSS différentes (`color` pour notre attribut `class` et `font-size` pour notre `id`). Il n'y a donc pas de risque de conflit.

En revanche, il y aurait eu conflit si on avait précisé des comportements différents pour la même propriété avec nos deux sélecteurs.

Un point sur l'ordre de priorité d'application de styles CSS

Imaginons maintenant qu'on passe un attribut `class` et un attribut `id` à un même élément et qu'on définisse une même propriété CSS de manière différente pour ces `id` et `class`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1 class="bleu" id="orange">Un titre de niveau 1</h1>
        <p class="bleu">Un premier paragraphe</p>
        <p class="bleu">Un autre paragraphe avec un <a href="#">lien</a></p>
        <ul>
            <li>Elément de liste 1</li>
            <li class="vert">Elément de liste 2</li>
        </ul>
        <p class="bleu" id="gros">Un troisième paragraphe</p>
    </body>
</html>
```

```
#orange{
    color: orange;
}

#gros{
    font-size: 24px;
}

.bleu{
    color: blue;
}

.vert{
    color: green;
}
```

Ici on passe un attribut `class="bleu"` et `id="orange"` à notre titre `h1`. Or, on définit le comportement de la même propriété (la propriété `color`) de manière différente dans les sélecteurs `.bleu` et `#orange`. Il y a donc conflit sur les styles.

Comme vous pouvez le voir, notre titre s'affiche en orange ce qui signifie que ce sont les styles liés à l'`id` qui vont être pris en compte plutôt que ceux liés à la `class`.

Vous pouvez ici retenir la règle suivante dans l'application des styles CSS : ce seront toujours les styles liés au sélecteur le plus précis qui seront appliqués en cas de conflit.

Par « précis », on entend le sélecteur qui permet d'identifier le plus précisément l'élément auxquels vont être appliqués les styles.

Ici, comme chaque `id` doit posséder une valeur unique dans une page, le sélecteur CSS lié à notre `id` est très précis et beaucoup plus précis que le sélecteur lié à l'attribut `class` puisqu'il permet d'identifier un élément de manière unique alors qu'un attribut `class` peut être partagé par plusieurs éléments et ne permet donc pas d'identifier un élément en particulier.

Cette notion de précision peut vous sembler un peu floue pour le moment car c'est le genre de notion qu'il est difficile de comprendre sans connaitre l'ensemble du langage. Pas d'inquiétude, tout cela va se préciser au fil des leçons et à chaque nouvelle notion que nous allons aborder.

Attribuer plusieurs attributs class à un élément HTML

L'un des grands intérêts de l'attribut HTML `class` est qu'un même attribut (et donc les styles CSS liés) va pouvoir être partagé par différents éléments. Cela facilite grandement la gestion des styles de nos fichiers HTML et nous permet de gagner beaucoup de temps.

Réciproquement, un même élément HTML va tout à fait pouvoir recevoir différents attributs. Pour cela, il va suffire d'indiquer les différentes valeurs séparées par un espace. Ainsi, une très bonne pratique en CSS et pour la création d'un site va être de na pas surcharger un sélecteur `.class` avec de nombreux styles CSS mais au contraire d'utiliser

de multiples sélecteurs .class qui se contenteront de définir chacun un comportement ou plusieurs propriétés d'un même « type ».

Fonctionner comme cela permet d'avoir un code beaucoup plus clair et d'avancer beaucoup plus vite dans la création d'un site. Regardez plutôt l'exemple ci-dessous pour bien comprendre cette utilisation des attributs class :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1 class="gros souligne orange">Un titre de niveau 1</h1>
        <p class="bleu">Un premier paragraphe</p>
        <p>Un autre paragraphe avec un <a href="#">lien</a></p>
        <ul>
            <li>Elément de liste 1</li>
            <li class="souligne">Elément de liste 2</li>
        </ul>
        <h2 class="gros italique bleu">Un titre de niveau 2</h2>
    </body>
</html>
```

```css
/*On commence par définir des styles CSS qu'on lie a des sélecteurs de
 *class. On n'aura plus qu'à utiliser les attributs class correspondants
 *en HTML ensuite pour appliquer les styles liés à nos éléments*/
.gros{
    font-size: 24px;
}
/*Texte en italique*/
.italique{
    font-style: italic;
}
/*Crée un trait de soulignement sous le texte*/
.souligne{
    text-decoration: underline;
}
.orange{
    color: orange;
}
.bleu{
    color: blue;
}

.vert{
    color: green;
}
```

Notez que faire ceci avec des attributs `id` n'aurait aucun sens puisque les styles des sélecteurs liés n'ont pas vocation à être partagés entre différents éléments (c'est-à-dire à être « réutilisés ») mais sont liés à un élément en particulier et il est donc logique ici de placer tous les styles voulus pour l'élément dans un seul `id`.

Ordre d'application (cascade) et héritage des règles en CSS

Dans cette nouvelle leçon, nous allons étudier et comprendre les mécanismes de cascade et d'héritage en CSS qui sont deux mécanismes fondamentaux de ce langage.

Comprendre comment fonctionne ces mécanismes va nous permettre de savoir quelle règle CSS va être appliquée à quel élément et pourquoi et ainsi de véritablement contrôler le résultat graphique de nos pages HTML.

Comprendre l'importance d'établir un ordre d'application des règles CSS : le problème des conflits

Pour comprendre les mécanismes fondamentaux de cascade et d'héritage en CSS, il faut avant tout comprendre ce qu'est un conflit CSS.

Parfois, plusieurs sélecteurs différents vont nous permettre d'appliquer des styles CSS à un même élément.

Imaginons par exemple un élément `p` auquel on attribuerait un attribut `class` et un attribut `id`. Nous allons pouvoir appliquer des styles CSS à cet élément de trois façons évidentes différentes :

- en utilisant un sélecteur élément ;
- en le ciblant via son attribut `class` ;
- en le ciblant via son attribut `id`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1 class="bigorange">Un titre de niveau 1</h1>

        <p>Un premier paragraphe</p>
        <p class="bigorange">Un autre paragraphe</p>
        <p class="bigorange" id="green">Un troisième paragraphe</p>
        <p style="color:purple;" class="bigorange" id="yellow">Un dernier paragraphe</p>
    </body>
</html>
```

```css
p{
  color: red;
  text-decoration: underline;
}
.bigorange{
  color: orange;
  font-size: 24px
}
#green{
  color: green;
}
#yellow{
  color: yellow;
}
```

Dans l'exemple ci-dessus, par exemple, nous avons une page HTML qui contient un titre de niveau 1 et trois paragraphes. Un de nos paragraphes possède un attribut `class="bigorange"` tandis qu'un autre possède à la fois un attribut `class="bigorange"` et un attribut `id="green"`.

Enfin, un dernier paragraphe possède à la fois un attribut `class="bigorange"`, un attribut `id="yellow"` et un attribut `style` dans lequel nous allons directement préciser un comportement pour la propriété `color` qui ne s'appliquera donc qu'à cet élément.

Du côté du CSS, on cible nos éléments HTML via quatre sélecteurs : un sélecteur éléments `p`, un sélecteur `.bigorange` et deux sélecteurs `#green` et `#yellow`. Certains de nos paragraphes vont donc être ciblés plusieurs fois avec plusieurs sélecteurs différents et recevoir les styles définis dans ces différents sélecteurs.

Ici, on voit que le sélecteur `p` est le seul sélecteur qui définit le comportement de la propriété `text-decoration` tandis que le sélecteur `.bigorange` est le seul qui définit le comportement de la propriété `font-size`. Il n'y aura donc pas de conflit sur ces deux propriétés puisqu'elles ne sont définies qu'une fois pour les mêmes éléments en CSS.

En revanche, on définit un comportement différent pour la propriété `color` au sein de chaque sélecteur. Dans ce cas-là, il va y avoir un conflit puisque le CSS va devoir déterminer quelle valeur de la propriété appliquer pour chaque élément ciblé avec plusieurs sélecteurs.

Pour comprendre comment le CSS va procéder dans ce cas, il faut avant tout bien se persuader que le CSS (comme tout autre langage web) repose sur un ensemble de règles. Les règles définissant l'ordre de préférence d'application des propriétés définies dans différents sélecteurs sont contrôlées par un mécanisme qu'on appelle la cascade. Connaitre ces règles va nous permettre de prédire quel style sera appliqué dans telle ou telle situation.

Le mécanisme de cascade CSS

Il n'est pas toujours simple de prédire quels styles CSS vont s'appliquer à quel élément pour la simple et bonne raison que le CSS peut être défini à des endroits différents (dans un élément `style`, dans la balise ouvrante d'un élément dans un attribut `style` ou dans un fichier CSS séparé) et qu'on va également pouvoir appliquer des styles à un élément en particulier en le ciblant via plusieurs sélecteurs CSS différents.

Il est donc essentiel de bien comprendre comment le CSS va fonctionner pour déterminer quels styles devront être appliqués à tel élément. L'ordre de préférence et d'application d'un style va dépendre de trois grands facteurs qui vont être :

- La présence ou non du mot clef `!important` ;
- La précision du sélecteur ;
- L'ordre de déclaration dans le code.

A noter que ces trois facteurs vont être analysés dans l'ordre donné et que le premier va primer sur le deuxième qui va primer sur le dernier : par exemple, si une règle utilise la syntaxe `!important` elle sera jugée comme prioritaire peu importe la précision du sélecteur ou sa place dans le code.

Le mot clef !important

La mot clef `!important` sert à forcer l'application d'une règle CSS. La règle en question sera alors considérée comme prioritaire sur toutes les autres déclarations et le style sera appliqué à l'élément concerné.

Nous allons placer ce mot clef à la fin d'une déclaration CSS lorsqu'on souhaite qu'un style s'applique absolument à un élément.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1 class="bigorange">Un titre de niveau 1</h1>

        <p>Un premier paragraphe</p>
        <p class="bigorange">Un autre paragraphe</p>
        <p class="bigorange" id="green">Un troisième paragraphe</p>
        <p style="color:purple;" class="bigorange" id="yellow">Un dernier paragraphe</p>
    </body>
</html>
```

```css
p{
  color: red !important;
  text-decoration: underline;
}
.bigorange{
  color: orange;
  font-size: 24px
}
#green{
  color: green;
}
#yellow{
  color: yellow;
}
```

Comme vous pouvez le constater dans l'exemple ci-dessus, le fait d'ajouter `!important` dans la définition du comportement de la propriété `color` liée à notre sélecteur `p` fait que c'est cette définition qui s'appliquera par-dessus toutes les autres.

Ici, en particulier, vous pouvez voir que tous nos paragraphes sont rouges, même lorsque la propriété `color` a été définie différemment dans un sélecteur de `class` ou d'`id` et même lorsqu'un comportement différent a été précisé dans un attribut `style` dans la balise ouvrante d'un élément en particulier.

Le mot clef `!important` est donc extrêmement puissant en CSS et peut ainsi sembler très pratique et très utile aux yeux des débutants. Cependant, en pratique, nous essaierons tant que possible de nous en passer tout simplement car ce mot clef est une sorte de « joker » qui court-circuite toute la logique normale du CSS.

L'utiliser à outrance et lorsque ce n'est pas strictement nécessaire peut donc amener de nombreux problèmes par la suite comme par exemple des problèmes de styles définis autrement et qui ne s'appliqueraient pas car déjà définis avec `!important` ailleurs dans le code.

De manière générale, on préfèrera toujours aller dans le sens des langages et essayer de respecter et d'utiliser les normes qu'ils ont mis en place.

Le degré de précision du sélecteur

Le deuxième critère déterminant dans l'application d'un style plutôt que d'un autre va être le degré de précision du sélecteur où le style a été défini une première fois par rapport aux autres degrés de précision des autres sélecteurs où le style a été à nouveau défini.
Le sélecteur le plus précis imposera ses styles aux sélecteurs moins précis en cas de conflit.

Pour rappel, la « précision » désigne ici le fait d'identifier de manière plus ou moins unique un élément. Les sélecteurs peuvent être rangés dans l'ordre suivant (du plus précis au moins précis) :

- Un style défini dans un attribut HTML `style` sera toujours le plus précis et notamment plus précis qu'un style défini avec un sélecteur CSS ;
- Le sélecteur `#id` va être le sélecteur le plus précis mais sera moins précis qu'un style défini dans un attribut HTML `style` ;
- Un sélecteur `.class` ou un autre sélecteur d'attribut* (*les autres sélecteurs d'attributs sont des sélecteurs complexes que nous étudierons plus tard) ou un sélecteur de pseudo-classe** (**nous verrons ce qu'est une pseudo-classe plus tard dans ce cours) sera moins précis qu'un sélecteur `#id` ;
- Un sélecteur d'élément ou de pseudo-élément*** (***nous étudierons les pseudo éléments plus tard dans ce cours) sera moins précis qu'un sélecteur d'attribut ou de pseudo-classe.

Si deux sélecteurs différents sont au même degré de précision, alors c'est le sélecteur le plus « complet » c'est-à-dire celui qui utilisera le plus de combinateurs qui sera jugé le plus précis.

Prenons immédiatement un exemple pour illustrer cela :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1 class="bigorange">Un titre de niveau 1</h1>

        <p class="petit">Un premier paragraphe</p>
        <p class="bigorange">Un autre paragraphe</p>
        <ul>
            <li>Elément de liste 1</li>
            <li class="bigorange petit">Elément de liste 2</li>
        </ul>
        <p class="bigorange" id="green">Un troisième paragraphe</p>
        <p style="color:purple;" class="bigorange" id="yellow">Un dernier paragraphe</p>
    </body>
</html>
```

```css
p{
  color: red;
  text-decoration: underline;
}
ul .petit{
  font-size: 12px;
}
.bigorange{
  color: orange;
  font-size: 24px
}
#green{
  color: green;
}
#yellow{
  color: yellow;
}
```

Ici, on voit que deux propriétés sont définies dans plusieurs sélecteurs qui servent à sélectionner le même élément : les propriétés `color` et `font-size`. Ce sont donc nos deux propriétés qui vont générer des conflits.

On commence par vérifier la présence du mot clef `!important` : il n'est défini nulle part ici. On passe donc au deuxième critère qui est le degré de précision.

On regarde déjà si des attributs `style` sont présents dans le code. C'est le cas pour notre dernier paragraphe qui possède un attribut `style="color:purple"`. Comme la règle ne peut pas être appliquée plus précisément, on sait que ce paragraphe sera de couleur violette.

Ensuite, on s'intéresse à la présence d'attributs `id`. Notre troisième paragraphe possède un `id="green"` et le sélecteur correspondant définit la règle `color : green`. Ce paragraphe sera donc vert.

Ensuite, on regarde la présence de sélecteur `.class` ou de sélecteurs d'autres attributs ou de sélecteurs de pseudo-classes. Notre deuxième élément de liste possède deux attributs class : `bigorange` et `petit` et on va définir le comportement de la propriété `font-size` dans chacun des deux sélecteurs associés.

Ici, les deux sélecteurs sont des sélecteurs `.class` et possèdent donc le même degré de précision à priori. Il va donc falloir regarder si un sélecteur est plus complet que l'autre c'est-à-dire s'il utilise différents combinateurs pour le rendre plus précis ou pas. C'est le cas de notre sélecteur `ul .petit` qui va finalement nous servir à ne cibler que les éléments possédant un attribut `class= "petit"` contenus dans un élément `ul`.

L'ordre d'écriture des règles

Le troisième et dernier critère qui va nous permettre de définir quel style doit primer sur tel autre et doit donc être appliqué à un élément va tout simplement être l'ordre d'écriture d'une règle dans le code.

Ce critère va être utilisé dans le cas où plusieurs sélecteurs concurrents définissent le comportement d'une même propriété et ont la même importance et la même spécificité. La règle ici est très simple : c'est la dernière déclaration dans le code qui primera sur des déclarations précédentes.

Regardez plutôt l'exemple suivant :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
        <style>
            h1{
                color: orange;
                font-size: 30px;
            }
        </style>
    </head>

    <body>
        <h1 class="petit">Un titre de niveau 1</h1>

        <p class="petit grand">Un premier paragraphe</p>
        <p class="orange bleu">Un autre paragraphe</p>
    </body>
</html>
```

```
h1{
    color: blue;
}

.petit{
  font-size: 12px;
}
.grand{
  font-size: 24px;
}
.bleu{
  color: blue;
}
.orange{
  color: orange;
}
```

Ici, chacun de mes deux paragraphes possèdent deux attributs `class` qui vont à chaque fois définir le comportement d'une même propriété. Les sélecteurs CSS associés ont la même importance et le même degré de spécificité. Il va donc falloir regarder leur ordre d'écriture pour savoir quelles règles vont être appliquées.

Ici, le sélecteur `.grand` apparait après le sélecteur `.petit` dans le code. C'est donc la taille de texte définie dans `.grand` qui va être appliquée à notre premier paragraphe.

De même, le sélecteur `.orange` apparait après le sélecteur `.bleu` dans le code. Le texte de notre deuxième paragraphe sera donc orange et non pas bleu.

Notez que c'est exactement la même règle d'ordre d'écriture des styles qui va s'appliquer, à sélecteur égal, pour déterminer si ce sont les styles définis dans un élément `style` ou si ce sont ceux définis dans un fichier CSS séparés qui vont s'appliquer.

Ici, notre titre `h1` s'affiche en orange car nous avons précisé l'élément `style` après l'élément `link` qui fait appel à notre fichier CSS dans notre fichier HTML. Les styles définis dans l'élément `style` seront donc lus après ceux définis dans notre fichier CSS liés et seront donc appliqués dans le cas où plusieurs sélecteurs concurrents définissent le comportement d'une même propriété et ont la même importance et la même spécificité.

Pour vous en convaincre, échangeons la place des éléments `link` et `style` dans notre code HTML et observons le résultat sur notre code HTML :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <style>
            h1{
                color: orange;
                font-size: 30px;
            }
        </style>
        <link rel="stylesheet" href="cours.css">

    </head>

    <body>
        <h1 class="petit">Un titre de niveau 1</h1>

        <p class="petit grand">Un premier paragraphe</p>
        <p class="orange bleu">Un autre paragraphe</p>
    </body>
</html>
```

```css
h1{
    color: blue;
}

.petit{
  font-size: 12px;
}
.grand{
  font-size: 24px;
}
.bleu{
  color: blue;
}
.orange{
  color: orange;
}
```

Une convention en HTML va être de toujours préciser notre élément style après notre élément link dans le code pour ne pas s'embrouiller et c'est la raison pour laquelle on

retient généralement qu'à sélecteur égal les styles définis dans l'élément `style` sont prioritaires sur ceux définis dans un fichier CSS séparé.

Notez qu'ici notre titre `h1` va toujours avoir la taille définie dans le sélecteur `.petit` puisqu'un sélecteur d'attribut `class` est toujours plus précis qu'un sélecteur élément et que ce critère de précision passe avant le critère de l'ordre d'écriture des styles dans le code.

L'héritage en CSS

La notion d'héritage est une autre notion fondamentale du CSS. Elle signifie que certains styles CSS appliqués à un élément vont être hérités par les éléments enfants de cet élément, c'est-à-dire par les éléments contenus dans cet élément.

Cette notion d'héritage est conditionnée par deux choses :

- Toutes les propriétés ne vont pas être héritées pour la simple et bonne raison que cela ne ferait aucun sens pour certaines de l'être ;
- Les éléments enfants n'hériteront des styles de leur parent que si il n'y a pas de conflit c'est-à-dire uniquement dans la situation où ces mêmes styles n'ont pas été redéfinis pour ces éléments enfants en CSS.

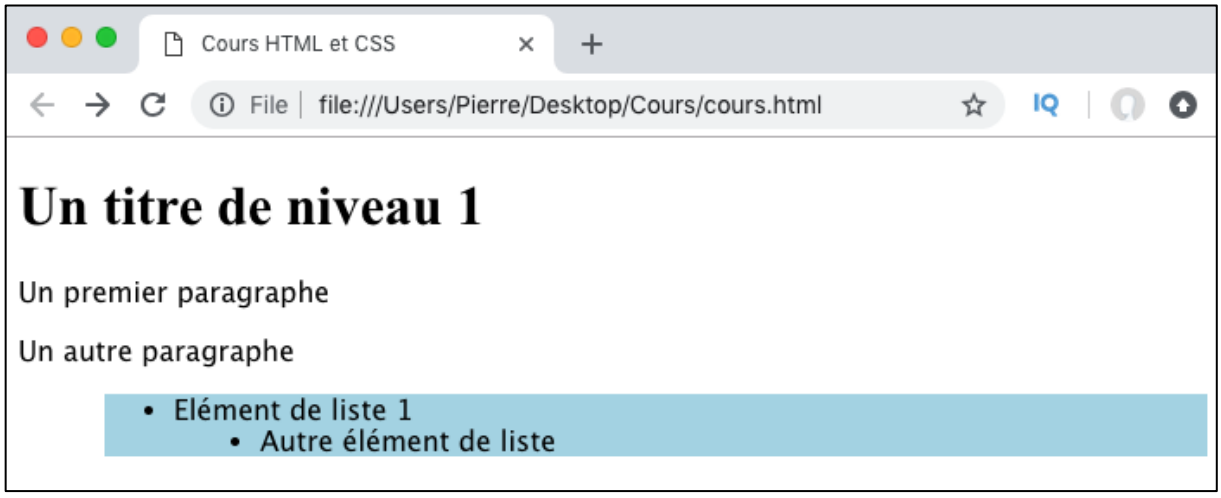
Pour savoir quelles propriétés vont pouvoir être héritées et quelles autres ne vont pas pouvoir l'être il va soit falloir faire preuve de logique (et bien connaitre le langage CSS), soit falloir apprendre par cœur pour chaque propriété si elle peut être héritée ou pas.

Les propriétés qui vont pouvoir être héritées sont en effet celles dont l'héritage fait du sens. Par exemple, la propriété `font-family` qui sert à définir un jeu de polices à utiliser pour du texte va pouvoir être hérité car il semble logique que l'on souhaite avoir la même police pour tous les textes de nos différents éléments par défaut.

En revanche, les propriétés liées aux marges par exemple ou plus généralement les propriétés de mise en page et de positionnement des éléments ne vont pas pouvoir être héritées car cela ne ferait aucun sens d'un point de vue design de rajouter une marge d'une taille définie pour chaque élément enfant.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p>Un premier paragraphe</p>
        <p>Un autre paragraphe</p>

        <ul>
            <li>Elément de liste 1</li>
            <li id="marge">Autre élément de liste</li>
        </ul>
    </body>
</html>
```

```css
html{
  font-family: "Lucida Grande", sans-serif;
}

h1{
  font-family: "Times New Roman", serif
}

ul{
  margin-left: 50px;
  background-color: lightBlue
}

#marge{
  margin-left: 50px;
}
```



Dans l'exemple ci-dessus, je définis un jeu de police avec la propriété `font-family` dans mon sélecteur `html`. Comme tous les éléments d'une page HTML sont des enfants de cet élément (ils sont contenus dans l'élément `html`) et comme la propriété `font-family` peut être héritée, tous les textes de ma page utiliseront la police d'écriture définie dans cette propriété sauf si une autre police est définie de manière plus précise avec un sélecteur plus précis comme c'est le cas pour mon titre `h1` ici.

J'attribue ensuite une marge extérieure gauche égale à 50px à mon élément `ul` représentant ma liste. Ma liste sera donc décalée de 50px par rapport au bord gauche de son élément parent qui est ici l'élément `body` qui représente la page. Cependant, comme la propriété `margin` ne peut pas être héritée, les éléments de liste de vont pas hériter de ce même `margin-left : 50px` par défaut.

Ici, vous devez bien comprendre que la marge se calcule par rapport au début de l'élément parent. La liste entière est décalée de 50px par rapport à l'élément `body` mais les éléments de liste ne sont pas décalés par défaut de 50px par rapport à l'élément `ul` qui est leur élément parent. Pour bien illustrer cela, j'ai ajouté manuellement un `margin-left : 50px` au deuxième élément de liste afin de vous prouver que le premier élément de liste n'a pas hérité de la propriété `margin` appliquée à son élément parent `ul`.

Notez que le CSS nous laisse toutefois la possibilité de « forcer » un héritage pour des propriétés non héritées par défaut ou plus exactement la possibilité de définir des comportements d'héritage pour chaque propriété définie dans chaque sélecteur.

Pour faire cela, nous allons pouvoir utiliser quatre valeurs qui vont fonctionner avec toutes les propriétés CSS (elles sont dites universelles) et qui vont nous permettre d'indiquer que telle propriété définie dans tel sélecteur doit avoir le même comportement que celle définie pour l'élément parent ou pas.

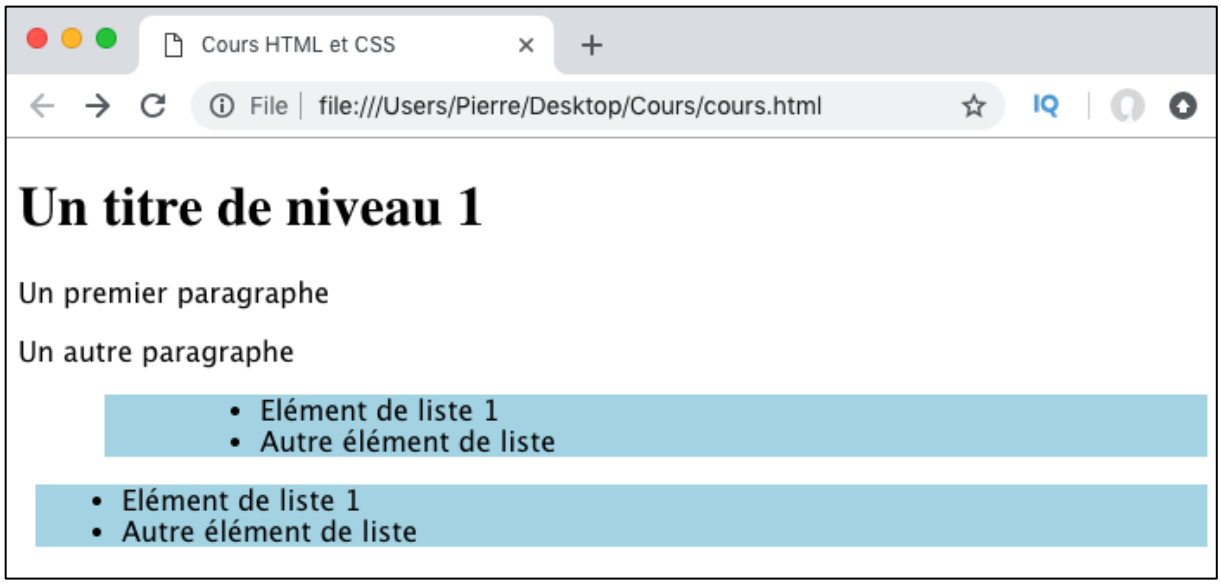
Ces valeurs sont les suivantes :

Valeur	Signification
`inherit`	Sert à indiquer que la valeur de propriété appliquée à l'élément sélectionné est la même que celle de l'élément parent
`initial`	Sert à indiquer que la valeur de propriété appliquée à l'élément sélectionné est la même que celle définie pour cet élément dans la feuille de style par défaut du navigateur
`unset`	Permet de réinitialiser la propriété à sa valeur naturelle, ce qui signifie que si la propriété est naturellement héritée elle se comporte comme si on avait donné la valeur `inherit`. Dans le cas contraire, son comportement sera le même que si on lui avait donné la valeur `initial`
`revert`	Permet la propriété à la valeur qu'elle aurait eue si aucun style ne lui avait été appliqué. La valeur de la propriété va donc être fixée à celle de la feuille de style de l'utilisateur si elle est définie ou sera récupérée dans la feuille de style par défaut de l'agent utilisateur

En pratique, la valeur la plus utilisée parmi ces quatre va être `inherit`. Notez également que le support pour la valeur `revert` n'est pas encore acquis pour la plupart des navigateurs. Je n'ai évoqué cette valeur ici que par souci d'exhaustivité mais vous déconseille de l'utiliser pour le moment. Pour cette raison, je ne l'évoquerai plus dans la suite de ce cours.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p>Un premier paragraphe</p>
        <p>Un autre paragraphe</p>

        <ul id="liste1">
            <li>Elément de liste 1</li>
            <li>Autre élément de liste</li>
        </ul>

        <ul id="liste2">
            <li>Elément de liste 1</li>
            <li>Autre élément de liste</li>
        </ul>
    </body>
</html>
```

```css
html{
  font-family: "Lucida Grande", sans-serif;
}

/*Pour info, la police utilisée par défaut pour les h1 ici est Times*/
h1{
  font-family: initial;
}

ul{
  background-color: lightBlue
}
#liste1{
    margin-left: 50px;
}
#liste2{
    margin-left: 10px;
}

/*On demande explicitement à ce que les éléments li héritent du même
 comportement pour la propriété margin-left que leur parent (qui est ul)*/
li{
  margin-left: inherit;
}
```



Ici, on définit un `h1{font-family: initial;}` en CSS. Ainsi, c'est la valeur de `font-family` définie pour cet élément dans la feuille de style par défaut du navigateur qui va être appliquée. En l'occurrence, dans mon cas, cela va être la valeur `Times`.

Ensuite, on demande explicitement à ce que les éléments de liste `li` héritent de la valeur donnée à la propriété `margin-left` à leur parent. Pour notre première liste, on définit `margin-left: 50px`. Les éléments `li` vont donc également posséder une marge extérieure gauche de 50px par rapport à la liste en soi qui est leur élément parent.

Pour notre deuxième liste, en revanche, on a défini une marge gauche de 10px seulement. Les éléments de liste vont donc utiliser cette même valeur pour leur propriété `margin-left` et être décalés de 10px par rapport à la liste en soi.

Conclusion sur les mécanismes de cascade et d'héritage en CSS

Les mécanismes de cascade et d'héritage en CSS vont permettre de définir via un ensemble de règles quels styles vont être appliqués à quel élément en cas de conflit.

Ces mécanismes vont en pratique très souvent entrer en jeu. En effet, la plupart des sites sont aujourd'hui complexes et vont utiliser plusieurs feuilles de styles (fichiers CSS) différentes qui vont définir de nombreuses règles à appliquer à chaque élément.

Comprendre comment ces mécanismes fonctionnent et connaitre ces règles est essentiel et fondamental puisque cela va nous permettre de toujours obtenir le résultat visuel espéré.

Notez que la cascade et l'héritage sont le cœur même du CSS et sont en grande partie sa puissance puisque ces mécanismes vont nous permettre d'un côté de pouvoir « surcharger » des styles en utilisant des sélecteurs plus ou moins précis et de l'autre côté de transmettre des styles d'un élément parent à ces enfants et donc nous éviter de définir tous les styles voulus pour chaque élément.

Les éléments HTML div et span

Dans cette nouvelle leçon, nous allons nous intéresser à deux éléments HTML très spéciaux qui sont les éléments `div` et `span`.

Ces éléments sont très particuliers puisqu'ils ne servent pas à préciser la nature d'un contenu mais vont simplement nous servir de conteneurs génériques en HTML.

Nous allons ici comprendre l'intérêt de ces deux éléments et en particulier leur intérêt pour l'application de styles CSS et les cas d'utilisation de ces éléments.

Le HTML et la valeur sémantique des éléments

Il est toujours bon de commencer par rappeler le rôle du HTML : le HTML a pour but de structurer du contenu et de lui donner du sens.

Les éléments HTML vont nous servir à marquer les différents contenus et donc à indiquer aux navigateurs et moteurs de recherche de quoi est composé une page. On va ainsi pouvoir dire grâce au HTML que tel contenu doit être considéré et traité comme un paragraphe, que tel autre contenu est un titre, que tel texte est plus important qu'un autre, que ceci est une liste, que cet objet est une image, etc.

A ce titre, les éléments HTML `div` et `span` sont très spéciaux puisque ce sont deux éléments HTML qui ne possèdent aucune valeur sémantique, c'est-à-dire qu'ils ne servent pas à préciser la nature d'un contenu.

Ces deux éléments sont en effet des conteneurs génériques qui ont été créés pour nous permettre d'ordonner nos pages plus simplement ensuite en CSS.

Quels usages pour les éléments div et span ?

Le fait que les éléments `div` et `span` ne possèdent aucune valeur sémantique peut faire penser qu'ils vont à l'encontre même du rôle du HTML. C'est tout à fait vrai en soi, et c'est la raison pour laquelle on essaiera idéalement de n'utiliser ces éléments qu'en dernier recours et si nous n'avons aucun autre choix crédible.

Les éléments HTML `div` et `span` ont été créés principalement pour simplifier la mise en page de nos pages HTML en CSS c'est-à-dire pour simplifier l'application de certains styles CSS.

Exemple d'utilisation de l'élément div

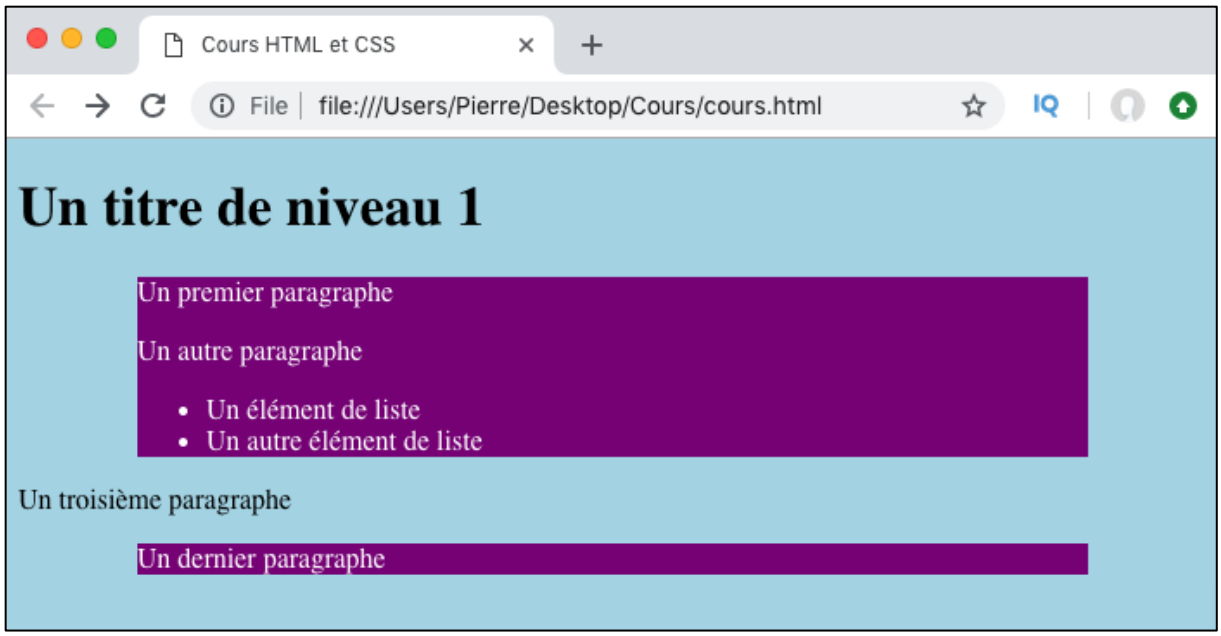
Nous allons utiliser l'élément `div` comme conteneur pour différents éléments afin de pouvoir ensuite facilement appliquer les mêmes styles CSS à tous les éléments contenus

dans notre `div` par héritage ou pour les mettre en forme en appliquant un style spécifique au `div`.

Ici, le terme de « conteneur » est l'équivalent du terme « parent » : nous allons simplement placer nos différents éléments à l'intérieur de nos balises `<div>` et `</div>` puis appliquer les styles CSS directement au `div`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <div>
            <p>Un premier paragraphe</p>
            <p>Un autre paragraphe</p>

            <ul>
                <li>Un élément de liste</li>
                <li>Un autre élément de liste</li>
            </ul>
        </div>

        <p>Un troisième paragraphe</p>

        <div>
            <p>Un dernier paragraphe</p>
        </div>
    </body>
</html>
```

```css
/*Couleur de fond du body (donc de la partie visible de la page) : bleu clair*/
body{
    background-color: lightBlue;
}

div{
  color: white; /*Textes des éléments dans les div blancs par héritage*/
  background-color: purple; /*Les div auront un fond violet*/
  width: 80%; /*Définit la largeur des div à 80% de leur parent (body ici)*/
  margin: 0 auto; /*Permet de centrer les div dans leur élément parent (body ici)*/
}
```



On peut ici penser qu'on peut arriver au même résultat en utilisant plusieurs attributs `class` possédant la même valeur pour les différents éléments. Ce n'est pas tout à fait vrai.

Tout d'abord, ce n'est pas normalement exactement le rôle de base des attributs `class` : les sélecteurs `.class` sont censés être liés à un style CSS particulier et chaque élément doit pouvoir utiliser la `class` adaptée pour appliquer ce style. Ici, nous utilisons plutôt le sélecteur `.class` de manière exclusive en définissant de nombreux styles pour un groupe d'éléments en particulier. La logique de code est donc inversée.

En dehors de cette considération sur le rôle des `class`, il est beaucoup plus simple et rapide dans le cas présent d'utiliser un `div` que de renseigner un attribut `class` à chaque fois.

De plus, dans certaines situations, nous allons vouloir pour des raisons de mise en page appliquer des styles spécifiquement au conteneur et nos pas à chaque élément contenu comme par exemple des marges externes.

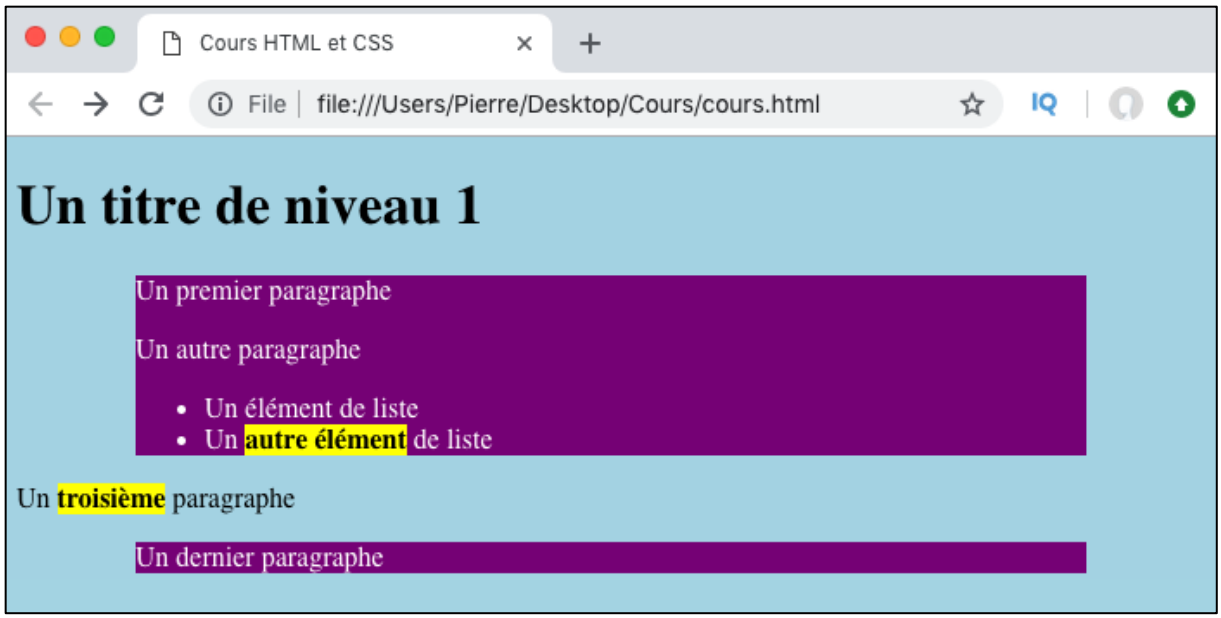
Exemple d'utilisation de l'élément span

L'élément `span` va lui servir de conteneur à un autre niveau : il va servir de conteneur interne à un élément plutôt que de conteneur pour plusieurs éléments.

On va par exemple pouvoir placer une certaine partie du texte d'un titre ou d'un paragraphe dans un élément `span` pour ensuite pouvoir lui appliquer un style CSS particulier, chose qu'il nous était impossible de faire jusqu'à présent.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <div>
            <p>Un premier paragraphe</p>
            <p>Un autre paragraphe</p>

            <ul>
                <li>Un élément de liste</li>
                <li>Un <span>autre élément</span> de liste</li>
            </ul>
        </div>

        <p>Un <span>troisième</span> paragraphe</p>

        <div>
            <p>Un dernier paragraphe</p>
        </div>
    </body>
</html>
```

```css
/*Couleur de fond du body (donc de la partie visible de la page) : bleu clair*/
body{
    background-color: lightBlue;
}

div{
  color: white; /*Textes des éléments dans les div blancs par héritage*/
  background-color: purple; /*Les div auront un fond violet*/
  width: 80%; /*Définit la largeur des div à 80% de leur parent (body ici)*/
  margin: 0 auto; /*Permet de centrer les div dans leur élément parent (body ici)*/
}

span{
    font-weight: bold; /*Les textes seront en gras*/
    background-color: yellow; /*Fond des span jaune*/
    color: black; /*Couleur du texte noire*/
}
```



Les éléments div et span et les attribut class et id

Les attributs class et id sont des attributs universels ce qui signifie qu'on va pouvoir les utiliser avec n'importe quel élément HTML, et notamment avec les éléments div et span.

En pratique, il va être très courant de préciser des attributs class et id pour nos éléments div et span pour pouvoir appliquer des styles à un div (ou span) ou à un groupe d'éléments div ou span) définis.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <div class="conteneur" id="cp">
            <p>Un premier paragraphe</p>
            <p>Un autre paragraphe</p>

            <ul>
                <li>Un élément de liste</li>
                <li>Un <span class="gras">autre élément</span> de liste</li>
            </ul>
        </div>

        <p>Un <span class="gras fondjaune">troisième</span> paragraphe</p>

        <div>
            <p class="gras">Un dernier paragraphe</p>
        </div>
    </body>
</html>
```

```css
/*Couleur de fond du body (donc de la partie visible de la page) : bleu clair*/
body{
    background-color: lightBlue;
}

div{
    background-color: purple; /*Les div auront un fond violet*/
    color: white; /*Textes des éléments dans les div blancs par héritage*/
}
.conteneur{
  width: 80%; /*La largeur de l'élément sera égale à 80% de son parent*/

}
#cp{
  margin: 0 auto; /*Permet de centrer le div dans son élément parent (body ici)*/
}

.gras{
    font-weight: bold; /*Les textes seront en gras*/
}
.fondjaune{
    background-color: yellow; /*Fond des span jaune*/
}
.noir{
    color: black; /*Couleur du texte noire*/
}
```

Avec ce qu'on a appris dans la dernière leçon, vous devriez être capable de comprendre les styles appliqués ici en vous concentrant. Je vous laisse donc essayer, ça vous fera un bon exercice !

Quelles différences entre les éléments div et span et quand utiliser l'un plutôt que l'autre ?

La grande différence entre les éléments `div` et `span` va concerner ce qu'ils vont pouvoir contenir : un élément `div` va pouvoir conteneur plusieurs éléments et va donc nous servir de conteneurs d'éléments tandis que l'élément `span` va nous servir de conteneur pour une partie du contenu d'un élément et va donc être utilisé à l'intérieur d'un élément.

Cette différence est due au fait que les éléments `div` et `span` sont de niveau ou au « type » différents : l'élément `div` est un élément de niveau `block` tandis que l'élément `span` est un élément de niveau `inline`.

Nous découvrirons plus tard dans ce cours ce que ces différents « niveaux » ou « types » d'éléments signifient et les différences entre eux.

Les niveaux ou types d'éléments block et inline

Nous avons pour le moment défini et étudié les grands mécanismes de fonctionnement du CSS tout en présentant certaines propriétés CSS impactant l'aspect visuel des éléments.

Pour aller plus loin dans notre étude du CSS, nous allons devoir maintenant comprendre comment est définie la place prise par chaque élément dans une page.

Les niveaux ou « types » d'éléments HTML

De manière schématique, on peut considérer qu'Il existe deux grands types d'affichage principaux pour les éléments HTML : un élément HTML va pouvoir être soit de niveau (ou de type) `block`, soit de niveau (ou de type) `inline`.

Ces types d'affichage vont définir la façon dont les éléments vont se comporter dans une page par rapport aux autres et la place qu'ils vont prendre dans la page.

Connaître le type d'affichage d'un élément HTML va donc être essentiel pour créer et mettre en forme nos pages web car les éléments de type `block` et ceux de type `inline` vont se comporter de façon radicalement différente dans une page et certaines propriétés CSS vont avoir des comportements différents selon le type d'affichage d'un élément.

Comprendre comment est défini le type d'affichage d'un élément HTML

Le type d'affichage d'un élément va toujours être défini en CSS par la propriété `display`. Si cette propriété n'est pas explicitement renseignée pour un élément, c'est la valeur par défaut de `display` qui va être appliquée à l'élément c'est-à-dire `display: inline`.

Ainsi, par défaut, on peut dire que tout élément HTML va posséder un type d'affichage `inline` (nous allons voir par la suite ce que signifie ce type d'affichage).

Cependant, rappelez-vous que chaque navigateur possède une feuille de styles (c'est-à-dire un fichier CSS) qui sera appliquée par défaut pour les différents éléments dont nous ne précisons pas le comportement dans nos propres feuilles de style.

La plupart des navigateurs possèdent aujourd'hui des feuilles de style similaires notamment pour la définition des styles basiques et c'est une très bonne chose pour nous développeur puisque cela va nous éviter d'avoir à définir le comportement de chaque propriété pour chaque élément de notre page.

Parmi ces styles par défaut appliqués par n'importe quel navigateur se trouve la définition du type d'affichage ou du `display` pour chaque élément.

Aujourd'hui, la plupart des navigateurs suivent les recommandations du W3C (l'organisme en charge de l'évolution et des standards des langages Web). Ce W3C spécifie pour chaque élément HTML quelle devrait être la valeur de son `display`.

Attention cependant : encore une fois, ce ne sont que des recommandations et chaque navigateur est libre de ne pas en tenir compte et de définir une autre valeur de `display` pour chaque élément !

C'est la raison pour laquelle il reste important de définir nous-mêmes la plupart des styles CSS impactant et qui pourraient être définis différemment par défaut par différents navigateurs.

Le W3C va donc indiquer quel devrait être le type d'affichage d'un élément par défaut et l'immense majorité des navigateurs va appliquer ces recommandations ce qui fait que l'un des `display` suivants à la plupart des éléments HTML en fonction de l'élément :

- `display : block` : affichage sous forme d'un bloc ;
- `display : inline` : affichage en ligne ;
- `display : none` : l'élément n'est pas affiché.

Notez que le W3C préconise d'autres types d'affichage pour certains éléments HTML particuliers. Les deux autres valeurs de `display` généralement respectées et appliquées par les navigateurs sont :

- `display : list-item` va être appliquée par défaut pour les éléments de liste `li`. L'affichage se fait sous forme de bloc mais une boite avec un marqueur est également générée ;
- `display : table` va être appliquée par défaut pour les éléments de tableau `table`. L'affichage se fait sous forme de bloc.

Certains navigateurs dans certains cas très particuliers peuvent également utiliser la valeur `inline-block`pour afficher certains éléments.

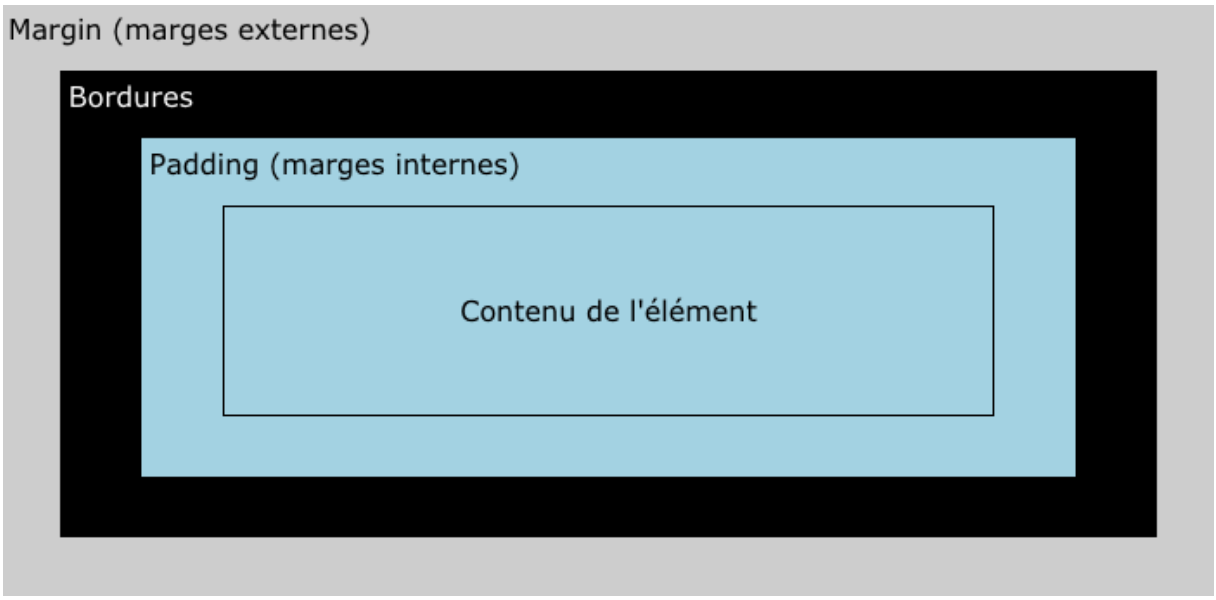
Ici, vous devez bien comprendre que ces valeurs ne sont que les valeurs préconisées par le W3C. Rien ne nous empêche de définir un type d'affichage différent de celui préconisé pour un élément en utilisant la propriété `display` avec la valeur souhaitée. Cela va pouvoir être utile pour aider à la mise en page de certains éléments.

Bon à savoir : Jusqu'à récemment (jusqu'au HTML 4.1), le W3C utilisait la simple distinction « bloc-level elements » vs « inline-level elements » (éléments de type `block` vs éléments de type `inline`) pour catégoriser les éléments HTML. Cependant, l'évolution des langages HTML et CSS et de leur complexité a amené le W3C à repenser la façon dont les éléments devaient être catégorisés. Aujourd'hui, les éléments sont classés selon des catégories ou des modèles de contenus (« content categories » ou « content models »).

Rapide introduction au modèle des boites

Le modèle des boites (que nous étudierons plus tard dans ce cours) nous dit que tout élément HTML peut être représenté sous forme d'une boite rectangulaire. C'est une représentation qu'il vous faut connaitre et qu'il vous faudra comprendre.

Cette boite rectangulaire représentant l'élément contient d'autres boites (une qui contient le contenu, une autre qui contient en plus les marges intérieures, etc.) et ces différentes boites vont se comporter de manière différente selon le type d'affichage qui va lui être attribué, c'est-à-dire selon la valeur donnée à la propriété `display` pour cet élément.



Cependant, retenez bien ici que quelle que soit la valeur donnée à la propriété `display`, l'élément va toujours pouvoir être représenté sous forme d'une boite. Cela peut vous sembler flou pour le moment, mais intégrer cela va beaucoup vous aider pour bien comprendre comment créer des mises en page efficaces en CSS.

Les éléments de type inline

Par simplicité, on appellera « élément de type inline » (ou « inline level element » en anglais) un élément auquel a été appliqué un `display: inline`.

Les éléments de type `inline` vont posséder les caractéristiques suivantes qui vont les différencier des éléments de type `block` :

- Un élément de type `inline` ne va occuper que la largeur nécessaire à l'affichage de son contenu par défaut ;
- Les éléments de type `inline` vont venir essayer de se placer en ligne, c'est-à-dire à côté (sur la même ligne) que l'élément qui les précède dans le code HTML ;
- Un élément de type `inline` peut contenir d'autres éléments de type `inline` mais ne devrait pas contenir d'éléments de type `block`.

De plus, notez qu'on ne va pas par défaut pouvoir appliquer de propriété `width` ou `height` à un élément de type `inline` puisque la caractéristique principale de ce type d'éléments est de n'occuper que la place nécessaire à l'affichage de leur contenu.

Les éléments HTML dont le type d'affichage recommandé par le W3C est le type `inline` les plus courants sont les suivants :

- Les éléments de distinction d'importance du contenu `em` et `strong` ;

- L'élément `span` ;
- L'élément de liens `a` ;
- L'élément `button` ;
- Les éléments de formulaire `input`, `label`, `textarea` et de liste de choix `select` ;
- L'élément d'insertion d'images `img` (cas intéressant et souvent source de confusions car on va pouvoir passer une largeur et une hauteur à l'image à afficher en soi qui va « remplacer » l'élément `img` lors de l'affichage, mais il n'empêche que l'élément `img` est bien `inline` en soi);
- Les éléments `code`, `script`, etc.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>
        <p>L'élément p est un élément de type block</p>
        <p>L'élément <strong>strong</strong> est de type inline,
        tout comme <span>span</span></p>
    </body>
</html>
```

```css
strong, span{
    background-color: orange;
}
```

Dans l'exemple ci-dessus, j'ai rajouté une couleur de fond aux éléments `inline` afin que vous puissiez bien voir l'espace qu'ils prennent dans la page.

Les éléments de type block

Par simplicité, on appellera « élément de type block » (ou « block level element » en anglais) un élément auquel on va appliquer un `display: block`.

Les éléments de type `block` vont posséder les caractéristiques de disposition suivantes :

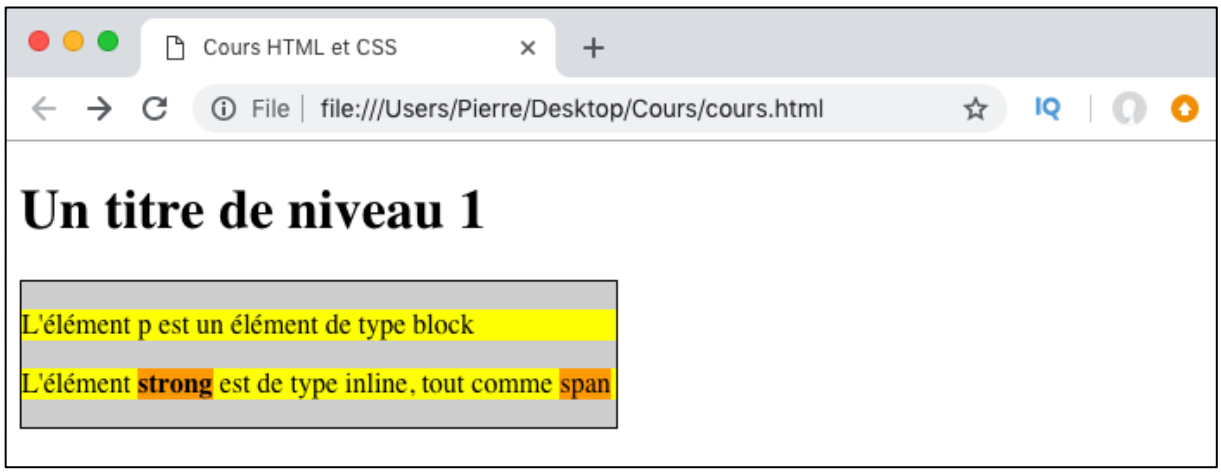
- Un élément de type `block` va toujours prendre toute la largeur disponible au sein de son élément parent (ou élément conteneur) ;
- Un élément de type `block` va toujours « aller à la ligne » (créer un saut de ligne avant et après l'élément), c'est-à-dire occuper une nouvelle ligne dans une page et ne jamais se positionner à côté d'un autre élément par défaut ;
- Un élément de type `block` peut contenir d'autres éléments de type `block` ou de type `inline`.

Les éléments HTML dont le type d'affichage recommandé par le W3C est le type `block` les plus communs sont les suivants :

- L'élément `body`, cas particulier mais qui est concrètement considéré comme un élément `block` ;
- L'élément de division du contenu `div` ;
- Les paragraphes `p` et titres `h1`, `h2`, `h3`, `h4`, `h5`, `h6` ;
- Les éléments structurants `article`, `aside`, `header`, `footer`, `nav` et `section` ;
- Les listes `ul`, `ol`, `dl` et éléments de listes de définition `dt` et `dd` ;
- L'élément de définition de tableaux `table` ;
- L'élément de définition de formulaires `form` et l'élément `fieldset` ;
- Les éléments `figure` et `figcaption` ;
- Les éléments `canvas`, `video`, etc.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>
        <div class="conteneur">
            <p>L'élément p est un élément de type block</p>
            <p>L'élément <strong>strong</strong> est de type inline,
            tout comme <span>span</span></p>
        </div>
    </body>
</html>
```

```css
.conteneur{
  border: 1px solid black;
  background-color: lightGrey;
  width: 50%
}

p{
  width: 100%;
  background-color: yellow;
}

strong, span{
    background-color: orange;
}
```



Bon à savoir : Les éléments HTML comme `video` et `img` décrits ci-dessus comme étant respectivement de types `block` et `inline` sont des éléments HTML très particuliers : ils sont dans la catégorie des éléments « void » (j'utilise ici le mot anglais car utiliser « vide » porterait à confusion) et ce sont également ce qu'on appelle des éléments HTML remplacés. Un élément void est un élément qui ne peut pas posséder de contenu qui lui soit propre et un élément remplacé est un élément qui fait appel à du contenu extérieur (qui va être « remplacé » par un contenu extérieur) possédant déjà des dimensions propres. Notez que la plupart des éléments remplacés sont également des éléments void.

Les autres valeurs de la propriété display

Nous allons passer de nombreuses autres valeurs à la propriété `display` en plus des valeurs `inline`, `block` et `none` comme par exemple `inline-block`, `table`, `list-item`, `flex`, etc.

Nous aurons l'occasion de reparler des différentes valeurs de la propriété `display` dans la leçon qui lui est dédiée plus tard dans ce cours.

Pour le moment, retenez simplement ici que toutes ces « sous » valeurs d'affichage vont toujours reposer sur un affichage pour l'élément en soi `block` ou `inline` auquel vont pouvoir s'ajouter différentes règles, contraintes ou variations.

Je vous demande donc pour l'instant de considérer que tous les éléments sont soit de type block, soit de type inline et de bien retenir les différences entre ces deux types d'affichage.

Notations complètes (longhand) et raccourcies (shorthand) en CSS

Dans cette nouvelle leçon, nous allons définir ce que sont les notations CSS raccourcies ou « short hand » en anglais et voir à quel moment il est intéressant de les utiliser par rapport aux notations complètes ou « long hand ».

Définition d'une notation CSS raccourcie ou notation « short hand »

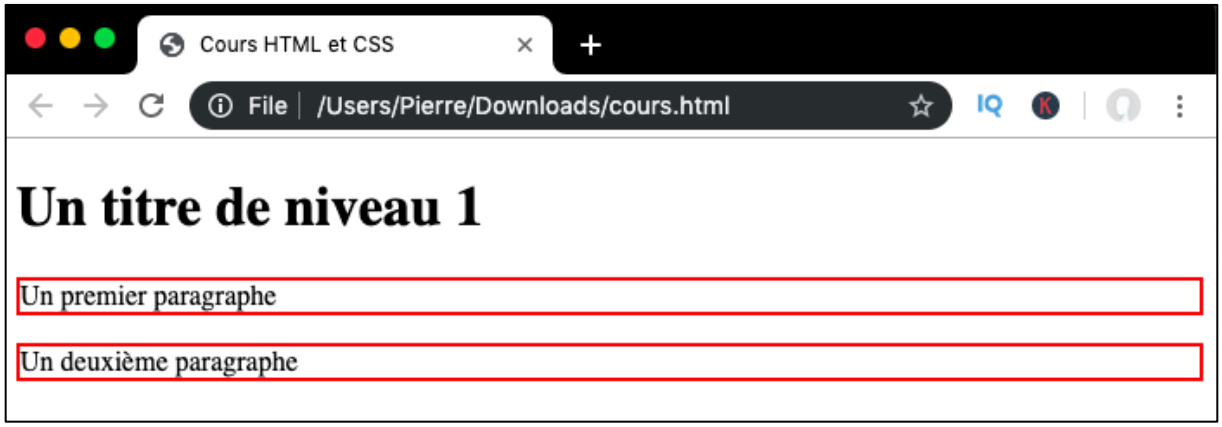
Une notation raccourcie ou notation « short hand » ou encore « propriété raccourcie / short hand » correspond à une propriété à laquelle on va pouvoir passer les valeurs d'un ensemble d'autres propriétés et qui va donc nous permettre de définir de valeur de plusieurs propriétés d'un coup.

Nous avons déjà été amené à en rencontrer certaines dans ce cours comme par exemple la propriété `border` qui correspond en fait à la notation raccourcie des propriétés `border-width`, `border-style` et `border-color` et qui nous permet ainsi de définir d'un coup les valeurs pour ces trois propriétés pour un élément.

De manière générale, il va souvent être équivalent de préciser le comportement d'un aspect d'un élément HTML en utilisant une notation raccourcie ou en utilisant les propriétés long hand.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Un titre de niveau 1</h1>
        <p id="p1">Un premier paragraphe</p>
        <p id="p2">Un deuxième paragraphe</p>
    </body>
</html>
```

```css
/*Notation complète ou "long hand"*/
#p1{
    border-width: 2px;
    border-style: solid;
    border-color: red;
}

/*Notation short hand équivalente*/
#p2{
    border: 2px solid red;
}
```



Cependant, les notations short hand possèdent certains avantages sur les notations long hand mais également certaines limites dans certaines situations que nous allons voir dans la suite de cette leçon.

L'ordre de déclaration des valeurs des propriétés short hand

Une propriété short-hand est une propriété à laquelle on va pouvoir passer les valeurs de plusieurs autres propriétés.

L'ordre de déclaration des valeurs ne va compter que dans le cas où la notation raccourcie va accepter plusieurs valeurs d'un type similaire et donc dans le cas où il peut y avoir ambiguïté sur ce à quoi doit s'appliquer une valeur.

Dans ce cas-là, il faudra respecter un ordre précis. J'indiquerai l'ordre de déclaration pour chaque notation raccourcie que nous allons étudier dans ce cours dans la leçon qui lui est relative.

Par exemple, la propriété raccourcie `padding` va permettre de définir le comportement des marges internes d'un élément HTML. Cette propriété est la version short hand des propriétés suivantes :

- `padding-top` : définition de la marge interne supérieure ;
- `padding-right` : définition de la marge interne droite ;
- `padding-bottom` : définition de la marge interne inférieure ;

- `padding-left` : définition de la marge interne gauche.

Chacune de ces quatre propriétés va accepter le même type de valeur et notamment des valeurs de type « longueur » en `px` par exemple. Lors de la définition de `padding`, il faudra donc faire attention à l'ordre des valeurs pour définir la bonne taille pour chacune des marges internes.

Que se passe-t-il en cas d'oubli de déclaration de certaines valeurs dans les notations raccourcies ?

Avant tout, vous devez bien comprendre que nous ne sommes jamais obligés de définir le comportement de chacune des propriétés long hand dans la propriété short hand associée.

En d'autres termes, on va tout à fait pouvoir omettre de déclarer certaines valeurs dans nos propriétés raccourcies.

Ici, il va y avoir principalement deux cas à distinguer : le cas où il peut y avoir ambiguïté sur les valeurs et le cas où il n'y en a pas.

Dans le cas où il peut y avoir ambiguïté, la définition de la propriété raccourcie va nous indiquer son comportement. Prenons l'exemple de notre propriété `padding` par exemple. Si on omet une valeur, la propriété ne peut pas savoir à première vue si c'est la valeur de la marge haute, droite, basse ou gauche que l'on ne souhaite pas définir.

Pour gérer ce genre de situations, des règles ont donc été définies lors de la création de la version raccourcie. Dans le cas de `padding`, par exemple, si on ne passe que trois valeurs alors la deuxième va s'appliquer à la fois à la marge droite et à la marge gauche.

Dans le cas où il n'y a pas d'ambiguïté, ce sont les valeurs par défaut des propriétés relatives qui vont être utilisées. Cela signifie que les valeurs qui ne sont pas définies avec la propriété raccourcie sont définies avec leur valeur initiale.

Faites bien attention donc ici car omettre des valeurs dans une propriété raccourcie va tout de même définir le comportement des propriétés relatives avec leur valeur initiale.

En particulier, dans le cas où la propriété en question avait déjà été définie auparavant avec sa notation long hand, la valeur sera surchargée par la propriété raccourcie ce qui signifie que c'est la valeur par défaut transmise par la version raccourcie qui sera utilisée. Faites donc bien attention aux comportements inattendus !

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Un titre de niveau 1</h1>
        <p>Un premier paragraphe</p>
        <p id="p1">Un deuxième paragraphe</p>
    </body>
</html>
```

```css
p{
    font-style: italic;
    font-variant: small-caps;
    font-weight: bold;
    font-size: 1em;
    line-height: 1.5em;
    font-family: verdana;
}

#p1{
    font: 1em verdana;
}
```

Dans l'exemple ci-dessus, par exemple, on définit des comportements pour chacune des propriétés qui peuvent être définies avec la notation raccourcie `font` pour nos éléments `p`.

Ensuite, on utilise la notation raccourcie `font` en ciblant un paragraphe en particulier. Dans `font`, on ne définit que les valeurs relatives à la taille de la police et à la famille de polices utilisée.

Pour toutes les autres valeurs, ce sont les valeurs par défaut qui vont être utilisées. Ainsi, par exemple, le style (`font-style`) et le poids (`font-weight`) vont être `normal`, ce qui est la valeur par défaut de ces deux propriétés et non pas `italic` et `bold` comme on l'a défini au-dessus.

Les limites des propriétés short hand par rapport aux notations long hand

La première limite des propriétés short hand par rapport à leurs équivalentes long hand est qu'on ne va pas pouvoir utiliser les valeurs globales `inherit`, `initial` et `unset` dans la déclaration des valeurs de nos propriétés raccourcies.

En effet, si on les utilisait, il serait impossible pour les navigateurs de savoir à quelle propriété correspond quelle valeur dans le cas de l'oubli de certaines valeurs.

Une deuxième limite évidente est que l'héritage des propriétés de va pas être possible avec les propriétés raccourcies puisque les valeurs oubliées dans les propriétés raccourcies vont être remplacées par leurs valeurs initiales. Il ne va donc pas être possible de pouvoir faire hériter les valeurs de certaines propriétés en les omettant dans la déclaration short hand puisque le CSS va tout de même automatiquement les définir en utilisant les valeurs initiales des propriétés « non définies ».

Quelques notations short hand courantes et les propriétés long hand associées

Avant tout, notez que chacune des propriétés long hand ne va pas forcément avoir de propriété short hand associée.

Les propriétés short hand ont été créées pour simplifier et raccourcir l'écriture du CSS tout en gardant une cohérence et une bonne lisibilité du code. C'est la raison pour laquelle les propriétés short hand regroupent toujours des ensembles de propriétés qui agissent sur un même aspect d'un élément HTML.

Vous pourrez trouver dans le tableau ci-dessous les propriétés short hand les plus courantes et qu'il vous faut connaitre avec l'ensemble des propriétés long hand qu'elles permettent de définir.

Nous n'avons pour le moment pas étudié la majorité de ces propriétés. Pas d'inquiétude, nous allons le faire au cours de ce cours.

Short Hand	Équivalent Long Hand
`font`	`font-style`, `font-variant`, `font-weight`, `font-size`, `line-height`, `font-family`
`border`	`border-width`, `border-style`, `border-color`
`margin`	`margin-top`, `margin-right`, `margin-bottom`, `margin-left`
`padding`	`padding-top`, `padding-right`, `padding-bottom`, `padding-left`

Short Hand	Équivalent Long Hand
background	background-image, background-position, background-size, background-repeat, background-origin, background-clip, background-attachment, background-color
transition	transition-property, transition-duration, transition-timing-function, transition-delay
animation	animation-name, animation-duration, animation-timing-function, animation-delay, animation-iteration-count, animation-direction, animation-fill-mode, animation-play-state
flex	flex-shrink, flex-grow, flex-basis

PARTIE IV

Mise en forme de textes en CSS

La propriété CSS font-family et les Google Fonts

Dans cette nouvelle partie, nous allons nous intéresser aux propriétés CSS permettant de mettre en forme des textes. Les propriétés CSS liées au texte peuvent être séparées en deux grandes catégories :

- Les propriétés de type `font-` qui vont définir l'aspect des caractères en soi en qui agissent directement sur la police d'écriture (choix de la police, de la taille des caractères ou de leur poids entre autres) ;
- Les propriétés de type `text-` qui ne vont pas impacter directement l'aspect des caractères du texte mais nous permettre d'ajouter des effets de style autour de celui-ci (alignement du texte, soulignement ou encore ajout d'ombres autour des textes par exemple).

Dans cette leçon, nous allons nous intéresser à une première propriété de type `font-` qui va nous permettre de définir la police de nos textes : la propriété `font-family`.

La propriété CSS font-family : définition et exemples d'utilisation

La propriété CSS `font-family` va nous permettre de définir la police de nos textes, c'est-à-dire le rendu graphique de chaque caractère.

Il existe aujourd'hui un très grand nombre de polices d'écritures disponibles et parmi lesquelles on va pouvoir choisir mais il faut savoir que certaines versions de certains navigateurs que peuvent posséder vos visiteurs ne vont pas supporter certaines polices.

Pour cette raison, nous indiquerons toujours plusieurs noms de polices (une « famille de polices ») à utiliser en valeur de la propriété `font-family` en commençant par celle préférée et en séparant chaque valeur par une virgule.

Ainsi, le navigateur va lire les différentes polices renseignées dans `font-family` dans l'ordre et utiliser la première qu'il supporte.

Notez ici que si vous renseignez un nom de police qui contient des espaces, vous devrez le mettre entre guillemets ou apostrophes.

Voyons un premier exemple d'utilisation de `font-family` ensemble. On va définir une même police pour toute la partie visible de notre page HTML en appliquant notre propriété à l'élément `body` (les éléments dans le `body` hériteront par défaut de la valeur passée à la propriété `font-family` pour l'élément `body`).

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p>Un premier paragraphe</p>
        <p>Un autre paragraphe</p>
    </body>
</html>
```

```css
body{
    font-family: "Source code pro", Verdana, sans-serif;
}
```

Dans cet exemple, j'indique trois polices d'écriture différentes en valeur de ma propriété `font-family` : Source code pro, Verdana et sans-serif.

Lors de l'affichage, mon navigateur va lire les valeurs dans leur ordre d'écriture et va appliquer la première valeur de police qu'il supporte / reconnait.

Dans cet exemple, j'ai indiqué intentionnellement trois polices particulières : la police Source code pro est ce qu'on appelle une « Google Font » (nous reviendrons dessus plus tard) tandis que Verdana fait partie des « web safe fonts » (en français « famille de polices sûres ») et sans-serif est ce qu'on appelle une « famille générique ».

Les web safe fonts et les familles génériques

Les web safe fonts où « polices d'écriture sûres pour le web » sont un ensemble de polices qui sont lues de manière universelle par toutes les versions de tous les navigateurs dignes de ce nom.

En précisant une police qui appartient à cette catégorie en valeur de font-family, on s'assure donc que le navigateur de nos visiteurs sera à minima capable de lire et d'afficher cette police (à priori).

Une famille générique représente pas une police en soi mais va servir à définir la forme des caractères à afficher de manière sommaire. Les familles génériques doivent toujours être mentionnées en dernière valeur de la propriété font-family. Elles vont être utilisées en dernier recours si le navigateur n'arrive à lire aucune des polices indiquées dans la propriété font-family.

La famille générique serif par exemple est caractérisée l'empattement de ses caractères tandis que ce sera le contraire pour sans-serif. Notez bien que chaque police connue appartient automatiquement à une famille de polices par défaut.

Il existe 5 familles génériques disponibles en CSS aujourd'hui :

Famille générique	Description
Serif	Les caractères possèdent un empattement
Sans-serif	Les caractères ne possèdent pas d'empattement
Monospace	Les caractères ont tous la même taille et sont espacés équitablement
Cursive	Les caractères sont souvent fins et ressemblent à une écriture manuelle soignée
Fantasy	Les caractères sont généralement difformes ou très stylisés, au contraire de la famille générique monospace

En général, on indiquera toujours au moins une police appartenant aux web safe fonts en valeur de notre propriété font-family ainsi qu'une famille générique associée afin d'être certain qu'au moins une valeur puisse être lue par le navigateur de vos visiteurs.

Comme je vous l'ai dit précédemment, chaque police connue appartient automatiquement à une famille de polices. Une bonne pratique va donc être de préciser une série de polices appartenant à la même famille générique (par exemple, que des polices sans-serif) puis de préciser en dernière valeur de font-family la famille générique associée afin de garder un maximum de cohérence.

Voici une liste des web safe fonts les plus utilisées et de leur famille générique associée :

Famille générique	Police
Serif	Times New Roman, Georgia

Sans-serif	Arial, Verdana
Monospace	Courier New, Lucida Console
Cursive	Comic sans MS

Définition, intégration et utilisation des Google Fonts

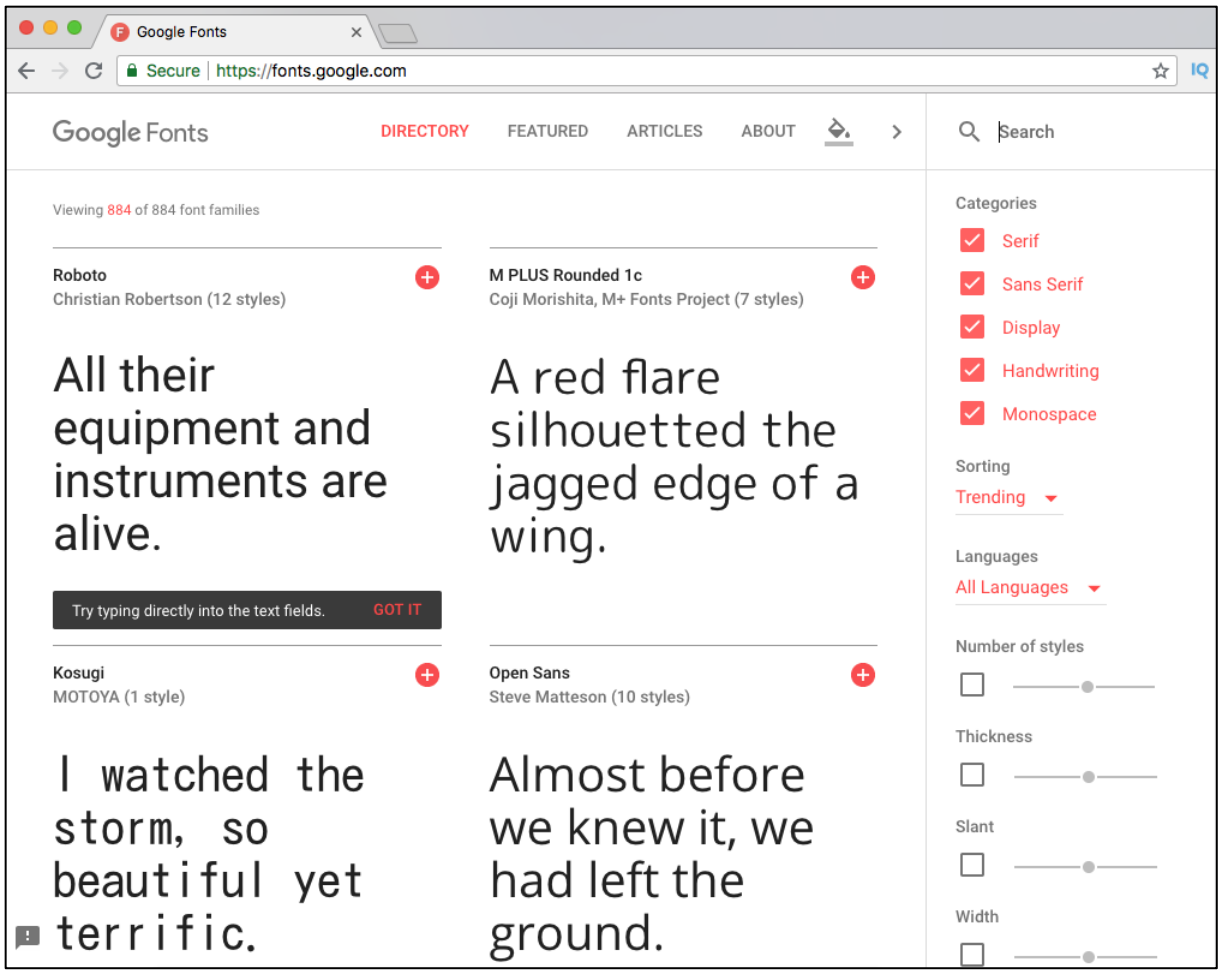
Google a développé sa propre liste de polices d'écriture appelées les Google Fonts.

Ces polices peuvent être utilisées sur n'importe quel site, le tout est de faire un lien en HTML vers la police Google désirée puis de l'utiliser comme valeur de notre propriété `font-family` en CSS.

Un des grands intérêts des Google Fonts est que les polices de Google ne dépendent pas d'un navigateur mais peuvent être lues par (quasiment) tous les navigateurs. En effet, l'idée va être de pré-charger le jeu complet de caractères de la police via Google dans le navigateur d'un utilisateur lorsque celui-ci tente d'accéder à une page de notre site. Ainsi, les risques de non-comptabilité d'une police sont très sérieusement réduits.

L'autre grand avantage est que nous allons pouvoir choisir parmi une très longue liste de polices et ainsi totalement personnaliser l'écriture sur votre site.

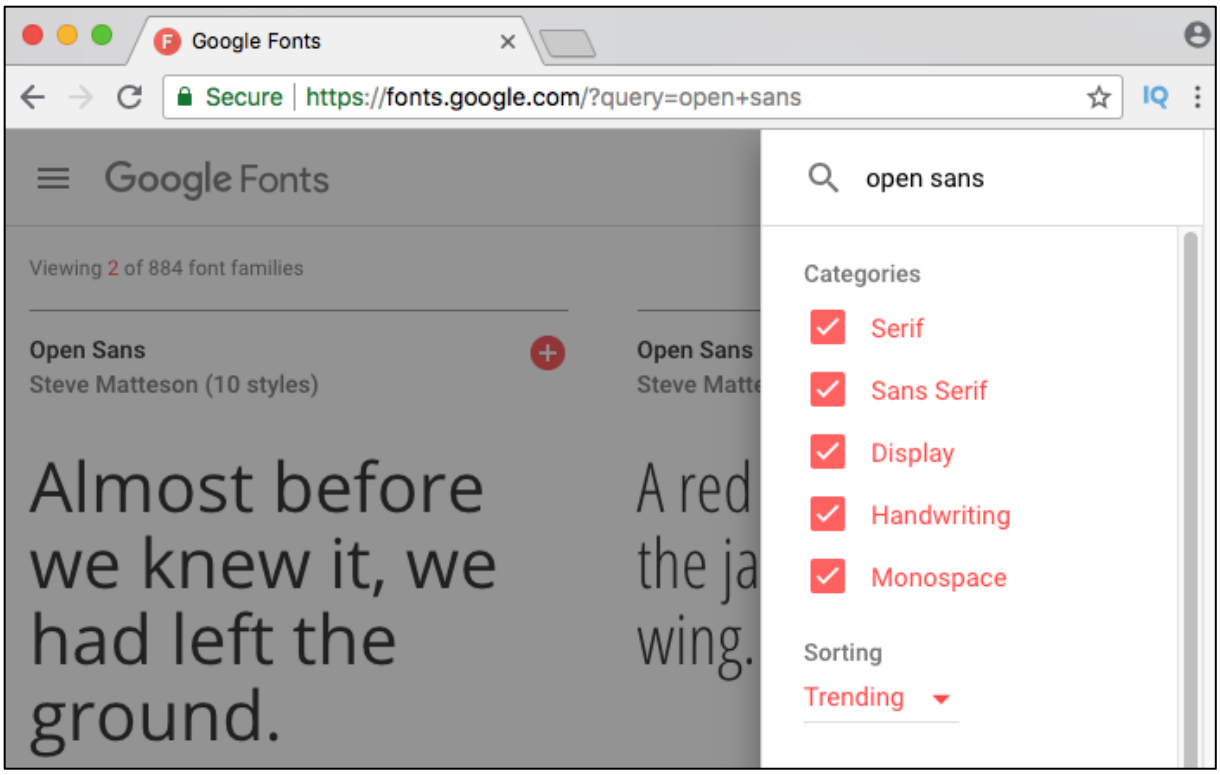
Pour utiliser les Google Fonts, il va tout d'abord nous falloir choisir une police dans la liste proposée sur le site de Google dédié : https://fonts.google.com/.



Sur ce site, vous allez pouvoir soit rechercher une police en particulier en utilisant la recherche en haut à droite, soit pouvoir définir des critères de recherche pour trier les polices et n'afficher que certains types de polices.

Vous allez ainsi pouvoir filtrer les polices selon leur popularité pour voir quelles sont les polices les plus tendances ou encore pouvoir choisir de n'afficher les polices que d'un type de famille générique ce qui va pouvoir s'avérer très utile.

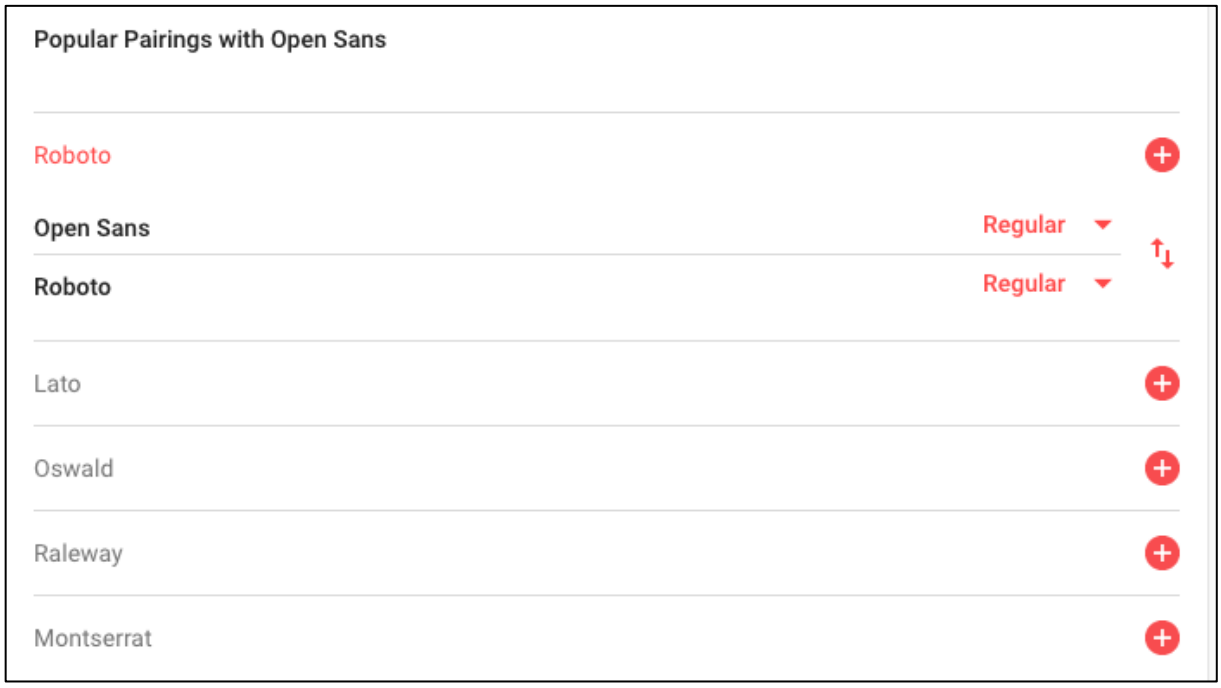
Pour ma part, je vais choisir la police « Open sans », l'une des polices les plus communément utilisées ces dernières années.



Nous allons ici soit pouvoir sélectionner directement la police en question avec les options par défaut en cliquant sur le « + », soit pouvoir cliquer sur la police en soi pour avoir davantage d'informations par rapport à cette Google Font.

Cela va également nous permettre de voir l'aspect des caractères de notre police selon l'épaisseur de celle-ci.

On va également nous dire avec quelle autre police notre police est souvent couplée sur un site. Il est en effet reconnu comme une bonne pratique d'utiliser deux polices différentes pour les titres et les autres contenus textuels et la plupart des sites appliquent cette règle.

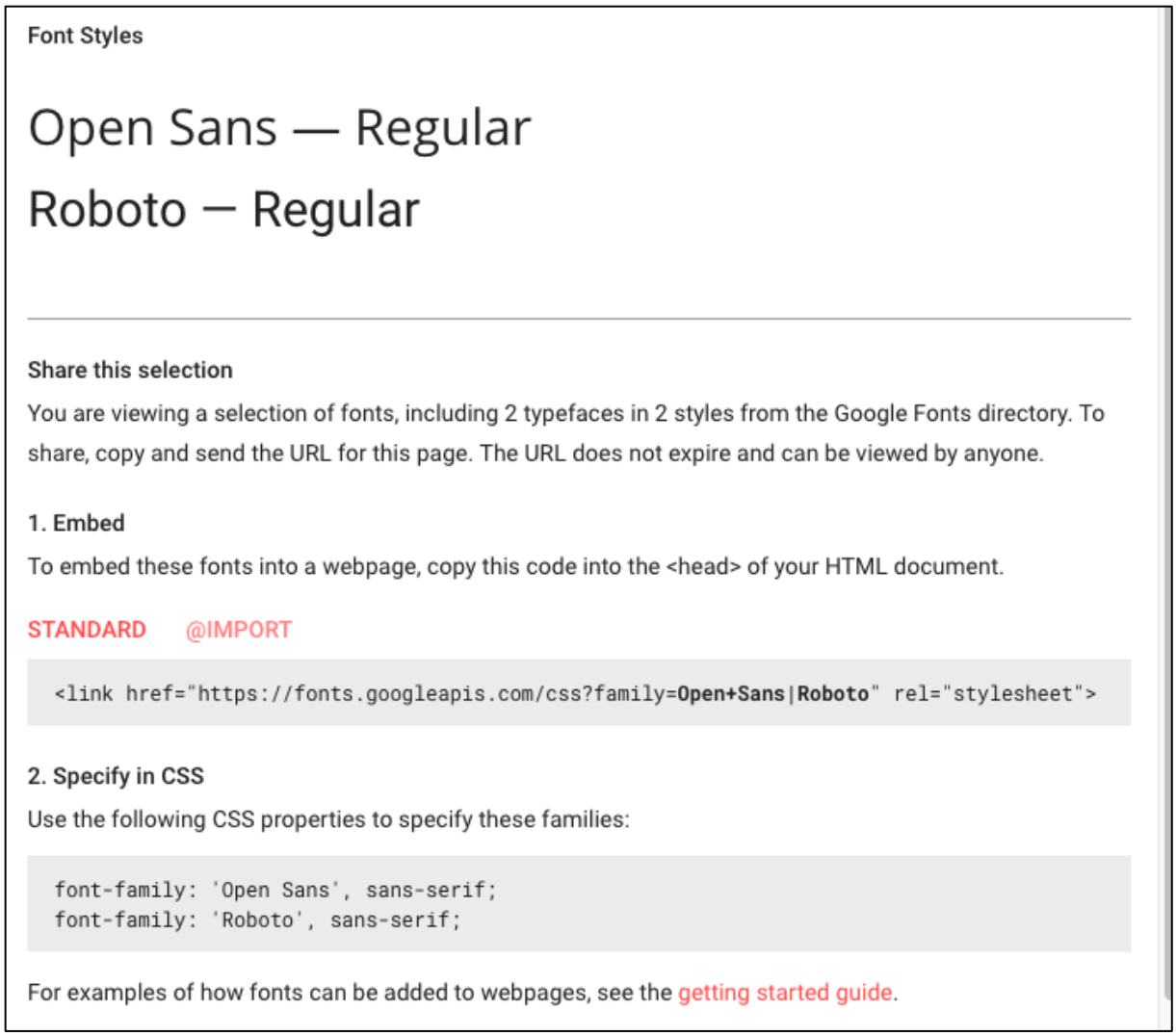


Ici, par exemple, on s'aperçoit que la police « Open Sans » est souvent utilisée avec une autre police bien connue qui est Roboto. Nous pouvons également à cet endroit choisir le style de notre ou de nos polices (Light, Regular, Bold, etc.).

Sélectionnons ici les polices Google « Open sans » et Roboto en version Regular. Pour cela, commencez par sélectionner la police « Open sans » en cliquant sur le « + » en haut à droite de la page puis sélectionnez de la même façon Roboto dans la liste des polices souvent utilisées ensemble. Vous devriez alors avoir un encadré en bas de page avec vos polices sélectionnées.

Cliquez sur cet encadré puis sur la flèche à l'intérieur.

Vous êtes alors finalement ramené sur la page vous expliquant comment intégrer ces deux polices dans vos pages et vous donnant les codes d'intégration de ces deux polices.



Comme vous pouvez le constater, la manipulation est très simple : il suffit d'ajouter dans l'élément `head` de notre page HTML un élément `link` et d'indiquer en valeur de `href` l'URL fournie (ici, par exemple, ce sera `href="https://fonts.googleapis.com/css?family=Open+Sans|Roboto"`) puis de préciser finalement un attribut `rel="stylesheet"`.

Ensuite, nous n'avons plus qu'à indiquer le nom de nos polices en valeur de notre propriété `font-family` dans notre feuillet CSS et le tour est joué ! Regardez plutôt l'exemple suivant dans lequel j'ai intégré mes deux Google Fonts :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
        <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Open+Sans|Roboto">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p>Un premier paragraphe</p>
        <p id="p2">Un autre paragraphe</p>
    </body>
</html>
```

```css
body{
    font-family: "Open sans", Georgia, serif;
}

#p2{
  font-family: Roboto, Verdana, sans-serif;
}
```

Note : Si vous utilisez comme moi le navigateur Google Chrome il est possible que les Google Fonts soient disponibles pour vous sans que vous ayez besoin du lien d'intégration. Cependant, il faut ici penser à vos visiteurs : tous ne vont pas utiliser Google Chrome et c'est la raison pour laquelle il est obligatoire d'insérer le lien de pré chargement des polices dans votre code via l'élément link et son attribut href.

Autres propriétés CSS liées à la police

Les propriétés CSS de type `font-` vont nous permettre de modifier l'apparence des caractères de notre police d'écriture.

Dans la leçon précédente, nous avons étudié la propriété `font-family` qui sert à définir une police d'écriture qui devra être utilisée pour nos textes.

Il existe d'autres propriétés CSS de type `font-` qu'il est bon de connaitre. Dans cette nouvelle leçon, nous allons présenter les propriétés suivantes :

- La propriété `font-size` qui va nous permettre de modifier la taille de notre police ;
- La propriété `font-weight` qui va nous permettre de modifier le poids de notre police ;
- La propriété `font-style` qui va nous permettre de modifier l'inclinaison de notre police.

Notez déjà que les propriétés qui modifient le comportement des polices vont être soumises à la police qu'elles modifient : certaines polices ne supportent pas certaines modifications et les propriétés en question seront alors ignorées.

La propriété font-size

La propriété CSS `font-size` va nous permettre de modifier la taille de notre police d'écriture lors de l'affichage des différents textes. Cette première propriété va pouvoir s'appliquer à toutes les polices puisque toutes les polices supportent un changement de taille aujourd'hui.

Cette propriété va accepter deux grands types de valeurs : des valeurs absolues et des valeurs relatives. Chaque type de valeur va posséder des avantages et des inconvénients et on utilisera plutôt une valeur ou une autre selon la situation. Vous pouvez déjà trouver la liste des différentes valeurs acceptées ci-dessous :

```css
/*Valeurs "mot clef" absolues*/
font-size: xx-small;
font-size: x-small;
font-size: small;
font-size: medium;
font-size: large;
font-size: x-large;
font-size: xx-large;

/*Valeurs "mot clef" relatives*/
font-size: larger;
font-size: smaller;

/*Exemple de valeur "longueur" absolue et fixe*/
font-size: 14px;

/*Exemple de valeurs "longueur" relatives (dynamiques)*/
font-size: 1.5ex;
font-size: 2em;
font-size: 2rem;

/*Exemple de valeur "pourcentage" (relative)*/
font-size: 80%;

/*Valeurs universelles ou globales*/
font-size: inherit;
font-size: initial;
font-size: unset;
```

Une valeur dite « absolue » est une valeur qui ne va pas dépendre d'autres éléments dans la page tandis qu'une valeur « relative » va au contraire être calculée en fonction de la valeur d'autres éléments dans la page.

Attention ici : une valeur absolue n'est pas forcément fixe ou statique puisque certaines valeurs absolues vont dépendre des réglages par défaut de l'utilisateur. « Absolu » ici veut simplement dire que la valeur ne va pas être calculée en fonction des autres éléments dans la page.

Les valeurs de type « mot clef »

On va donc déjà pouvoir passer un mot clef à la propriété `font-size`. Les mots clefs suivants correspondent à des valeurs absolues de police mais qui sont relatives aux réglages faits par l'utilisateur dans son navigateur.

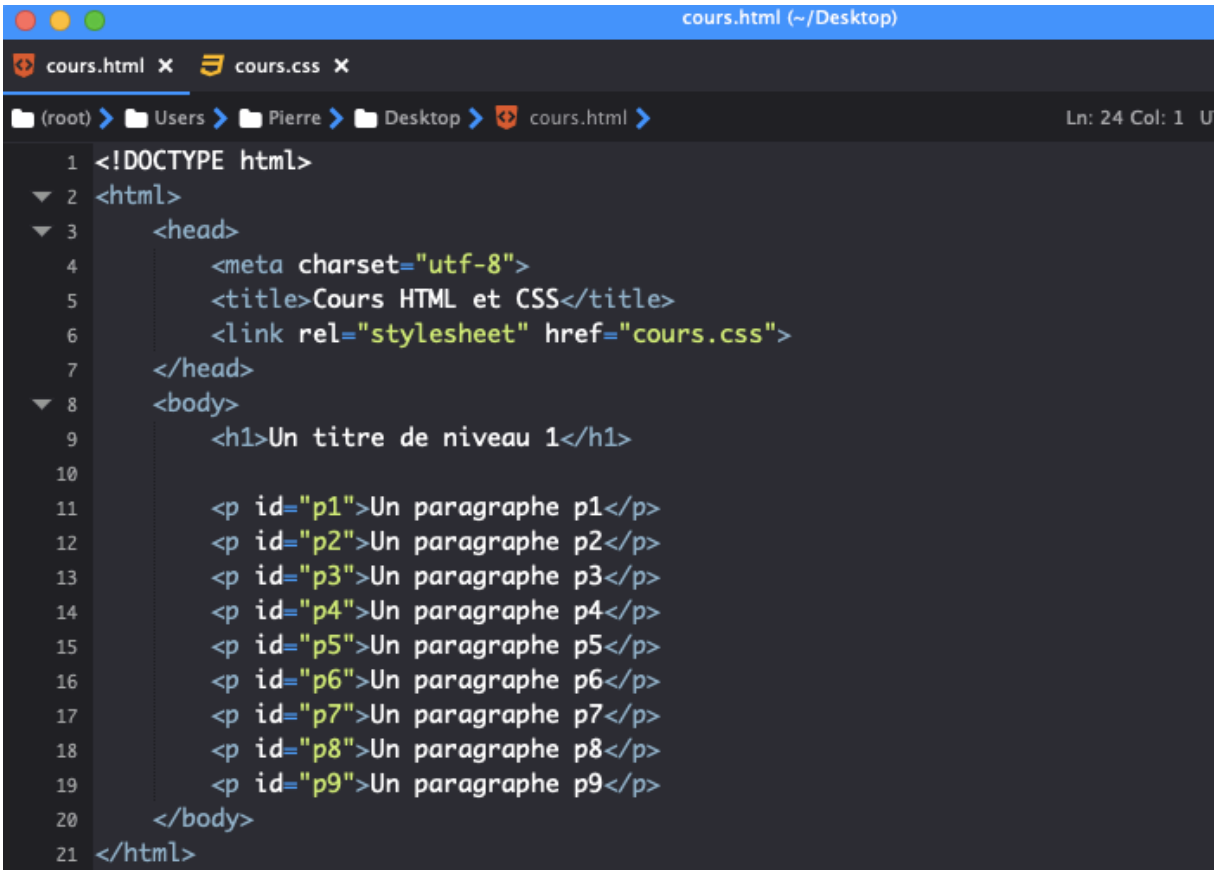
Notez ici que la taille utilisée par défaut par les navigateurs (sans réglage spécifique de l'utilisateur donc) est généralement égale à 16px pour un paragraphe mais va être plus grande pour un titre.

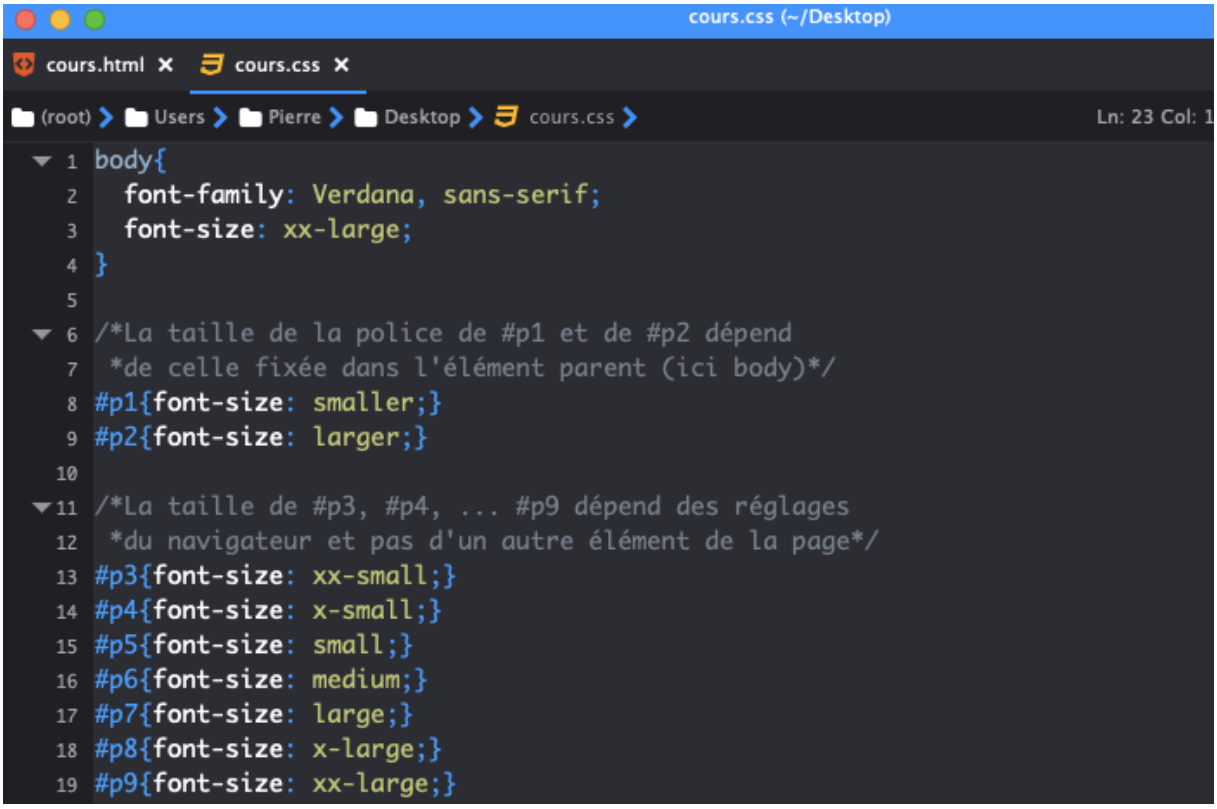
- `xx-small` : la taille de la police sera égale à la moitié de celle définie dans le navigateur ;
- `x-small` : la taille de la police sera égale à 60% de celle définie dans le navigateur ;
- `small` : la taille de la police sera égale à 80% de celle définie dans le navigateur ;

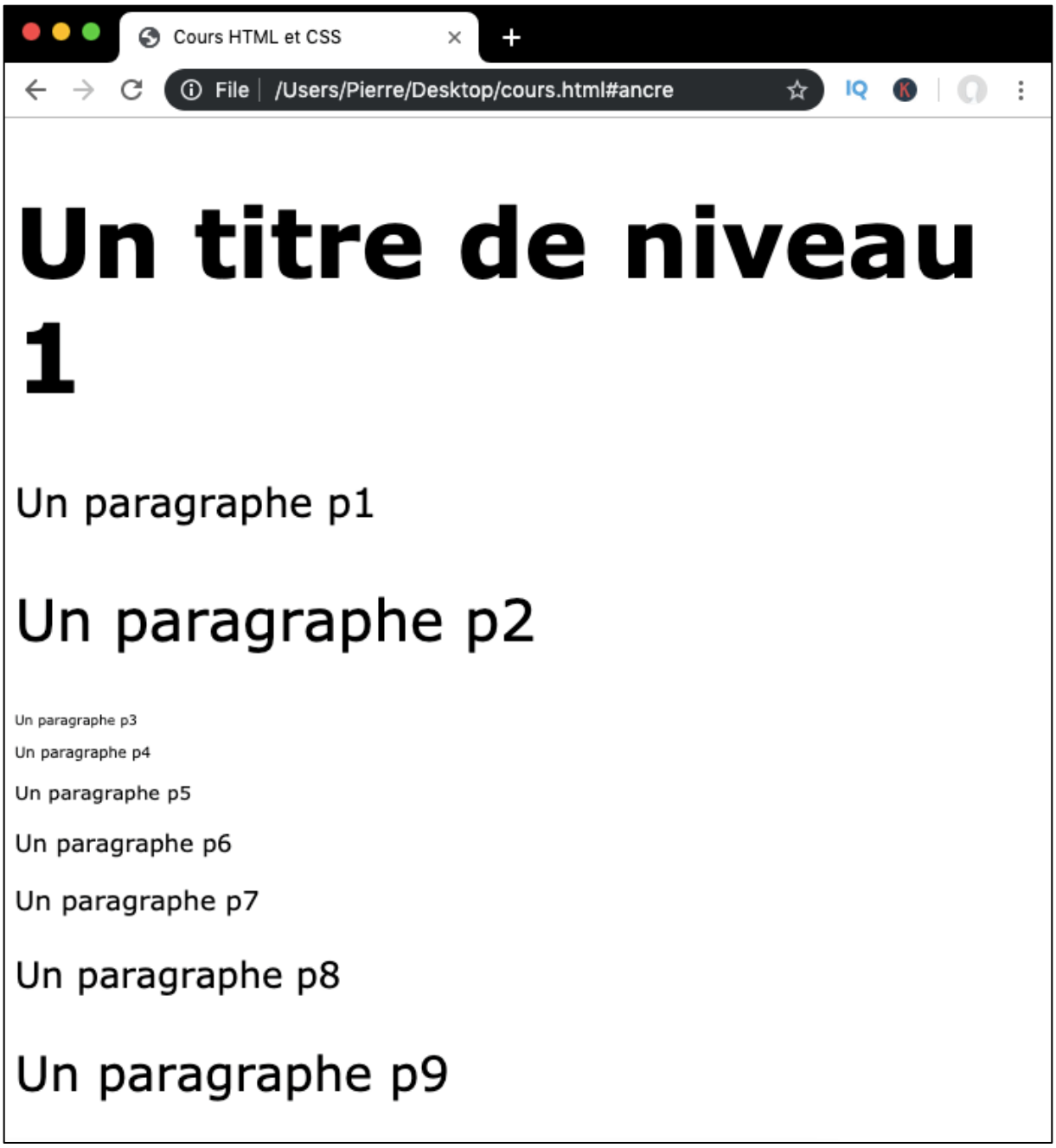
- medium : la taille de la police sera égale à celle définie dans le navigateur ;
- large : la taille de la police sera 10% plus grande que celle définie dans le navigateur ;
- x-large : la taille de la police sera 50% plus grande que celle définie dans le navigateur ;
- xx-large : la taille de la police sera deux fois plus grande que celle définie dans le navigateur.

Les mots clefs suivants permettent de définir une taille de police relativement à celle de l'élément parent :

- smaller : la taille de la police de l'élément sera plus petite que celle de son élément parent ;
- larger : la taille de la police de l'élément sera plus grande que celle de son élément parent.




cours.css (~/Desktop)
cours.html
cours.css
(root) > Users > Pierre > Desktop > cours.css
Ln: 23 Col: 1
body{
font-family: Verdana, sans-serif;
font-size: xx-large;
}
/*La taille de la police de #p1 et de #p2 dépend
de celle fixée dans l'élément parent (ici body)/
#p1{font-size: smaller;}
#p2{font-size: larger;}
/*La taille de #p3, #p4, ... #p9 dépend des réglages
du navigateur et pas d'un autre élément de la page/
#p3{font-size: xx-small;}
#p4{font-size: x-small;}
#p5{font-size: small;}
#p6{font-size: medium;}
#p7{font-size: large;}
#p8{font-size: x-large;}
#p9{font-size: xx-large;}



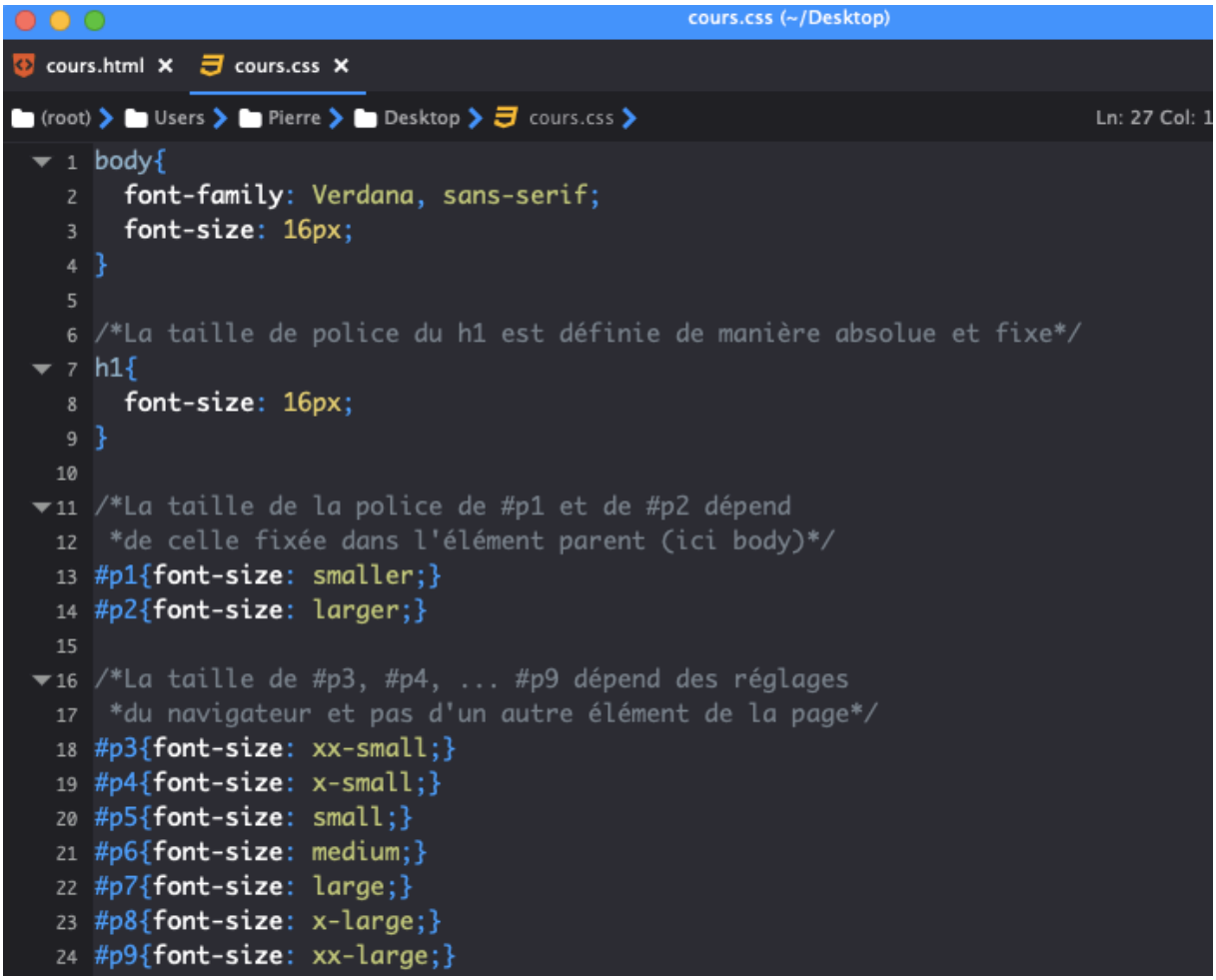
Ici, la taille de la police des paragraphes `p1` et `p2` dépend de la taille de la police de leur élément parent qui est dans ce cas l'élément `body` et qui a été réglée à `font-size: xx-large`.

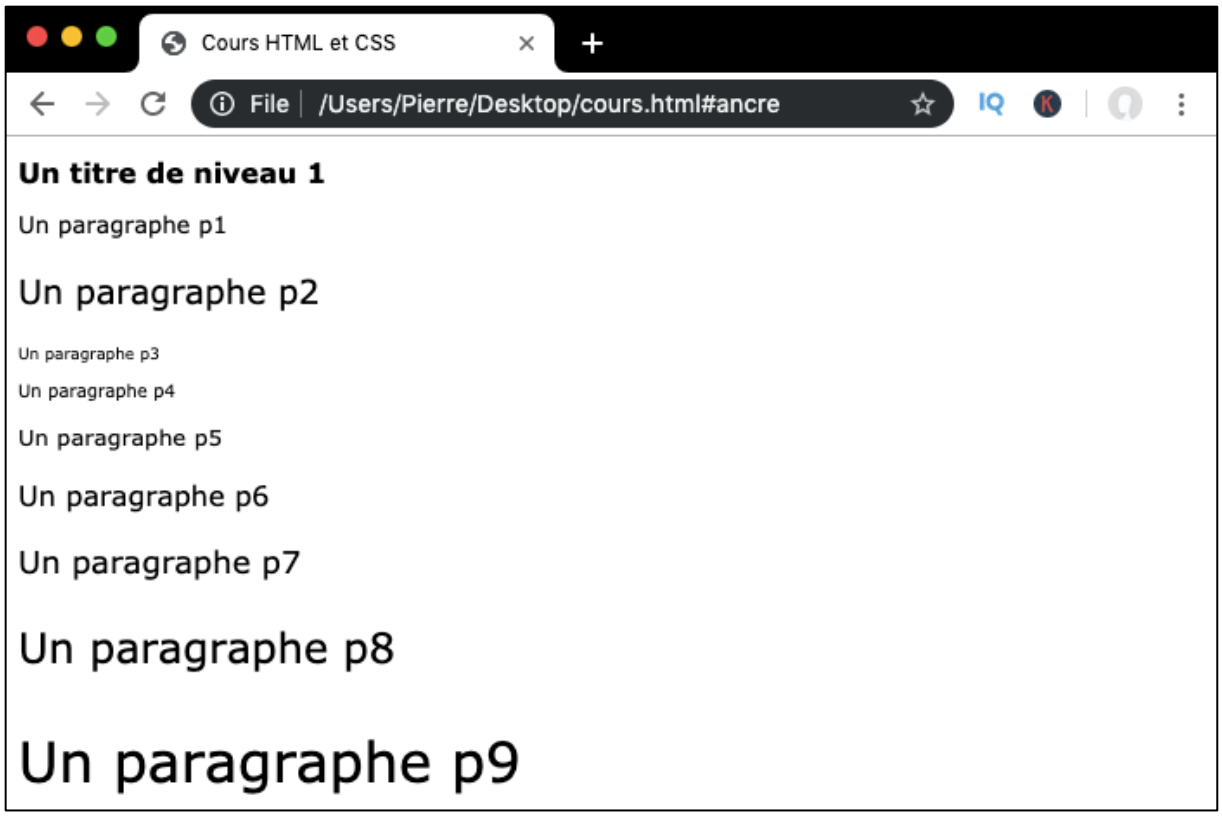
Notez également qu'ici notre titre `h1` hérite de la valeur de la propriété de son parent, c'est-à-dire `font-size: xx-large` et va donc s'afficher en très gros puisque le réglage par défaut des navigateurs pour un titre `h1` définit généralement une taille de police deux fois plus grande que pour un paragraphe.

Les valeurs de type « longueur »

On va également pouvoir passer une longueur en valeur de la propriété `font-size`. Une longueur peut être exprimée en `px` (pixels). Dans ce cas, la taille de la police sera absolue et statiques c'est-à-dire indépendante de tout réglage de l'utilisateur ou des valeurs par défaut du navigateur.

Utiliser des valeurs en px permet donc d'avoir un résultat très prévisible mais empêche les utilisateurs d'adapter la taille du texte selon leurs réglages, ce qui va être très problématique pour les utilisateurs souffrant de déficiences visuelles.





Une longueur peut également être exprimée en `em` ou plus rarement en `ex`. Les valeurs en `em` (et en `ex`) vont être dynamiques et relatives à la valeur de l'élément parent pour la propriété concernée.

En déclarant `font-size : 1em` pour un élément on demande à ce que la valeur de la propriété soit la même que celle définie pour l'élément parent, en déclarant `font-size : 1.5em` on demande à ce que la valeur de la propriété soit égale à 1,5 fois celle définie pour l'élément parent et etc.

Note : Attention ici à bien utiliser des points à la place des virgules (notations anglo-saxonnes) lorsque vous préciser des valeurs non entières en CSS !

Si aucune taille n'a été définie pour l'élément parent alors la taille définie dans le navigateur sera utilisée (généralement 16px par défaut sauf réglage utilisateur spécifique).

Les valeurs en `em` possèdent donc l'avantage de pouvoir s'adapter mais peuvent entrainer des problèmes de composition ou de cohérence / d'homogénéisation. En effet, imaginons qu'un attribue un `font-size : 1.5em` pour un type d'éléments comme les éléments `span` par exemple.

Si dans ce cas un élément `span` contient lui-même un élément `span`, les tailles de police des différents éléments `span` vont être différentes puisque la police du `span` contenu dans l'autre `span` va être égale à 1,5 fois celle du `span` parent.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Un titre de niveau 1</h1>

        <p>Un paragraphe p1</p>
        <p>Un paragraphe p2</p>
        <p>Un paragraphe <span>avec un span qui
        <span>contient</span> lui même un span</span></p>
    </body>
</html>
```

```css
body{
    font-family: Verdana, sans-serif;
    font-size: 16px;
}

/*Ici, 2em = 32px (2 fois la taille de
l'élément parent qui est l'élément body)*/
h1{
    font-size: 2em;
}

p{
    font-size: 0.8em;
}

span{
  font-size: 1.5em;
}
```

Ici, la taille de la police du premier span dans notre paragraphe est égale à 1,5 fois celle du paragraphe (qui est son parent) tandis que la taille de la police du span enfant dans le span parent est égale à 1,5 fois celle du span parent.

Souvent, ce ne sera pas le comportement souhaité et ce genre de situations est ce qui rend ce type de valeurs (et les valeurs relatives en général) très difficile à manier en pratique.

Pour pallier ce problème, nous pouvons plutôt utiliser l'unité rem. Les valeurs fournies en rem vont être relatives à la taille définie pour la propriété de l'élément racine html et non pas à la taille définie pour l'élément parent.

```
body{
    font-family: Verdana, sans-serif;
    font-size: 16px;
}

/*Ici, 2em = 32px (2 fois la taille de
l'élément parent qui est l'élément body)*/
h1{
    font-size: 2em;
}

p{
    font-size: 0.8em;
}

/*La taille de la police des span dépend de la taille
 *réglée dans l'élément html ou de celle par défaut du
 *navigateur si rien n'a été précisé (16px = 1em généralement)*/
span{
  font-size: 1.5rem;
}
```

Les valeurs de type « pourcentage »

Nous allons enfin également pouvoir passer un pourcentage en valeur de la propriété font-size. Les valeurs exprimées en pourcentage vont être proportionnelles à la valeur renseignée pour la propriété dans l'élément parent.

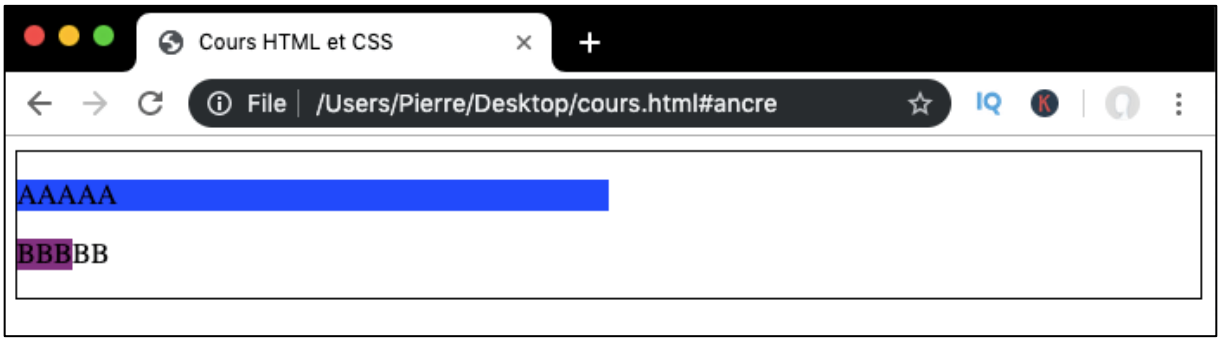
```css
body{
  font-family: Verdana, sans-serif;
  font-size: 16px;
}

/*Ici, 2em = 32px (2 fois la taille de l'élément
 *parent qui est l'élément body)*/
h1{
  font-size: 2em;
}

/*80% = 0.8e*/
p{
  font-size: 80%;
}

span{
  font-size: 150%;
}
```

Les unités en em et les % vont donc être équivalent pour définir la taille de la police d'un élément. Cependant, notez bien que ces unités vont pouvoir être réutilisées avec d'autres propriétés et dans ce cas elles ne vont plus l'être puisque les unités en em vont toujours être exprimées en fonction de la taille de la police.

Regardez plutôt l'exemple suivant. La propriété width sert ici à définir la largeur d'un élément et peut accepter des valeurs aussi bien en em qu'en % :

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur">
            <div id="d1"><p>AAAAA</p></div>
            <div id="d2"><p>BBBBB</p></div>
        </div>
    </body>
</html>
```

```css
.conteneur{
  border: 1px solid black;/*Crée une bordure autour du div*/
}

#d1{
  font-size: 16px; /*Taille (hauteur) du texte : 16px*/
  width: 50%;/*Largeur égale à 50% de celle de l'élément parent*/
  background-color: blue;
}
#d2{
  font-size: 16px; /*Taille (hauteur) du texte : 16px*/
  width: 2em; /*Largeur égale à 2 fois la taille du texte*/
  background-color: purple;
}
```



Ici, la largeur du `div #d1` est égale à la moitié de la largeur de son élément parent (`width : 50%`) tandis que la largeur du `div #d2` est égale à deux fois la taille de la police définie dans l'élément (`width : 2em`). J'ai ajouté une couleur de fond aux deux `div` afin qu'on puisse bien voir l'espace pris par chacun d'entre eux.

Les valeurs universelles

On va également bien évidemment pouvoir passer une valeur universelle à la propriété `font-size` comme `inherit` ou `initial` par exemple pour que la propriété hérite du comportement défini dans un élément parent ou pour qu'elle soit réinitialisée à sa valeur d'origine définie dans la feuille de style par défaut du navigateur.

Note importante : les valeurs inherit, initial et unset sont des valeurs dites « globales » ou « universelles » car elles vont fonctionner avec toutes les propriétés CSS. Dans la suite de ce cours, je ne préciserai donc pas pour chaque propriété étudiée qu'elle peut utiliser ces valeurs car cela est évident.

La propriété font-weight

La propriété CSS font-weight va nous permettre de définir le poids d'une police, c'est-à-dire son épaisseur.

Cette propriété va pouvoir prendre un mot clef ou un chiffre exprimé en centaine(s) en valeur. Nous allons ainsi pouvoir choisir parmi les valeurs suivantes :

- Le mot clef normal : valeur par défaut qui correspond à un poids de police « normal » ;
- Le mot clef lighter qui va définir une police d'écriture plus fine que pour la valeur normal ;
- Le mot clef bold qui va définir une police d'écriture plus épaisse que pour la valeur normal ;
- Le mot clef bolder qui va définir une police d'écriture très épaisse ;
- Une centaine entre 100 (qui correspond à une police très fine) et 900 (pour une police très épaisse). En termes d'équivalence avec les mots clefs, vous pouvez retenir que la valeur 400 est équivalente au mot clef normal et que la valeur 700 est équivalente au mot clef bold.

Notez bien ici que toutes les valeurs évoquées ci-dessus ne vont pas toujours pouvoir être appliquées : en effet, certaines polices ne supportent tout simplement pas de manière intrinsèque certains poids.

Dans le cas où l'on passe une valeur à la propriété font-weight qui ne peut pas être appliquée à une certaine police, alors elle sera tout simplement ignorée.

Les deux valeurs généralement bien supportées sont font-weight : normal ou font-weight : 400 et font-weight : bold ou font-weight : 700.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p id="p1">Un premier paragraphe</p>
        <p id="p2">Un autre paragraphe</p>
        <p>Un troisième paragraphe</p>
    </body>
</html>
```

```css
body{
  font-family: Verdana, sans-serif;
}

h1{
  font-weight: 400;/*Identique à font-weight: normal*/
}

#p1{
    font-weight: bold;/*Identique à font-weight: 700*/
}

#p2{
    font-weight: lighter;
}
```

La propriété font-style

La propriété `font-style` va nous permettre de gérer l'inclinaison de notre police. On va pouvoir lui passer l'une des valeurs suivantes :

- `normal` : valeur par défaut, les caractères seront droits ;
- `italic` : la police va s'afficher en italique ;
- `oblique` : la police va être tordue pour être rendue de façon oblique.

Les valeurs italic et oblique produisent souvent un résultat très similaire. Sachez cependant qu'elles ne sont pas strictement équivalentes pour autant.

En effet, certaines polices ne supportent pas l'état italique (car elles n'ont pas été conçues pour pouvoir être rendues en italique). Dans ces cas-là, si on passe la valeur `italic` à la propriété `font-style` alors celle-ci sera simplement ignorée.

La valeur `oblique`, au contraire, va forcer l'inclinaison de n'importe quelles polices et mêmes de celles qui n'ont pas été conçues pour pouvoir être rendues de façon oblique. Dans ce cas-là, cette valeur va « casser » la police afin de la rendre quand même oblique (ce qui peut amener à un rendu visuel non optimal).

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p id="p1">Un premier paragraphe</p>
        <p id="p2">Un autre paragraphe</p>
        <p>Un troisième paragraphe</p>
    </body>
</html>
```

```css
body{
  font-family: Verdana, sans-serif;
}

/*Le texte de notre paragraphe p1 s'affiche en gras italique*/
#p1{
    font-weight: bold;
    font-style: italic;
}

/*Le texte de notre paragraphe p1 s'affiche en oblique*/
#p2{
    font-style: oblique;
}
```

La notation raccourcie font

Les différentes propriétés de type font- peuvent être combinées en une seule qui représente leur notation raccourcie ou « short hand » en anglais et qui va être tout simplement la propriété font.

Nous reparlerons de l'utilisation des propriétés short-hand plus tard dans ce cours plus en détail.

Pour le moment, retenez simplement qu'une notation raccourcie ou « short-hand » va se définir par opposition à la version complète ou « long hand » des propriétés.

Les notations short-hand vont être un condensées de différentes propriétés et vont donc nous permettre d'écrire notre code CSS plus rapidement en déclarant d'un coup les valeurs relatives à plusieurs propriétés.

Cependant, ici, il faudra souvent respecter un ordre précis de déclarations des différentes valeurs relatives aux propriétés long-hand agrégées dans la version short-hand afin qu'il n'y ait pas d'ambiguïté sur ce à quoi va correspondre chaque valeur déclarée.

Nous allons pouvoir passer les valeurs suivantes à la notation short-hand font (dans l'ordre donné ci-après) :

1. La valeur relative à la propriété font-style (facultatif) ;
2. La valeur relative à la propriété font-variant (facultatif) ;
3. La valeur relative à la propriété font-weight (facultatif) ;
4. La valeur relative à la propriété font-size (obligatoire)/line-height (facultatif) ;
5. La valeur relative à la propriété font-family (obligatoire).

La propriété font-variant n'est pas beaucoup utilisée et c'est la raison pour laquelle je n'en ai pas parlé pour le moment. Pour faire simple, elle est elle-même une notation raccourcie des sous propriétés font-variant-caps, font-variant-numeric, font-variant-alternates, font-variant-ligatures et font-variant-east-asian qui permettent d'utiliser des glyphes alternatifs pour les différents caractères d'une police.

La propriété line-height permet de définir la taille de l'espace entre les lignes (interligne). Nous étudierons cette propriété en détail plus tard dans cette partie.

Notez que pour que font fonctionne correctement, il faudra obligatoirement à minima préciser les valeurs liées aux propriétés font-size et font-family. Les autres valeurs sont facultatives et pourront être omises.

Voilà comment nous allons pouvoir utiliser cette propriété raccourcie en pratique :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p id="p1">Un premier paragraphe</p>
        <p id="p2">Un autre paragraphe</p>
        <p>Un troisième paragraphe</p>
    </body>
</html>
```

```css
body{
  font-family: Verdana, sans-serif;
}

/*On déclare les valeurs relatives à chacune de nos propriétés long-hand*/
#p1{
  font: normal small-caps bold 18px/1.5em Verdana, sans-serif;
}

/*On omet certaines valeurs facultatives*/
#p2{
  font: italic 18px Verdana, sans-serif;
}
```

Expliquons rapidement le code ci-dessus. Ma première définition de `font` liée au sélecteur `#p1` est complète : j'ai passé les valeurs relatives à toutes les propriétés décrites ci-dessus et celles-ci vont donc être appliquées à l'élément portant l'`id="p1"`. Notez que les valeurs relatives aux propriétés `font-size` et `line-height` doivent être séparées par un slash.

Dans mon sélecteur `#p2`, cependant, j'ai omis certaines valeurs qui sont facultatives et n'ai passé que les valeurs relatives aux propriétés `font-style`, `font-size` et `font-family`.

Vous devriez avoir une question par rapport à cette deuxième déclaration : comment savoir à quelle propriété correspond une valeur lorsque celle-ci n'est pas unique à la propriété ?

Imaginons ici qu'il y ait écrit `normal` à la place de `italic`. Comment savoir si cette valeur `normal` a été déclarée comme valeur pour ma propriété `font-style`, `font-variant` ou `font-weight` ?

Le CSS va en fait lire l'ensemble des valeurs fournies à la notation short hand `font` et, s'il y a ambiguïté, les valeurs fournies vont être liées à la première propriété qui l'accepte. C'est tout l'intérêt d'avoir défini un ordre des valeurs et de le respecter. Ainsi, dans le cas présent, comme `font-style` accepte la valeur `normal`, celle-ci lui sera attribuée.

Notez qu'ici ce n'est pas impactant puisque dans tous les cas la valeur `normal` est la valeur par défaut pour les trois propriétés `font-style`, `font-variant` ou `font-weight` mais ça pourra l'être dans d'autres cas où des propriétés partagent des valeurs en commun qui ne sont pas leurs valeurs par défaut.

Notez également qu'il est généralement déconseillé d'utiliser à la fois les notations short-hand et long-hand pour définir le comportement des mêmes propriétés car cela peut mener à de nombreux problèmes dans le code.

Si pour une raison ou une autre vous devez redéfinir vos propriétés de cette manière, retenez bien que pour toute notation long-hand déclarée avant la version short-hand associée sera ignorée même dans le cas où la version short-hand ne définit pas de comportement pour la propriété long-hand en question de manière explicite.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p id="p1">Un premier paragraphe</p>
        <p id="p2">Un autre paragraphe</p>
        <p>Un troisième paragraphe</p>
    </body>
</html>
```

```css
#p1{
  font-style: italic;/*Va être ignorée*/
  font: normal small-caps bold 18px/1.5em Verdana, sans-serif;
  font-weight: normal;/*Va être appliquée*/
}

#p2{
  font-style: italic;/*Va être ignorée*/
  font: 18px Verdana, sans-serif;
  font-weight: normal;/*Va être appliquée*/
}
```

Les propriétés CSS liées au texte

Les propriétés liées au texte ne vont pas nous permettre de modifier la forme des caractères de la police mais plutôt d'ajouter des effets autour de notre texte ou de le formater dans la page d'une manière ou d'une autre.

Dans cette nouvelle leçon, nous allons présenter et étudier quelques-unes des propriétés CSS de type `text-` les plus utiles et les plus intéressantes et notamment :

- La propriété `text-align` qui va nous permettre de gérer l'alignement du texte ;
- La propriété `text-transform` qui va nous permettre de gérer la casse du texte (le fait que le texte soit en majuscules ou en minuscules) ;
- La propriété `text-decoration` qui va nous permettre d'ajouter des éléments de décoration autour du texte comme un trait de soulignement par exemple ;
- La propriété `text-indent` qui va nous permettre de définir l'indentation d'un texte ;
- La propriété `text-shadow` qui va nous permettre d'ajouter des ombres autour d'un texte.

La propriété text-align

La propriété `text-align` va nous permettre de définir l'alignement d'un texte par rapport à son élément conteneur.

Nous allons pouvoir choisir parmi les valeurs d'alignement suivantes :

- `left` : valeur par défaut. Le texte sera aligné contre le bord gauche de l'élément qui le contient ;
- `center` : Le texte sera centré dans l'élément qui le contient ;
- `right` : Le texte sera aligné contre le bord droite de l'élément qui le contient ;
- `justify` : Le texte sera justifié (les écarts entre les mots vont être calculés de façon à ce que chaque ligne de texte occupe la même largeur).

Notez bien que la propriété `text-align` va toujours aligner le texte dans / par rapport à l'élément qui le contient. Ainsi, appliquer cette propriété à un élément de type `inline` comme un `span` par exemple n'aurait aucun sens puisque ce type d'élément possède la même taille que son contenu.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur">
            <h1>Un titre de niveau 1</h1>

            <p class="p1">Un premier paragraphe</p>
            <p class="p2">Un autre paragraphe</p>
        </div>
    </body>
</html>
```

```css
/*On ajoute une bordure autour de l'élément conteneur.
 *Le div fait 70% de la largeur de son parent (le body)*/
.conteneur{
  border: 1px solid black;
  width: 70%;
}

/*Le texte est centré p/r à l'élément parent (le div conteneur)*/
h1{
    text-align: center;
}
/*Texte aligné à gauche p/r à l'élément parent (le div)*/
.p1{
    text-align: left;
}
/*Texte aligné à droite p/r à l'élément parent (le div)*/
.p2{
    text-align: right;
}
```

La propriété text-transform

La propriété `text-transform` va nous permettre de modifier la casse d'un texte ou de certaines parties d'un texte, c'est-à-dire de gérer le fait qu'un texte ou qu'une partie de texte s'affiche en majuscules ou en minuscules.

Cette propriété peut être utile dans le cas de textes générés automatiquement ou dans les cas où nous n'avons pas accès ou ne pouvons pas modifier le texte dans le code HTML directement.

Nous allons pouvoir passer l'une des valeurs suivantes à `text-transform` :

- `none` : Valeur par défaut. Pas de transformation du texte. Utile pour annuler une transformation transmise par héritage par exemple ;
- `lowercase` : Transforme tout le texte d'un élément en minuscules ;
- `uppercase` : Transforme tout le texte d'un élément en majuscules ;
- `capitalize` : Transforme la première lettre de chaque mot en majuscule.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>
    </body>
</html>
```

```css
/*Tout en majuscules*/
h1{
    text-transform: uppercase;
}
/*Tout en minuscules*/
.p1{
    text-transform: lowercase;
}
/*Première lettre de chaque mot en majuscules*/
.p2{
    text-transform: capitalize;
}
```

La propriété text-decoration

La propriété `text-decoration` va nous permettre d'ajouter des décorations à un texte, comme un trait de soulignement ou de surlignement par exemple.

On va pouvoir lui passer jusqu'à trois valeurs qui vont correspondre au type de décoration (valeur obligatoire), à la couleur de la décoration (valeur facultative, couleur actuelle utilisée par défaut) et au style de la décoration (solide par défaut).

Concernant le type de décoration, nous allons pouvoir choisir parmi les valeurs :

- `underline` : ajoute un trait de soulignement au texte ;
- `overline` : ajoute un trait de surlignement au texte ;
- `line-through` : ajoute un trait qui va barrer le texte ;
- `underline overline` : ajoute un trait de soulignement et un trait de surlignement au texte.

Concernant le style de la décoration, nous allons pouvoir choisir parmi les valeurs :

- `solid` : valeur par défaut ; le trait de décoration sera solide ;
- `double` : le trait de décoration sera double ;
- `dotted` : le trait de décoration sera en pointillés ;
- `dashed` : le trait de décoration sera en tirets ;
- `wavy` : le trait de décoration sera courbé.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>
        <p class="p3">Un troisième paragraphe</p>
        <p>Un <a href="http://wikipedia.org">lien</a></p>
    </body>
</html>
```

```css
h1{
    text-decoration: underline;
}
.p1{
    text-decoration: underline overline red;
}
.p2{
    text-decoration: overline dashed;
}
.p3{
  text-decoration: underline wavy;
}
a{
    text-decoration: none;
}
```

Notez également qu'à l'heure actuelle la définition de la propriété `text-decoration` est en train d'évoluer puisque de nouvelles sous propriétés CSS nous permettant de gérer un aspect de la décoration à la fois ont été dernièrement proposées. Si cette nouvelle

définition est acceptée, alors la propriété `text-decoration` deviendra la notation raccourcie des propriétés suivantes :

- `text-decoration-line` qui permet de définir une décoration autour du texte ;
- `text-decoration-color` qui permet de choisir la couleur de la décoration (noire par défaut) ;
- `text-decoration-style` qui permet de choisir le style de la décoration (solide par défaut).

La propriété `text-decoration-line` va accepter les valeurs suivantes :

- `underline` : ajoute un trait de soulignement au texte ;
- `overline` : ajoute un trait de surlignement au texte ;
- `line-through` : ajoute un trait qui va barrer le texte ;
- `underline overline` : ajoute un trait de soulignement et un trait de surlignement au texte.

La propriété `text-decoration-style` va accepter les valeurs suivantes :

- `solid` : valeur par défaut ; le trait de décoration sera solide ;
- `double` : le trait de décoration sera double ;
- `dotted` : le trait de décoration sera en pointillés ;
- `dashed` : le trait de décoration sera en tirets ;
- `wavy` : le trait de décoration sera courbé.

A noter cependant que pour le moment ces propriétés ne sont pas encore officielles et vont pas encore forcément supportées par tous les navigateurs. Elles devraient cependant bientôt le devenir et c'est pourquoi je les mentionne dans ce cours.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p1bis">Un premier paragraphe (bis)</p>
        <p class="p2">Un autre paragraphe</p>
        <p class="p3">Un troisième paragraphe</p>
        <p>Un <a href="http://wikipedia.org">lien</a></p>
    </body>
</html>
```

```css
h1{
    text-decoration: underline;
}

.p1{
    text-decoration: underline overline red;
}
/*Même chose qu'au dessus mais en version longue*/
.p1bis{
  text-decoration-line: underline overline;
  text-decoration-color: red;
}

.p2{
    text-decoration: overline dashed;
}

.p3{
  text-decoration: underline wavy;
}

a{
    text-decoration: none;
}
```

La propriété text-indent

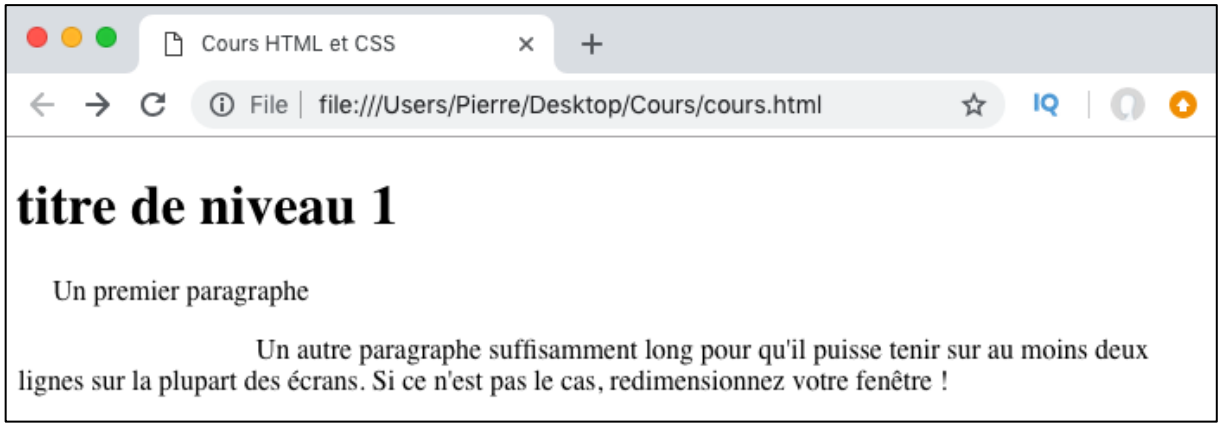
La propriété text-indent va nous permettre de préciser l'indentation c'est-à-dire le retrait de la première ligne d'un texte par rapport au bord de l'élément parent.

Nous allons ici pouvoir passer une valeur en px, em, %, etc. Les valeurs en % vont être exprimées en fonction de la largeur de l'élément parent.

Notez que les valeurs négatives sont acceptées.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe suffisamment long pour qu'il
        puisse tenir sur au moins deux lignes sur la plupart des écrans.
        Si ce n'est pas le cas, redimensionnez votre fenêtre !</p>
    </body>
</html>
```

```css
h1{
    text-indent: -50px;
}

.p1{
    text-indent: 20px;
}

.p2{
    text-indent: 20%;/*De l'élément parent c'est-à-dire du body*/
}
```



La propriété text-shadow

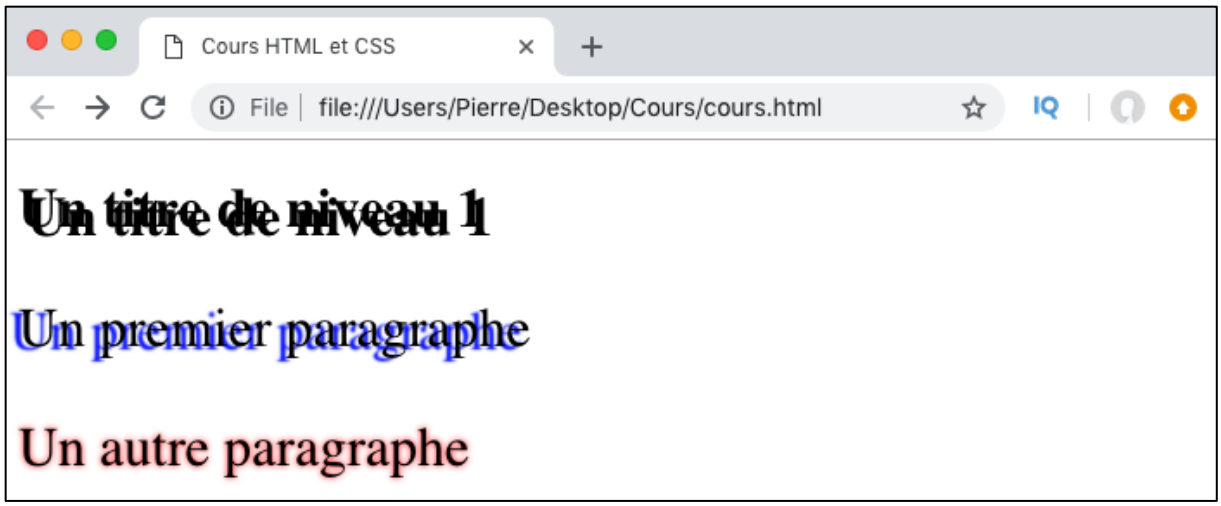
La propriété text-shadow va nous permettre d'ajouter des ombres autour de nos textes. Cette propriété est relativement complexe à maitriser puisqu'elle va pouvoir utiliser jusqu'à 4 valeurs pour définir précisément une ombre et surtout puisqu'on va pouvoir appliquer plusieurs ombres différentes à un même texte grâce à elle.

Les 4 valeurs de text-shadow vont correspondre :

1. Au déplacement (ou « projection ») horizontal de l'ombre par rapport au texte. En passant une valeur positive, l'ombre est projeté à droite du texte. En passant une valeur négative, l'ombre est projetée à gauche de celui-ci. Cette valeur doit obligatoirement être renseignée ;
2. Au déplacement vertical de l'ombre par rapport au texte. En passant une valeur positive, l'ombre est projeté sous le texte. En passant une valeur négative, l'ombre est projetée au-dessus de celui-ci. Cette valeur doit obligatoirement être renseignée ;
3. Au rayon de flou de l'ombre. Un flou Gaussien est utilisé ici : plus la valeur est grande, plus l'ombre sera étendue et floue. Cette valeur est facultative ;
4. A la couleur de l'ombre. On va ici pouvoir passer toutes les valeurs de couleurs (que nous étudierons plus tard) disponibles.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>
    </body>
</html>
```

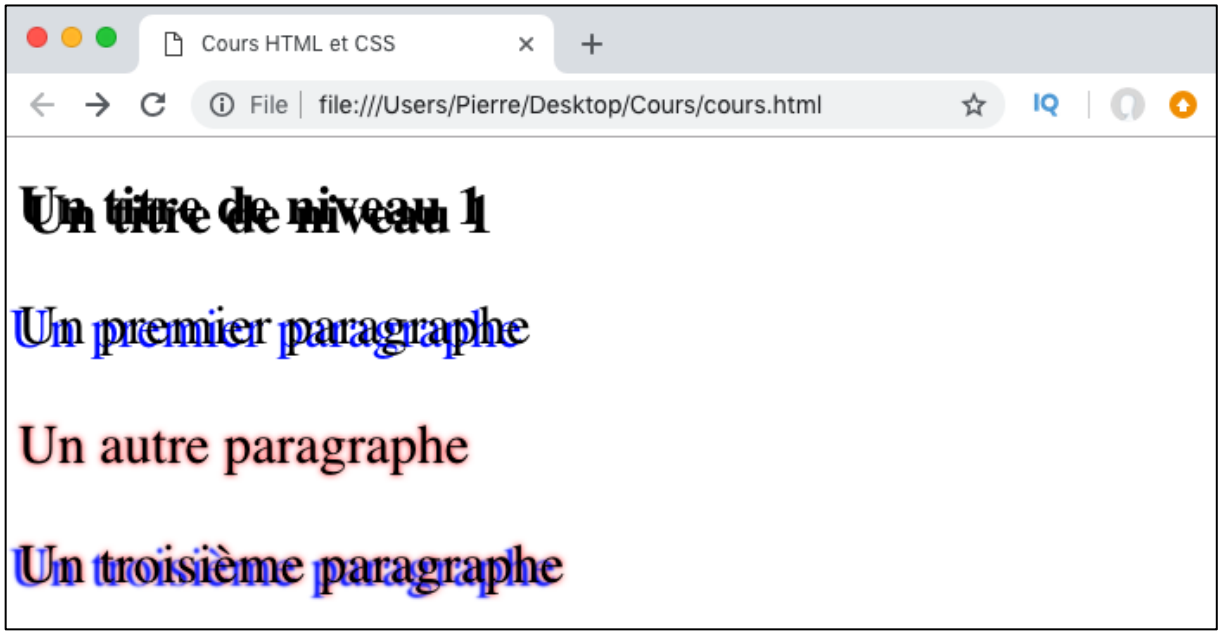
```css
h1,p{
  font-size: 2em;
}

/*Ombre noire en bas à droite, nette*/
h1{
    text-shadow: 5px 5px;
}

/*Ombre bleue en bas à gauche, nette*/
.p1{
    text-shadow: -5px 2px 1px blue;
}

/*Ombre rouge centrée sur le texte (décalement horizontal
 *et vertical de 0px), floue*/
.p2{
    text-shadow: 0px 0px 5px red;
}
```



En plus de cela, nous allons pouvoir définir plusieurs ombres différentes pour un même texte. Pour faire cela, il suffit de séparer les différentes déclarations relatives à chaque ombre par une virgule dans la propriété `text-shadow`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>
        <p class="p3">Un troisième paragraphe</p>
    </body>
</html>
```

```css
h1,p{
  font-size: 2em;
}

/*Ombre noire en bas à droite, nette*/
h1{
    text-shadow: 5px 5px;
}

/*Ombre bleue en bas à gauche, nette*/
.p1{
    text-shadow: -5px 2px blue;
}

/*Ombre rouge centrée sur le texte (décalement
 *horizontal et vertical de 0px), floue*/
.p2{
    text-shadow: 0px 0px 5px red;
}

/*Double ombre de .p1 et de .p2*/
.p3{
  text-shadow: -5px 2px 1px blue, 0px 0px 5px red;
}
```



Comme vous pouvez le voir ci-dessus, il faut faire bien attention avec cette propriété à ne pas rendre le texte illisible pour vos visiteurs ! Pour cela, nous allons pouvoir ajouter de la transparence à nos ombres en utilisant par exemple une notation RGBa pour la couleur. Nous reviendrons sur ces notations dans le chapitre de ce cours dédié à la couleur.

Gestion des interlignes et des espaces

Dans cette nouvelle leçon, nous allons étudier trois propriétés CSS qui vont nous permettre de gérer l'espace entre chaque lettre, entre chaque mot ainsi qu'entre chaque ligne de texte.

Ces propriétés sont les suivantes :

- La propriété `line-height` ;
- La propriété `letter-spacing` ;
- La propriété `word-spacing`.

La propriété line-height

La propriété `line-height` va nous permettre de définir la hauteur de chaque ligne d'un texte et donc de fait l'espace entre les lignes.

Cette propriété va pouvoir accepter en valeur un nombre simple, une unité de longueur en `px`, `em` etc. ou un pourcentage.

Les valeurs nombre simple et pourcentage vont nous permettre de définir la hauteur totale de la ligne par rapport à la taille de la police. Par exemple, indique `line-height : 2` ou `line-height : 200%` signifie que chaque ligne de notre élément aura une hauteur totale égale à deux fois la taille de la police, c'est-à-dire que l'interligne (espace entre deux lignes) sera égal à la hauteur de la police.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe <br>sur deux lignes</p>
        <p class="p2">Un autre paragraphe <br>sur deux lignes</p>
        <p class="p3">Un troisième paragraphe <br>sur deux lignes</p>
        <p class="p4">pppqqqgggjjj<br>dddfffhhhjjjkkkllbbb</p>
    </body>
</html>
```

```css
p{
  font-size: 1em;
}

/*Hauteur totale de ligne (texte + interligne) = 2 fois
 *la hauteur du texte*/
.p1{
    line-height: 2em;
}

/*Hauteur totale de ligne (texte + interligne) = 3 fois
 *la hauteur du texte*/
.p2{
    line-height: 300%;
}

/*La hauteur totale de la ligne est égale à la hauteur du
 *texte. La hauteur de l'interligne est donc nulle*/
.p3, .p4{
    line-height: 1em;
}
```

Ici, je donne un `line-height` égal à la taille de mon texte pour mon paragraphe `p3`. Chaque ligne va donc avoir exactement la même hauteur que le texte et il n'y aura pas d'espace entre les lignes.

Si on regarde attentivement, cependant, on observe qu'il reste tout de même un espace. Cela est dû à la définition même de la taille de la police. En effet, lorsqu'on définit une taille de police, on définit en fait la hauteur maximale que peut avoir un caractère.

Cependant, vous pouvez observer que tous les caractères ne font pas la même hauteur. Certains, comme le « e », le « a » ou le « r » occupent moins d'espace que le « l », le « p » ou que des caractères en majuscules par exemple. Notez que la hauteur réelle de chaque caractère va dépendre de la police utilisée.

C'est la raison pour laquelle il reste un espace visible entre les lignes lorsque les caractères ne sont pas des caractères occupant la hauteur maximale de police définie. Pour vous convaincre de cela, vous pouvez regarder le paragraphe `p4` pour lequel nous avons défini la même taille de police et la même hauteur de ligne que pour `p3` mais qui ne contient que des caractères occupant la taille maximale de police définie : il n'y a plus aucun espace visible entre les lignes dans ce cas.

La propriété letter-spacing

La propriété `letter-spacing` va nous permettre de définir l'espace entre les différents caractères de notre police.

Nous allons passer à cette propriété une valeur de type longueur en `px` ou en `em` par exemple. Les valeurs passées vont pouvoir être positives ou négatives.

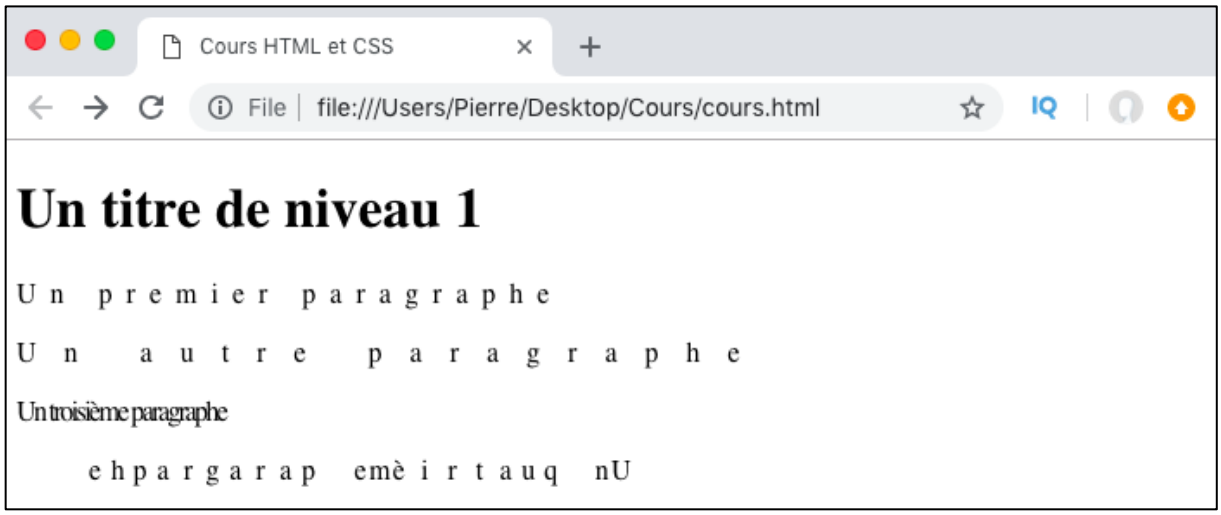
La valeur passée à `letter-spacing` va venir s'ajouter à l'espace par défaut entre les caractères. Une valeur positive va donc augmenter l'espace entre les caractères tandis qu'une valeur négative va le réduire.

Attention toutefois : cette police est dépendante de la police utilisée et certaines valeurs passées vont possiblement ne pas être acceptées par la police et ainsi être modifiées automatiquement, notamment dans le cas de valeurs réduisant l'espace entre les caractères.

Notez également qu'en passant une valeur négative plus importante que la taille de votre police le texte s'affichera « à l'envers » c'est-à-dire que les lettres suivantes vont se retrouver avant les lettres précédentes.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>
        <p class="p3">Un troisième paragraphe</p>
        <p class="p4">Un quatrième paragraphe</p>
    </body>
</html>
```

```css
p{
  font-size: 1em;
}

/*On augmente l'écart par défaut entre les caractères de 0.5em*/
.p1{
    letter-spacing: 0.5em
}

/*On augmente l'écart par défaut entre les caractères de 1em*/
.p2{
    letter-spacing: 1em
}

/*On réduit l'écart par défaut entre les caractères de 0.1em*/

.p3{
    letter-spacing: -0.1em;
}

/*On réduit l'écart par défaut entre les caractères de 1.3em*/
.p4{
    letter-spacing: -1.3em;
    text-align: center;
}
```



La propriété word-spacing

La propriété CSS word-spacing va fonctionner de manière similaire à letter-spacing mais va cette fois-ci nous permettre de définir l'espace entre les différents mots d'un texte.

Une nouvelle fois, passer une valeur positive à word-spacing va augmenter l'espace défini par défaut par la police entre deux mots tandis que passer une valeur négative va le réduire.

De manière analogue à la propriété letter-spacing, il faudra faire très attention avec l'utilisation de valeurs négatives avec word-spacing car le texte peut très vite devenir illisible si plusieurs mots commencent à se chevaucher.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>
        <p class="p3">Un troisième paragraphe</p>
        <p class="p4">Un quatrième paragraphe</p>
    </body>
</html>
```

```css
p{
  font-size: 1em;
}

/*On augmente l'écart par défaut entre les mots de 0.5em*/
.p1{
    word-spacing: 0.5em
}

/*On augmente l'écart par défaut entre les mots de 1em*/
.p2{
    word-spacing: 1em
}

/*On réduit l'écart par défaut entre les mots de 0.1em*/

.p3{
    word-spacing: -0.1em;
}

/*On réduit l'écart par défaut entre les mots de 1.3em*/
.p4{
    word-spacing: -1.3em;
    text-align: center;
}
```


Cours HTML et CSS
File | file:///Users/Pierre/Desktop/Cours/cours.html
Un titre de niveau 1
Un premier paragraphe
Un autre paragraphe
Un troisième paragraphe
Un quatrième paragraphe

Gestion de la couleur et de l'opacité des textes

Dans cette nouvelle leçon, nous allons apprendre à modifier la couleur et l'opacité de nos textes grâce à la propriété `color` que nous avons déjà rencontré précédemment.

Cette propriété est à la fois très simple à utiliser et relativement complexe à parfaitement maitriser car nous allons lui passer des valeurs de couleurs très différentes les unes des autres.

La propriété `color` va en effet pouvoir accepter des valeurs comme :

- Un nom de couleur (en anglais) ;
- Une notation hexadécimale ;
- Une notation RGB ou RGBa ;
- Une notation HSL ou HSLa.

Toutes ces notations vont nous permettre, in fine, d'attribuer une couleur particulière à notre texte. Le but de cette leçon est de comprendre comment chaque type de valeur fonctionne et les avantages et inconvénients de chacun.

Les valeurs de type « nom de couleur »

Les 16 premières couleurs normalisées

Il y a des années de cela, les langages de programmation ne disposaient pas de toutes les fonctionnalités d'aujourd'hui tout simplement car les infrastructures étaient beaucoup moins puissantes que de nos jours.

Ainsi, au départ, seules 16 couleurs ont été normalisées en CSS. C'était déjà un grand pas en avant pour les utilisateurs : ils n'avaient plus qu'à passer le nom (en anglais) de la couleur normalisée en valeur de la propriété CSS `color` afin de changer la couleur d'un élément.

Ces seize couleurs CSS sont les suivantes. Notez que j'ai déjà renseigné l'équivalent de chaque nom de couleur en notation hexadécimale dans le tableau ci-dessous. Nous reparlerons de ces notations plus tard dans cette leçon.

Nom de la couleur	Hexadécimal	Couleur
Aqua	#00FFFF	
Black	#000000	
Blue	#0000FF	
Fuschia	#FF00FF	

Gray	#808080	
Green	#008800	
Lime	#00FF00	
Maroon	#800000	
Navy	#000080	
Olive	#808000	
Purple	#800080	
Red	#FF0000	
Silver	#C0C0C0	
Teal	#008080	
White	#FFFFFF	
Yellow	#00FFFF	

Voyons immédiatement en pratique comment utiliser ces noms de couleurs en CSS avec différents exemples utilisant `color` en CSS :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="rouge">Un premier paragraphe</p>
        <p class="bleu">Un autre paragraphe</p>
    </body>
</html>
```

```css
h1{
    color: orange;
}

.rouge{
    color: red;
}

.bleu{
    color: blue;
}
```

Dans l'exemple ci-dessus, vous pouvez voir que nous attribuons en CSS une couleur orange à notre titre de niveau 1 avec le code `h1{color : orange}`. Nous définissons également des couleurs rouge (red) et bleu (blue) pour les textes de nos éléments `p1` et `p2`.

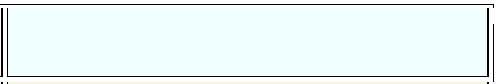
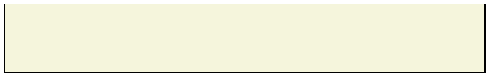
Les autres couleurs nommées

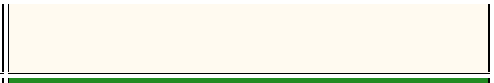
Avec l'évolution des performances et des langages, le support pour de nouvelles couleurs a progressivement été ajouté.

Ainsi, aujourd'hui, les navigateurs reconnaissent et supportent l'utilisation de plus de 140 noms de couleurs différents. Nous allons donc pouvoir passer chacune de ces valeurs à la propriété `color` pour définir une nouvelle couleur pour un texte.

Voici la liste de ces couleurs CSS (noms en anglais) ainsi que leur code hexadécimal :

Nom de la couleur	Hexadécimal	Couleur
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFD4	

Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	
Chartreuse	#7FFF00	
Chocolate	#D2691E	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGray	#A9A9A9	
DarkGrey	#A9A9A9	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	

DarkOliveGreen	#556B2F	
DarkOrange	#FF8C00	
DarkOrchid	#9932CC	
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSeaGreen	#8FBC8F	
DarkSlateBlue	#483D8B	
DarkSlateGray	#2F4F4F	
DarkSlateGrey	#2F4F4F	
DarkTurquoise	#00CED1	
DarkViolet	#9400D3	
DeepPink	#FF1493	
DeepSkyBlue	#00BFFF	
DimGray	#696969	
DimGrey	#696969	
DodgerBlue	#1E90FF	
FireBrick	#B22222	
FloralWhite	#FFFAF0	
ForestGreen	#228B22	
Fuchsia	#FF00FF	
Gainsboro	#DCDCDC	
GhostWhite	#F8F8FF	
Gold	#FFD700	
GoldenRod	#DAA520	
Gray	#808080	

Grey	#808080	
Green	#008000	
GreenYellow	#ADFF2F	
HoneyDew	#F0FFF0	
HotPink	#FF69B4	
IndianRed	#CD5C5C	
Indigo	#4B0082	
Ivory	#FFFFF0	
Khaki	#F0E68C	
Lavender	#E6E6FA	
LavenderBlush	#FFF0F5	
LawnGreen	#7CFC00	
LemonChiffon	#FFFACD	
LightBlue	#ADD8E6	
LightCoral	#F08080	
LightCyan	#E0FFFF	
LightGoldenRodYellow	#FAFAD2	
LightGray	#D3D3D3	
LightGrey	#D3D3D3	
LightGreen	#90EE90	
LightPink	#FFB6C1	
LightSalmon	#FFA07A	
LightSeaGreen	#20B2AA	
LightSkyBlue	#87CEFA	
LightSlateGray	#778899	

LightSlateGrey	#778899	
LightSteelBlue	#B0C4DE	
LightYellow	#FFFFE0	
Lime	#00FF00	
LimeGreen	#32CD32	
Linen	#FAF0E6	
Magenta	#FF00FF	
Maroon	#800000	
MediumAquaMarine	#66CDAA	
MediumBlue	#0000CD	
MediumOrchid	#BA55D3	
MediumPurple	#9370DB	
MediumSeaGreen	#3CB371	
MediumSlateBlue	#7B68EE>	
MediumSpringGreen	#00FA9A	
MediumTurquoise	#48D1CC<	
MediumVioletRed	#C71585	
MidnightBlue	#191970	
MintCream	#F5FFFA	
MistyRose	#FFE4E1	
Moccasin	#FFE4B5	
NavajoWhite	#FFDEAD	
Navy	#000080	
OldLace	#FDF5E6	
Olive	#808000	

OliveDrab	#6B8E23	
Orange	#FFA500	
OrangeRed	#FF4500	
Orchid	#DA70D6	
PaleGoldenRod	#EEE8AA<	
PaleGreen	#98FB98	
PaleTurquoise	#AFEEEE	
PaleVioletRed	#DB7093	
PapayaWhip	#FFEFD5	
PeachPuff	#FFDAB9	
Peru	#CD853F	
Pink	#FFC0CB	
Plum	#DDA0DD	
PowderBlue	#B0E0E6	
Purple	#800080	
RebeccaPurple	#663399	
Red	#FF0000	
RosyBrown	#BC8F8F	
RoyalBlue	#4169E1	
SaddleBrown	#8B4513	
Salmon	#FA8072	
SandyBrown	#F4A460	
SeaGreen	#2E8B57	
SeaShell	#FFF5EE	
Sienna	#A0522D	

Silver	#C0C0C0	
SkyBlue	#87CEEB	
SlateBlue	#6A5ACD	
SlateGray	#708090	
SlateGrey	#708090	
Snow	#FFFAFA	
SpringGreen	#00FF7F	
SteelBlue	#4682B4	
Tan	#D2B48C	
Teal	#008080	
Thistle	#D8BFD8	
Tomato	#FF6347	
Turquoise	#40E0D0	
Violet	#EE82EE	
Wheat	#F5DEB3	
White	#FFFFFF	
WhiteSmoke	#F5F5F5	
Yellow	#FFFF00	
YellowGreen	#9ACD32	

Fait intéressant ici, vous pouvez noter que :

- Fuchsia possède le même code couleur que Magenta ;
- Aqua possède le même code couleur que Cyan.

Utilisons immédiatement ces noms de couleurs CSS avec la propriété `color` à travers de nouveaux exemples :

Avantages et limitations des valeurs de type « nom de couleur »

L'utilisation des valeurs de type « nom de couleur » avec la propriété CSS `color` est très pratique puisqu'il suffit d'indiquer le nom de la couleur souhaitée.

Cependant, ce type de valeurs possède une limitation majeure : nous sommes limités en termes de couleurs à ces quelques 140 noms. Or, parfois, nous voudrons utiliser une couleur ou une variation de couleur différente de ces 140 disponibles pour définir une identité visuelle précise.

Dans ces cas-là, nous utiliserons alors plutôt l'un des autres types de notation, que ce soit des notations RGB, HEX, ou HSL.

En effet, chacun de ces nouveaux types de valeurs va nous permettre de créer et d'utiliser jusqu'à 16 millions de variations de couleurs afin de trouver la couleur exacte voulue en CSS. Ils vont tous reposer sur une logique similaire de mélange des couleurs rouge, vert et bleu.

Les notations de type RGB

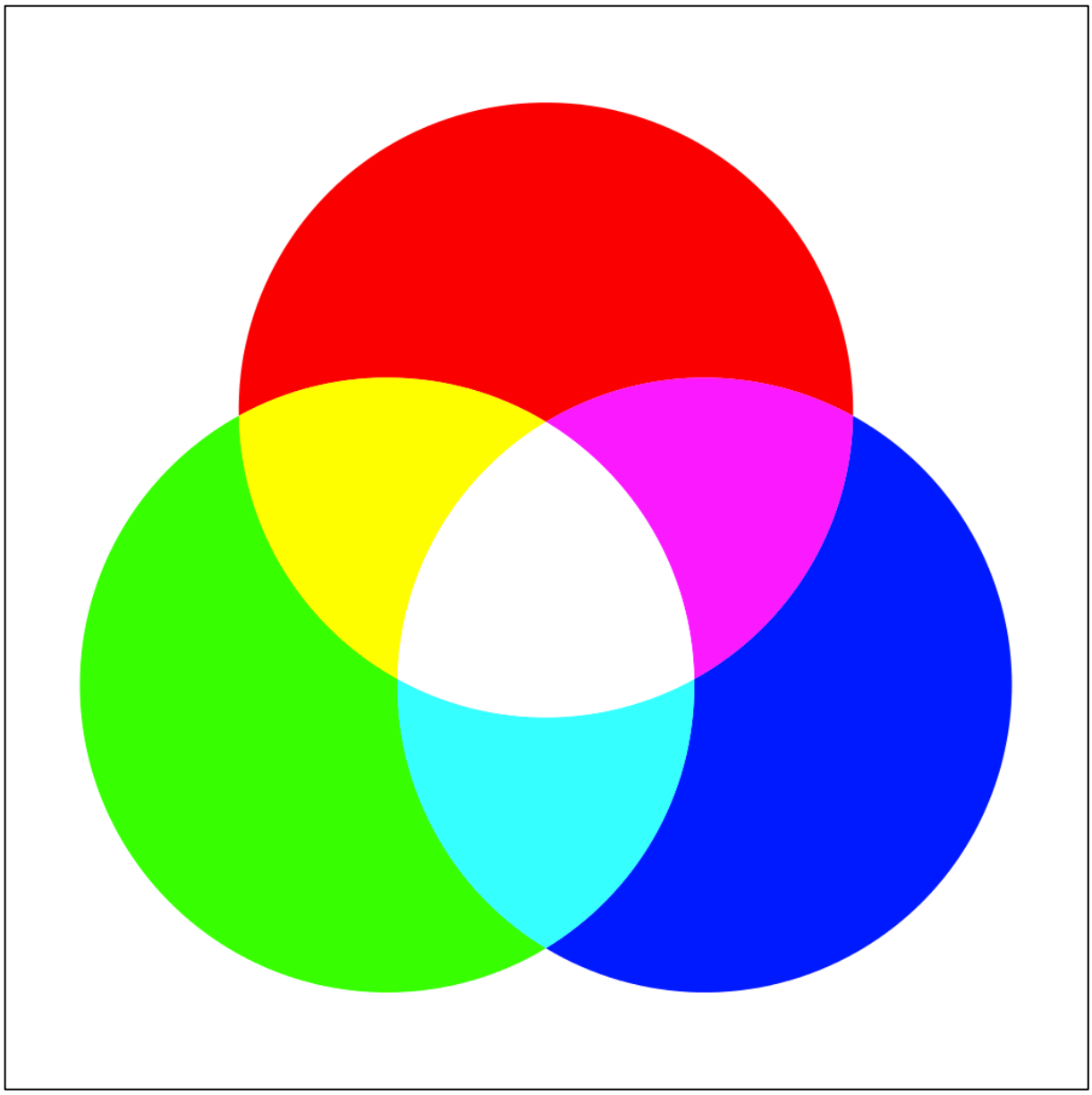
Commençons déjà par expliquer comment fonctionnent les valeurs de type RGB et comment les utiliser avec `color` en CSS.

Avant tout, vous devez savoir que les lettres « RGB » sont les abréviations de « Red Green Blue » soit « Rouge Vert Bleu » en français. Effectivement, pour créer une couleur RGB, nous allons devoir préciser trois niveaux d'intensité de Rouge, de Vert, et de Bleu qui vont ensuite être mélangés pour créer la couleur finale.

Chaque niveau d'intensité qu'on va renseigner va être compris entre 0 (intensité nulle ou absence de la couleur en question) et 255 (intensité maximale ou couleur pure). En précisant une intensité de rouge de 0, par exemple, on signifie qu'on ne souhaite pas utiliser de rouge dans notre couleur finale. En précisant une intensité de 255 de rouge, en revanche, on indique qu'on souhaite utiliser beaucoup de rouge pour créer notre couleur finale.

Vous vous rappelez à l'école quand vous mélangiez différentes couleurs en peinture entre elles pour en créer une nouvelle ? Cela fonctionne également de la même manière !

Pour rappel, je vous donne une image avec les couleurs obtenues lorsqu'on mélange en quantité équivalente nos trois couleurs de base. Cela vous aidera pour comprendre la suite.



Illustrons immédiatement avec quelques exemples le fonctionnement des notations RGB en CSS.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Les notations RGB en CSS</h1>

        <p>On va pouvoir <strong>créer une couleur RGB en CSS</strong>
        en mixant différentes intensités de Rouge, de Vert, et de Bleu.</p>

        <p><em>L'intensité</em> de chacune de ces couleurs de base doit être
        comprise entre <em>0 (intensité minimale)</em> et
        <em>255 (intensité maximale)</em>.</p>
    </body>
</html>
```

```css
h1{
  color : RGB(255, 180, 0);
}

p{
  color : RGB(50, 50, 50);
}

strong{
  color : RGB(0, 0, 200);
}

em{
  color : RGB(0, 200, 50);
}
```

Regardons ces exemples de plus près, en commençant déjà par nous intéresser à la syntaxe des notations RGB en CSS. Comme vous pouvez le voir, lorsqu'on renseigne une valeur de type RGB en valeur de la propriété CSS `color`, il faut le déclarer en écrivant `RGB()`.

Ensuite, vous pouvez observer dans chaque cas 3 chiffres séparés par des virgules à l'intérieur des parenthèses. Ces trois chiffres correspondent respectivement aux niveaux d'intensité de Rouge, de Vert et de Bleu qui seront utilisés pour créer notre couleur finale.

A ce niveau, vous pouvez noter que plus un niveau d'intensité va de rapprocher de 0, plus cela va correspondre à une couleur foncée. A l'inverse, plus un niveau d'intensité va se rapprocher de 255, plus la couleur va être claire. Pour retenir cela, retenez que `color : RGB(0, 0, 0)` correspond à du noir tandis que `color : RGB(255, 255, 255)` correspond à du blanc en CSS.

Intéressons-nous maintenant de plus près aux différents cas ci-dessus. On commence avec le code CSS `h1{color : RGB(255, 180, 0);}`. Vous remarquez que ce code me donne une variation d'orange en résultat. Essayons de déterminer pourquoi. Pour créer ce orange, j'ai précisé une intensité maximale de rouge (255) que j'ai mélangé avec une intensité forte de vert (180) et une intensité nulle de bleu (0).

Vous devriez normalement savoir que lorsqu'on mélange du rouge et du vert en quantité égale, on obtient du jaune. Ici, nous avons mélangé le maximum de rouge avec beaucoup de vert (mais en mettant moins de vert que de rouge). Notre couleur finale va donc se trouver entre du jaune et du rouge… C'est-à-dire de l'orange !

Pour nos paragraphes, nous précisons la même intensité de rouge, de vert et de bleu et nos trois intensités sont relativement basses. Je vous ai dit plus haut que `RGB(0, 0, 0)` donnait du noir tandis que `RGB(255, 255, 255)` donnait du blanc. Nous allons donc ici obtenir une sorte de gris foncé.

Essayez de comprendre par vous-même les couleurs obtenues pour les textes contenus dans nos éléments `strong` et `em`, ça vous fera un bon exercice.

Bon à savoir : Vous vous rappelez lorsque je vous ai dit que le type de valeurs RGB permettait de créer plus de 16 millions de couleurs en CSS ? Comprenez-vous maintenant d'où vient ce chiffre ?

Explication : nous pouvons choisir parmi 256 niveaux d'intensité de rouge, vert et bleu pour créer notre couleur finale. Oui, j'ai bien dit 256 et pas 255 car le 0 compte comme un niveau d'intensité : l'intensité nulle. Ainsi, on va pouvoir créer 256 * 256 * 256 = 16 777 216 couleurs différentes en utilisant les notations RGB avec `color` en CSS ! Les valeurs hexadécimales vont fonctionner sur le même principe.

Les valeurs de type hexadécimales

Les valeurs de type hexadécimale vont reposer sur le même principe que les valeurs de type RGB : nous allons à nouveau pouvoir préciser trois niveaux d'intensité de rouge, de vert et de bleu pour créer une couleur personnalisée.

La seule différence entre ces deux types de notation va être la façon dont on va compter : le mot « hexadécimal » signifie « qui fonctionne en base 16 ». Cela veut dire que le système hexadécimal utilise 16 symboles différents pour compter et représenter les chiffres.

Dans la vie de tous les jours, nous utilisons le système décimal : nous comptons en base 10. Cela signifie que nous utilisons 10 symboles pour compter : ce sont le 0, le 1, le 2, 3, 4, 5, 6, 7, 8, et le 9.

Le système hexadécimal, comme je vous l'ai dit, va utiliser non pas 10 mais 16 symboles pour compter. Ces symboles vont être : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F. Comme vous pouvez le remarquer, nous comptons de la même manière en hexadécimal qu'en décimal jusqu'à 9 puis nous utilisons les lettres A B C D E F qui vont être l'équivalent des unités « 10 », « 11, », « 12 », « 13 », « 14 » et « 15 » en système décimal.

Comment représente-t-on le 16 en hexadécimal me direz-vous ? Nous allons utiliser le même principe qu'avec le système décimal, excepté que cette fois-ci nous utilisons des seizaines et non pas des dizaines. Ainsi, pour représenter le « 16 » en hexadécimal, nous utiliserons le symbole « 10 » (comprendre 1 seizaine + 0 unité).

Regardez plutôt le tableau ci-dessous pour mieux comprendre :

Décimal	**Équivalent hexadécimal**
0 (ou 00)	0 (ou 00)
1 (ou 01)	1 (ou 01)
2 (ou 02)	2 (ou 02)
9 (ou 09)	9 (ou 09)
10	A (ou 0A)
11	B (ou 0B)
16	10
17	11
25	19
26	1A
32	20

Au final, le système hexadécimal n'est qu'une façon différente de compter en utilisant 16 unités de base et non pas 10 comme on en a l'habitude. Cela peut désorienter au premier abord mais c'est en fait très simple. Faites l'effort de comprendre cela pour bien comprendre l'utilisation des valeurs hexadécimales avec les couleurs en CSS !

Nous allons en effet pouvoir utiliser les notations hexadécimales pour créer une couleur avec la propriété CSS color. Les valeurs hexadécimales sont même les plus utilisées en CSS pour créer des couleurs !

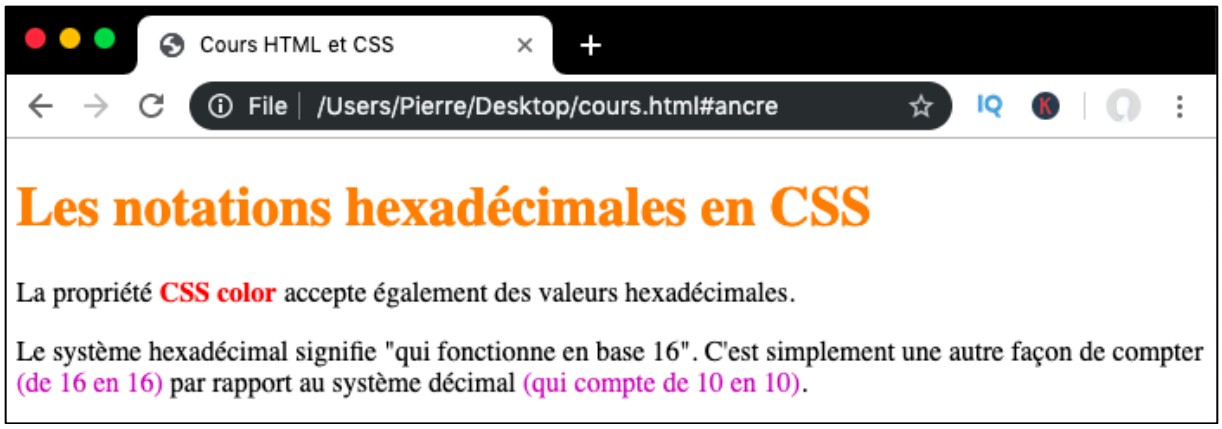
Nous allons ici utiliser exactement le même principe qu'avec les valeurs RGB en précisant trois intensités de Rouge, Vert et Bleu pour créer une couleur. Là encore, chacune des trois intensités des couleurs de base va pouvoir être comprise entre 0 et 255 ou plus exactement entre 00 et FF en notation hexadécimale (FF en hexadécimal est équivalent à 255 en décimal).

Pour créer une couleur en utilisant les notations hexadécimales en CSS, nous allons donc devoir préciser trois intensités de Rouge, Vert et de Bleu entre 00 et FF à la suite. De plus, nous devrons faire précéder cette notation par un dièse (symbole #).

Prenons immédiatement quelques exemples pour illustrer cela :

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Les notations hexadécimales en CSS</h1>

        <p>La propriété <strong>CSS color</strong> accepte également des
        valeurs hexadécimales.</p>

        <p>Le système hexadécimal signifie "qui fonctionne en base 16".
        C'est simplement une autre façon de compter <span>(de 16 en 16)</span>
        par rapport au système décimal <span>(qui compte de 10 en 10)</span>.</p>
    </body>
</html>
```

```css
/*Intensité max de rouge, médiane de vert, et pas de bleu*/
h1{
  color : #FF8800;
}

/* Rouge pur : intensité max de rouge, pas de vert ni de bleu*/
strong{
  color : #FF0000;
}

/*Grande intensité de rouge et de bleu, petite intensité de vert*/
span{
  color : #B904B9;
}
```



On applique une `color : #FF8800` à notre titre `h1` en CSS. On utilise ici une intensité maximale de rouge (FF = 255 en décimal), moyenne de vert (88 = 136 en décimal) et minimale (00) de bleu pour créer notre couleur finale. Comme on mélange beaucoup de rouge avec un peu de vert, on obtient naturellement de l'orange.

Notez ici quelque chose d'intéressant : lorsqu'on souhaite utiliser deux fois la même unité pour préciser une intensité (par exemple « 88 », « BB » ou « EE », on peut utiliser une notation raccourcie en se contentant de ne préciser l'unité qu'une seule fois.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Couleur et opacité des textes</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>
    </body>
</html>
```

```css
h1,p{
  font-size: 1.5em;
  font-weight: normal;
}

/*Intensité max de rouge, médiane de vert, et pas de bleu*/
h1{
  color : RGB(255,136,000);
}

/*Equivalent en hexadécimal (version longue)*/
.p1{
  color : #FF8800;
}
/*Equivalent en hexadécimal (version courte)*/
.p2{
  color : #F80;
}
```

Ici, vous pouvez remarquer qu'il est strictement équivalent d'écrire `color : #FF8800` et `color : #F80`.

Notez par ailleurs qu'il est strictement équivalent d'utiliser des majuscules ou des minuscules pour les lettres des valeurs hexadécimales.

Les notations de type HSL

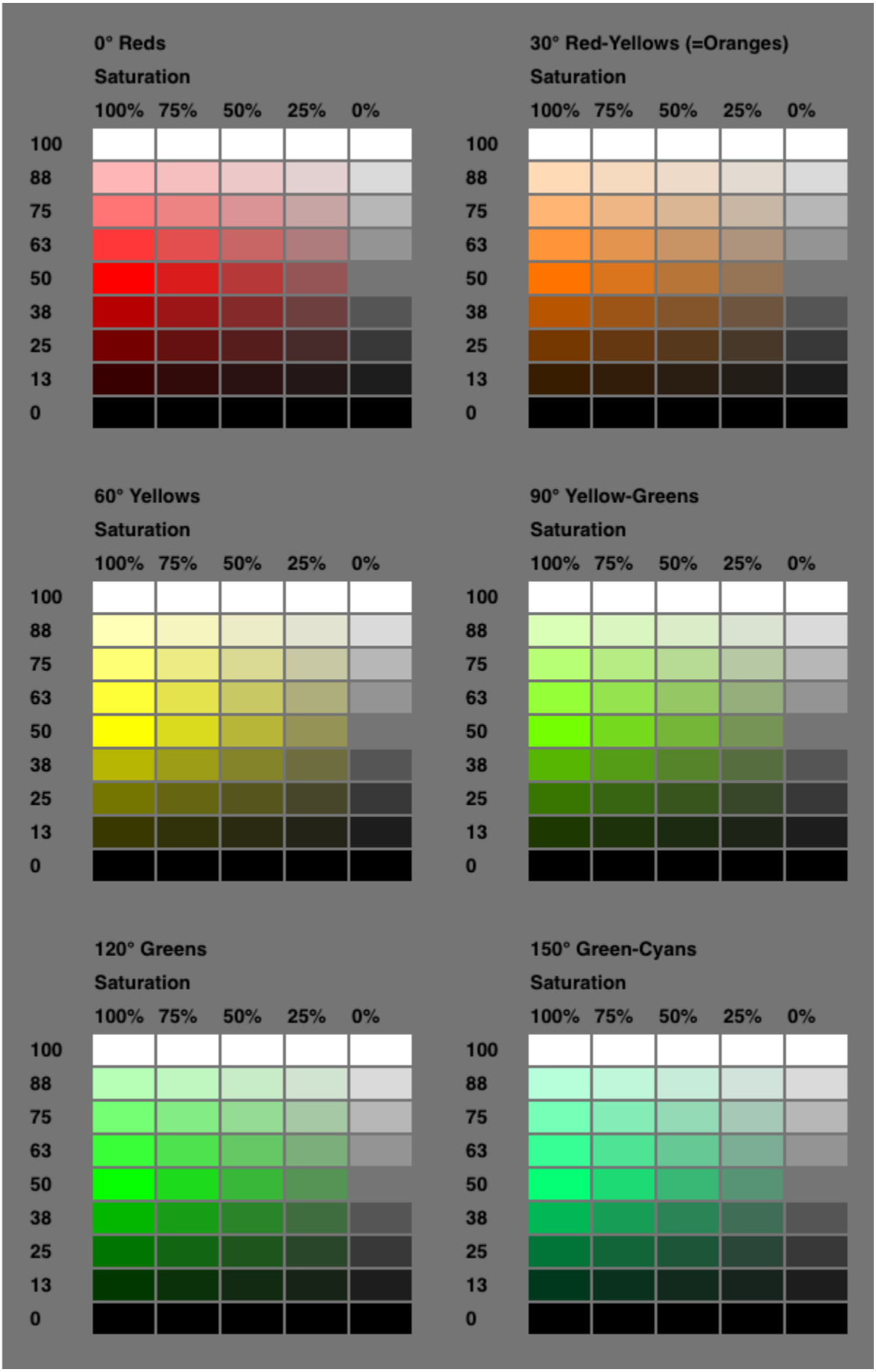
« HSL » est l'abréviation de « Hue-Saturation-Lightning », c'est-à-dire « teinte-saturation-luminosité » en français.

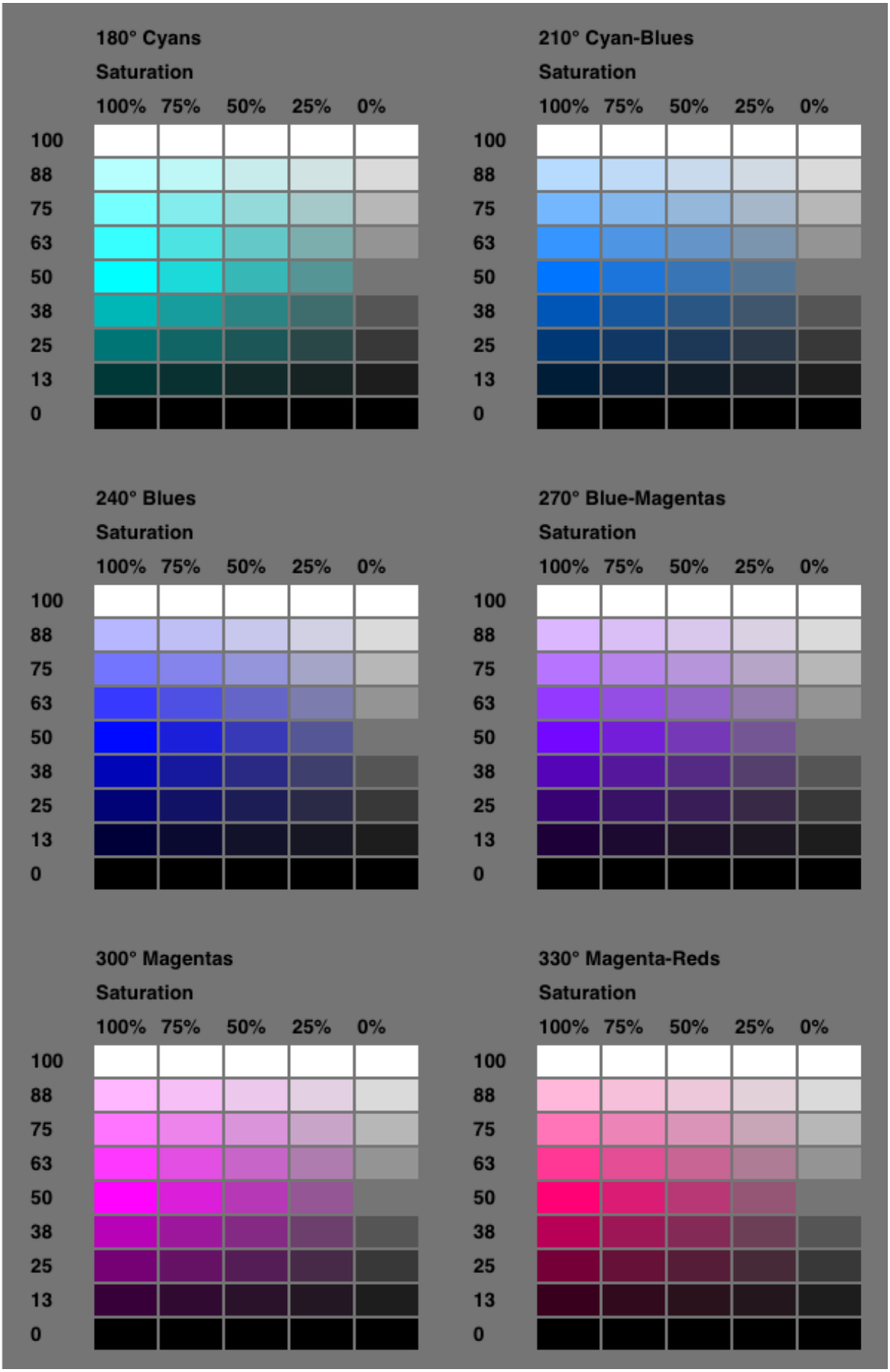
Pour créer une couleur en utilisant les notations HSL, nous allons devoir renseigner trois valeurs :

- La teinte. On va devoir ici renseigner un nombre entre 0 et 360 qui représente un angle du cercle chromatique (c'est-à-dire l'arc en ciel représenté dans un cercle). Ici, vous pouvez retenir que la couleur rouge correspond à un angle de 0 (degré) ou de 360 (degrés), tandis que le vert va se situer à 120 degrés et le bleu à 240 degrés. Nous allons pouvoir préciser des valeurs décimales ici pour choisir précisément une couleur ;

- La saturation. Celle-ci va être représentée sous forme de pourcentage. 100% correspond à une saturation maximale tandis que 0% correspond à une saturation minimale ;
- La luminosité. Celle-ci va également être représentée par un pourcentage. 100% de luminosité correspond à du blanc tandis que 0% correspond à du noir.

Vous pouvez vous aider des repères suivants fournis par le W3C pour bien comprendre comment fonctionnent les couleurs HSL. Ici, le pourcentage de saturation est indiqué sur l'axe horizontal tandis que le pourcentage de luminosité est indiqué sur l'axe vertical :


0° Reds
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0
30° Red-Yellows (=Oranges)
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0
60° Yellows
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0
90° Yellow-Greens
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0
120° Greens
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0
150° Green-Cyans
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0


180° Cyans
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0
210° Cyan-Blues
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0
240° Blues
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0
270° Blue-Magentas
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0
300° Magentas
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0
330° Magenta-Reds
Saturation
100% 75% 50% 25% 0%
100
88
75
63
50
38
25
13
0

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Couleur et opacité des textes</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>
    </body>
</html>
```

```css
/*Teinte = 45° (à mi-chemin entre le orange et le jaune),
 *Saturation à 100% et luminosité à 60%*/
h1{
  color: HSL(45, 100%, 60%);
}

/*Teinte à 150° (dans les vert), saturation à 50% et luminosité à 50%*/
.p1{
  color: HSL(150, 50%, 50%);
}

/*Teinte à 300° (dans les rose), saturation à 50% et luminosité à 50%*/
.p2{
  color: HSL(300, 50%, 50%);
}
```

L'opacité des éléments et des textes en CSS

Jusqu'à présent, nous n'avons pas parlé d'opacité des couleurs en CSS. Nos couleurs étaient donc par défaut complètement opaques. Le CSS nous permet cependant de préciser un niveau d'opacité (ou de transparence, comme vous préférez) pour nos différents textes ou pour nos éléments de différentes façons.

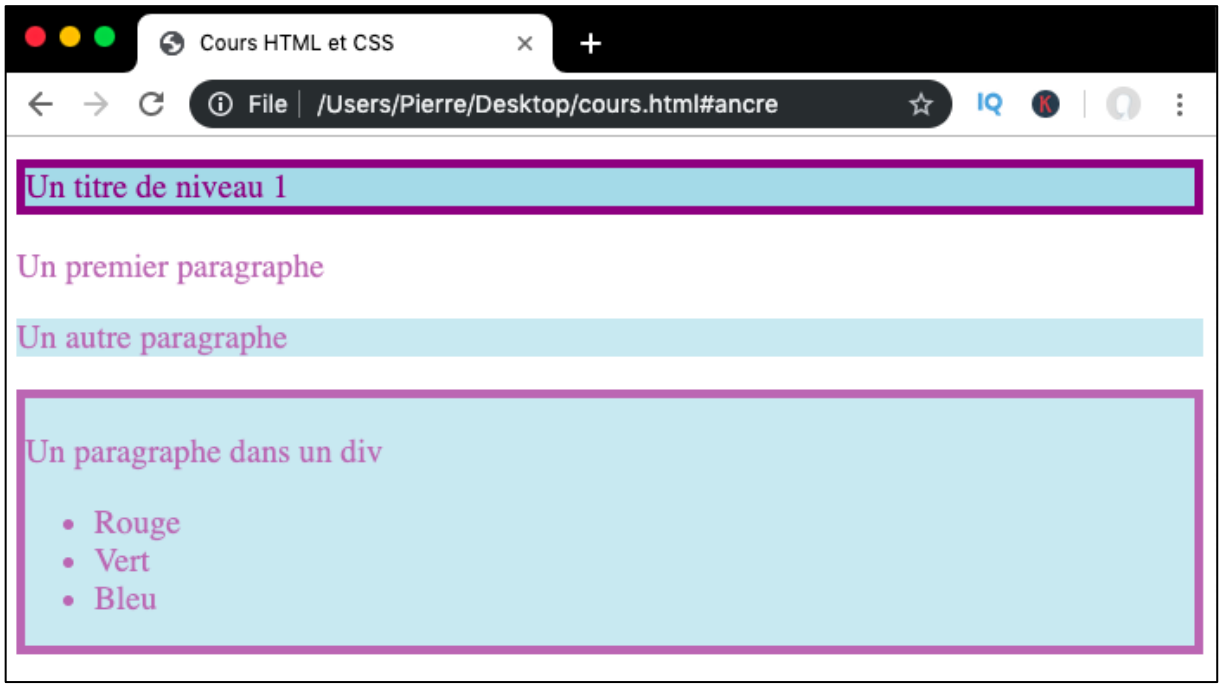
Gérer l’opacité des éléments avec la propriété opacity

Tout d’abord, on va pouvoir rendre un élément HTML plus ou moins transparent grâce à la propriété CSS `opacity`.

Cette propriété va accepter une valeur comprise entre 0 et 1 et qui va déterminer le niveau d’opacité d’un élément : la valeur 0 va rendre l’élément totalement transparent tandis que la valeur 1 (valeur par défaut) le rend totalement opaque.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>

        <div class="conteneur">
          <p>Un paragraphe dans un div</p>
          <ul>
            <li>Rouge</li>
            <li>Vert</li>
            <li>Bleu</li>
          </ul>
        </div>
    </body>
</html>
```

```css
.conteneur{
  background-color: lightBlue;
  color: purple;
  opacity: 0.6;
  border: 5px solid purple;
}

h1, p, li{
  font-size: 1.2em;
  font-weight: normal;
}

h1{
  background-color: lightBlue;
  color: purple;
  border: 5px solid purple;
}

.p1{
  color: purple;
  opacity: 0.6;
}

.p2{
  background-color: lightBlue;
  color: purple;
  opacity: 0.6;
}
```



Ici, j'attire cependant votre attention sur un point important : la propriété `opacity` ne va pas gérer la transparence d'un texte mais bien définir le niveau d'opacité d'un élément en soi.

Ainsi, si l'élément possède une couleur de fond ou des bordures, celles-ci vont également être impactées par `opacity` et vont donc possiblement être semi transparentes. De même, en appliquant cette propriété à un élément conteneur `div`, nous allons rendre le `div` en soi et tous les éléments qu'il contient semi-transparents (sauf si une règle contraire est précisée pour chaque élément du `div` bien évidemment).

Ce comportement va parfois être le comportement voulu mais ce n'est pas celui attendu dans le cas où l'on souhaite simplement gérer l'opacité des textes (et uniquement des textes) de nos éléments.

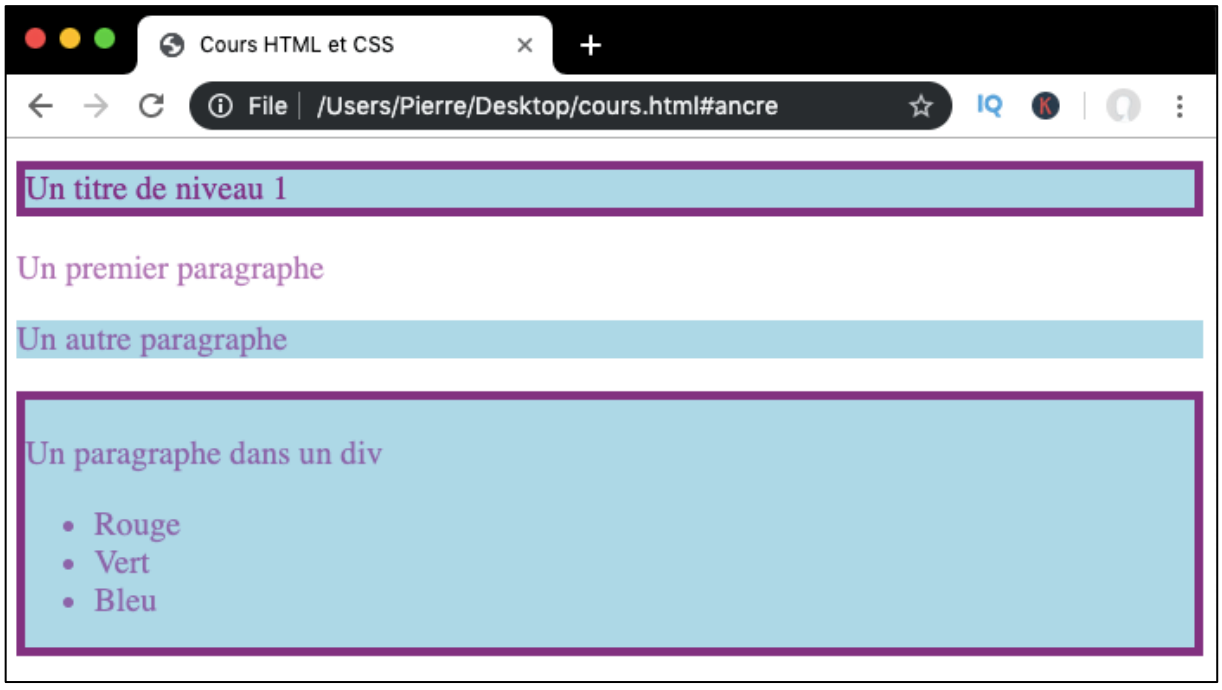
Pour faire cela, nous allons plutôt devoir utiliser des variantes des notations RGB et HSL : les notations RGBa et HSLa.

Gérer l'opacité des textes avec les notations RGBa ou HSLa

Les notations RGBa et HSLa vont accepter une valeur de plus qui va correspondre au niveau d'opacité de la couleur (le « a » est l'abréviation de « alpha channel »).
Nous allons ici à nouveau devoir préciser un chiffre compris entre 0 et 1 et indiquant le niveau d'opacité de nos textes (0 = texte transparent / 1 = texte opaque).

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>

        <div class="conteneur">
          <p>Un paragraphe dans un div</p>
          <ul>
            <li>Rouge</li>
            <li>Vert</li>
            <li>Bleu</li>
          </ul>
        </div>
    </body>
</html>
```

```css
.conteneur{
  background-color: lightBlue;
  color: HSLa(300, 100%, 25%, 60%);
  border: 5px solid purple;
}

h1, p, li{
  font-size: 1.2em;
  font-weight: normal;
}

h1{
  background-color: lightBlue;
  color: RGB(128,0,128);/*Equivalent à HSL(300, 100%, 25%)*/
  border: 5px solid RGBa(128,0,128);
}

.p1{
  color: RGBa(128,0,128,0.6)
}

.p2{
  background-color: lightBlue;
  color: RGBa(128,0,128, 0.6);
}
```



Comme vous pouvez le voir, cette fois-ci seul le niveau d'opacité de nos textes change.

Notez que les notations hexadécimales supportent également la notion de transparence. Pour indiquer un niveau d'opacité, nous allons devoir ici préciser un quatrième jeu de caractères compris entre 00 (couleur totalement transparente) et FF (couleur totalement opaque).

Cependant, je vous déconseille d'utiliser les notations hexadécimales de cette manière pour le moment car la gestion de la transparence pour ce type de valeur n'est pas encore une recommandation officielle et donc le support par les différents navigateurs n'est pas forcément assuré. Utiliser plutôt les notations RGBa ou HSLa si vous souhaitez définir des couleurs avec un niveau de transparence.

PARTIE V

Le modèle des boites

Le modèle des boites

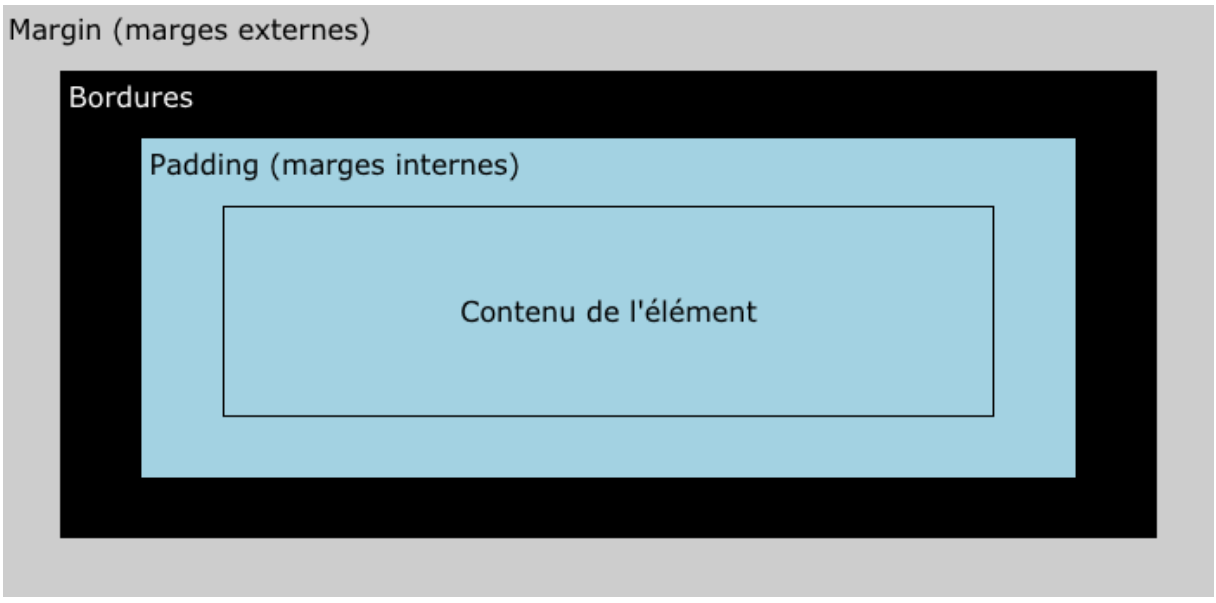
Dans cette nouvelle leçon, nous allons présenter le fameux « modèle des boites » CSS. Comprendre le concept de « boites » va être essentiel construire des designs de pages efficaces puisque cela va nous permettre d'appréhender la façon dont vont être calculées les dimensions de chaque élément.

Le concept de « boites » et le modèle des boites

L'idée centrale du modèle des boites est que tout élément HTML peut être représenté par un empilement de différentes boites rectangulaires :

- La première boite, centrale, va être composée du contenu de l'élément en soi ;
- La deuxième boite va être composée de la première boite ainsi que des marges internes de l'élément ;
- La troisième boîte va être composée de la deuxième boite et des bordures de l'élément ;
- La quatrième boite va être composée de la troisième boite et des marges externes de l'élément.

Voici la représentation d'un élément selon le modèle des boites :



Les propriétés CSS liées aux différentes boites

Le CSS va déjà nous fournir différentes propriétés qui vont nous permettre de spécifier la taille des différents éléments composants les différentes boites :

- Les propriétés `width` et `height` vont nous permettre de définir la largeur et la hauteur de la boite « contenu » ;
- La propriété `padding` va nous permettre de définir la taille des marges internes ;

- La propriété `border` va nous permettre de définir des bordures pour notre élément ;
- La propriété `margin` va nous permettre de définir la taille des marges externes.

Nous allons dans cette partie commencer avec l'étude de ces différentes propriétés qui sont fondamentales. Nous parlerons également de la propriété `box-sizing` qui va nous permettre de changer la façon dont la largeur et la hauteur d'un élément vont être calculées et donc de modifier le modèle des boites par défaut.

Le modèle des boites et le positionnement

Comprendre comment est calculée la taille de chaque élément et de quoi chaque élément est composé est essentiel pour créer des mises en page efficaces.

Pour choisir le type d'affichage et de positionnement des éléments HTML, le CSS va nous fournir des propriétés très puissantes qui vont nous permettre de modifier le flux normal de la page, c'est-à-dire de modifier l'ordre d'affichage des éléments ou la place réservée par défaut à chacun d'entre eux.

Ici, nous allons nous intéresser aux propriétés suivantes :

- La propriété `display` qui va nous permettre de définir un type d'affichage pour un élément ;
- La propriété `position` qui va nous permettre de positionner nos éléments de différentes façons dans une page ;
- La propriété `float` qui va nous permettre de faire « flotter » des éléments HTML dans la page.

Largeur (width) et hauteur (height) de la boite de contenu des éléments HTML

Tout contenu d'un élément HTML va prendre un certain espace dans une page, c'est-à-dire posséder une largeur et une hauteur. Cet espace pris, c'est-à-dire la largeur et la hauteur de ce contenu vont être représentées respectivement par les propriétés CSS `width` (largeur) et `height` (hauteur).

Dans cette leçon, nous allons étudier ces deux propriétés et apprendre à les manipuler.

L'impact du type d'affichage d'un élément sur ses dimensions

Pour comprendre comment fonctionnent les propriétés `width` et `height` et comment les manipuler il est avant tout nécessaire d'avoir une vue claire sur les types d'affichage principaux des éléments : l'affichage `block` (sous forme de bloc) et `inline` (en ligne).

En effet, je vous rappelle que les éléments HTML peuvent être affichées de deux grandes façons différentes : soit sous forme de bloc, soit en ligne.

Les dimensions par défaut du contenu des éléments HTML vont avant tout être déterminées par le type d'affichage des éléments : en effet, les éléments de type `block` occuperont par défaut toute la largeur disponible dans leur élément parent tandis que les éléments de type `inline` n'occuperont que la largeur nécessaire à leur contenu.

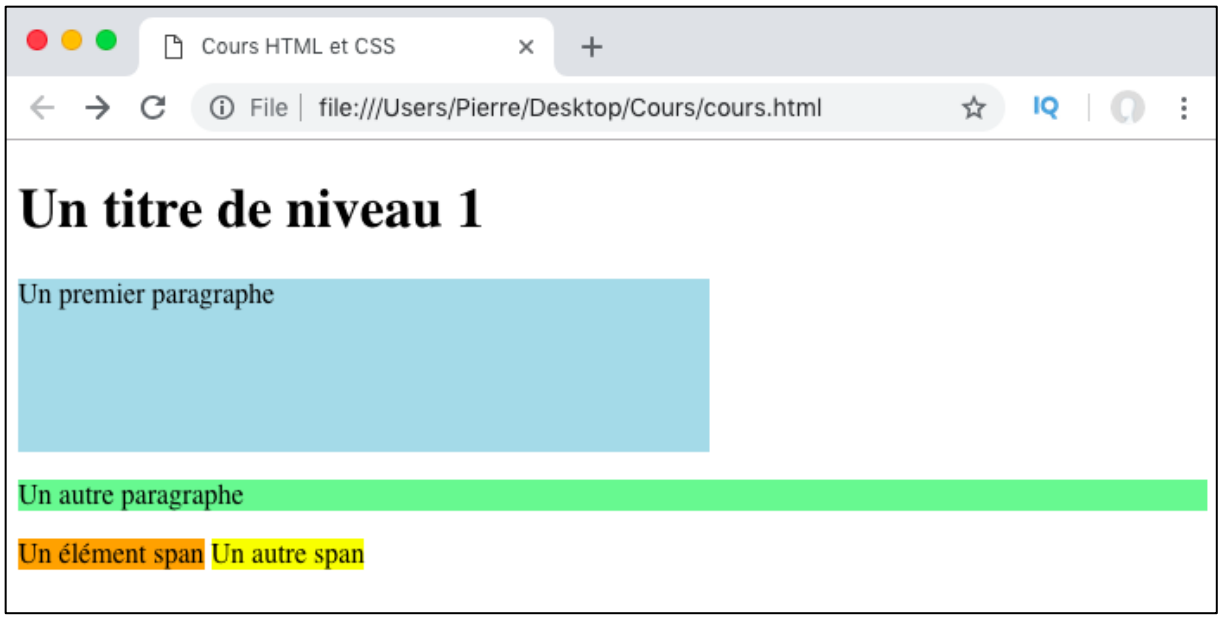
Vous pouvez ainsi déjà retenir ici que nous n'allons pas pouvoir modifier la taille de l'espace pris par le contenu des éléments de type `inline` avec les propriétés `width` et `height` : les valeurs qu'on va pouvoir définir vont tout simplement être ignorées.

En effet, le principe de base d'un élément de type `inline` est que l'espace pris par sa boite « contenu » soit toujours égal à l'espace strictement nécessaire à l'affichage de ce contenu.

Regardez l'exemple ci-dessous pour vous en convaincre. Je vous rappelle ici qu'un élément `p` possède un type d'affichage `block` tandis qu'un élément `span` possède un type d'affichage `inline`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un autre paragraphe</p>
        <span class="s1">Un élément span</span>
        <span class="s2">Un autre span</span>
    </body>
</html>
```

```css
p, span{
  font-size: 1em;
}

.p1{
  background-color: lightBlue;
  width: 400px;
  height: 100px;
}

.p2{
  background-color: lightGreen;
}

.s1{
  background-color: orange;
  width: 400px;
  height: 100px;
}

.s2{
  background-color: yellow;
}
```



Comme vous pouvez le constater, les valeurs passées à `width` et à `height` sont ignorées pour mon `span s1`.

Notez bien ici que modifier la taille de la boite du contenu d'un élément de type `block` ne change pas les propriétés fondamentales de celui-ci. J'entends par là qu'un élément de type `block` commencera toujours sur une nouvelle ligne et ne viendra jamais se positionner à côté d'un autre élément de type `block` quelle que soit sa taille.

Les types de valeur des propriétés width et height et les problèmes de dépassement

Les propriétés `width` et `height` vont pouvoir accepter plusieurs types de valeurs :

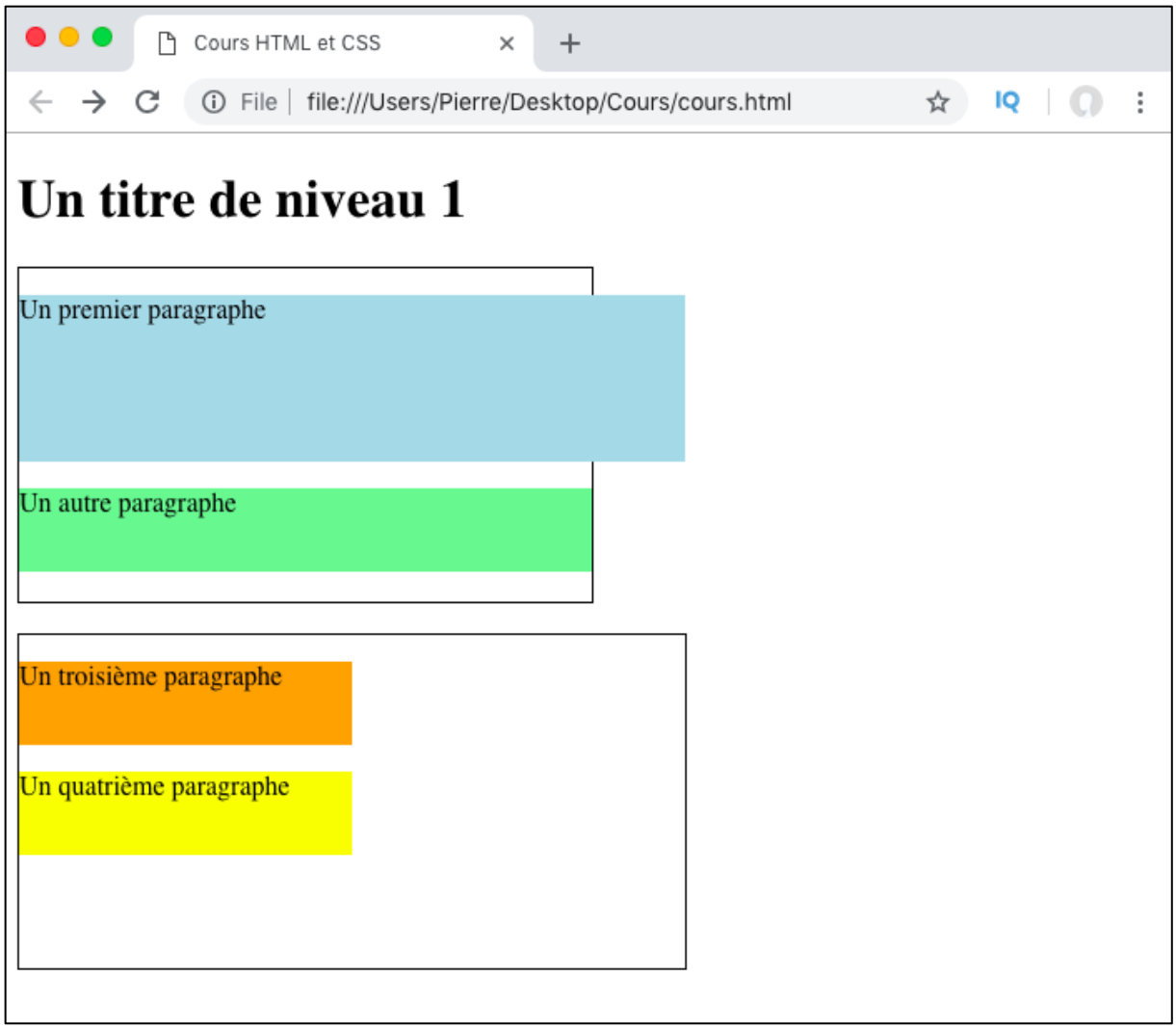
- Des valeurs de type « longueur » qui vont être généralement exprimées en `px` ou en `em` ;
- Des valeurs en pourcentage, auquel cas le pourcentage donné sera relatif à la dimension de l'élément parent.

Problème de dépassement n°1 : l'élément dépasse de son parent

Pour bien identifier ce premier problème de dépassement, prenons immédiatement un exemple qui va également nous permettre de comprendre pourquoi il est généralement déconseillé de mélanger des unités fixes et relatives lorsqu'on définit des dimensions.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <div class="d1">
            <p class="p1">Un premier paragraphe</p>
            <p class="p2">Un autre paragraphe</p>
        </div>
        <br>
        <div class="d2">
            <p class="p3">Un troisième paragraphe</p>
            <p class="p4">Un quatrième paragraphe</p>
        </div>
    </body>
</html>
```

```css
p, span{
  font-size: 1em;
}

.d1{
  width: 50%;
  height: 200px;
  border: 1px solid black;
}
.d2{
  width: 400px;
  height: 200px;
  border: 1px solid black;
}
.p1{
  background-color: lightBlue;
  width: 400px;
  height: 100px;
}
.p2{
  background-color: lightGreen;
  width: 100%;
  height: 25%;
}
.p3{
  background-color: orange;
  width: 200px;
  height: 50px;
}
.p4{
  background-color: yellow;
  width: 50%;
  height: 25%;
}
```



Dans cet exemple, la largeur de notre premier conteneur `div class="d1"` est définie de manière relative par rapport à son parent (qui est le `body`). La largeur du `div` va donc changer en fonction de la largeur de la fenêtre.

Ensuite, dans ce `div` conteneur, on spécifie une largeur pour notre premier paragraphe en unités absolues et fixes. Le problème ici va être que pour des fenêtres trop petites notre élément conteneur `div` va être plus petit que le paragraphe qu'il contient et cela va poser de nombreux problèmes de design.

Pour éviter ce genre de problèmes, la première règle est de bien faire attention lorsqu'on définit des dimensions aux différents types de valeurs utilisés et aux interactions entre ces différentes valeurs.

Ensuite, afin d'être certain que le design général de notre page ne sera pas impacté, on peut également utiliser la propriété `overflow` et en particulier sa valeur `hidden` sur l'élément parent. Cela va avoir pour effet de tronquer tout le contenu qui dépasse de l'élément. Si l'on souhaite que le contenu reste accessible, on peut utiliser `overflow : scroll` qui va proposer une barre de défilement dans l'élément parent.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <div class="d1">
            <p class="p1">Un premier paragraphe avec du texte qui risque
            d'être coupé si le contenu dépasse du parent</p>
            <p class="p2">Un autre paragraphe</p>
        </div>
        <br>
        <div class="d2">
            <p class="p3">Un troisième paragraphe</p>
            <p class="p4">Un quatrième paragraphe</p>
        </div>
    </body>
</html>
```

```css
p, span{
  font-size: 1em;
}

.d1{
  width: 50%;
  height: 200px;
  border: 1px solid black;
  overflow: hidden;
}
.d2{
  width: 400px;
  height: 200px;
  border: 1px solid black;
}
.p1{
  background-color: lightBlue;
  width: 400px;
  height: 100px;
}
.p2{
  background-color: lightGreen;
  width: 100%;
  height: 25%;
}
.p3{
  background-color: orange;
  width: 200px;
  height: 50px;
}
.p4{
  background-color: yellow;
  width: 50%;
  height: 25%;
}
```

Problème de dépassement n°2 : le contenu dépasse de l'élément

Un autre problème de dépassement courant va être le problème « contraire » au précédent, à savoir le cas où le contenu de mon élément dépasse de sa boite.

Ce problème va survenir si on définit une taille trop petite pour la boite de contenu de l'élément par rapport au contenu de l'élément.

Dans ce cas-là, il va falloir utiliser la propriété `overflow` sur l'élément en soi afin que le contenu qui dépasse soit caché.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <div class="d1">
            <p class="p1">Un premier paragraphe avec beaucoup de texte
            qui risque de dépasser de l'élément en soi</p>
            <p class="p2">Un autre paragraphe</p>
        </div>
    </body>
</html>
```

```css
p, span{
  font-size: 1em;
}

.d1{
  width: 300px;
  height: 200px;
  border: 1px solid black;
}
.p1{
  background-color: lightBlue;
  width: 200px;
  height: 60px;
  overflow: scroll;
}
.p2{
  background-color: lightGreen;
  width: 100%;
  height: 25%;
}
```

Problème de dépassement n°3 : l'élément dépasse globalement de son parent

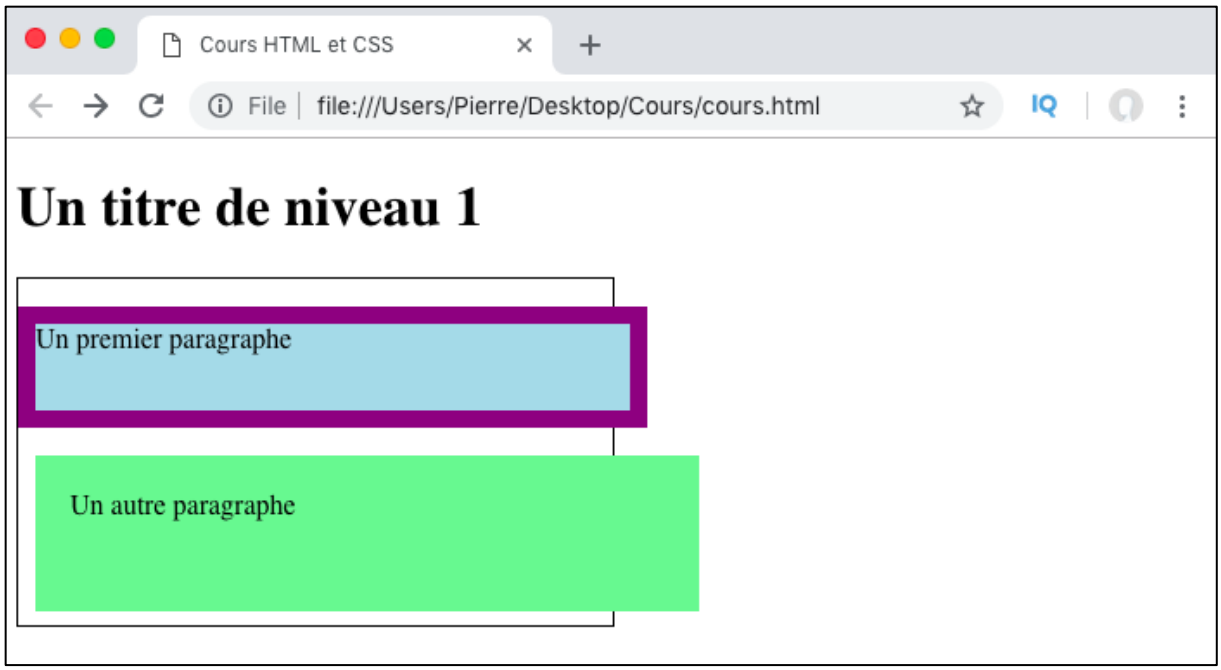
Une autre erreur couramment faite se situe dans la mauvaise compréhension du modèle des boites. En effet, il faut bien ici se rappeler que les propriétés `width` et `height` ne servent qu'à définir les dimensions de la boite liée au contenu d'un élément.

A ces dimensions vont venir s'ajouter les tailles des marges intérieures, des bordures et celles des marges extérieures pour former la taille totale de l'élément. Ainsi, si l'on ne fait pas attention, on peut très rapidement se retrouver avec des éléments enfants plus grands et qui vont dépasser de leur parent.

Imaginons par exemple qu'on fixe la largeur d'un élément enfant à `100%` de celle de son parent. Si on attribue ensuite une quelconque marge interne, bordure, ou marge externe à l'élément, celui-ci va dépasser de son élément parent.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <div class="d1">
            <p class="p1">Un premier paragraphe</p>
            <p class="p2">Un autre paragraphe</p>
        </div>
    </body>
</html>
```

```css
p, span{
  font-size: 1em;
}

.d1{
  width: 50%;
  height: 200px;
  border: 1px solid black;
}
.d2{
  width: 400px;
  height: 200px;
  border: 1px solid black;
}
.p1{
  background-color: lightBlue;
  width: 100%;
  height: 50px;
  border: 10px solid purple; /*Bordure violette de 10px*/
}
.p2{
  background-color: lightGreen;
  width: 100%;
  height: 25%;
  padding: 20px; /*Marge interne de 20px*/
  margin: 10px; /*Marge externe de 10px*/
}
```


Cours HTML et CSS
File | file:///Users/Pierre/Desktop/Cours/cours.html
Un titre de niveau 1
Un premier paragraphe
Un autre paragraphe

Gestion des marges internes (« padding ») en CSS

En CSS, nous allons devoir distinguer deux types de marges : les marges intérieures (“padding”) et les marges extérieures (“margin”).

Les marges intérieures se trouvent entre le contenu de l’élément et sa bordure. Ainsi, définir une marge intérieure importante va éloigner la bordure de l’élément de son contenu. Si on définit une couleur de fond pour notre élément, celle-ci s’applique également dans l’espace correspondant aux marges intérieures.

La propriété CSS padding

Nous allons pouvoir ajouter des marges internes à un élément grâce à la propriété CSS `padding`. Notez déjà que la propriété `padding` est la version raccourcie des propriétés `padding-top`, `padding-left`, `padding-bottom` et `padding-right` qui vont servir à définir les marges internes de chaque côté d’un élément.

Ces propriétés vont pouvoir accepter deux types de valeurs :

- Des valeurs de type longueur, généralement en `px` ou en `em` ;
- Des valeurs de type pourcentage. Dans ce cas, le `%` indiqué est calculé par rapport à la taille de l’élément parent.

Notez qu’il n’est pas possible de passer des valeurs de `padding` négatives.

Ajouter une même marge interne de chaque côté d’un élément

Pour définir une même marge interne de chaque côté d’un élément, il suffit de passer la valeur de la marge intérieure que l’on souhaite appliquer à l’élément à la propriété `padding` en CSS.

Par exemple, pour appliquer une marge intérieure de 25px de chaque côté d’un élément, on écrira en CSS `padding: 25px`. Pour que la marge intérieure soit égale à 10% de la taille de l’élément parent de notre élément, on écrira `padding: 10%` tout simplement.

Pour bien constater l’effet de la propriété `padding`, je vous conseille d’ajouter une couleur de fond ou une bordure aux éléments pour les exemples suivants.

Notez qu’on va tout aussi bien pouvoir utiliser les propriétés `padding-top`, `padding-left`, `padding-bottom`et `padding-right` et leur passer la même valeur pour arriver au même résultat. C’est juste plus long à écrire !

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un deuxième paragraphe</p>
        <p class="p3">Un troisième paragraphe</p>
    </body>
</html>
```

```css
p{
  background-color: lightGreen;
  border: 1px solid black;
}

.p2{
  padding: 20px;
}

.p3{
  padding-top: 20px;
  padding-right: 20px;
  padding-bottom: 20px;
  padding-left: 20px;
}
```

Définir des marges internes différentes de chaque côté d'un élément

Nous allons également pouvoir appliquer des marges intérieures de taille différentes de chaque côté d'un élément.

Pour cela, nous avons deux façons de faire : soit en passant plusieurs valeurs à la propriété `padding`, soit en utilisant les propriétés `padding-top`, `padding-left`, `padding-bottom` et `padding-right`.

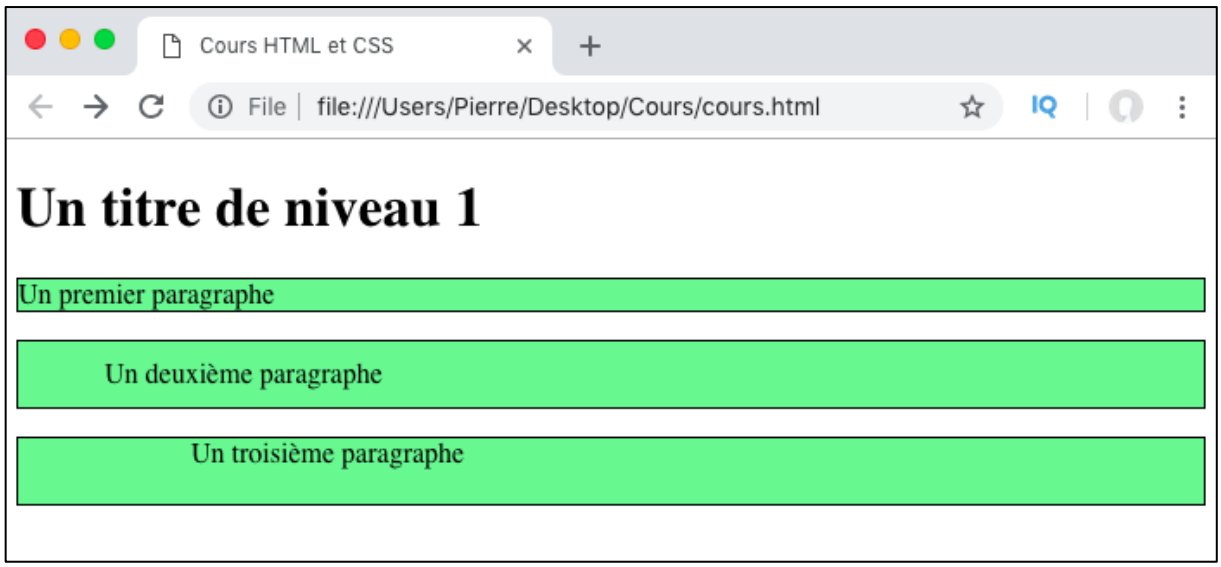
On va en effet pouvoir passer entre 1 et 4 valeurs à la propriété raccourcie `padding` :

- En passant une valeur à `padding`, la valeur passée définira le comportement des 4 marges intérieures de l'élément ;
- En passant deux valeurs à `padding`, la première valeur passée définira le comportement des marges intérieures supérieure et inférieure de l'élément tandis que la seconde valeur définira le comportement des marges intérieures gauche et droite de l'élément ;
- En passant trois valeurs à `padding`, la première valeur passée définira le comportement de la marge interne supérieure, la deuxième définira le comportement des marges intérieures gauche et droite tandis que la troisième définira le comportement de la marge interne basse ;
- En passant quatre valeurs à `padding`, la première valeur passée définira le comportement de la marge interne supérieure, la deuxième définira le comportement de la marge interne droite, la troisième celui de la marge interne basse et la quatrième celui de la marge interne gauche.

Si on choisit d'utiliser les propriétés `padding-top`, `padding-left`, `padding-bottom` et `padding-right`, il suffit de suivre le nom pour comprendre à quelle marge interne est liée chaque propriété (top = haute, left = gauche, bottom = basse, right = droite).

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un deuxième paragraphe</p>
        <p class="p3">Un troisième paragraphe</p>
    </body>
</html>
```

```css
p{
  background-color: lightGreen;
  border: 1px solid black;
}

/*Padding top et bottom = 10px
 *Padding left et right = 50px*/
.p2{
  padding: 10px 50px;
}

.p3{
  padding-left: 100px;
  padding-bottom: 20px;
}
```



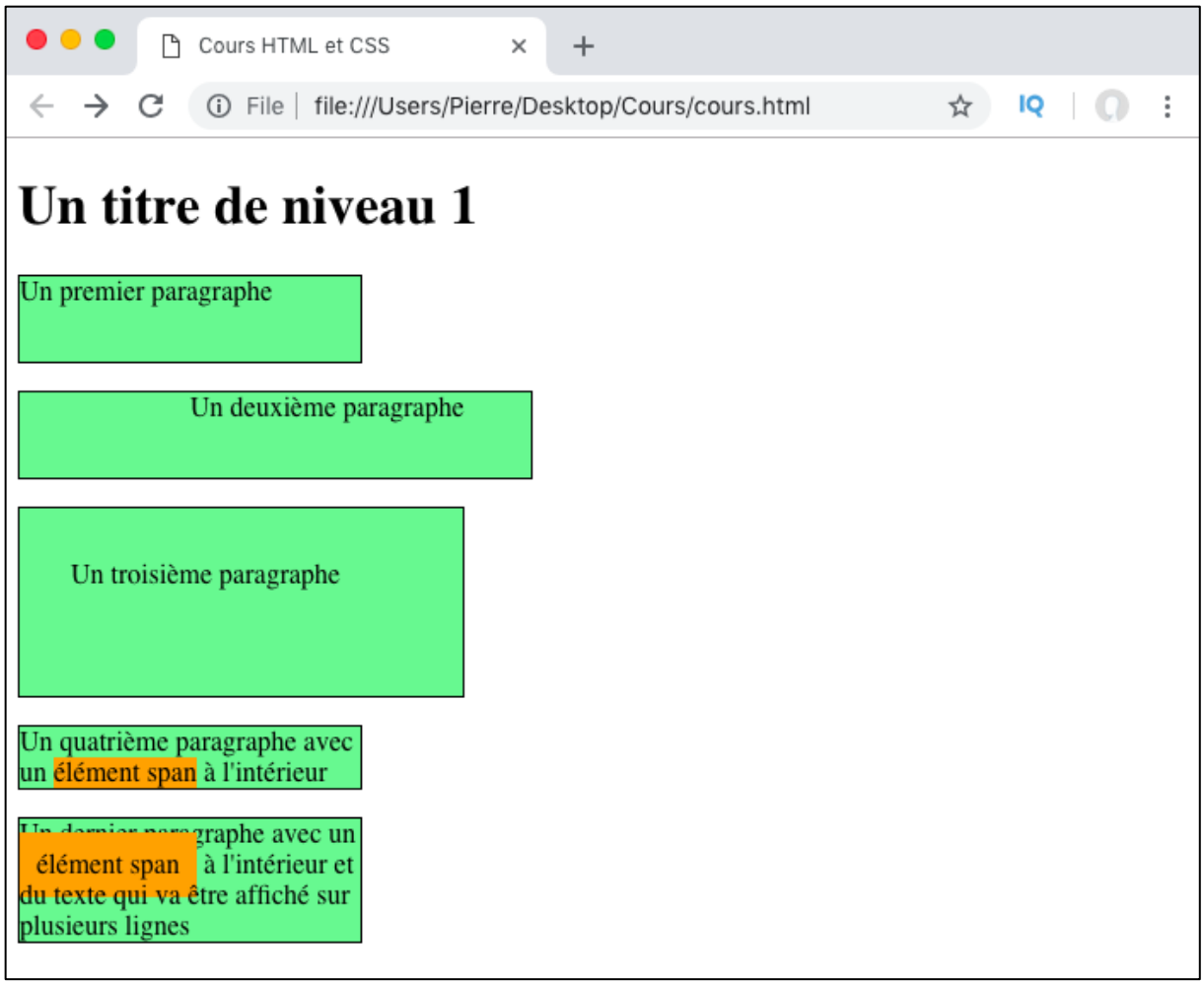
L'effet du padding sur les éléments environnants

L'ajout de marges internes ou `padding` va augmenter la taille totale de l'élément. En effet, la valeur donnée au `padding` va venir s'ajouter à celles des propriétés `width` et `height` par défaut.

Regardez plutôt le code ci-dessous pour vous en convaincre :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un deuxième paragraphe</p>
        <p class="p3">Un troisième paragraphe</p>

        <p class="p4">Un quatrième paragraphe avec
        un <span>élément span</span> à l'intérieur</p>

        <p class="p4">Un dernier paragraphe avec un
        <span class="s1">élément span</span> à l'intérieur et du
        texte qui va être affiché sur plusieurs lignes</p>
    </body>
</html>
```

```css
p{
  background-color: lightGreen;
  border: 1px solid black;
  width: 200px;
}
.p1, .p2, .p3{
  height: 50px;
}
.p2{
  padding-left: 100px;
}
.p3{
  padding: 30px;
}
span{
  background-color: orange;
}
.s1{
  padding: 10px;
}
```



Ici, on voit clairement que le `padding` a un impact sur les dimensions totales des éléments, et ceci qu'ils soient de type `block` ou `inline`.

Cependant, l'impact sur les éléments environnants ne va pas être le même selon le type d'élément auquel on applique un `padding` et selon le `padding` appliqué.

Ici, il y a notamment un cas à noter : celui des marges internes haute et basse d'un élément de type `inline`. Comme vous pouvez le voir ci-dessus, les marges internes haute et basse sont bien appliquées à mon élément `span s1` (on le voit grâce à la couleur en fond de mon élément).

Cependant, le navigateur ne va pas en tenir compte pour l'affichage et le positionnement des autres éléments. C'est la raison pour laquelle notre élément `span` chevauche le texte des lignes précédente et suivante du paragraphe.

Notez par ailleurs ici les différentes « couches » de texte : la ligne de texte précédent celle du `span` va se trouver en dessous de lui tandis que la suivante va être au-dessus (on voit que la couleur de fond du `span` cache le texte précédent mais est sous le texte de la ligne suivante).

Encore une fois, notez que le `padding` est bien appliqué de chaque côté pour chaque type d'éléments, cependant les `padding` haut et bas n'auront pas d'impact sur le positionnement d'un élément `inline` ni sur celui des éléments autour de lui.

Gestion des bordures en CSS

Nous allons pouvoir définir des bordures de largeurs, de styles ou de couleurs différents autour de nos éléments HTML en CSS.

L'espace pris par la bordure va se trouver entre la marge intérieure et la marge extérieure d'un élément HTML.
Nous pouvons définir les bordures d'un élément de différentes manières en CSS : soit en utilisant les trois propriétés `border-width`, `border-style` et `border-color`, soit un utilisant directement la notation raccourcie `border`.

Nous allons également pouvoir créer des bordures arrondies à l'aide de la propriété `border-radius` que nous étudierons plus tard dans ce cours car elle est relativement complexe à comprendre et à maitriser.

Les caractéristiques des bordures en CSS

Les bordures vont être définies par trois caractéristiques en CSS : une épaisseur (ou largeur), un style et une couleur.

Nous allons pouvoir définir ces différentes caractéristiques d'un coup au sein de la propriété raccourcie `border` ou les définir une à une avec chacune des propriétés de type `border-`. Plus précisément :

- La propriété `border-width` va nous permettre de définir la largeur (ou « l'épaisseur) d'une bordure ;
- La propriété `border-style` va nous permettre de définir le style d'une bordure ;
- La propriété `border-color` va nous permettre de définir la couleur d'une bordure.

Définir la largeur ou l'épaisseur d'une bordure

Nous allons pouvoir utiliser différents types de valeurs pour définir la largeur d'une bordure :

- Une valeur de type « mot clef » à choisir parmi `thin` (bordure fine), `medium` (bordure moyenne) et `thick` (bordure épaisse) ;
- Une valeur de type « longueur » en `px` ou en `em` par exemple.

Généralement, nous utiliserons des unités en `px` pour définir la largeur de nos bordures.

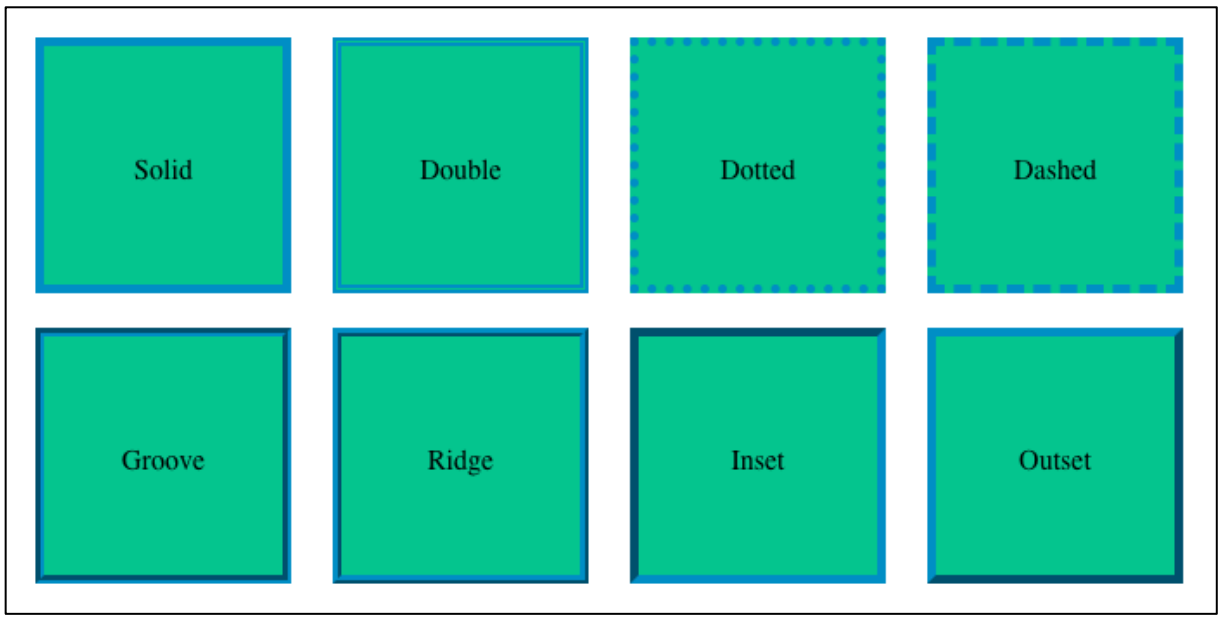
Choisir le style d'une bordure

Le « style » d'une bordure correspond à son aspect : une bordure peut prendre la forme d'un simple trait, d'un trait double, ou être constituée de pointillés, avoir un effet 3D, etc.
Pour définir le style d'une bordure, nous allons devoir choisir parmi les mots clefs suivants :

Valeur	Description

solid	Bordure solide simple (un trait)
double	Bordure solide double
dotted	Bordure en pointillés
dashed	Bordure constituée de tirets
groove	Bordure incrustée avec effet 3D. L'effet produit est l'inverse de `ridge`
ridge	Bordure en relief avec effet 3D. L'effet produit est l'inverse de `groove`
inset	La bordure donne l'effet que la boite représentant l'élément est enfoncée dans la page. L'effet produit est l'inverse de `outset`
outset	La bordure donne l'effet que la boite représentant l'élément est en relief par rapport au reste de la page. L'effet produit est l'inverse de `inset`

Chaque mot clef va créer un type de bordure différent. Voici à quoi correspond chaque effet visuellement :



Définir la couleur d'une bordure

Finalement, nous allons devoir définir la couleur d'une bordure. Pour cela, nous allons pouvoir piocher parmi toutes les valeurs de type « couleur » connues et notamment :

- Les notations de type « nom de couleur » ;
- Les notations hexadécimales ;
- Les notations RGB() et RGBa() ;
- Les notations HSL() et HSLa().

Exemples de création de bordures en CSS

La suite de cette leçon va être pour nous l'occasion de s'exercer et de créer toutes sortes de bordures. Je vous rappelle ici que nous aborderons les bordures arrondies dans la prochaine leçon.

Commençons déjà avec des exemples simples de bordures créées en CSS en appliquant ce que nous avons vu ci-dessous.

Ici, nous allons créer 4 types de bordures différentes : une bordure simple, une bordure double, une avec effet 3D de type « ridge » et une avec un style « outset ».

Nous allons pouvoir faire cela en utilisant soit les propriétés `border-width`, `border-style` et `border-color`, soit la notation raccourcie `border`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Titre de niveau 1</h1>
        <p class="p1">Solid</p>
        <p class="p2">Double</p>
        <p class="p3">Ridge</p>
        <p class="p4">Outset</p>
    </body>
</html>
```

```css
p{
  padding: 20px;
  margin: 20px 10px;
}

.p1{
  border-width: 5px;
  border-style: solid;
  border-color : #09C;
}
.p2{
  border: 15px double #09C;
}
.p3{
  background-color: #0C9;
  border: 5px ridge #09C;
}
.p4{
  background-color: #0C9;
  border: 5px outset #09C;
}
```

Ici, nous nous contentons d'ajouter des bordures différentes autour de nos paragraphes. J'ai également rajouté une couleur de fond aux deux derniers paragraphes afin que l'on puisse bien observer l'effet de 3D des deux dernières bordures.

Notez une chose intéressante par rapport à notre bordure double : la taille de la bordure correspond à la taille totale de la bordure, c'est-à-dire dans ce cas du double trait et de l'espace entre ces deux traits. La largeur totale va être répartie régulièrement : en indiquant une bordure double de `15px`, chaque élément de la bordure (les deux traits plus l'espace entre les traits) va faire 5px de large (15px / 3).

Définir des bordures différentes pour chaque côté d'un élément

Le CSS va également nous permettre de définir chacune des quatre bordures de nos éléments indépendamment les unes des autres afin de pouvoir appliquer des effets intéressants.

Nous avons deux façons de faire cela selon que nous utilisions la notation raccourcie `border` pour définir nos bordures ou que nous utilisions chacune des propriétés `border-`.

Dans le cas où nous souhaitons utiliser `border`, nous allons devoir utiliser 4 sous propriétés CSS qui sont :

- `border-top` pour définir l'aspect (taille, style et couleur) de la bordure supérieure de l'élément ;

- `border-right` pour définir l'aspect (taille, style et couleur) de la bordure droite de l'élément ;
- `border-bottom` pour définir l'aspect (taille, style et couleur) de la bordure inférieure de l'élément ;
- `border-left` pour définir l'aspect (taille, style et couleur) de la bordure gauche de l'élément.

Bon à savoir : Les propriétés `border-top`, `border-right`, `border-bottom` et `border-left` sont à nouveau des écritures raccourcies des propriétés CSS `border-top-width`, `border-top-style`, `border-top-color`, etc.

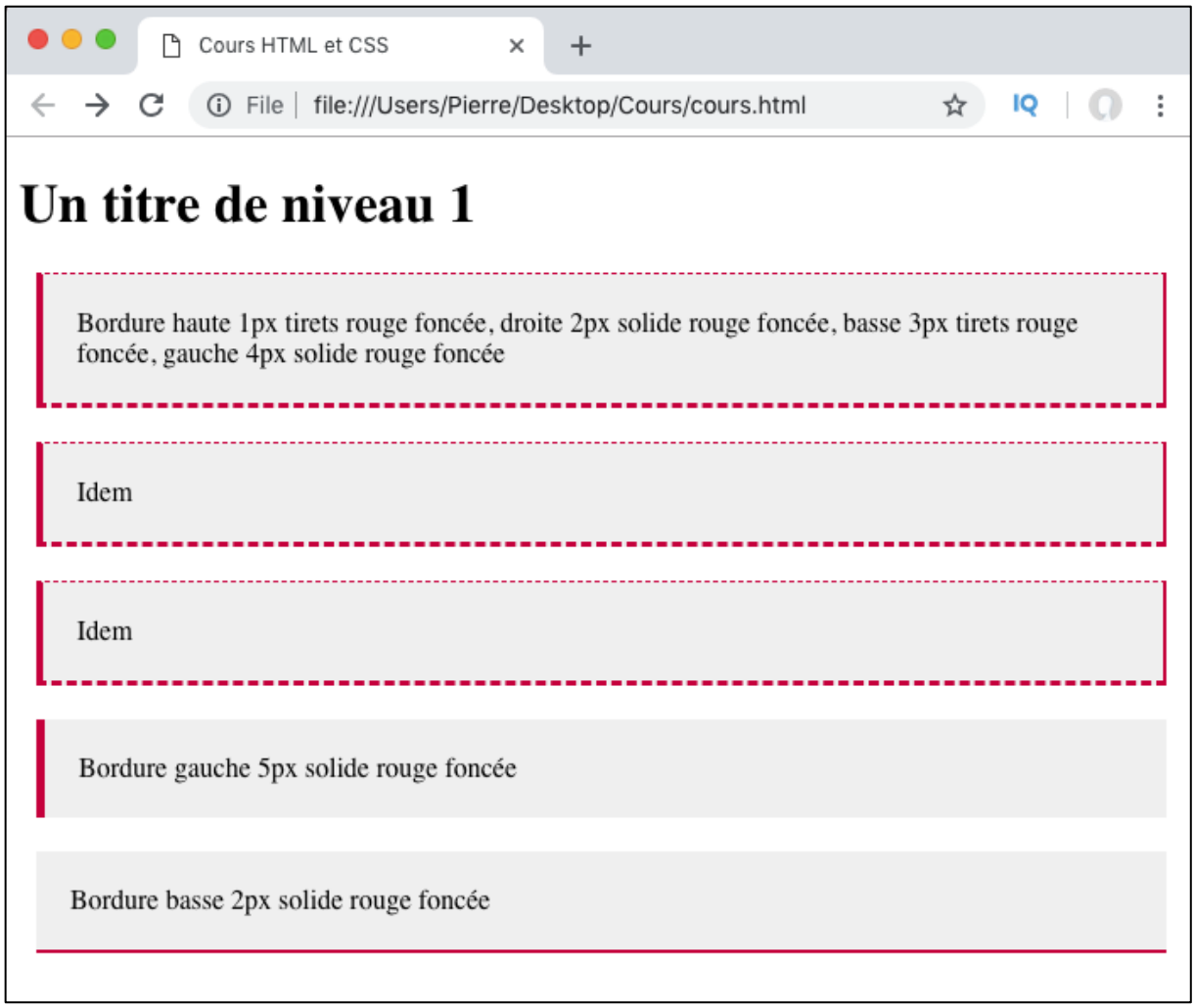
Dans le cas où nous utilisons les propriétés `border-width`, `border-style` et `border-color` alors nous pourrons indiquer 4 valeurs à la suite qui définiront le comportement des bordures supérieure, droite, inférieure et gauche dans cet ordre.

Notez ici qu'il est tout à fait possible de ne renseigner qu'une valeur pour l'une de ces 3 propriétés si vous voulez que vos 4 bordures aient le même comportement. Vous pouvez également ne mentionner que deux valeurs : dans ce cas, la première valeur définira le comportement des bordures supérieure et inférieure tandis que la seconde définira le comportement des bordures droite et gauche de l'élément.

Regardez plutôt les exemples ci-dessous qui résument toutes les situations possibles pour bien comprendre :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Bordure haute 1px tirets rouge foncée,
        droite 2px solide rouge foncée,
        basse 3px tirets rouge foncée,
        gauche 4px solide rouge foncée</p>
        <p class="p2">Idem</p>
        <p class="p3">Idem</p>
        <p class="p4">Bordure gauche 5px solide rouge foncée</p>
        <p class="p5">Bordure basse 2px solide rouge foncée</p>
    </body>
</html>
```

```css
p{
  padding: 20px;
  margin: 20px 10px;
  background-color: #f1f1f1;
}

.p1{
  border-width: 1px 2px 3px 4px;
  border-style: dashed solid dashed solid;
  border-color: #C04 #C04 #C04 #C04; /*Rouge foncé*/
}

.p2{
  border-width: 1px 2px 3px 4px;
  border-style: dashed solid;
  border-color: #C04;
}

.p3{
  border-top: 1px dashed #C04;
  border-right: 2px solid #C04;
  border-bottom: 3px dashed #C04;
  border-left : 4px solid #C04;
}

.p4{
  border-left-width: 5px;
  border-left-style: solid;
  border-left-color: #C04;
}
.p5{
  border-bottom: 2px solid #C04;
}
```



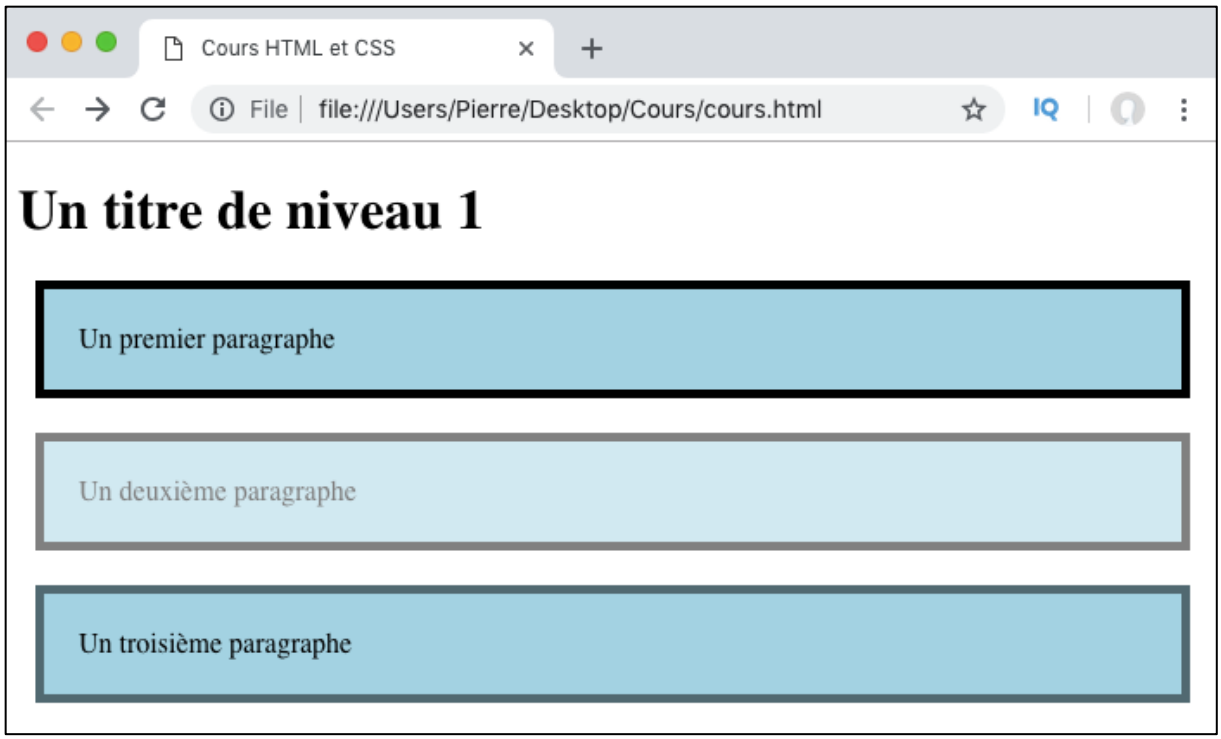
Créer des bordures semi transparentes

Nous allons encore pouvoir ajouter un effet de transparence à nos bordures en utilisant tout simplement une notation RGBa lorsque l'on précisera la paramètre couleur de notre bordure.

Notez ici qu'utiliser la propriété CSS `opacity` ne produirait pas le comportement voulu puisque l'effet de transparence serait appliqué à l'élément entier et non pas seulement à la bordure comme on le souhaite.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p class="p1">Un premier paragraphe</p>
        <p class="p2">Un deuxième paragraphe</p>
        <p class="p3">Un troisième paragraphe</p>
    </body>
</html>
```

```css
p{
  padding: 20px;
  margin: 20px 10px;
  background-color: lightBlue;
}
.p1{
    border: 5px solid black;
}

.p2{
    border: 5px solid black;
    opacity: 0.5;
}

.p3{
    border: 5px solid RGBa(0, 0, 0, 0.5);
}
```



Les bordures dans le modèle des boites

Les bordures d'un élément HTML se situent entre les marges internes et externes de l'élément.

La taille des bordures va par défaut venir s'ajouter aux dimensions de l'élément définies avec les propriétés `width` et `height` ainsi qu'aux marges internes.

Notez que nous allons pouvoir définir des bordures de la même façon pour des éléments de type `block` et `inline` et que ces bordures auront exactement le même comportement.

Gestion des marges externes en CSS

En CSS, nous allons devoir distinguer deux types de marges : les marges intérieures (“padding”) et les marges extérieures (“margin”).

Les marges intérieures se trouvent entre le contenu de l’élément et sa bordure. Les marges extérieures, au contraire, vont définir la taille de l’espace autour d’un élément.

Les marges extérieures se trouvent en dehors des bordures d’un élément et servent généralement à éloigner un élément d’un autre. Lorsqu’on définit une couleur de fond pour un élément, cette couleur de fond ne va pas s’appliquer dans l’espace des marges extérieures car encore une fois celles-ci se trouvent « en dehors » de l’élément.

La propriété CSS margin

Nous allons pouvoir ajouter des marges externes à un élément grâce à la propriété CSS `margin`. Cette propriété est en fait la notation raccourcie des propriétés `margin-top`, `margin-left`, `margin-bottom` et `margin-right` qui vont servir à définir les marges externes de chaque côté d’un élément.

Ces propriétés vont pouvoir accepter différents types de valeurs :

- Des valeurs de type longueur, généralement en `px` ou en `em` ;
- Des valeurs de type pourcentage. Dans ce cas, le `%` indiqué est calculé par rapport à la taille de l’élément parent ;
- Le mot clef `auto`. Ce mot clef va surtout nous servir pour centrer des blocs.

Notez qu’à la différence de la propriété `padding`, nous allons pouvoir passer des valeurs négatives à `margin` pour créer des marges externes négatives même si en pratique cela est généralement déconseillé pour les problèmes d’ergonomie et de consistance du design que ça peut causer au niveau de la page.

Notez également que la plupart des navigateurs définissent des marges par défaut pour de nombreux éléments. Il faudra donc souvent effectuer un reset des marges dans nos feuilles de styles (en définissant des marges nulles pour les différents éléments) pour être certain d’obtenir le résultat attendu.

Ajouter une même marge externe de chaque côté d’un élément

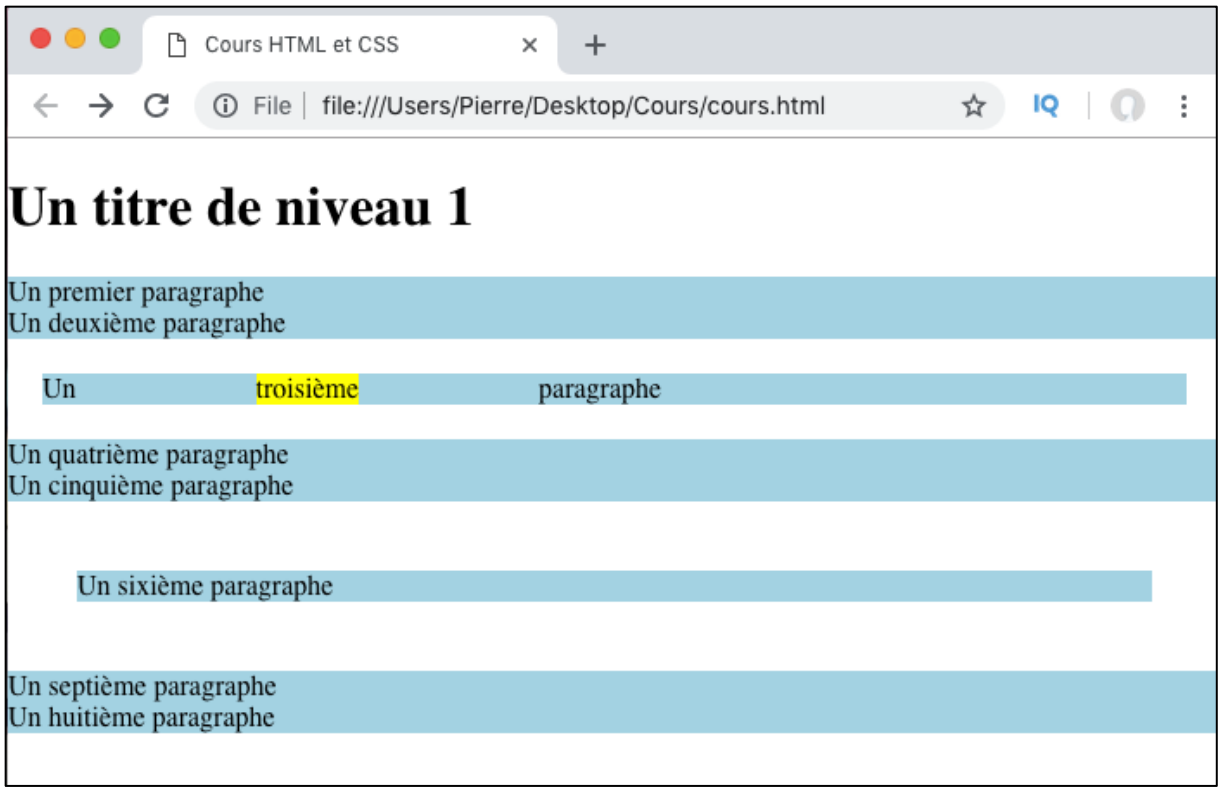
Pour définir une même marge externe de chaque côté d’un élément, il suffit de passer la valeur de la marge extérieure que l’on souhaite appliquer à l’élément à la propriété `margin` en CSS.

Par exemple, pour appliquer une marge extérieure de 25px de chaque côté d'un élément, on écrira en CSS `margin : 25px`. Pour que la marge extérieure représente 10% de la taille de l'élément parent de notre élément, on écrira `margin : 10%` tout simplement.

Nous allons bien évidemment également pouvoir définir des marges externes égales pour chaque côté d'un élément en utilisant les propriétés `margin-top`, `margin-left`, `margin-bottom` et `margin-right` et en leur passant la même valeur à chacune.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p id="p1">Un premier paragraphe</p>
        <p id="p2">Un deuxième paragraphe</p>
        <p id="p3">Un <span>troisième</span> paragraphe</p>
        <p id="p4">Un quatrième paragraphe</p>
        <p id="p5">Un cinquième paragraphe</p>
        <p id="p6">Un sixième paragraphe</p>
        <p id="p7">Un septième paragraphe</p>
        <p id="p8">Un huitième paragraphe</p>
    </body>
</html>
```

```css
/*On reset les marges pour ensuite bien les redéfinir*/
body, p{
  margin: 0px;
}

p{
  background-color: lightBlue;
}

#p3{
  margin: 20px;
}
span{
  margin: 100px;
  background-color: yellow;
}
#p6{
  margin-top: 40px;
  margin-right: 40px;
  margin-bottom: 40px;
  margin-left: 40px;
}
```



Ici, vous pouvez noter que les bordures d'un élément définissent la « limite » de cet élément. Les marges externes sont un espace entre les éléments et c'est la raison pour laquelle nous ne pouvons pas appliquer de couleur de fond (entre autres) dans l'espace des marges externes de l'élément.

Notez également qu'on ne va pas pouvoir appliquer de marges haute ou basse aux éléments de type `inline` comme c'était déjà le cas pour les marges internes. En revanche, les marges externes droite et gauche s'appliquent normalement.

Définir des marges externes différentes de chaque côté d'un élément

Nous allons également pouvoir appliquer des marges extérieures de taille différentes de chaque côté d'un élément.

Pour cela, nous avons deux façons de faire : soit en passant plusieurs valeurs à la propriété `margin`, soit en utilisant les propriétés `margin-top`, `margin-left`, `margin-bottom` et `margin-right`.

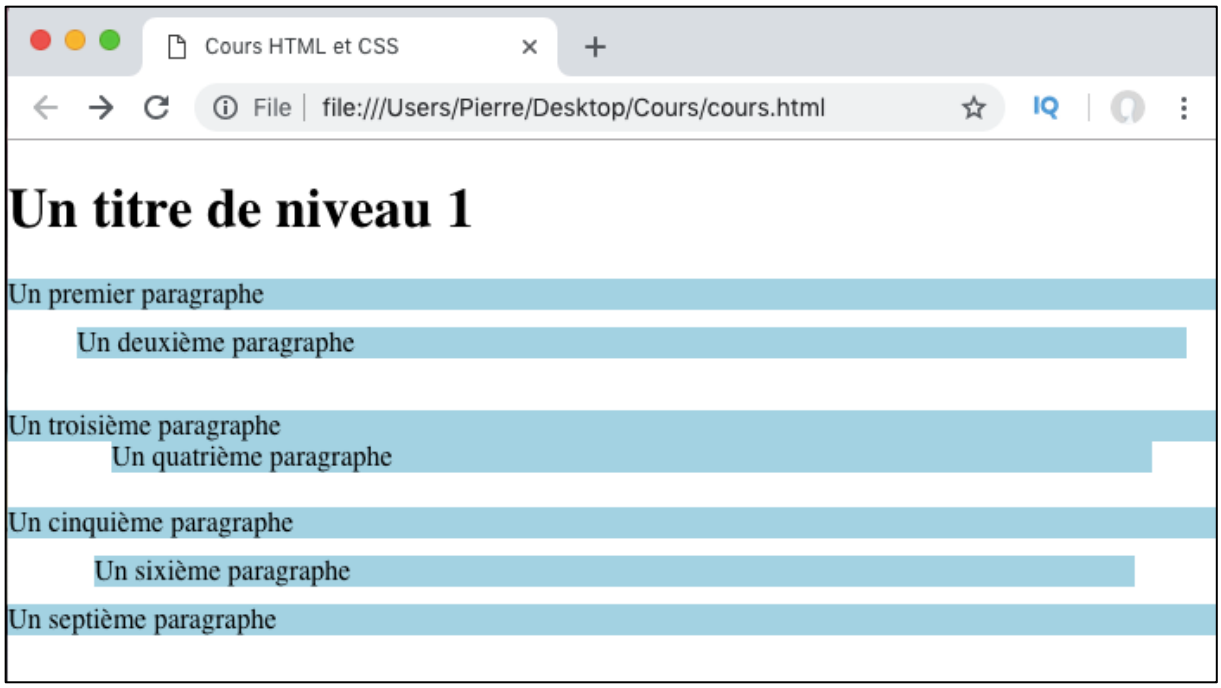
On va en effet pouvoir passer entre 1 et 4 valeurs à la propriété raccourcie `margin` :

- En passant une valeur à `margin`, la valeur passée définira le comportement des 4 marges extérieures de l'élément ;
- En passant deux valeurs à `margin`, la première valeur passée définira le comportement des marges extérieures supérieure et inférieure de l'élément tandis que la seconde valeur définira le comportement des marges extérieures gauche et droite de l'élément ;

- En passant trois valeurs à margin, la première valeur passée définira le comportement de la marge externe supérieure, la deuxième définira le comportement des marges extérieures gauche et droite tandis que la troisième définira le comportement de la marge externe basse ;
- En passant quatre valeurs à margin, la première valeur passée définira le comportement de la marge externe supérieure, la deuxième définira le comportement de la marge externe droite, la troisième celui de la marge externe basse et la quatrième celui de la marge externe gauche.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <p id="p1">Un premier paragraphe</p>
        <p id="p2">Un deuxième paragraphe</p>
        <p id="p3">Un troisième paragraphe</p>
        <p id="p4">Un quatrième paragraphe</p>
        <p id="p5">Un cinquième paragraphe</p>
        <p id="p6">Un sixième paragraphe</p>
        <p id="p7">Un septième paragraphe</p>
    </body>
</html>
```

```css
/*On reset les marges pour ensuite bien les redéfinir*/
body, p{
  margin: 0px;
}
p{
  background-color: lightBlue;
}

#p2{
  margin: 10px 20px 30px 40px ;
}
#p4{
  margin-right: 40px;
  margin-bottom: 20px;
  margin-left: 60px;
}
#p6{
  margin: 10px 50px;
}
```



La fusion ou « collapsing » des marges externes verticales

Une des grandes différences entre l'implémentation des marges internes et des marges externes est que les marges externes haute et basse de deux éléments vont pouvoir « fusionner » selon certaines conditions.

La fusion des marges externes d'éléments consécutifs

Si on définit une marge basse à un premier élément de type `block` et une marge haute à l'élément suivant, alors les hauteurs des deux marges ne vont pas s'additionner mais vont « fusionner ». En effet, seule la marge la plus importante sera appliquée.

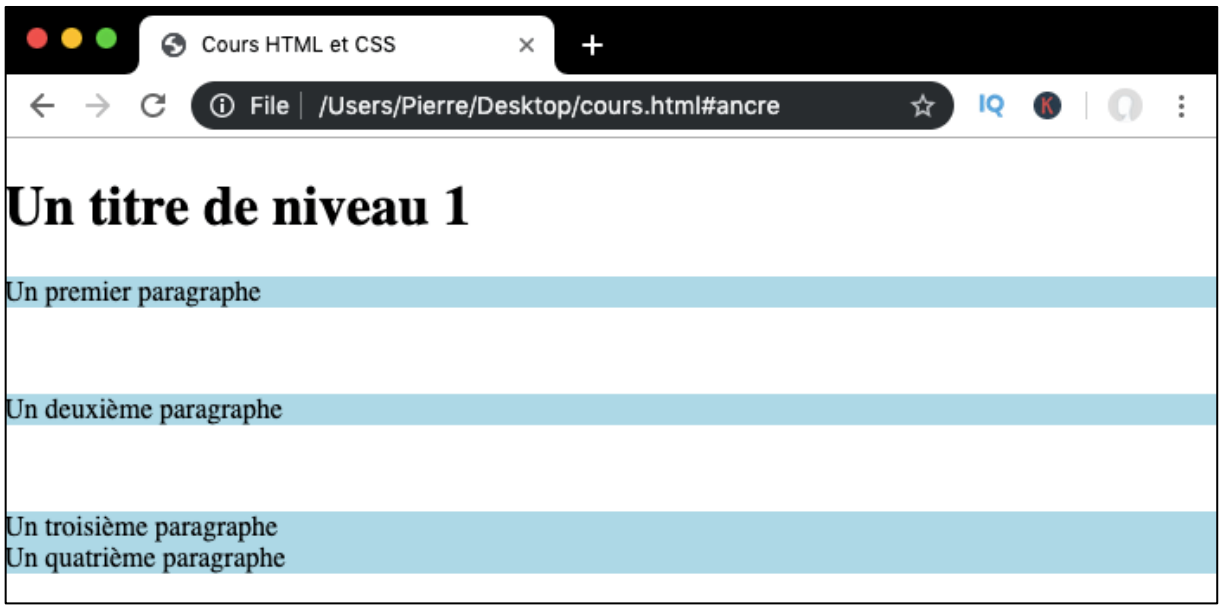
Par exemple, imaginons qu'on ait défini une `margin-bottom : 30px` pour notre premier élément et une `margin-top : 50px` pour l'élément suivant. Les deux éléments ne seront pas séparés de 30 + 50 = 80px mais seulement de la taille de la marge la plus importante parmi les deux, à savoir ici 50px.

De même, si deux éléments consécutifs possèdent deux marges négatives, alors seule la marge négative la plus importante sera conservée.

Dans le cas où un élément possède une marge basse externe positive et le suivant possède une marge haute externe négative (ou le contraire), alors la marge négative va se soustraire à la marge positive. Par exemple, si le premier élément possède une `margin-bottom : 50px` et le deuxième élément a une `margin-top : -30px`, alors la marge appliquée sera de 50 – 30 = 20px.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Un titre de niveau 1</h1>

        <p id="p1">Un premier paragraphe</p>
        <p id="p2">Un deuxième paragraphe</p>
        <p id="p3">Un troisième paragraphe</p>
        <p id="p4">Un quatrième paragraphe</p>
    </body>
</html>
```

```css
body, p{
  margin: 0px;
}
p{
  background-color: lightBlue;
}

#p1{
  margin-bottom: 30px;
}

#p2, #p3{
  margin-top: 50px;
}
```



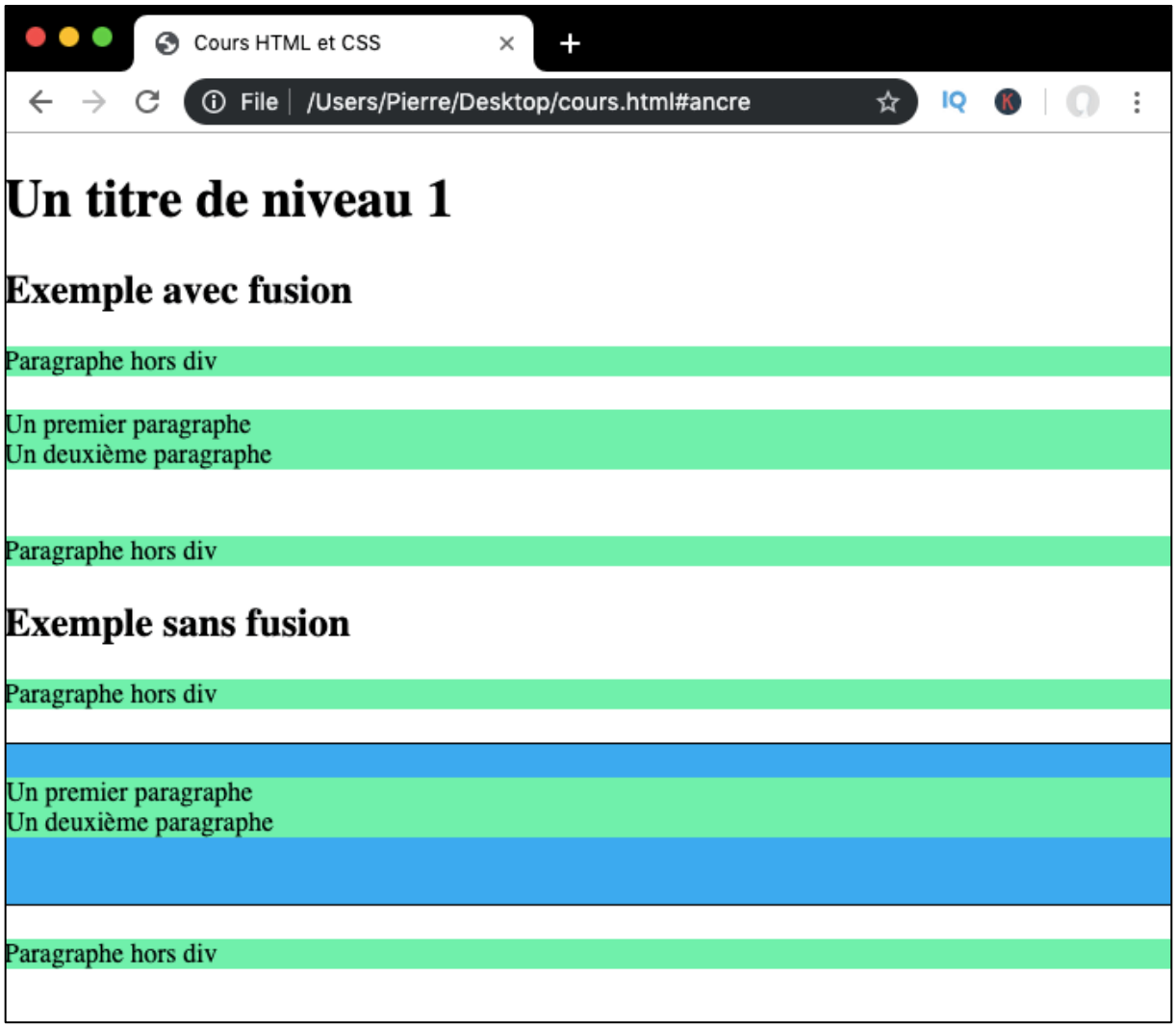
La fusion des marges entre un élément parent et son premier et dernier enfant

Cette deuxième situation de fusion des marges est plus complexe que la première. Pour faire très simple, vous pouvez retenir qu'il va y avoir fusion entre un élément parent et son premier enfant s'il n'y a aucune bordure ou remplissage entre sa marge haute et la marge haute de son enfant.

De même, il y aura fusion entre la marge basse du parent et la marge basse de son dernier enfant s'il n'y a aucune bordure ou remplissage pour les séparer et si aucune propriété `height`, `min-height` ou `max-height` n'est définie bien évidemment.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Un titre de niveau 1</h1>
        <h2>Exemple avec fusion</h2>
        <p>Paragraphe hors div</p>
        <div class="conteneur">
          <p class="p1">Un premier paragraphe</p>
          <p class="p2">Un deuxième paragraphe</p>
        </div>
        <p>Paragraphe hors div</p>

        <h2>Exemple sans fusion</h2>
        <p>Paragraphe hors div</p>
        <div class="conteneur bordure">
          <p class="p1">Un premier paragraphe</p>
          <p class="p2">Un deuxième paragraphe</p>
        </div>
        <p>Paragraphe hors div</p>
    </body>
</html>
```

```css
body, p{
    margin: 0px;
}
div{
    background-color: #0AE;/*Bleu*/
    margin: 20px 0px;
}
p{
    background-color: #0EA;/*Vert*/
}
.p1{
    margin-top: 20px;
}
.p2{
    margin-bottom: 40px;
}
.bordure{
    border: 1px solid black;
}
```



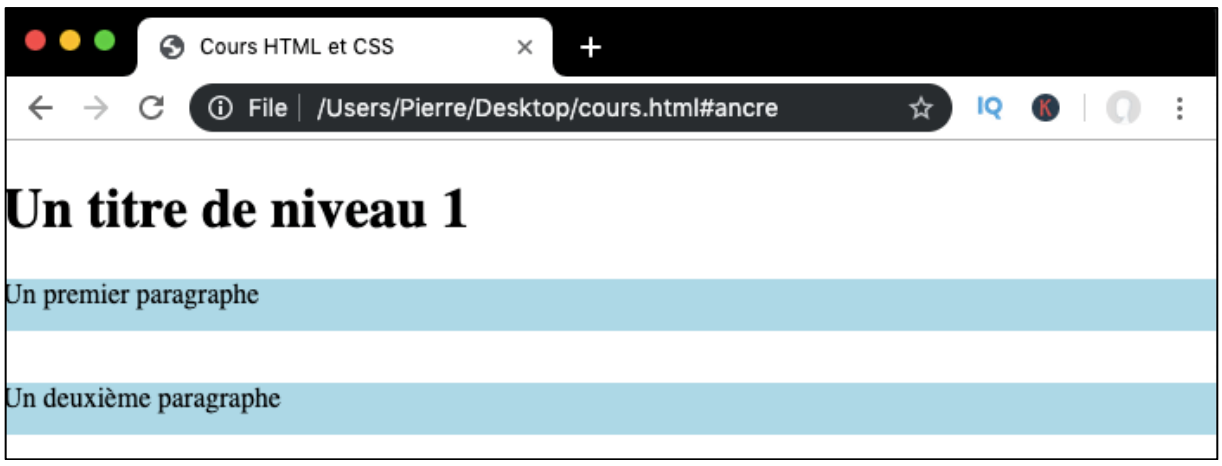
Fusion des marges d'un bloc vide

Finalement, si un élément de type `block` est vide, c'est-à-dire ne possède aucun contenu, bordure, remplissage et qu'on ne lui a pas défini de hauteur fixe alors ses marges hautes et basses vont également fusionner.

Notez bien à nouveau que cet effet de collapse ne va pouvoir avoir lieu qu'avec les marges supérieure et inférieure des éléments et ne va pas se manifester pour les marges gauches et droite.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Un titre de niveau 1</h1>
        <p id="p1">Un premier paragraphe</p>
        <div></div>
        <p id="p2">Un deuxième paragraphe</p>
    </body>
</html>
```

```css
body, p{
  margin: 0px;
}

/*On fixe une hauteur de 30px pour nos paragraphes pour
  *bien voir que cette hauteur est équivalente à l'espace pris par le div*/
p{
  height: 30px;
  background-color: lightBlue;
}

div{
  margin: 30px 0px;
}
```



Utiliser margin pour centrer un élément dans son parent

La propriété `margin` va souvent être utilisée pour centrer horizontalement un élément dans son parent.

Pour faire cela, nous allons devoir définir des marges externes `auto` à gauche et à droite de l'élément qui devra être centré.

Attention toutefois : cela ne va fonctionner que sur des éléments de type `block` car centrer des éléments `inline` (en ligne) dans leur parent n'aurait aucun sens.

De plus, afin de voir l'effet du centrage, il va falloir définir explicitement une largeur pour l'élément qui devra être centré avec la propriété `width` et lui passer une largeur plus petite que celle de son parent.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <div class="conteneur">
            <p class="p1">Un premier paragraphe</p>
            <p class="p2">Un deuxième paragraphe</p>
        </div>
    </body>
</html>
```

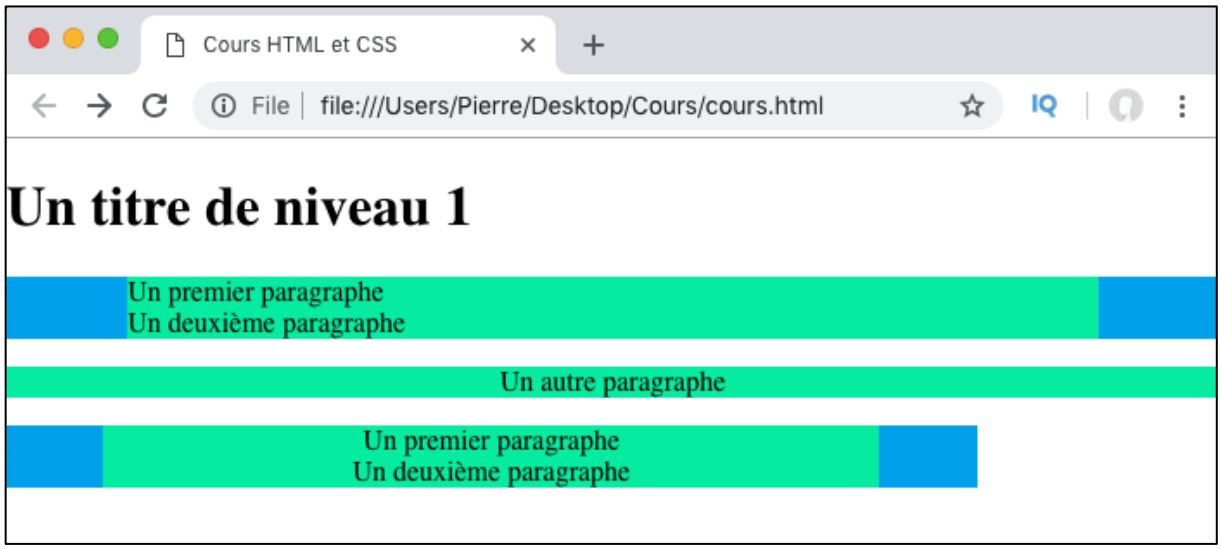
```css
body{
  margin: 0px;
}

div{
  background-color: #0AE;/*Bleu*/
}
p{
  background-color: #0EA;/*Vert*/
  width: 80%;
  margin: 0 auto;
}
```

Notez bien ici que c'est l'élément entier qui est centré dans son conteneur et non pas le contenu de l'élément qui est centré dans l'élément. Pour centrer horizontalement le contenu d'un élément dans l'élément en soi, on utilisera la propriété `text-align` et sa valeur `center`.

Pour centrer l'élément et son contenu, il suffit donc d'utiliser les propriétés `margin` et `text-align` ensemble.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <!--Le div occupe toute la largeur du body et les paragraphes
        sont centrés dans le div-->
        <div>
            <p class="bloc-centre">Un premier paragraphe</p>
            <p class="bloc-centre">Un deuxième paragraphe</p>
        </div>

        <!--Le texte est centré dans l'élément p qui occupe toute la
        largeur de la page-->
        <p class="texte-centre">Un autre paragraphe</p>

        <!--Texte centré dans l'élément p et éléments p centrés dans leur
        parent div. Le div n'est lui pas centré dans la page-->
        <div class="conteneur">
            <p class="bloc-centre texte-centre">Un premier paragraphe</p>
            <p class="bloc-centre texte-centre">Un deuxième paragraphe</p>
        </div>
    </body>
</html>
```

```css
body{
  margin: 0px;
}

div{
  background-color: #0AE;/*Bleu*/
}
p{
  background-color: #0EA;/*Vert*/
}
.bloc-centre{
  width: 80%;
  margin: 0 auto;
}

.texte-centre{
  text-align: center;
}
.conteneur{
  width: 80%;
}
```



La propriété CSS box-sizing

La propriété `box-sizing` va nous permettre de définir quelles boites doivent être inclues dans le calcul de la largeur et de la hauteur d'un élément.

Cette propriété va être très utile pour éviter qu'un élément ne dépasse de son parent à cause de bordures trop large ou de marges internes trop grandes par exemple.

Les interactions entre les différentes propriétés du modèle des boites

Commençons déjà par souligner comment fonctionnent les propriétés liées au modèle des boites ensemble.

Vous pouvez retenir que lorsqu'aucune largeur n'est explicitement définie pour un bloc, alors l'ajout de marges (externes comme internes) et de bordures va compresser le contenu ou les boites internes afin que l'élément ne dépasse pas de son parent conteneur.

En revanche, dès qu'on définit une largeur pour l'élément auquel on applique des marges et / ou des bordures, les différentes tailles des marges et bordures vont venir s'ajouter par défaut à la taille définie et l'élément va ainsi pouvoir potentiellement dépasser de son conteneur.

Le fonctionnement de la propriété box-sizing

La propriété `box-sizing` va nous permettre d'indiquer que l'on souhaite inclure les marges internes et les bordures dans le calcul de la taille d'un élément.

Nous allons pouvoir fournir l'un des mots clefs suivants à cette propriété :

- `content-box` : valeur par défaut. Les dimensions définies pour l'élément vont s'appliquer à sa boite de contenu. Toute marge interne ou bordure ajoutées ensuite vont augmenter la taille de l'élément ;
- `border-box` : les dimensions définies pour l'élément vont s'appliquer à la boite contenant le contenu + le padding + les bordures. En ajoutant ou en augmentant la taille des marges internes ou des bordures, la taille de l'élément ne change pas mais son contenu sera compressé.

Exemple d'utilisation de la propriété box-sizing

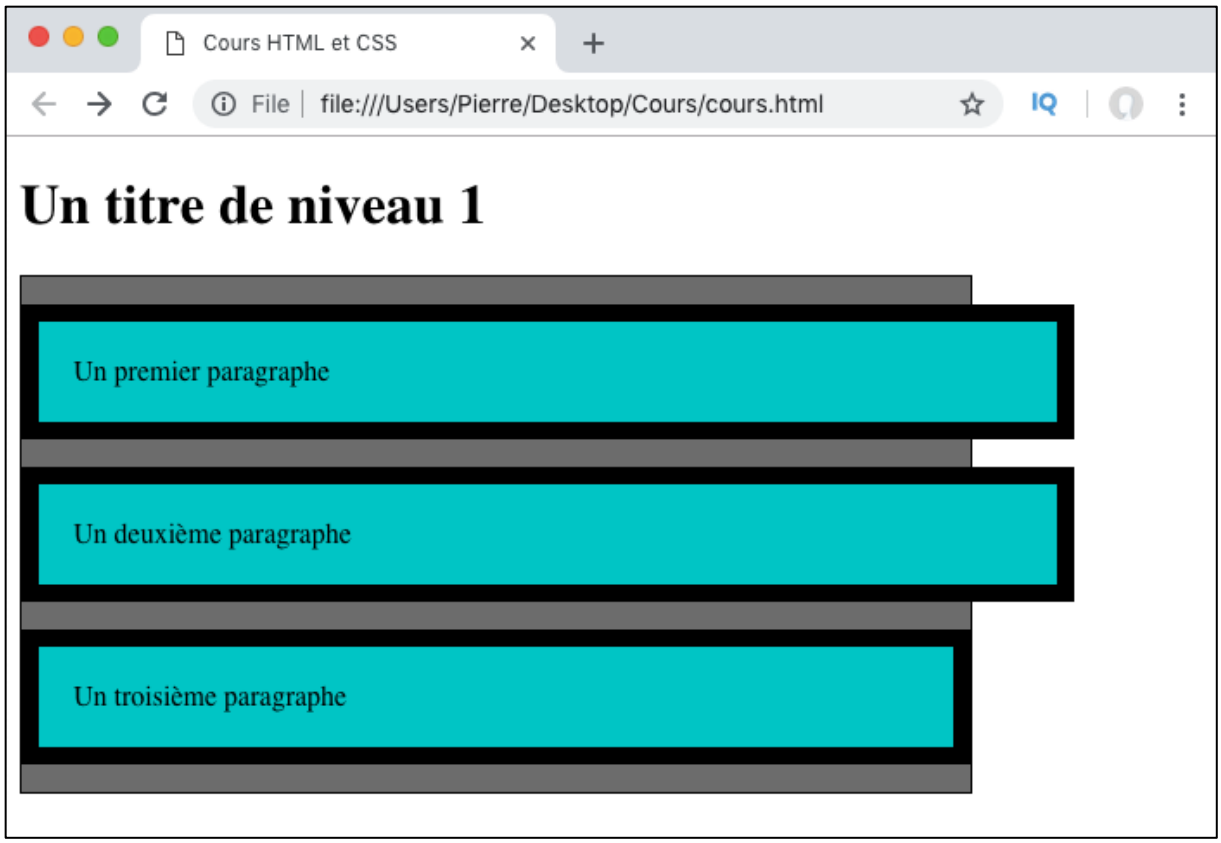
Dans l'exemple ci-dessous, nos paragraphes sont tous des enfants d'un `div class="conteneur"`. La largeur des éléments `p` est fixée à 100% de la taille de leur élément parent.

Par défaut, la largeur de la boite de contenu des paragraphes sera égale à celle du `div` parent. Ensuite, nous ajoutons des marges internes et des bordures à nos paragraphes. La taille des marges et de la bordure va par défaut s'ajouter à la taille définie avec `width` pour nos paragraphes et ceux-ci vont donc dépasser de leur parent.

Nous allons donc utiliser la propriété `box-sizing` et sa valeur `border-box` pour que la largeur définie inclue les marges internes et bordures dans son calcul et afin que notre paragraphe ne dépasse pas de son élément parent.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Un titre de niveau 1</h1>

        <div class="conteneur">
            <p id="p1">Un premier paragraphe</p>
            <p id="p2">Un deuxième paragraphe</p>
            <p id="p3">Un troisième paragraphe</p>
        </div>
    </body>
</html>
```

```css
.conteneur{
  width: 80%;
  background-color: #777;/*Gris*/
  border:1px solid black;
}
p{
  width:100%;
  background-color: #0CC;/*Bleu-vert*/
  padding: 20px;
  border: 10px solid black;
}
#p2{
  box-sizing: content-box;/*Valeur par défaut*/
}
#p3{
  box-sizing: border-box;
}
```


Cours HTML et CSS
File | file:///Users/Pierre/Desktop/Cours/cours.html
Un titre de niveau 1
Un premier paragraphe
Un deuxième paragraphe
Un troisième paragraphe

Créer des bordures arrondies en CSS

Le module CSS3 relatif aux bordures a apporté de nouvelles fonctionnalités nous permettant de personnaliser encore davantage nos bordures. Parmi celles-ci, l'une des nouveautés les plus attendues était la possibilité de créer des bordures arrondies.

Nous allons donc pouvoir créer des bordures arrondies en CSS (ou plus exactement arrondir les bords d'un élément HTML) en utilisant la propriété `border-radius`.

Notez que l'arrondi créé va s'appliquer aux bords de l'élément en soi (ou à son arrière-plan si vous préférez). Cela signifie que nous n'avons pas forcément besoin de définir une quelconque bordure avec `border` pour que `border-radius` s'applique.

Comment sont définis les arrondis avec border-radius ?

La propriété `border-radius` va prendre deux valeurs représentant la dimension du « rayon » sur l'axe des X (axe horizontal) et la dimension du « rayon » sur l'axe des Y (axe vertical) d'une ellipse qui va servir à définir la forme de l'arrondi.

Ces deux valeurs dont être séparées par un slash comme ceci : `border-radius: X/Y`. La deuxième valeur est facultative et si elle est omise elle sera considérée comme égale à la première par défaut.

Note : ici, j'appelle « rayon » sur l'axe de X et des Y les longueurs égales à la moitié de la largeur et à la moitié de la hauteur totale de l'ellipse, c'est à dire la distance du point central de l'ellipse à son bord gauche ou droit dans le plan horizontal et la distance du point central de l'ellipse à son bord supérieur ou inférieur dans le plan vertical. Bien évidemment, parler de rayon pour une ellipse n'a aucun sens d'un point de vue mathématique, ce n'est que pour vous donner une image.

Ces deux valeurs vont pouvoir être des valeurs de type longueur (en `px` par exemple) ou des pourcentages (`%`).

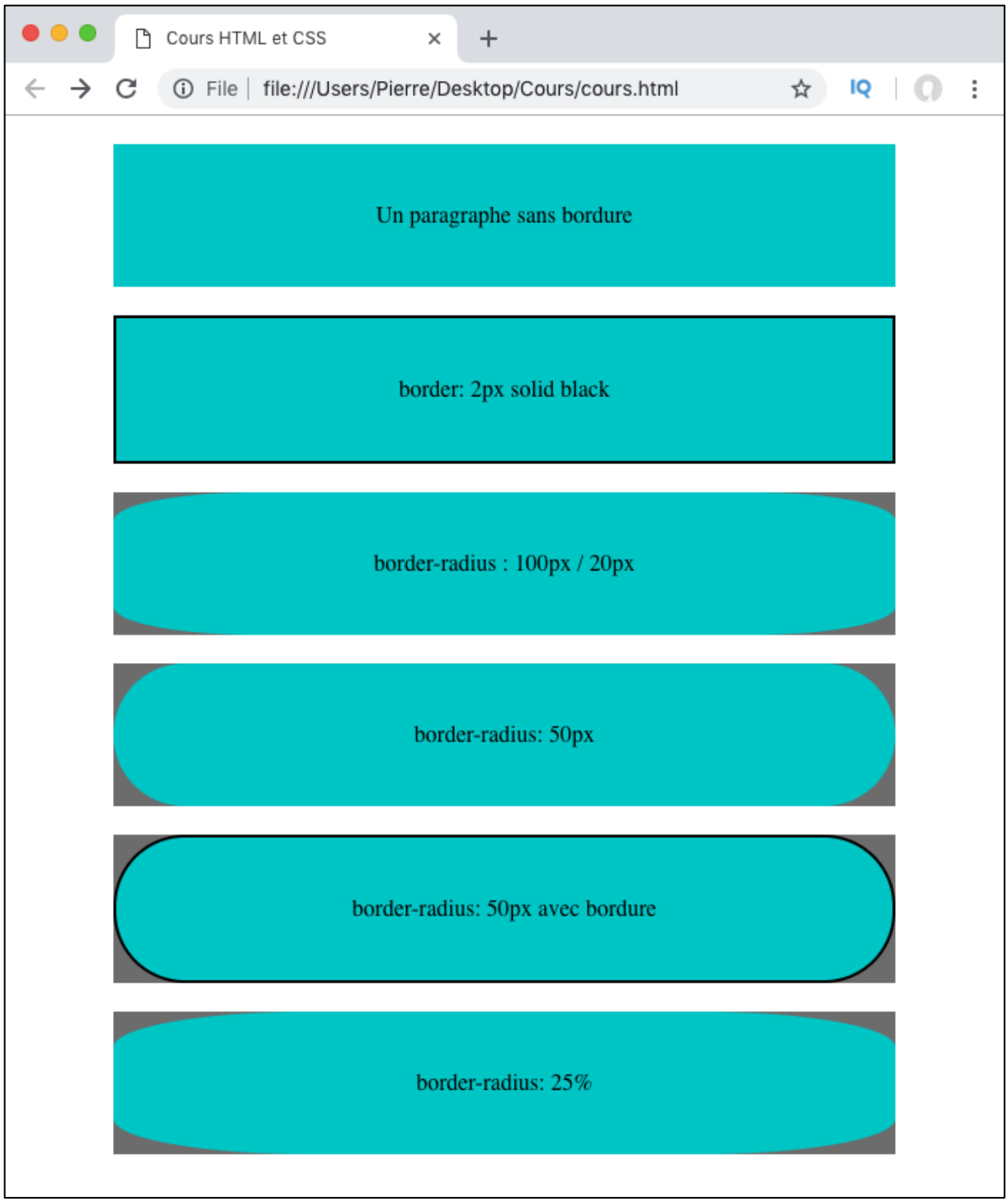
Pour être tout à fait précis, notez déjà que ce sont 4 ellipses qui vont être utilisées pour définir les bordures arrondies d'un élément (une ellipse pour chaque coin).

Pour le moment, nous n'allons mentionner qu'une valeur de rayon horizontale et une valeur de rayon vertical avec `border-radius` ce qui fait que les 4 ellipses vont être de tailles identiques (c'est la raison pour laquelle je parlerai de « l'ellipse » plutôt que « des ellipses »). Nous verrons par la suite comment définir des bordures arrondies différentes pour chaque côté d'un élément.

Prenons directement un premier exemple pour illustrer tout ça :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div><p>Un paragraphe sans bordure</p></div>
        <div><p class="bordure">border: 2px solid black</p></div>
        <div><p class="arrondi1">border-radius : 100px / 20px</p></div>
        <div><p class="arrondi2">border-radius: 50px</p></div>
        <div><p class="arrondi2 bordure">border-radius: 50px avec bordure</p></div>
        <div><p class="arrondi3">border-radius: 25%</p></div>
    </body>
</html>
```

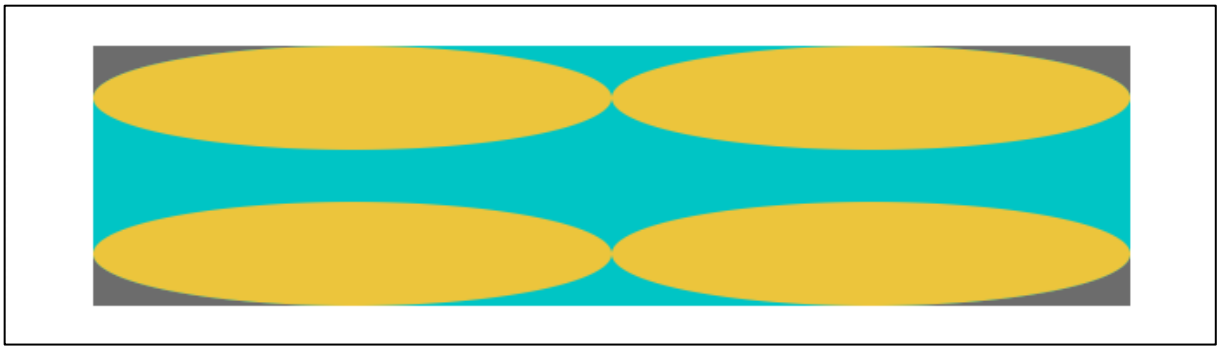
```css
div{
    width: 80%;
    margin: 20px auto; /*Centre les div dans la page*/
    background-color: #777; /*Fond gris*/
}

p{
  width: 100%;
  height: 100%;
  box-sizing: border-box;
  text-align:center; /*Texte centré horizontalement*/
  line-height: 100px; /*Texte centré verticalement*/
  background-color: #0CC; /*Fond bleu vert*/
}

.bordure{
    border: 2px solid black;
}
.arrondi1{
    border-radius : 100px / 20px;
}
.arrondi2{
    border-radius: 50px;/*Identique à border-radius: 50px/50px*/
}
.arrondi3{
  border-radius: 25%;/*Identique à border-radius: 25%/25%*/
}
```



Ici, nous voulons appliquer des bordures arrondies à nos différents paragraphes. On commence par placer chaque paragraphe dans un `div` et par donner une couleur de fond à nos éléments `div` et `p` pour bien voir la bordure arrondie par la suite.

On attribue également une largeur et une hauteur pour nos différents paragraphes égales à celles de leur parent `div` pour pouvoir ensuite utiliser la propriété `box-sizing` et régler d'ores-et-déjà les potentiels problèmes de dépassement des bordures définies avec `border`.

Ensuite, on définit 3 arrondis différents avec `border-radius`. Le premier arrondi est défini avec `border-radius : 100px / 20px`. Cela signifie que chacun des bords arrondis de notre paragraphe va être formé à partir d'une ellipse de largeur maximale de 200px (le « rayon » sur l'axe horizontal * 2) et de hauteur maximale de 40px (le « rayon » sur l'axe vertical * 2). Cela devient très net si on trace ces ellipses à l'intérieur de notre paragraphe :

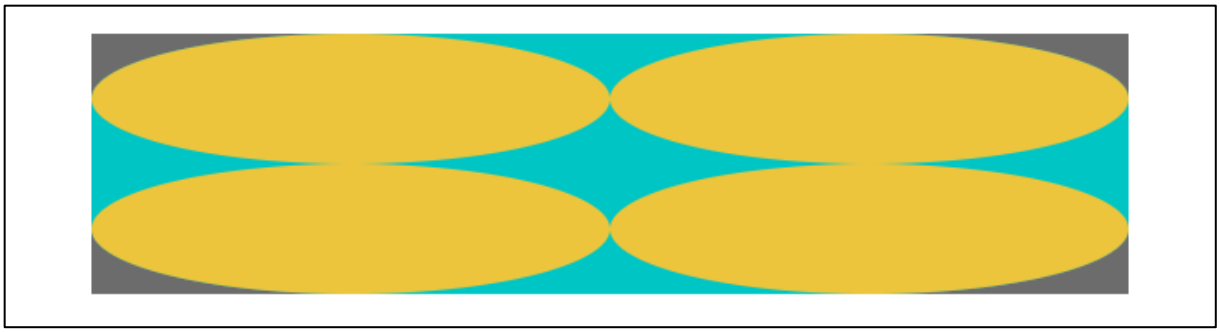


Pour notre deuxième arrondi, on ne précise qu'une valeur pour notre propriété `border-radius` en pixels avec `border-radius : 50px`. Par défaut, la deuxième valeur va donc être également fixée à 50px. Dans ce cas, les bords de notre paragraphe vont être arrondis selon la forme d'une ellipse de rayon horizontal et vertical de 50px, ce qui n'est rien d'autre qu'un cercle.

En effet, lorsque les deux valeurs passées à `border-radius` sont les mêmes, alors nous sommes dans le cas particulier où notre ellipse est un cercle (à l'exception des valeurs en % dont nous allons parler par la suite). Ici, vous devez donc vous imaginer le contour d'un cercle de diamètre = 100px.

Nous utilisons deux fois cet arrondi : une fois dans un paragraphe sans bordures et une fois avec un paragraphe qui possède des bordures. Ici, on précise `border-radius : 25%` ce qui est équivalent à `border-radius : 25%/ 25%`. L'arrondi va ainsi être créé à partir d'une ellipse de rayon horizontal égal à 25% de la largeur du paragraphe et de rayon vertical égal à 25% de la hauteur du paragraphe.

Finalement, notre dernier arrondi utilise des valeurs en pourcentage. Ici, vous devez bien comprendre que le pourcentage défini va être un pourcentage de dimension associée (largeur ou hauteur) du paragraphe.



La propriété border-radius et les valeurs en pourcentage (%)

Il faut faire très attention lorsqu'on définit un `border-radius` avec des valeurs en pourcentage car le ou les pourcentage(s) donné(s) seront exprimés en fonction de la largeur et de la hauteur de la boite représentant l'élément auquel on applique la bordure.

Ainsi, écrire `border-radius : 25%` signifie que le « rayon » sur l'axe horizontal de l'ellipse utilisée pour créer la bordure sera égal à 25% de la largeur de l'élément et que le « rayon » sur l'axe vertical de l'ellipse utilisée pour créer la bordure sera égal à 25% de la hauteur de l'élément. Ainsi, si notre élément a par exemple une largeur de 400px et une hauteur de 100px, il est équivalent d'écrire `border-radius : 25%` et `border-radius : 100px / 25px`.

Ici, vous pouvez retenir la chose suivante : en ne passant qu'une seule valeur à `border-radius` (que ce soit des `px`, `em`, `in`, `viewport related units`, `cm`…), les bordures créées seront toujours issues d'un cercle EXCEPTE dans le cas où l'on utilise une valeur en %.

Bien évidemment, si le boite représentant notre élément est carrée (largeur = hauteur), alors l'arrondi créé sera toujours créé à partir d'un cercle si on ne passe qu'une valeur à `border-radius` et ceci même si la valeur passée est une valeur en `%`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div><p class="arrondi1">border-radius : 50%</p></div>
        <div class="carre"><p class="arrondi1">border-radius : 50%</p></div>
        <div><p class="arrondi2">border-radius: 25% / 50%</p></div>
    </body>
</html>
```

```css
div{
    width: 80%;
    margin: 20px auto; /*Centre les div dans la page*/
    background-color: #777; /*Fond gris*/
}

p{
  width: 100%;
  height: 100%;
  box-sizing: border-box;
  text-align:center; /*Texte centré horizontalement*/
  line-height: 100px; /*Texte centré verticalement*/
  background-color: #0CC; /*Fond bleu vert*/
}
.carre{
  width: 200px;
  height: 200px;
}
.carre p{
  line-height: 200px;
}

.bordure{
    border: 2px solid black;
}
.arrondi1{
    border-radius : 50%;
}
.arrondi2{
    border-radius: 25% / 50%;
}
```

Définir des bordures arrondies différentes

On va tout à fait pouvoir définir un arrondi différent pour chacune des bordures d'un élément. Pour cela, il suffira de passer 2, 3 ou 4 valeurs à la propriété `border-radius` :

- En passant une valeur : la valeur va définir les arrondis des 4 bords de l'élément ;
- En passant deux valeurs : la première valeur va définir l'arrondi des angles supérieur gauche et inférieur droit de l'élément tandis que la seconde valeur va définir l'arrondi des angles supérieur droit et inférieur gauche de l'élément ;
- En passant trois valeurs : la première valeur définit l'arrondi de l'angle supérieur gauche de l'élément, la deuxième valeur définit l'arrondi des angles supérieur droit et inférieur gauche de l'élément tandis que la troisième valeur définit l'arrondi de l'angle inférieur droit de l'élément ;
- En passant quatre valeurs : la première valeur définit l'arrondi de l'angle supérieur gauche de l'élément, la deuxième valeur définit l'arrondi de l'angle supérieur droit, la troisième valeur définit l'arrondi de l'angle inférieur droit tandis que la quatrième valeur définit l'arrondi de l'angle inférieur gauche.

Notez par ailleurs que la propriété `border-radius` est une notation raccourcie des propriétés suivantes :

- `border-top-left-radius` : Définit l'arrondi de l'angle supérieur gauche de l'élément ;

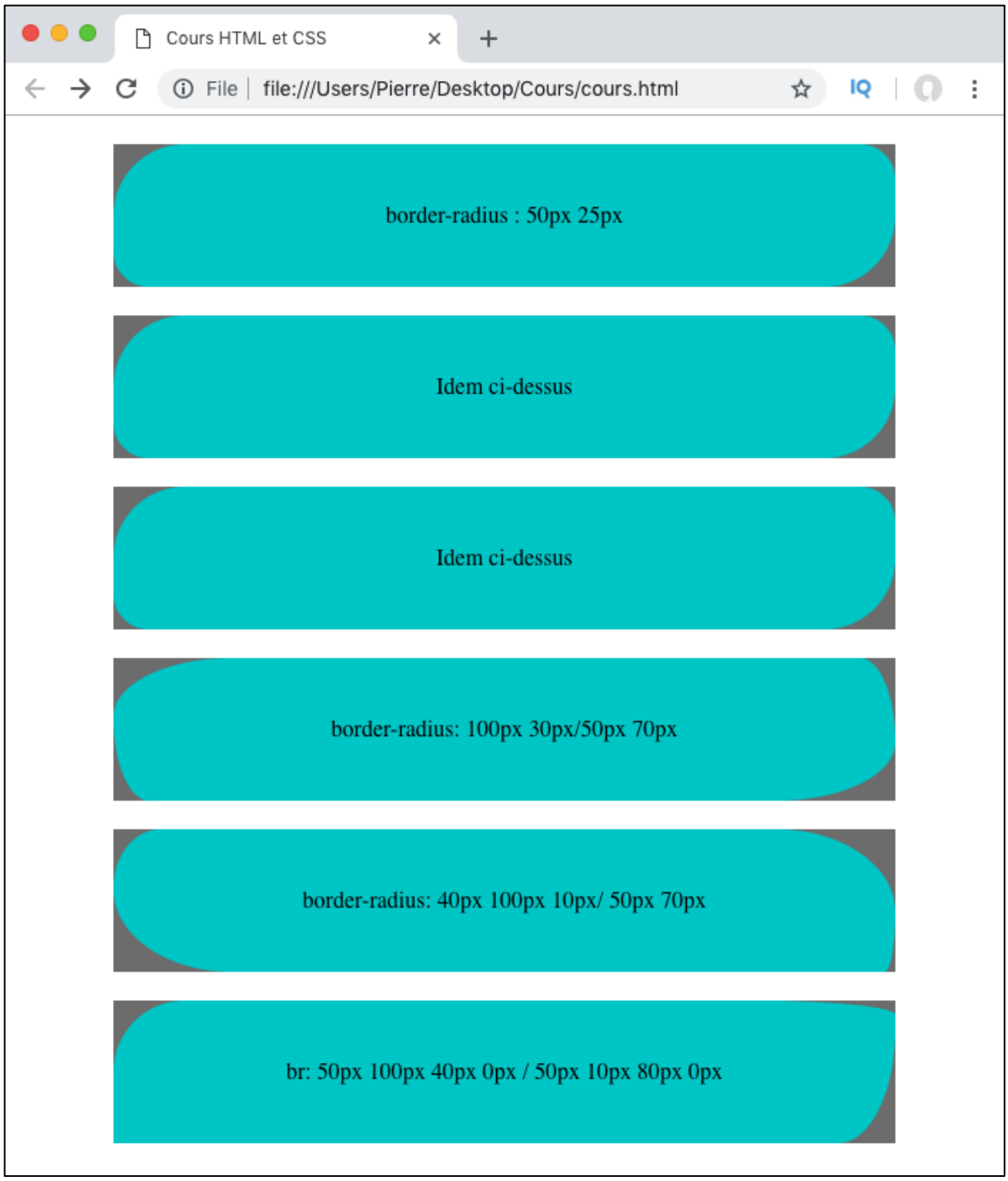
- `border-top-right-radius` : Définit l'arrondi de l'angle supérieur droit de l'élément ;
- `border-bottom-right-radius` : Définit l'arrondi de l'angle inférieur droit de l'élément ;
- `border-bottom-left-radius` : Définit l'arrondi de l'angle inférieur gauche de l'élément.

Une nouvelle fois, nous allons pouvoir mentionner deux « rayons » différents pour chacune des bordures arrondies. Dans le cas où on utilise la notation raccourcie `border-radius`, il va cependant falloir faire bien attention à l'ordre des valeurs lorsqu'on souhaite définir des arrondis différents car celui-ci peut sembler contre intuitif à première vue : nous allons devoir commencer par passer toutes les valeurs de « rayons » horizontaux pour nos arrondis PUIS les valeurs de « rayons » verticaux, en séparant ces deux groupes par un slash.

Notez également qu'en utilisant les propriétés complètes `border-top-left-radius`, `border-top-right-radius`, etc. plutôt que la notation raccourcie `border-radius` il ne faudra PAS indiquer de slash pour séparer les valeurs des deux « rayons » de l'ellipse si on souhaite en préciser deux différentes.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div><p class="arrondi1">border-radius : 50px 25px</p></div>
        <div><p class="arrondi2">Idem ci-dessus</p></div>
        <div><p class="arrondi3">Idem ci-dessus</p></div>
        <div><p class="arrondi4">border-radius: 100px 30px/50px 70px</p></div>
        <div><p class="arr5">border-radius: 40px 100px 10px/ 50px 70px</p></div>
        <div><p class="arr6">br: 50px 100px 40px 0px/50px 10px 80px 0px</p></div>
    </body>
</html>
```

```css
div{
    width: 80%;
    margin: 20px auto; /*Centre les div dans la page*/
    background-color: #777; /*Fond gris*/
}

p{
  width: 100%;
  height: 100%;
  box-sizing: border-box;
  text-align:center; /*Texte centré horizontalement*/
  line-height: 100px; /*Texte centré verticalement*/
  background-color: #0CC; /*Fond bleu vert*/
}

.arrondi1{
  border-radius: 50px 25px;
}
.arrondi2{
  border-radius: 50px 25px / 50px 25px;
}
.arrondi3{
  border-top-left-radius: 50px;
  border-bottom-right-radius: 50px;
  border-top-right-radius: 25px;
  border-bottom-left-radius: 25px;
}
.arrondi4{
  border-radius: 100px 30px/50px 70px;
}
.arr5{
  border-radius: 40px 100px 10px/ 50px 70px;
}
.arr6{
  border-top-left-radius: 50px;
  border-bottom-right-radius: 40px 80px;
  border-top-right-radius: 100px 10px;
}
```



Pour notre premier arrondi, on définit un `border-radius : 50px` pour les angles supérieur gauche et inférieur droit de l'élément et un `border-radius : 25px` pour les angles supérieur droit et inférieur gauche de l'élément.

Notre deuxième arrondi est l'équivalent du premier mais écrit en précisant les valeurs des deux rayons pour chacun de nos angles (ici, les valeurs des deux rayons sont identiques pour chaque angle).

Notre troisième arrondi correspond à une troisième façon d'arriver au même résultat que notre premier arrondi, en utilisant cette fois-ci les propriétés complètes `border-top-left-radius`, `border-top-right-radius`, etc. plutôt que la notation raccourcie.

Pour notre quatrième arrondi, les angles supérieur gauche et inférieur droit de l'élément seront formés à partir d'une ellipse de rayon 100px (horizontal) et 50px (vertical). Les angles supérieur droit et inférieur gauche de l'élément emprunteront eux leurs arrondis à une ellipse de rayon horizontal de 30px et de rayon vertical de 70px.

Notre cinquième déclaration d'arrondi est un peu plus complexe à appréhender. Ici, il faut bien comprendre qu'on définit 3 rayons horizontaux pour nos bordures arrondis de 40px, 100px et 10px. La première valeur sera utilisée pour définir l'arrondi horizontal de l'angle supérieur gauche de l'élément, la seconde pour les angles supérieur droit et inférieur gauche et la troisième pour l'angle inférieur droit.

Ensuite, on ne définit que 2 valeurs de rayons verticaux de 50px et 70px. La première valeur sera utilisée pour les angles supérieur gauche et inférieur droit tandis que la deuxième valeur sera utilisée pour les angles supérieur droit et inférieur gauche.

Finalement, le dernier exemple d'arrondi sert à illustrer comment créer des arrondis utilisant deux rayons différents avec nos propriétés complètes `border-top-left-radius`, `border-top-right-radius`, etc. Je vous rappelle qu'il ne faut pas préciser de slash en utilisant ces propriétés mais écrire les deux valeurs de rayons à la suite.

La gestion des valeurs d'arrondis trop grandes (valeurs aberrantes)

La création de bordures arrondies se fait au moyen d'ellipses ou, dans certains cas particuliers, de cercles. La création de bordures arrondies se fait au moyen d'ellipses ou, dans certains cas particuliers, de cercles. Parfois, cependant, il peut arriver que des valeurs aberrantes soient fournies.

Définition d'une valeur aberrante pour border-radius

Les valeurs passées à la propriété `border-radius` vont être considérées comme aberrantes et les bordures arrondies vont être redimensionnées dès que la règle « deux bordures adjacentes ne doivent pas se chevaucher » ne sera plus respectée c'est-à-dire dans les situations suivantes :

- Si l'une (ou chacune) des deux sommes des deux « rayons » dans l'axe vertical des ellipses servant à créer les bordures supérieure gauche et inférieure gauche et supérieure droite et inférieure droite dépasse la valeur de la hauteur de la boite de l'élément ;
- Si l'une (ou chacune) des deux sommes des deux « rayons » dans l'axe horizontal des ellipses servant à créer les bordures supérieure gauche et supérieure droite et inférieure gauche et inférieure droite dépasse la valeur de la largeur de la boite de l'élément.

Dans ce cas-là, les valeurs passées à `border-radius` vont être réduites de manière proportionnelle jusqu'à ce que la transition entre les deux bordures arrondies soit fluide (pour être tout à fait exact d'un point de vue mathématique, il faudrait dire « jusqu'à ce qu'on puisse tracer une tangente qui soit parallèle au côté de la boite de l'élément »).

Trouver les valeurs corrigées automatiquement par le CSS à partir des valeurs aberrantes

Imaginons par exemple qu'on ait un paragraphe avec les dimensions suivantes `width : 400px; height : 100px` et qu'on tente de lui appliquer les bordures arrondies suivantes : `border-radius : 100px 250px 200px 300px / 10px 30px 70px 80px`.

Cela est équivalent à définir les bordures arrondies suivantes :

- `border-top-left-radius : 100px 10px` ;
- `border-top-right-radius : 250px 30px` ;
- `border-bottom-right-radius : 200px 70px` ;
- `border-bottom-left-radius : 300px 80px`.

Faisons maintenant les sommes des rayons 2-à-2 de nos bordures dans les axes horizontal et vertical :

- Axe horizontal (top-left + top-right) : 100 + 250 = 350px ;
- Axe horizontal (bottom-right + bottom-left) : 200 + 300 = 500px ;
- Axe vertical (top-left + bottom-left) : 10 + 80 = 90px ;
- Axe vertical (top-right + bottom-right) : 30 + 70 = 100px.

Ici, nous allons avoir un problème pour l'axe horizontal avec les bordures inférieures droite et gauche puisque la valeur totale des deux rayon, 500px, est supérieure à la largeur de l'élément qui est 400px.

Ainsi, TOUTES les bordures arrondies de l'élément vont être redimensionnées de façon proportionnelle jusqu'à arriver à la première valeur acceptable pour les valeurs qui posent problème, c'est-à-dire lorsque la somme des rayons (axe horizontal) des deux bordures inférieures droite et gauche fera 400px.

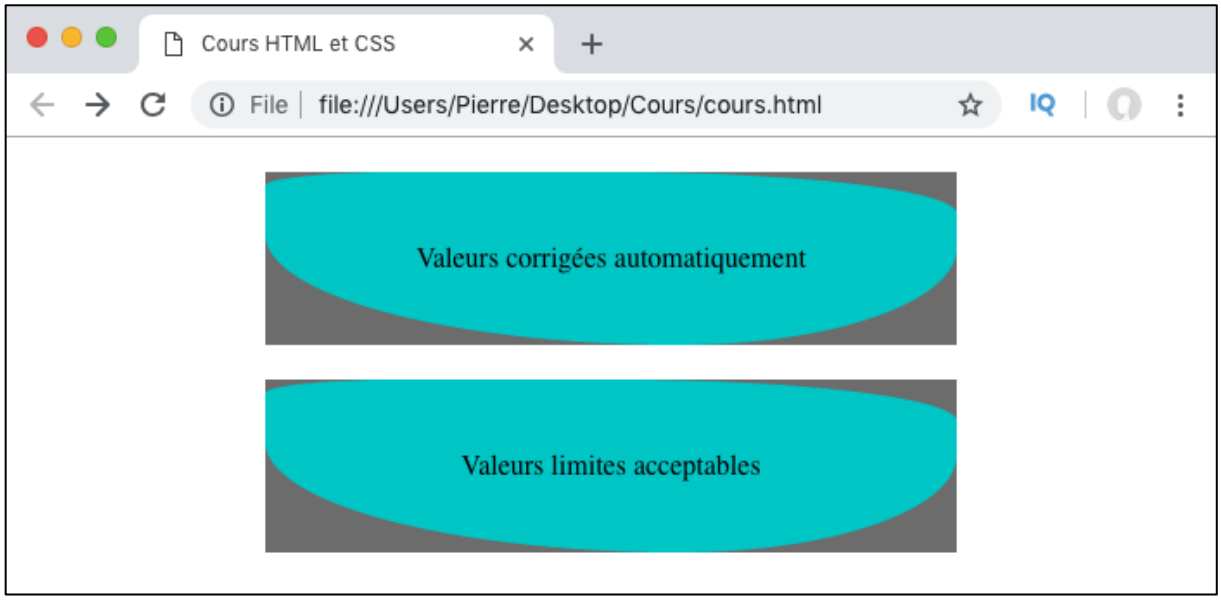
Nous n'avons donc qu'à faire un calcul de proportionnalité en utilisant une règle de 3 pour connaitre les valeurs qui vont être attribuées. Pour cela, il faut déjà trouver le coefficient de proportionnalité qui nous permet de passer de 500 à 400. On a donc : 400 * 100 / 500 = 0,8.

Il ne nous reste donc plus qu'à multiplier toutes les dimensions données à `border-radius` par 0,8 pour obtenir les valeurs corrigées automatiquement par le CSS soit : `border-radius : 80px 200px 160px 240px / 8px 24px 56px 64px.`

Regardez plutôt le résultat ci-dessous pour vous en convaincre. J'ai également recréé les bordures dans le dernier exemple telles qu'elles seraient si elles n'avaient pas été ajustées :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div><p class="arrondi1">Valeurs corrigées automatiquement</p></div>
        <div><p class="arrondi2">Valeurs limites acceptables</p></div>
    </body>
</html>
```

```css
div{
    width: 400px;
    margin: 20px auto; /*Centre les div dans la page*/
    background-color: #777; /*Fond gris*/
}

p{
  width: 100%;
  height: 100%;
  box-sizing: border-box;
  text-align:center; /*Texte centré horizontalement*/
  line-height: 100px; /*Texte centré verticalement*/
  background-color: #0CC; /*Fond bleu vert*/
}

.arrondi1{
    border-radius : 100px 250px 200px 300px / 10px 30px 70px 80px;
}
.arrondi2{
    border-radius : 80px 200px 160px 240px / 8px 24px 56px 64px;
}
```



Vous pouvez également noter ici que :

- Si nous avons plusieurs valeurs aberrantes, alors nous utiliserons le coefficient de proportionnalité de la valeur la plus aberrante afin que tous les arrondis soient de tailles acceptables ;
- Si une valeur est aberrante en soi (c'est-à-dire si une la valeur d'un rayon de l'ellipse servant à définir bordure arrondie est déjà supérieure à la taille de la boite de l'élément), nous procéderons exactement de la même manière que précédemment pour trouver la valeur qui sera définie automatiquement en CSS ;
- Ces règles de correction des valeurs s'appliquent pour tous types de valeurs passées, que ce soit des valeurs en `px` ou en `%`.

PARTIE VI

Positionnement et affichage

Gérer l'affichage des éléments en CSS

La propriété CSS display est une propriété très puissante puisqu'elle va nous permettre de modifier la façon dont un élément va s'afficher dans la page : en ligne, sous forme de bloc, etc. et donc la façon dont il va se comporter avec ses voisins.

Nous avons déjà eu l'occasion de parler de l'affichage des éléments dans la leçon expliquant les différences entre les éléments de niveau block et de niveau inline.

En CSS3, la propriété display accepte de nombreuses valeurs différentes ce qui va nous permettre de choisir précisément comment chaque élément HTML doit être affiché dans la page.

Dans cette nouvelle leçon, nous allons expliquer comment fonctionne la propriété display en détail et constater son impact sur l'affichage des éléments en utilisant ses différentes valeurs.

Rappels sur la définition du type d'affichage d'un élément par défaut

Le type d'affichage d'un élément va toujours être défini en CSS via la propriété display. Si cette propriété n'est pas explicitement renseignée pour un élément, la valeur par défaut de qui est display: inline sera appliquée à l'élément.

Cependant, vous devez bien comprendre que lorsque vous ouvrez une page dans un navigateur, le navigateur va appliquer des styles par défaut à chaque élément HTML afin d'améliorer le rendu et pour servir de solution de secours si des styles n'ont pas été appliqués par les développeurs de la page.

Parmi ces styles par défaut appliqués par n'importe quel navigateur (et dépendant de chaque navigateur, attention !) se trouve la définition du type d'affichage ou du display pour chaque élément.

Pour savoir quel type de display appliquer par défaut à chaque élément, la plupart des navigateurs sérieux et connus se basent sur les recommandations du W3C (ou du WHATWG, car il est souvent difficile de savoir lequel de ces deux groupes est à l'origine d'une recommandation).

Attention cependant : encore une fois, ce ne sont que des recommandations et chaque navigateur est libre de ne pas en tenir compte et de définir une autre valeur de display pour chaque élément, ce qui peut en pratique arriver pour certains éléments particuliers ou dans certaines situations peu courantes.

Ceci étant dit, nous allons maintenant apprendre à modifier nous-mêmes le type d'affichage d'un élément en utilisant la propriété display pour ne pas avoir à dépendre de l'implémentation par les différents navigateurs.

Inner display et outer display

Plus haut, j'ai dit que la propriété CSS `display` affectait la façon dont un élément s'affichait dans une page.

Plus précisément, cette propriété affecte la façon dont l'élément va générer les boites le composant. Je vous rappelle qu'en HTML5 et CSS3, tout élément HTML doit être vu comme un empilement de boites :

- Boite n°1 (la plus interne) : contenu de l'élément ;
- Boite n°2 : boite n°1 + marges internes ;
- Boite n°3 : boite n°2 + bordures ;
- Boite n°4 : boite n°3 + marges externes.

La propriété `display` va ainsi impacter la génération de l'*outer display* qui correspond au comportement de l'élément globalement (qu'on peut représenter via la boite globale ou boite n°4) dans le flux de la page et donc par rapport aux autres éléments mais également de l'*inner display* de l'élément qui correspond à la façon dont l'élément va créer ses boites internes (excepté pour les éléments remplacés qui sont en dehors de la portée du CSS).

Ainsi, la propriété `display` va toujours implicitement recevoir deux valeurs : une première pour définir l'*outer display* et une seconde pour définir l'*inner display*.

Cependant, en pratique, nous ne mentionnerons explicitement qu'une valeur cet laisserons le CSS définir la deuxième valeur par défaut. Retenez toutefois bien que la propriété `display` définira quand même toujours un *outer display* et un *inner display*.
Ainsi, lorsqu'on mentionne `display : block` par exemple, la « vraie » valeur complète de `display` va être `display : block flow`.

Dans le cas où nous voudrions une deuxième valeur autre que la valeur par défaut, nous n'aurons pas non plus besoin de préciser deux valeurs en pratique puisque le CSS a implémenté des mots clefs composés comme `inline-block` qui sont là pour répondre à ces cas.

Pour information, voici les versions « raccourcies » des valeurs données à `display` que vous devriez toujours utiliser et les valeurs complètes pour référence. Par d'inquiétude, nous allons indiquer par la suite à quoi correspond chaque valeur.

Valeur raccourcie	Valeur complète	Comportement
none	—	L'élément ne s'affiche pas et ne génère aucune boite
contents	—	L'élément ne génère aucune boite à l'affichage mais ses enfants oui
block	block flow	Elément de niveau block (block-level) et boites internes de type block (block container)

Valeur raccourcie	Valeur complète	Comportement
flow-root	block flow-root	Elément de niveau block (block-level) et boites internes de type block (block container) établissant un nouveau contexte de formatage
inline	inline flow	Elément de niveau inline (inline-level) et boites internes de type inline
inline-block	inline flow-root	Elément de niveau inline (inline-level) et boites internes de type block (block container)
run-in	run-in flow	Elément de type run-in (inline avec des règles spéciales)
list-item	block flow list-item	Elément block-level avec block container de type block qui crée également une boite contenant un marqueur
inline list-item	inline flow list-item	Elément inline-level qui crée également une boite contenant un marqueur
flex	block flex	Élément block-level avec boites internes flexibles (flex container)
inline-flex	inline flex	Élément inline-level avec boites internes flexibles (flex container)
table	block table	Élément block-level avec boites internes de type table
inline-table	inline table	Élément inline-level avec boites internes de type table
grid	block grid	Élément block-level avec boites internes de type grille (grid)
inline-grid	inline grid	Élément inline-level avec boites internes de type grille (grid)

Les valeurs de display-outside de display

Nous allons déjà pouvoir commencer par définir l'outer display d'un élément avec `display`, c'est-à-dire le type d'affichage de l'élément en soi et son comportement visuel par rapport aux autres.

Si on exclut les valeurs particulières `display : none` et `display : contents`, nous ne pouvons définir l'outer display d'un élément qu'avec trois valeurs différentes :

- Avec la valeur `inline` : l'élément va être de niveau inline ;

- Avec la valeur `block` : l'élément va être de niveau block ;
- Avec la valeur `run-in` : l'élément va être de niveau inline avec des règles particulières.

Notez que pour chacune de ces trois valeurs outer display, la valeur de l'inner display associée par défaut est `flow`.

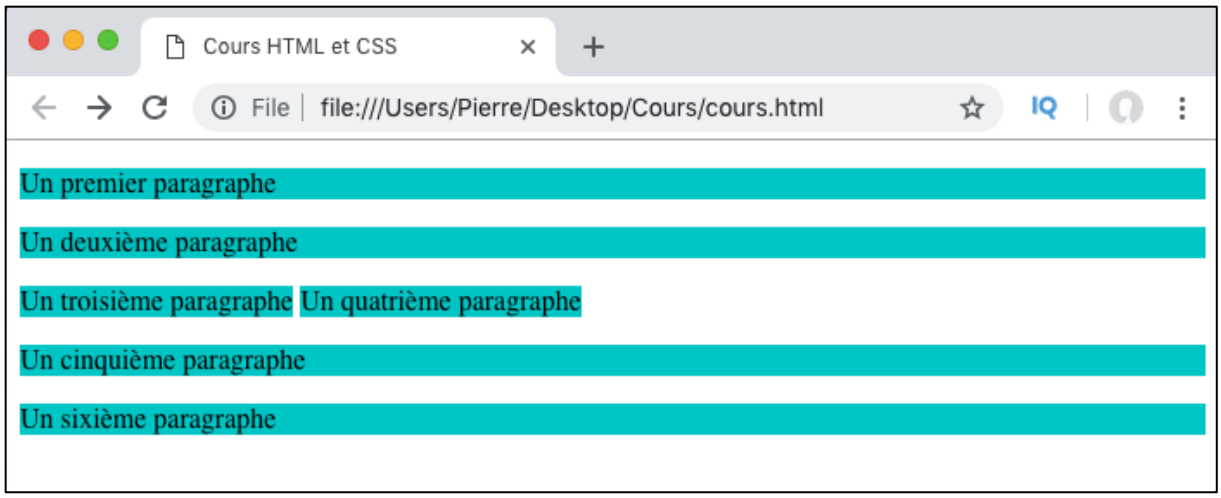
Display : inline

En précisant un `display : inline` (qui est la valeur raccourcie de `display : inline flow`), on définit un élément de niveau `inline` (inline-level element) ou tout simplement de « type » `inline`. Un élément de niveau `inline` a les caractéristiques suivantes :

- Un élément de type `inline` ne va occuper que la largeur nécessaire à l'affichage de son contenu par défaut ;
- Les éléments de type `inline` vont venir essayer de se placer en ligne, c'est-à-dire à côté (sur la même ligne) que l'élément qui les précède dans le code HTML ;
- Un élément de type `inline` peut contenir d'autres éléments de type `inline` mais ne peut pas contenir d'éléments de type `block`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un premier paragraphe</p>
        <p>Un deuxième paragraphe</p>
        <p class="enligne">Un troisième paragraphe</p>
        <p class="enligne">Un quatrième paragraphe</p>
        <p>Un cinquième paragraphe</p>
        <p>Un sixième paragraphe</p>
    </body>
</html>
```

```css
p{
  background-color: #0CC;
}

.enligne{
  display: inline;
}
```



Par défaut, les éléments `p` possèdent un `display : block`. Ici, nous définissons un `display : inline` pour deux d'entre eux ; ces deux paragraphes en particulier vont donc se comporter comme des éléments `inline`.

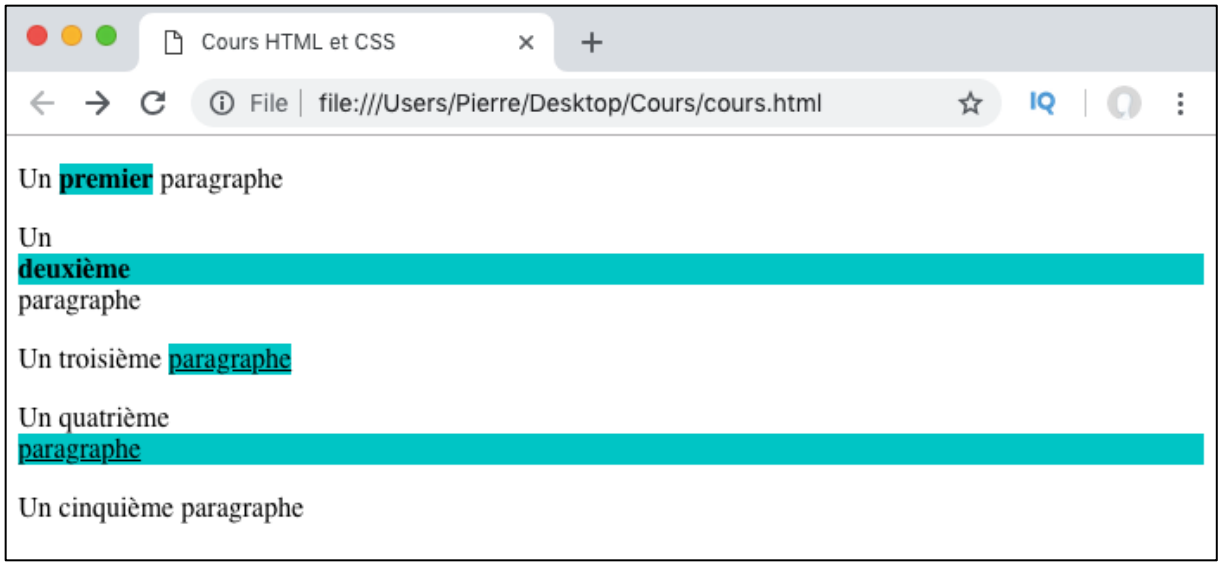
Display : block

En précisant un `display : block` (valeur raccourcie de `display : block flow`), on définit un élément de niveau `block` (block-level element) ou encore de « type » `block`. Un élément de niveau `block` va posséder les caractéristiques suivantes :

- Un élément de type `block` va toujours prendre toute la largeur disponible au sein de son élément parent (ou élément conteneur) ;
- Un élément de type `block` va toujours « aller à la ligne » (créer un saut de ligne avant et après l'élément), c'est-à-dire occuper une nouvelle ligne dans une page et ne jamais se positionner à côté d'un autre élément par défaut ;
- Un élément de type `block` peut contenir d'autres éléments de type `block` ou de type `inline`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un <strong>premier</strong> paragraphe</p>
        <p>Un <strong class="bloc">deuxième</strong> paragraphe</p>
        <p>Un troisième <a href="#">paragraphe</a></p>
        <p>Un quatrième <a href="#" class="bloc">paragraphe</a></p>
        <p>Un cinquième paragraphe</p>
    </body>
</html>
```

```css
strong, a{
  background-color: #0CC;
}

.bloc{
  display: block;
}
```



Les éléments `strong` et `a` ont un type d'affichage inline qui leur est attribué par défaut. Ici, nous changeons ce type d'affichage pour un `display : block` pour notre deuxième élément `strong` et deuxième lien. Ces deux éléments vont donc se comporter comme des éléments de type `block`.

Display : run-in

En précisant un `display : run-in` (valeur raccourcie de `display : run-in flow`), on définit un élément de type `inline` avec un comportement spécial : l'élément va essayer de fusionner / s'insérer dans l'élément de type `block` suivant.

Notez que cette valeur ne fait pas encore partie des recommandations officielles du W3C et est toujours en développement. A éviter pour un développement en production donc.

Les valeurs de display-inside de display

Comme je l'ai précisé plus haut, nous ne préciserons généralement qu'une seule valeur à `display` et laisserons le CSS appliquer la deuxième valeur par défaut. Dans la majorité des cas, ce sera la valeur liée à l'outer display qui sera précisée.

Cependant, pour certains éléments particuliers et dans certains contextes nous passerons plutôt une valeur d'inner display à la propriété `display` (et lui laisserons donc appliquer l'outer display lié par défaut).

Ce sont ces valeurs qui vont nous intéresser ici et notamment les valeurs d'affichage table, flex, grid et list-item.

Inner display: table

En précisant un display : table a un élément, l'élément va visuellement se comporter comme un tableau. Nous étudierons la création de tableaux en HTML plus tard dans ce cours.

La valeur complète du display par défaut est display : block table. A partir de là, vous pouvez déduire qu'un tableau en HTML a un outer display de type block par défaut.

Notez ici que les tableaux vont avoir des structures d'affichage complexes puisqu'ils vont être composés de lignes et de cellules.

Pour rendre complètement le comportement visuel d'un tableau, nous allons également pouvoir utiliser les valeurs suivantes pour display sur certains éléments en répliquant la structure d'un tableau « normal » même si nous essayerons d'éviter de faire ça tant que possible pour des raisons évidentes de sémantique.

- display : table-header-group : l'élément se comporte visuellement comme un élément thead ;
- display : table-footer-group : l'élément se comporte visuellement comme un élément tfoot ;
- display : table-row-group : l'élément se comporte visuellement comme un élément tbody ;
- display : table-row : l'élément se comporte visuellement comme un élément tr ;
- display : table-cell : l'élément se comporte visuellement comme un élément td ;
- display : table-column-group : l'élément se comporte visuellement comme un élément colgroup ;
- display : table-column : l'élément se comporte visuellement comme un élément col.

Nous pouvons également utiliser display : table-caption pour qu'un élément se comporte comme un élément caption. Cette valeur de display va créer un block avec un comportement relatif à celui du tableau.

Inner display: flex

En attribuant un display : flex a un élément, l'élément va visuellement se comporter comme une boite flexible, ce qui signifie que l'élément va se comporter comme un élément de type block pour l'outer display mais que l'intérieur de l'élément va suivre le modèle des boites flexibles ou flexbox (il va établir un nouveau contexte de formatage de type flex).

Nous allons consacrer une leçon au modèle des boites flexibles ou « flexbox » CSS dans ce cours car ce modèle est très intéressant pour créer des pages qui vont s'adapter à tous les écrans ou « responsives ».

La valeur complète de display : flex est display : block flex.

Inner display: grid

En attribuant un display : grid a un élément, l'élément va visuellement se comporter comme une grille : l'élément en soi va être de niveau block et va créer en interne un contexte de formatage de type grille ou grid c'est-à-dire disposer son contenu selon le modèle des grilles.

La valeur complète du display : grid est display : block grid.

Un cas particulier d'affichage : le display : list-item

Lorsque l'on crée un élément de liste en HTML avec l'élément li, on crée au final deux boites : une boite contenant la puce ou marqueur et une autre contenant le contenu textuel de notre élément de liste.

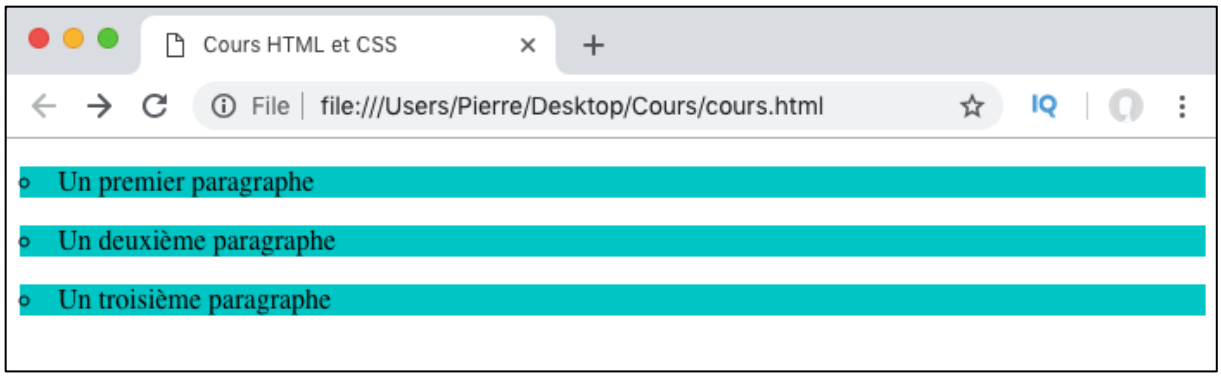
En attribuant un display : list-item a un élément, on va pouvoir recréer ce comportement et faire en sorte que l'élément se comporte comme un élément de liste et génère un marqueur et une boite de type block par défaut.

La valeur complète d'un display : list-item est display : block flow list-item. Ici, la valeur complète de display est composée de trois valeurs.

C'est tout à fait normal au sens où le mot clef list-item ne sert véritablement qu'à générer un marqueur de liste et ne dicte ni le comportement de l'outer display ni celui de l'inner display.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un premier paragraphe</p>
        <p>Un deuxième paragraphe</p>
        <p>Un troisième paragraphe</p>
    </body>
</html>
```

```
p{
  display: list-item;
  list-style-type: circle;/*Les marqueurs un puces seront des cercles*/
  list-style-position: inside;/*Les marqueurs sont dans l'élément*/
  background-color: #0CC;
}
```



Ici, nous attribuons un `display : list-item` à tous les paragraphes de nos pages. Ces derniers vont donc se comporter comme des éléments de liste et avoir chacun un marqueur ou puce.

On en profite pour définir l'apparence et la position des puces car par défaut la position est `outside` et comme mes paragraphes sont collés au bord gauche de ma page les puces seraient en dehors de la page. Notez qu'on pourrait également conserver un `list-style-position: outside` et ajouter une `margin-left`.

Les valeurs de display composées héritées du CSS2 : inline-block, etc.

Jusqu'au CSS2, on avait imaginé une syntaxe avec un mot clef unique à fournir en valeur de la propriété `display`.

Dans ce contexte, cependant, comment faire pour ne modifier que l'outer display ou que l'inner display quand on veut obtenir un comportement qui n'est pas le comportement par défaut ?

La réponse a été l'introduction de mots clefs composés dont nous héritons aujourd'hui comme par exemple le mot clef `inline-block` qui est l'équivalent d'un `display : inline flow-root`.

Ces mots clefs composés font toujours partie des recommandations du W3C et vous êtes donc invités à les utiliser même s'ils ne sont finalement que le reflet des limitations passées.

Display : inline-block

En attribuant un `display : inline-block` a un élément, l'élément en soi va être de niveau `inline` (l'outer display est `inline`) tandis que le contenu de l'élément va se comporter comme un `block`.

La valeur `display : inline-block` est l'équivalent de `display : inline flow-root`.

La valeur `display : inline-block` va ainsi être intéressante pour placer des éléments en ligne tout en pouvant mettre en forme le contenu de ceux-ci et notamment pour pouvoir attribuer

une taille précise à chaque contenu (je vous rappelle que cela n'est possible qu'avec des éléments / boites de type block).

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un <span>premier</span> paragraphe</p>
        <p>Un <span class="bloc">deuxième</span> paragraphe</p>
        <p>Un <span class="ib">troisième</span> paragraphe</p>
    </body>
</html>
```

```css
span{
  width: 200px;
  background-color: #0CC;
}

.bloc{
  display: block;
}
.ib{
  display: inline-block;
}
```

Ici, on commence par définir une largeur fixe pour nos différents éléments span avec la propriété width.

On n'indique pas de valeur particulière pour le display de notre premier élément span. Celui-ci aura donc un display : inline qui est le type d'affichage par défaut des éléments span. En tant qu'élément inline, il va ignorer la propriété width.

On indique un display : block pour notre deuxième span. Il va donc se comporter comme un élément de type block et occuper sa propre ligne mais tenir compte de la largeur passée.

On passe finalement un display : inline-block à notre troisième élément span. Celui-ci va donc se comporter comme un élément inline pour son outer display mais ses boites internes vont pouvoir être mises en forme comme un élément block. L'élément va donc rester en ligne mais nous allons pouvoir définir une largeur précise pour celui-ci.

Display : inline-table

En attribuant un display : inline-table à un élément, celui-ci va se comporter comme un tableau mais de niveau inline.

La valeur display : inline-table va donc nous permettre de placer un tableau à la suite d'un autre contenu plutôt que sur une ligne qui lui est propre.

La valeur display : inline-table est l'équivalent de display : inline table.

Display : inline-flex

En attribuant un display : inline-flex à un élément, celui-ci va se comporter comme un élément de niveau inline et organiser son contenu selon le modèle des boites flexibles. Notez que les flex-items ne sont pas affectés par le type du conteneur : ils vont continuer à se comporter comme des boites de niveau block (tout en possédant certaines propriétés liées aux inline-block).

La valeur display : inline-flex est l'équivalent de display : inline flex.

Display : inline-grid

De la même façon, attribuer un display : inline-grid a un élément va le faire se comporter comme un élément de type inline qui va organiser son contenu selon le modèle des grilles.

La valeur display : inline-grid est l'équivalent de display : inline grid.

Display : inline list-item

Attribuer un display : inline list-item a un élément de liste va modifier le comportement d'affichage de l'élément qui sera alors affiché comme un élément de niveau inline qui va également créer une boite contenant un marqueur.

Les valeurs de non-affichage display : none et display : contents

Finalement, nous allons terminer avec deux valeurs particulières de display : display : none et display : contents. Ces valeurs sont particulières car elles vont nous permettre de ne pas afficher certains éléments ou boites.

Display : none

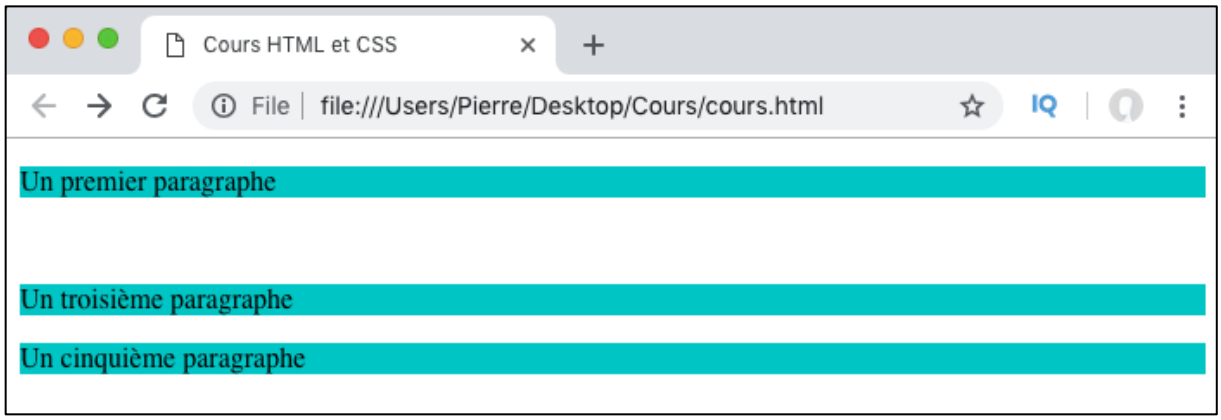
Si un élément possède un `display : none`, il ne sera tout simplement pas affiché dans la page et les autres éléments se comporteront comme s'il n'existait pas (il ne prendra aucune place dans la page).

Il convient de ne pas confondre `display : none` et `visibility : hidden`. La propriété `visibility` va en effet pouvoir faire disparaitre (visuellement) un élément mais l'espace qu'il occupe va être conservé dans la page à la différence de `display : none`.

Notez qu'en appliquant un `display : none` à un élément, ses éléments enfants ne seront pas non plus affichés.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un premier paragraphe</p>
        <p class="invisible">Un deuxième paragraphe</p>
        <p>Un troisième paragraphe</p>
        <p class="nonaffiche">>Un quatrième paragraphe</p>
        <p>Un cinquième paragraphe</p>
    </body>
</html>
```

```css
p{
  background-color: #0CC;
}

.invisible{
  visibility: hidden;
}
.nonaffiche{
  display: none;
}
```



Dans l'exemple ci-dessus, on utilise la propriété `visibility` et sa valeur `hidden` pour cacher un de nos éléments `p`. Vous pouvez cependant remarquer que si le contenu de l'élément ne s'affiche pas, la place qui lui était réservée dans le document est conservée.

Cela ne va pas être le cas pour le paragraphe auquel on a appliqué un `display : none` : non seulement l'élément ne va pas s'afficher mais le document va faire comme si le paragraphe n'existait pas du tout et celui-ci ne va donc pas prendre d'espace dans la page.

Display : contents

La valeur `display : contents` va également nous permettre de faire disparaitre les boites générées par un élément mais, à la différence de `display : none`, le contenu textuel de l'élément va être affiché normalement et les enfants de cet élément vont continuer à générer des boites et à apparaitre dans la page de manière normale.

Notez que cette valeur ne fait pas encore partie des recommandations du W3C et est toujours en cours de développement et peut donc être sujette à des changements ou à un abandon.

Gérer le positionnement des éléments en CSS

La propriété `position` est une propriété CSS très puissante qui va nous permettre de définir un type de positionnement pour nos éléments.

On va ainsi pouvoir positionner un élément relativement à partir de sa position par défaut ou de façon absolue par rapport à un point donné dans la page en utilisation `position` conjointement avec les propriétés `top`, `left`, `bottom` et `right`.

Dans cette leçon, nous allons découvrir les différentes valeurs qu'on va pouvoir donner à `position` et apprendre à les utiliser intelligemment en tentant de comprendre leurs implications.

Le fonctionnement et les valeurs de la propriété position

Nous allons pouvoir gérer et modifier le type de positionnement d'une élément HTML grâce à la propriété CSS `position`.

La propriété `position` ne va pas nous permettre de positionner un élément en soi dans une page mais simplement de définir un type de positionnement grâce aux valeurs suivantes :

- `position : static` ;
- `position : relative` ;
- `position : absolute` ;
- `position : fixed` ;
- `position : sticky`.

Une fois le type de positionnement défini avec `position`, nous allons pouvoir effectivement positionner un élément à un endroit précis dans une page grâce aux propriétés `top`, `left`, `bottom` et `right`.

Ces quatre propriétés vont pouvoir prendre des valeurs absolue ou relative et vont servir à indiquer où le coin supérieur gauche de la boite représentant un élément doit être positionné par rapport à un certain point de référence (50px à droite et 30px en dessous de ce point par exemple).

Le type de positionnement défini pour l'élément va servir à définir ce point de référence et va donc affecter le fonctionnement de ces propriétés qui vont produire des résultats différents.

Les types de positionnement d'un élément HTML dans une page

Il existe trois types de positionnement en CSS. Il est très intéressant de les connaitre et de les comprendre afin de mieux comprendre comment fonctionne le CSS et comment les différents éléments vont venir se positionner les uns par rapport aux autres.

Connaitre et comprendre les types de positionnement va également nous permettre de comprendre comment fonctionne la propriété CSS `position` puisque selon la valeur donnée à code>position, un élément HTML va se conformer à un type de positionnement plutôt qu'un autre.

1. Le premier type de positionnement est ce qu'on pourrait appeler le positionnement « normal » ou par défaut des éléments. Ici, les éléments vont respecter le flux normal de la page et s'y intégrer sans le casser. Ainsi, un élément de type `block` sera formaté comme tel (c'est-à-dire qu'il occupera tout l'espace possible et se placera à la ligne), un élément de type `inline` n'occupera que l'espace nécessaire et etc. ;
2. Ensuite, on va également pouvoir faire flotter des éléments HTML avec la propriété `float`. Ce type de positionnement est particulier puisque l'élément flotté va être retiré du flux normal de la page pour être repositionné ailleurs (généralement à gauche ou à droite) et va également permettre à d'autres éléments de type `inline` de se positionner à côté de notre élément flotté ;
3. Finalement, on va pouvoir positionner un élément de manière absolue dans notre page. Avec le type de positionnement absolu, un élément est complètement retiré du flux normal de la page pour être placé absolument par rapport à son élément parent direct et va ainsi pouvoir potentiellement passer au-dessus d'autres contenus.

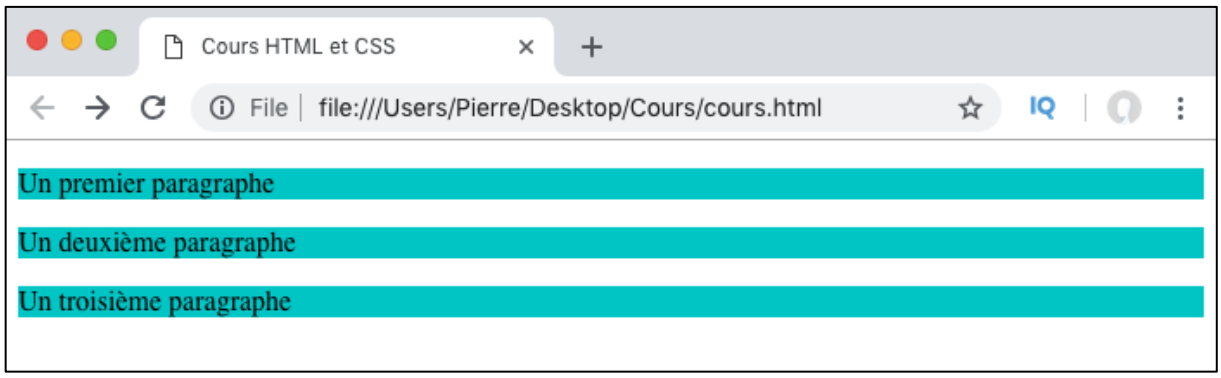
Position : static

La valeur `static` est la valeur par défaut de la propriété `position`. Ainsi, par défaut, tous les éléments HTML sont positionnés de manière `static`. Un élément HTML positionné avec `position : static` sera positionné selon le flux normal de la page.

Notez ici que les propriétés `top`, `left`, `bottom` et `right` n'auront aucun effet sur les éléments positionnés avec `position : static`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un premier paragraphe</p>
        <p class="statique">Un deuxième paragraphe</p>
        <p class="statique gauche">Un troisième paragraphe</p>
    </body>
</html>
```

```css
p{
  background-color: #0CC;
}

.static{
  position: static;/*Valeur par défaut*/
}

.gauche{
  left: 100px;/*La propriété ne s'applique pas sur un élément static*/
}
```



Position : relative

Attribuer une `position : relative` à un élément va positionner l'élément dans le flux normal de la page tout comme la valeur `static` de la propriété `position : static`.

Cependant, à l'inverse d'un élément HTML positionné avec `position : static`, un élément positionné avec `position : relative` va ensuite pouvoir être décalé par rapport à sa position initiale grâce aux propriétés `top`, `left`, `bottom` et `right`.

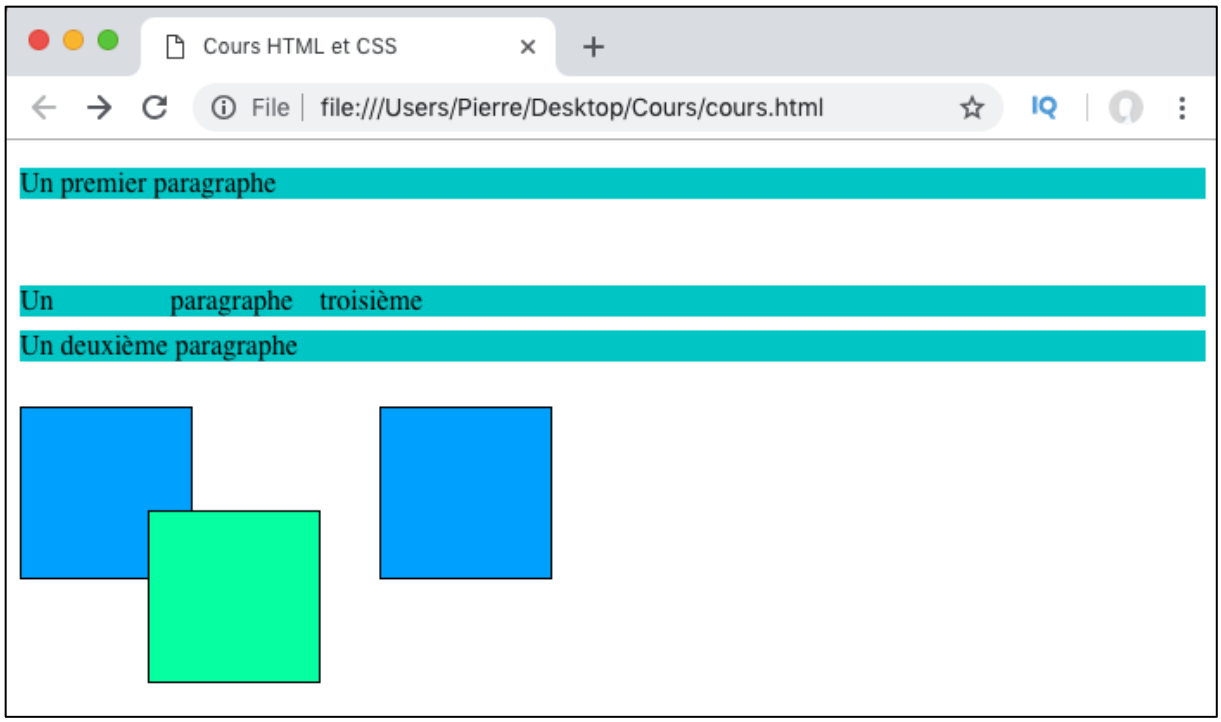
Ces propriétés vont prendre comme origine la position initiale de l'élément. Nous allons ainsi pouvoir positionner un élément relativement à sa position de départ.

Notez qu'ici l'espace occupé initialement par l'élément va continuer à lui appartenir : les autres éléments ne seront pas affectés par le décalage de notre élément et ne vont pas se repositionner en fonction de celui-ci.

Cela implique également que l'élément décalé va pouvoir être à cheval par-dessus d'autres éléments puisque a position de ces autres éléments ne va pas changer en fonction de l'élément décalé possédant une `position : relative`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un premier paragraphe</p>
        <p class="relatif haut">Un deuxième paragraphe</p>
        <p>Un <span class="relatif gauche">troisième</span> paragraphe</p>
        <br><br>
        <div class="carre bleu"></div>
        <div class="carre vert relatif haut droite"></div>
        <div class="carre bleu"></div>
    </body>
</html>
```

```css
p{
  background-color: #0CC;
}
div{
  width: 100px;
  height: 100px;
  display: inline-block;
  border: 1px solid black;
  box-sizing: border-box;
}

.relatif{
  position: relative;
}
.haut{
  top: 60px;/*Elément décalé de 50px vers le bas p/r à une origine*/
}
.gauche{
  left: 150px;/*Element décalé de 150px vers la droite p/r à une origine*/
}
.droite{
  right: 30px;/*Element décalé de 30px vers la gauchr p/r à une origine*/
}

.bleu{
  background-color: #0AF;
}
.vert{
  background-color: #0FA;
}
```



Position : absolute

Un élément positionné avec `position: absolute` va être positionné par rapport à son parent le plus proche positionné (avec une valeur de position différente de `static`).

Si aucun parent positionné n'est trouvé, alors l'élément sera positionné par rapport à l'élément racine représentant la page en soi.

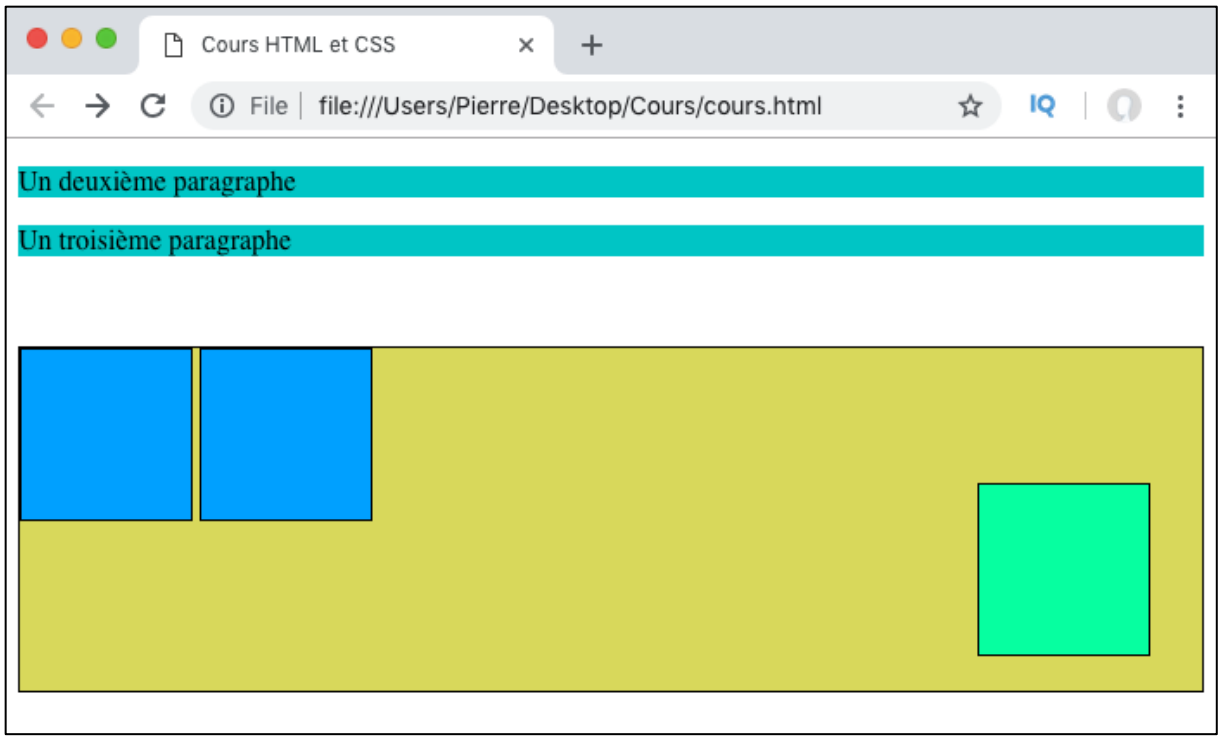
Le point de référence pour les propriétés `top`, `left`, `bottom` et `right` va ainsi être le côté de l'élément parent liée à la propriété (côté gauche pour `left`, supérieur pour `top` , etc.).

De plus, un élément positionné avec `position : absolute` va être retiré du flux normal de la page. Cela signifie et implique que l'espace initialement attribué à un élément au positionnement absolu (espace attribué selon le flux normal de la page) va être occupé par les éléments suivants.

Un élément positionné avec `position: absolute` va ainsi pouvoir se placer par-dessus d'autres éléments.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un premier paragraphe</p>
        <p class="absolu haut">Un deuxième paragraphe</p>
        <p>Un troisième paragraphe</p>
        <br><br>
        <div class="conteneur relative">
            <div class="carre bleu"></div>
            <div class="carre vert absolu bas droite"></div>
            <div class="carre bleu"></div>
        </div>
    </body>
</html>
```

```css
p{
  background-color: #0CC;
}
div{
  width: 100px;
  height: 100px;
  display: inline-block;
  border: 1px solid black;
  box-sizing: border-box;
}
.conteneur{
  width: 100%;
  height: 200px;
  display:block;;
  background-color: #DD6;/*Jaune*/
}

.absolu{position: absolute;}
.relative{position: relative}

.haut{top: 70px;}
.bas{bottom: 20px;}
.droite{right: 30px;}

.bleu{background-color: #0AF;}
.vert{background-color: #0FA;}
```



Ici, nous avons un paragraphe et un `div` positionnés de manière absolue. Notre paragraphe ne possède pas de parent positionné. Il va donc être positionné par rapport à l'élément racine de la page, c'est-à-dire par rapport à la page en soi. Ici, `top : 70px` signifie donc que notre paragraphe sera placé à 70px du point le plus haut de la page auquel il aurait pu être placé (en tenant compte des marges appliquées).

Notre `div` possède lui un parent positionné qui est le `div` jaune « conteneur ». Ce dernier est positionné de manière relative. Notez qu'on lui a déclaré une `position : relative` mais qu'on n'a pas modifié sa position avec une propriété `top`, `left`, etc. Cela n'empêche pas au `div` conteneur d'être positionné.

Notre `div` vert va donc être positionné par rapport au `div` jaune. Ici, `bottom : 20px` et `right : 30px` signifie que notre `div` vert sera positionné à 20px du bord inférieur et à 30px du bord droit de son parent.

Vous pouvez également observer que les éléments positionnés de manière absolue sont bien retirés du flux normal de la page et les autres éléments vont pouvoir venir se positionner à la place de ces éléments, comme s'ils n'existaient pas. On voit bien cela avec notre deuxième `div` bleu qui vient se coller au premier et vient donc prendre la place du `div` vert ainsi qu'avec notre troisième paragraphe qui prend la place initiale du deuxième.

Position : fixed

Le positionnement fixe est très proche du positionnement absolu. Un élément positionné avec `position: fixed` va également être retiré du flux de la page et l'espace qui lui était attribué selon le flux normal de la page va également pouvoir être utilisé par d'autres éléments.

La seule différence entre `position: fixed` et `position: absolute` est que l'élément ne va plus être positionné par rapport à son parent le plus proche mais par rapport au viewport, c'est-à-dire par rapport à la fenêtre visible à moins que l'un de ses parents possède une propriété `transform`, `filter` ou `perspective` dont la valeur est différente de `none`.

En dehors de ces cas particuliers, un élément positionné avec `position: fixed` apparaitra toujours à la même place même dans la fenêtre si on descend ou on monte dans la page : il sera fixe par rapport à la fenêtre. En effet, sa position va être calculée par rapport à la fenêtre visible.

A noter ici une exception pour les contenus paginés : dans ce cas-là, l'élément possédant une `position: fixed` sera répété dans chaque page.

Position : sticky

La dernière valeur de la propriété CSS `position` est la valeur `sticky`. Un élément positionné avec `position: sticky` sera d'abord positionné selon le flux normal de la page puis va pouvoir être décalé de manière similaire à un élément positionné de manière relative. Les éléments suivants ne verront pas leur position changée : ils seront toujours placé « comme si » l'élément positionné avec `position: sticky` occupait sa place d'origine.

La différence ici entre un élément positionné avec `position: sticky` et `position: relative` est que la position d'un élément sticky va être calculée par rapport à son parent possédant un mécanisme de défilement (scrolling) le plus proche.

Ainsi, un élément positionné avec `position: sticky` va avoir une position relative au départ puis son positionnement va devenir fixe dès qu'un certain point sera franchi, c'est-à-dire à partir d'un certain niveau de défilement de la page. Les propriétés `top`, `left`, `bottom` et `right` vont nous permettre de pouvoir préciser à partir de quel moment l'élément positionné avec `position: sticky` va devoir être fixe.

Notez que la valeur `sticky` est une valeur assez récente de la propriété `position` et est à ce titre toujours en développement. On évitera donc de l'utiliser pour le moment sur un site « live ».

Un mot sur l'accessibilité du contenu

Lorsqu'on code, il faut toujours s'efforcer de réfléchir en termes d'accessibilité à tous, et notamment pour les personnes souffrant de déficiences comme des déficiences visuelles.

En effet, un des principes de base du web est d'être accessible à tous ou du moins c'est l'une des valeurs fondamentales vers laquelle tendre.

Ici, il faudra donc faire bien attention à ce que les contenus positionnés ne cachent pas d'autres contenus de façon non désirée lorsqu'un utilisateur par exemple zoome sur la page pour augmenter la taille du texte.

Définir l'ordre d'affichage des éléments en cas de chevauchement avec la propriété z-index

Les éléments HTML vont pouvoir être positionné dans une page en CSS selon 3 dimensions : selon la largeur (axe horizontal ou axe des X en math), la hauteur (axe vertical ou axe des Y) et également selon une épaisseur ou un ordre d'empilement (axe des Z).

En effet, vous avez pu remarquer dans les exemples précédents que lorsque deux éléments se chevauchaient, il y en avait toujours un au-dessus de l'autre : il y a donc une notion d'ordre d'empilement selon cet axe 3D qui est l'axe des Z.

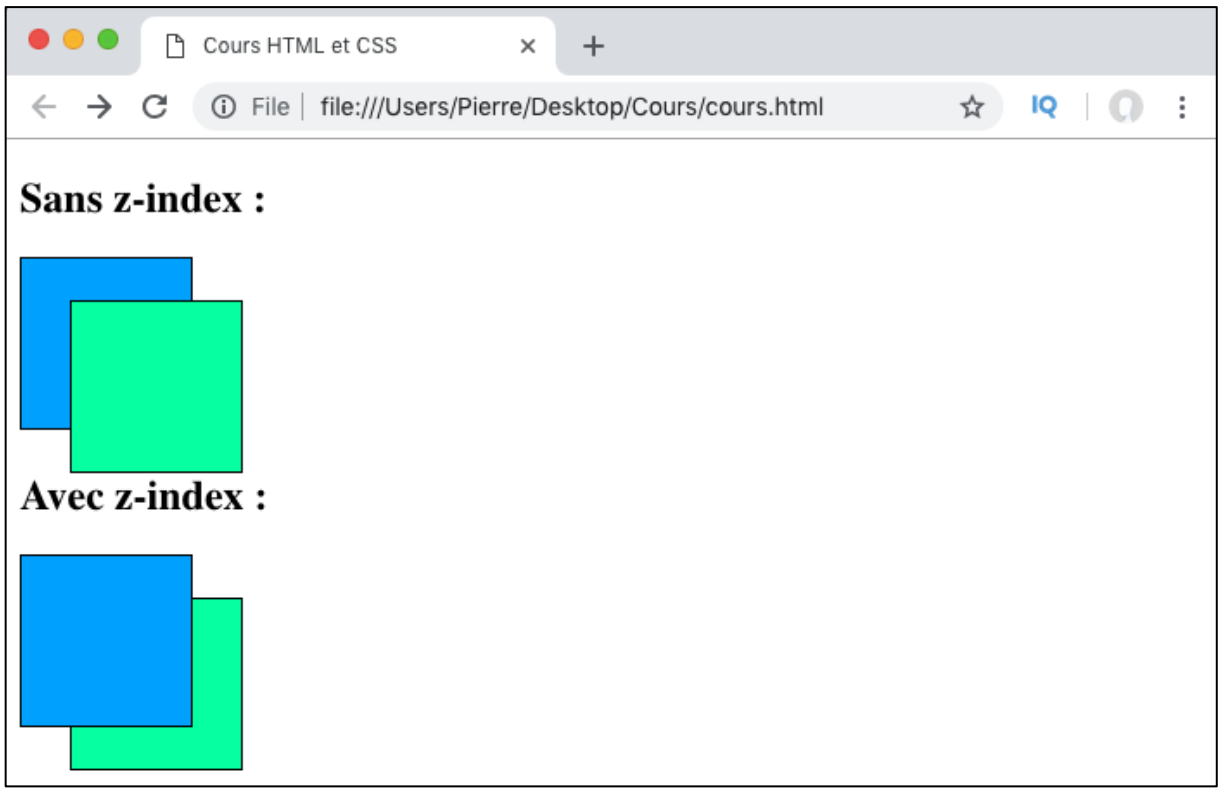
Par défaut, lorsque deux boites se chevauchent, l'élément déclaré en dernier apparaitra par-dessus l'élément déclaré avant en HTML. C'est une règle implicite de tout document HTML.

La propriété `z-index` va nous permettre de modifier ce comportement et de choisir quel élément doit apparaitre au-dessus de quel ordre en donnant un index sous forme de nombre à un ou plusieurs éléments. Ainsi, lorsque deux éléments se chevauchent, celui possédant la plus grande valeur pour son `z-index` apparaitra au-dessus de l'autre.

Notez que la propriété CSS `z-index` ne va fonctionner (et n'a de sens) qu'avec des éléments HTML positionnés, c'est-à-dire qu'avec des éléments possédant une propriété `position` dont la valeur est différente de `static` en CSS.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h2>Sans z-index :</h2>
        <div class="carre bleu relatif"></div>
        <div class="carre relatif vert decale"></div>
        <h2>Avec z-index :</h2>
        <div class="carre bleu relatif z100"></div>
        <div class="carre relatif vert decale"></div>
    </body>
</html>
```

```css
p{
  background-color: #0CC;
}
div{
  width: 100px;
  height: 100px;
  display: inline-block;
  border: 1px solid black;
  box-sizing: border-box;
}
.bleu{background-color: #0AF;}
.vert{background-color: #0FA;}

.relatif{position: relative}
.decale{
  right: 75px;
  top: 25px;
}

.z100{
  z-index: 100;
}
```



La propriété CSS float

La propriété CSS float permet de sortir un élément du flux normal de la page et de le faire “flotter” contre un bord de son élément parent conteneur ou contre un autre élément flottant.

Une utilisation bien connue de la propriété float est de s’en servir pour faire flotter une image à droite ou à gauche d’un texte et ainsi l’entourer avec du texte.

La propriété float est donc une autre propriété qui va impacter la disposition dans la page et qu’il convient de manier avec précaution pour ne pas obtenir de comportement indésirable. Le but de cette nouvelle leçon est d’apprendre à la manipuler.

Définition et fonctionnement de la propriété float

La propriété float va retirer un élément du flux normal de la page puis le placer contre un bord de son élément parent (ou élément conteneur) ou contre le bord d’un élément flottant le précédant.

Cette propriété va également impacter les éléments environnants puisque le texte et les éléments inline suivants un élément possédant un float différent de none vont essayer de venir se placer à ses côtés.

La propriété float était à l’origine principalement utilisée pour incruster des images dans du texte en les faisant flotter. L’utilisation « normale » de float est donc d’appliquer le float (de faire flotter) des éléments inline dans la page et de laisser les textes se positionner autour.

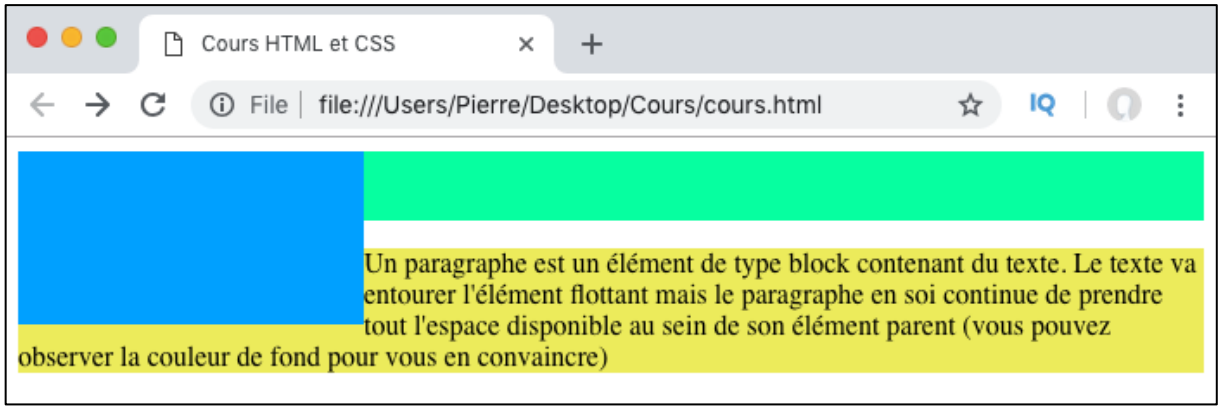
En effet, les éléments de type block suivants un élément flottant vont également se placer sur la même ligne que l’élément flottant mais continuer à prendre tout l’espace disponible dans la ligne. La partie des éléments block chevauchant un flottant va être cachée derrière le flottant.

On va cependant également tout à fait pouvoir faire flotter un élément block même si généralement nous utiliserons plutôt un display : inline-block pour placer deux éléments de type block côte-à-côte.

Notez que si vous voulez faire flotter un élément block sans contenu, alors il faudra lui donner une largeur et une hauteur explicites avec les propriétés width et height car dans le cas contraire les valeurs calculées seront égales à 0.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="flottant"></div>
        <div></div>
        <p>Un paragraphe est un élément de type block contenant du texte. Le
        texte va entourer l'élément flottant mais le paragraphe en soi continue
        de prendre tout l'espace disponible au sein de son élément parent
        (vous pouvez observer la couleur de fond ou inspecter l'élément
        pour vous en convaincre)</p>
    </body>
</html>
```

```css
.flottant{
  width: 200px;
  height: 100px;
  float: left;
  background-color: #0AF;/*Bleu*/
}
div{
  height: 40px;
  background-color: #0FA;/*Vert*/
}
p{
  background-color: #EE6;/*Jaune*/
}
```



Notez également déjà que la propriété `float` ne va pas fonctionner avec des éléments positionnés de manière absolue et ne va avoir aucun effet sur des éléments affichés avec `display : flex` ou `display : inline-flex`. Nous aurons l'occasion de revenir sur ces sujets plus tard.

Les valeurs de la propriété float

Historiquement, nous avions le choix entre 3 valeurs à passer à la propriété `float` :

- `float : left` : L'élément va venir se positionner à l'extrémité gauche de son élément conteneur ou va être décalé sur la gauche jusqu'à toucher un autre élément avec `float : left` ;
- `float : right` : L'élément va venir se positionner à l'extrémité droite de son élément conteneur ou va être décalé sur la droite jusqu'à toucher un autre élément avec `float : right` ;
- `float : none` : Valeur par défaut. L'élément n'est pas un élément flottant.

Le CSS3 va apporter un nouveau choix élargi de valeurs que nous allons pouvoir passer à `float`. Je vous rappelle ici que le CSS3 est toujours en développement et qu'ainsi tout ce qui est en train d'être établi par celui-ci n'est pas forcément encore passé comme recommandation du W3C. En effet, certaines valeurs et propriétés actuellement à l'étude dans le cadre du CSS3 vont potentiellement être modifiées ou abandonnées en cours de route.

Parmi les nouvelles valeurs apportées à `float`, cependant, nous pouvons déjà en citer deux qui possèdent déjà un bon support par les navigateurs :

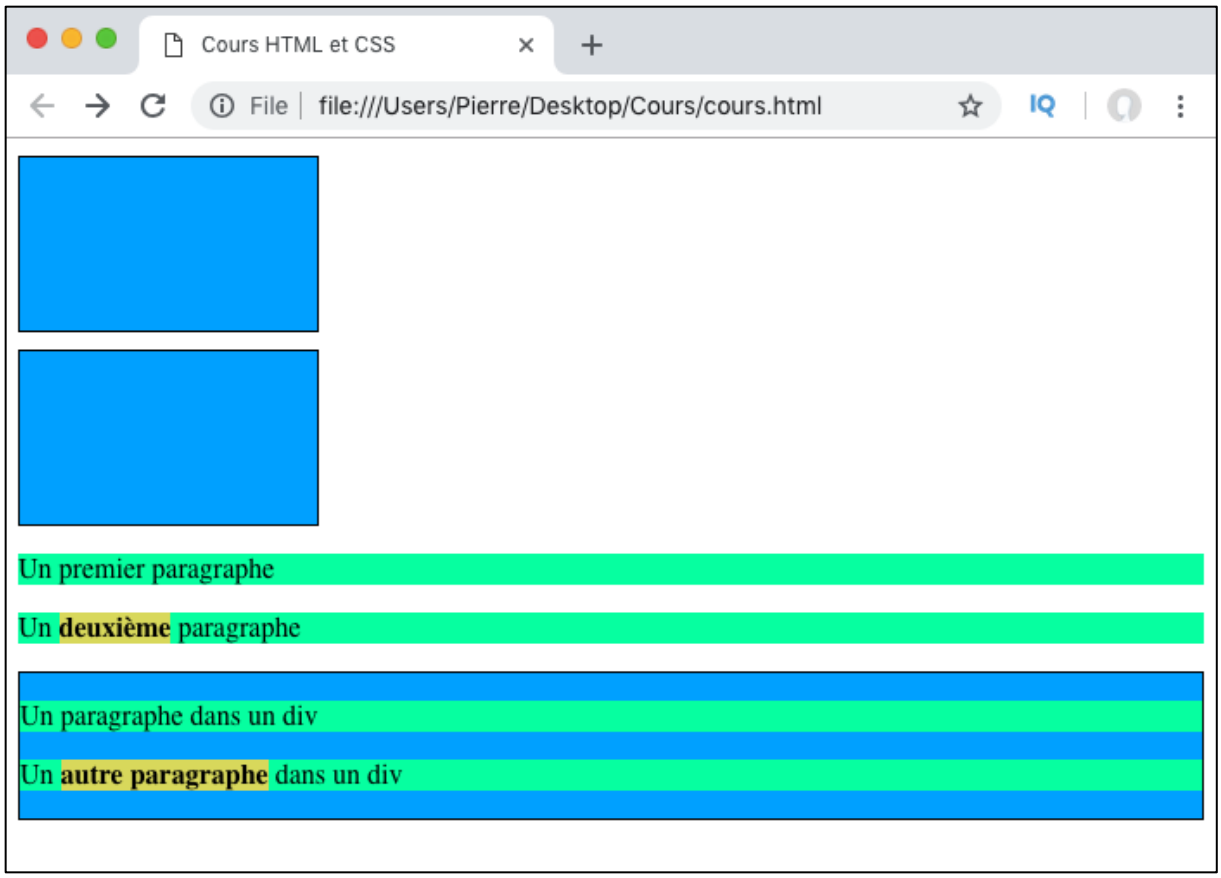
- `float : inline-start` : L'élément va venir se positionner au début de son élément conteneur (c'est-à-dire à gauche pour des documents dont l'écriture se fait de gauche à droite ou à droite dans le cas contraire) ou va être décalé vers le début de son conteneur jusqu'à toucher un autre élément avec `float : inline-start` ;
- `float : inline-end` : L'élément va venir se positionner à la fin de son élément conteneur (c'est-à-dire à droite pour des documents dont l'écriture se fait de gauche à droite ou à gauche dans le cas contraire) ou va être décalé vers la fin de son conteneur jusqu'à toucher un autre élément avec `float : inline-end`.

Float : none

La valeur par défaut de `float` est `none`. Cette valeur correspond à l'absence de flottement.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="w25 h100"></div>
        <div class="w25 h100"></div>
        <p>Un premier paragraphe</p>
        <p>Un <strong>deuxième</strong> paragraphe</p>
        <div>
            <p>Un paragraphe dans un div</p>
            <p>Un <strong>autre paragraphe</strong> dans un div</p>
        </div>
    </body>
</html>
```

```css
div{
  background-color: #0AF;/*Bleu*/
  border: 1px solid black;
  margin: 10px 0px;/*10px de marge haute et basse*/
}
p{
  background-color: #0FA;/*Vert*/
}
strong{
  background-color: #DD6;/*Jaune*/
}

.w25{
  width: 25%;
}
.h100{
  height: 100px;
}
```



Ici, nous avons deux éléments `div` de largeurs égales à `25%` de celle de leur parent et de hauteurs égales à `50px`. Les `div` sont des éléments de type `block` et vont donc aller à la ligne et occuper une ligne chacun quelle que soit leurs dimensions.

Ensuite, nous avons deux éléments `p` dont un qui contient un élément `strong`. Les éléments `p` sont également de type `block` et vont donc par défaut occuper tout l'espace disponible dans leur parent et occuper une ligne chacun. L'élément `strong` est lui un élément `inline` par défaut et va donc ne prendre que la place nécessaire dans son parent et ne pas aller à la ligne.

Finalement, nous avons créé un `div` qui contient deux paragraphes dont un contient lui-même un autre élément `strong`.

Par défaut, aucun de ces éléments ne flotte. Ne rien préciser ou préciser un `float : none` est ici identique et le comportement de ces éléments est connu.

Note : parfois, il sera utile de préciser un `float : none` pour un élément pour annuler par exemple un comportement d'héritage.

Float : left et float : right

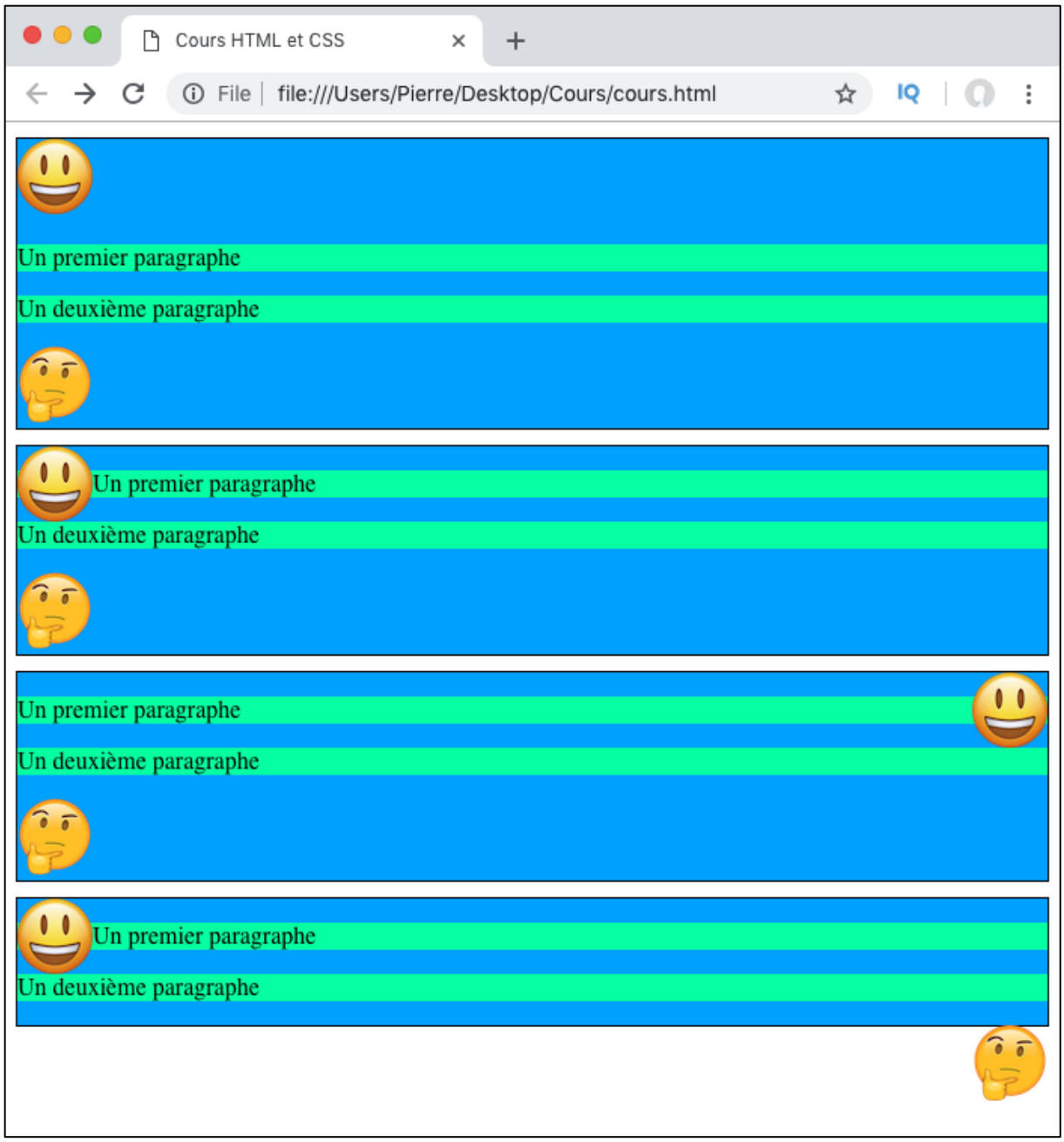
En appliquant un `float: left` à un élément, l'élément va venir se positionner contre le bord gauche de son élément conteneur ou va être décalé vers la gauche jusqu'à toucher un autre élément flottant.

En appliquant un `float: right` à un élément, l'élément va venir se positionner contre le bord droit de son élément conteneur ou va être décalé vers la droite jusqu'à toucher un autre élément flottant.

Ces deux valeurs vont se comporter et pouvoir être appliquées de manière similaire.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur">
            <img src="smile.png" alt="Un emoji content">
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
            <img src="think.png" alt="Un emoji penseur">
        </div>
        <div class="conteneur">
            <img src="smile.png" alt="Un emoji content" class="gauche">
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
            <img src="think.png" alt="Un emoji penseur">
        </div>
        <div class="conteneur">
            <img src="smile.png" alt="Un emoji content" class="droite">
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
            <img src="think.png" alt="Un emoji penseur">
        </div>
        <div class="conteneur">
            <img src="smile.png" alt="Un emoji content" class="gauche">
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
            <img src="think.png" alt="Un emoji penseur" class="droite">
        </div>
    </body>
</html>
```

```css
div{
  background-color: #0AF;/*Bleu*/
  border: 1px solid black;
  margin: 10px 0px;/*10px de marge haute et basse*/
}
p{
  background-color: #0FA;/*Vert*/
}
strong{
  background-color: #DD6;/*Jaune*/
}

.w25{
  width: 25%;
}
.h100{
  height: 100px;
}

.gauche{
  float: left;
}
.droite{
  float: right;
}
```



Il y a des choses intéressantes à noter dans cet exemple. Ici, on va faire flotter nos différentes images d'emoji à droite ou à gauche. La première chose à retenir est qu'un élément flottant va toujours flotter sur sa ligne. Dans le dernier exemple, par exemple, notre premier emoji est le premier élément déclaré dans notre conteneur, il flottera donc en haut tandis que notre deuxième emoji est le dernier élément déclaré et il flottera donc en bas du conteneur.

Dans notre dernier `div`, en particulier, l'emoji « penseur » se retrouve en dehors du `div`. Cela est dû au fait que `float` retire l'élément du flux normal de la page : le `div` ne tiendra plus compte de cet élément et va donc se redimensionner à la taille des éléments qu'il contient, c'est-à-dire dans ce cas les deux paragraphes. L'élément flotté va lui en revanche continuer à flotter sur sa ligne.

Si vous regardez de plus près, vous pouvez constater qu'il se passe exactement la même chose avec notre premier emoji mais c'est moins visible car c'est le premier élément

déclaré dans notre div et donc il va toujours être inclus dedans grâce aux paragraphes le suivant.

Cependant, ne vous y trompez pas : le div ne tient plus compte de l'emoji flotté dans le calcul de sa taille mais seulement des éléments non flottés et va se redimensionner en conséquence. C'est la raison pour laquelle nos deuxième et troisième div sont plus petits en hauteur que le premier. Pour éviter ce genre de rendu visuel, on peut toujours augmenter artificiellement la taille de notre élément conteneur en utilisant height.

La deuxième chose à bien comprendre est le fait qu'un élément flottant n'impacte pas les propriétés des éléments environnants à proprement parler : les éléments suivants un élément flottant vont pouvoir se placer à côté de l'élément flottant dans la limite de la hauteur de l'élément flottant. Les éléments sous l'élément flottant vont avoir un comportement « normal » comme on peut le voir avec notre deuxième paragraphe pour nos deuxième et troisième div.

Contrôler le comportement des éléments autour d'un flottant avec la propriété clear

La propriété CSS clear va nous permettre d'empêcher un élément de se positionner à côté d'un élément flottant. Cette propriété va être extrêmement utile dans de nombreuses situations pour mieux contrôler le design de nos pages.

Nous allons pouvoir lui passer l'une des valeurs suivantes :

- clear : none : Valeur par défaut. Laisse les éléments se positionner à côté d'éléments flottants ;
- clear : left : Empêche un élément de se positionner à côté d'éléments possédant un float : left ;
- clear : right : Empêche un élément de se positionner à côté d'éléments possédant un float : right ;
- clear : both : Empêche un élément de se positionner à côté d'éléments possédant un float : left ou un float : right ;
- clear : inline-start : Empêche un élément de se positionner à côté d'éléments possédant un float : inline-start ;
- clear : inline-end : Empêche un élément de se positionner à côté d'éléments possédant un float : inline-end.

Notez ici que la propriété clear va avoir un comportement légèrement différent selon qu'on l'applique à des éléments non flottants ou au contraire à des éléments déjà flottants.

Dans le cas où l'on applique clear à un élément non flottant, l'élément va être déplacé de telle sorte à ce que sa bordure supérieure se place directement sous le bord de la marge basse extérieure des éléments flottants concernés et il va y avoir fusion des marges (collapse en anglais).

Dans le cas où clear est appliquée à un élément flottant, l'élément va être déplacé de telle sorte à ce que l'extrémité de sa marge supérieure se place directement sous le bord de la marge basse des éléments flottants concernés. Les deux marges vont donc être conservées. La position des potentiels éléments flottants suivants va être impactée

puisqu'un élément flottant ne peut pas être situé plus haut qu'un autre élément flottant qui le précède.

Cas pratiques d'utilisation de la propriété clear

Généralement, nous allons donc utiliser la propriété `clear` après avoir appliqué un `float` à un élément pour empêcher certains éléments de venir flotter à côté de l'élément en question.

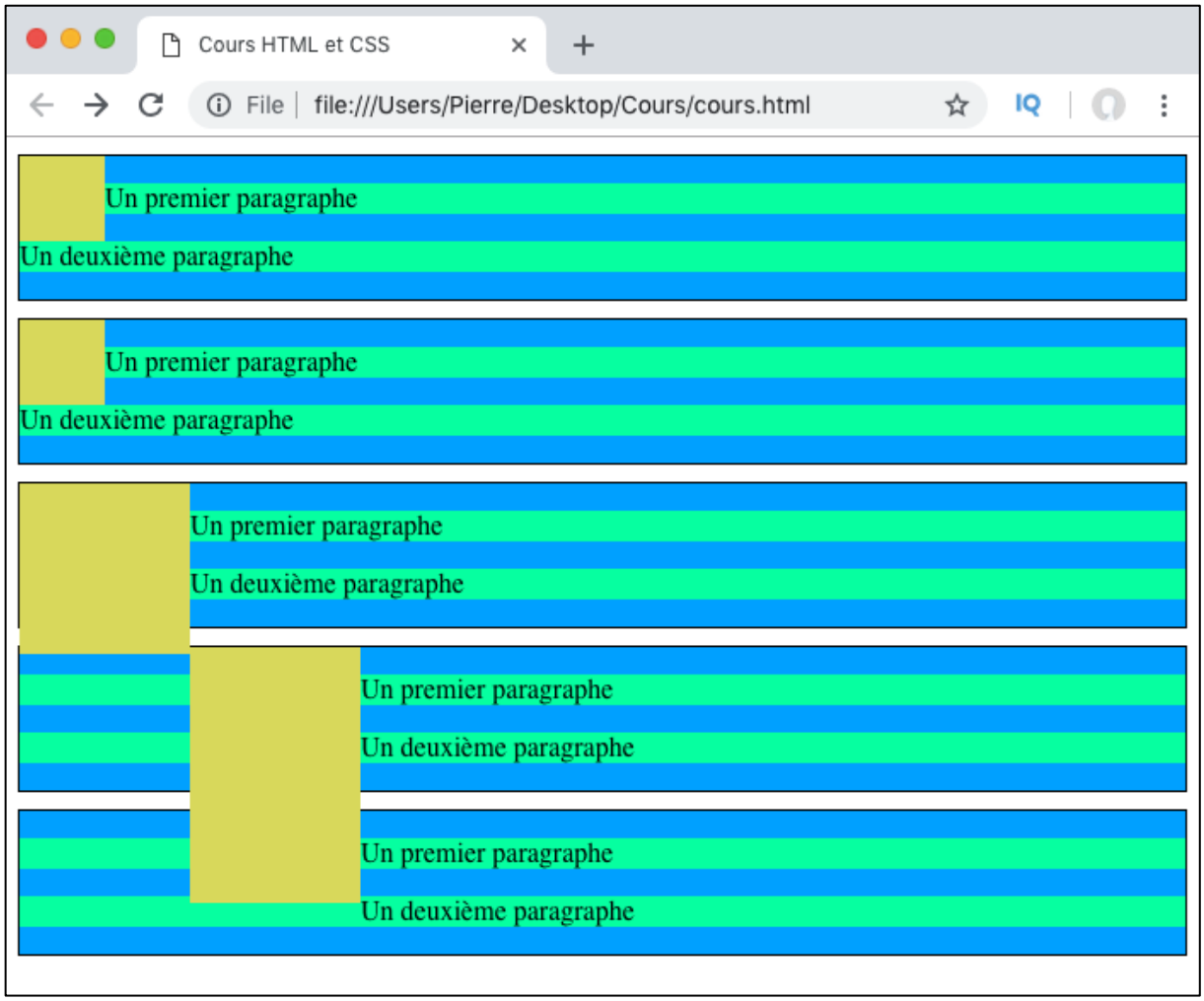
En effet, bien souvent, lorsque nous créerons le design d'une page, nous n'allons pas vouloir que les éléments se positionnent tant qu'ils le peuvent à côté d'un élément flottant mais pouvoir contrôler quels éléments vont pouvoir se positionner aux côtés d'un élément flottant et quels éléments ne doivent pas le faire.

Illustration du problème réglé par la propriété clear

Pour bien comprendre l'intérêt de la propriété `float`, je vous propose de regarder l'exemple suivant :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur">
            <div class="interne gauche w50 h50"></div>
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur">
            <div class="interne gauche w50 h50"></div>
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur">
            <div class="interne gauche w100 h100"></div>
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur">
            <div class="interne gauche w100 h150"></div>
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur">
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
    </body>
</html>
```

```css
p{
  background-color: #0FA;/*Vert*/
}
.conteneur{
  background-color: #0AF;/*Bleu*/
  border: 1px solid black;
  margin: 10px 0px;/*10px de marge haute et basse*/
}
.interne{
  background-color: #DD6;/*Jaune*/
}
.w50{width: 50px;}
.w100{width: 100px;}
.h50{height:50px;}
.h100{height:100px;}
.h150{height:150px;}

.gauche{
  float: left;
}
.droite{
  float: right;
}
```



Ici, nous faisons flotter chacun des div internes à gauche dans nos div conteneurs. Pour nos deux premiers div, le résultat est celui espéré. En revanche, on observe un décalage qui se crée pour nos trois derniers div.

En fait, c'est tout à fait normal : le fait que l'affichage se fasse bien avec nos deux premiers div est simplement un « coup de chance » dû au fait que nous n'avons pas deux éléments flottants qui se touchent puisque les div conteneurs soient plus grands que les div flottés et qu'ils ne contiennent qu'un élément flottant.

Ici, il faut savoir que par défaut tous les éléments suivants un flottant vont essayer de se positionner sur la même ligne que l'élément flottant et cela dans la limite de la hauteur de l'élément flottant. Dans nos deux premiers exemples, le deuxième paragraphe va à la ligne tout simplement car il n'a pas la place pour se positionner sur la même ligne que le flottant (la hauteur du flottant est déjà remplie par le premier paragraphe).

Pour les div flottants suivants, en revanche, la situation va être différente. Notre troisième div flottant dépasse en effet de son div conteneur et va arriver jusqu'à la ligne sur laquelle se situe le div flottant suivant.

Le quatrième div flottant va donc toucher le div flottant précédent. Or, un élément flottant va essayer se positionner soit contre le bord de son élément parent soit contre le bord d'un élément flottant précédent si un tel élément existe.

C'est ce qui se passe ici : notre dernier div flotté va rencontrer le div flotté précédent et va donc se coller contre son bord.

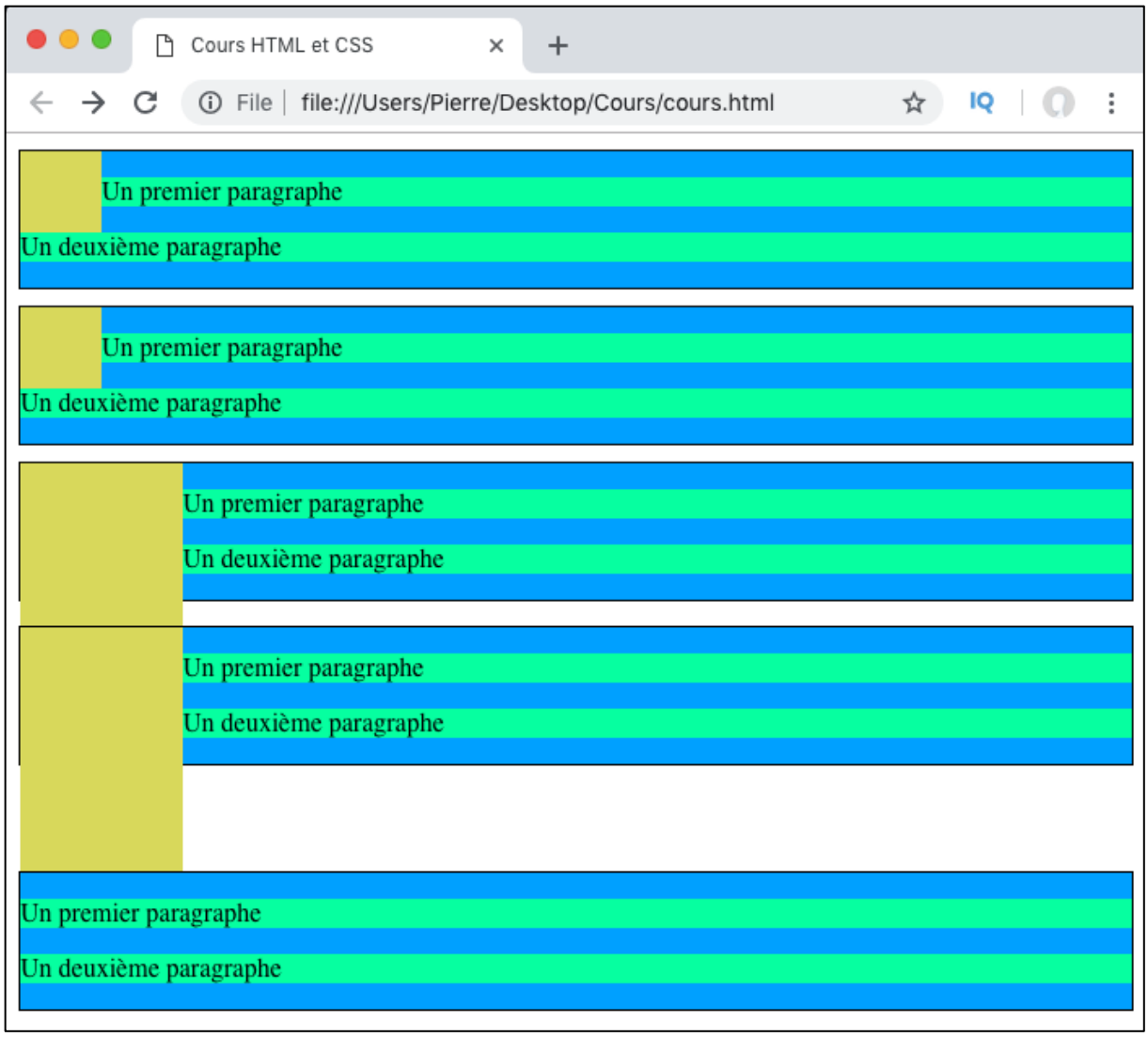
Enfin, notre quatrième et dernier div flottant est suffisamment haut pour arriver jusqu'aux lignes occupées par les paragraphes du dernier div conteneur. Le texte des paragraphes va donc se positionner contre le flottant du div conteneur précédent.

La propriété clear va justement nous permettre d'éviter ce genre de comportements généralement non souhaités.

Ici, par exemple, il suffirait d'appliquer un clear : left à chacun de nos div conteneurs pour résoudre une grande partie du problème :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur">
            <div class="interne gauche w50 h50"></div>
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur cleared">
            <div class="interne gauche w50 h50"></div>
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur cleared">
            <div class="interne gauche w100 h100"></div>
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur cleared">
            <div class="interne gauche w100 h150"></div>
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur cleared">
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
    </body>
</html>
```

```css
p{
  background-color: #0FA;/*Vert*/
}
.conteneur{
  background-color: #0AF;/*Bleu*/
  border: 1px solid black;
  margin: 10px 0px;/*10px de marge haute et basse*/
}
.interne{
  background-color: #DD6;/*Jaune*/
}
.w50{width: 50px;}
.w100{width: 100px;}
.h50{height:50px;}
.h100{height:100px;}
.h150{height:150px;}

.gauche{
  float: left;
}
.droite{
  float: right;
}

.cleared{
  clear: left;
}
```



La propriété `clear` empêche ici nos `div` conteneurs (et donc les éléments qu'ils contiennent) de venir se positionner à côté des éléments flottants précédents. Les `div` conteneurs vont donc se positionner juste en dessous des flottants si les éléments flottants précédents dépassaient de leur conteneur.

Notez qu'on aurait ici pu également tout-à-fait appliquer le `clear` sur un autre élément comme par exemple sur l'un des paragraphes de nos différents `div`. Dans ce cas, le paragraphe en question et tous les éléments suivants auraient été libérés du flottement et se seraient positionnés sous le flottant.

Bien évidemment, la propriété `clear` va fonctionner exactement de la même façon avec sa valeur `right` pour empêcher des éléments de se positionner à côté d'un élément possédant un `float : right`.

La valeur `both` va elle servir à libérer les éléments d'un flottant gauche ou d'un flottant droit. Regardez plutôt l'exemple ci-dessous que vous devriez être capable de comprendre par vous-même :

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur">
            <div class="interne gauche w50 h50"></div>
            <p class="cg">Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur">
            <div class="interne droite w50 h50"></div>
            <p>Un premier paragraphe</p>
            <p class="cd">Un deuxième paragraphe</p>
        </div>
        <div class="conteneur ">
            <div class="interne gauche w100 h100"></div>
            <p class="cg">Un premier paragraphe</p>
            <div class="interne droite w100 h150"></div>
            <p class="cd">Un deuxième paragraphe</p>
        </div>
        <div class="conteneur">
            <div class="interne gauche w100 h150"></div>
            <p>Un premier paragraphe</p>
            <div class="interne droite w100 h150"></div>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur cleared">
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
    </body>
</html>
```

```css
p{
    background-color: #0FA;/*Vert*/
}
.conteneur{
    background-color: #0AF;/*Bleu*/
    border: 1px solid black;
    margin: 10px 0px;/*10px de marge haute et basse*/
}
.interne{
    background-color: #DD6;/*Jaune*/
}
.w50{width: 50px;}
.w100{width: 100px;}
.h50{height:50px;}
.h100{height:100px;}
.h150{height:150px;}

.gauche{
    float: left;
}
.droite{
    float: right;
}

.cg{
    clear: left;
}
.cd{
    clear: right;
}
.cleared{
    clear: both;
}
```

Découverte et utilisation du « clearfix » CSS

Il nous reste un problème à régler : comment faire lorsque le flottant est le dernier élément dans son conteneur pour ne pas que celui-ci dépasse du conteneur ?

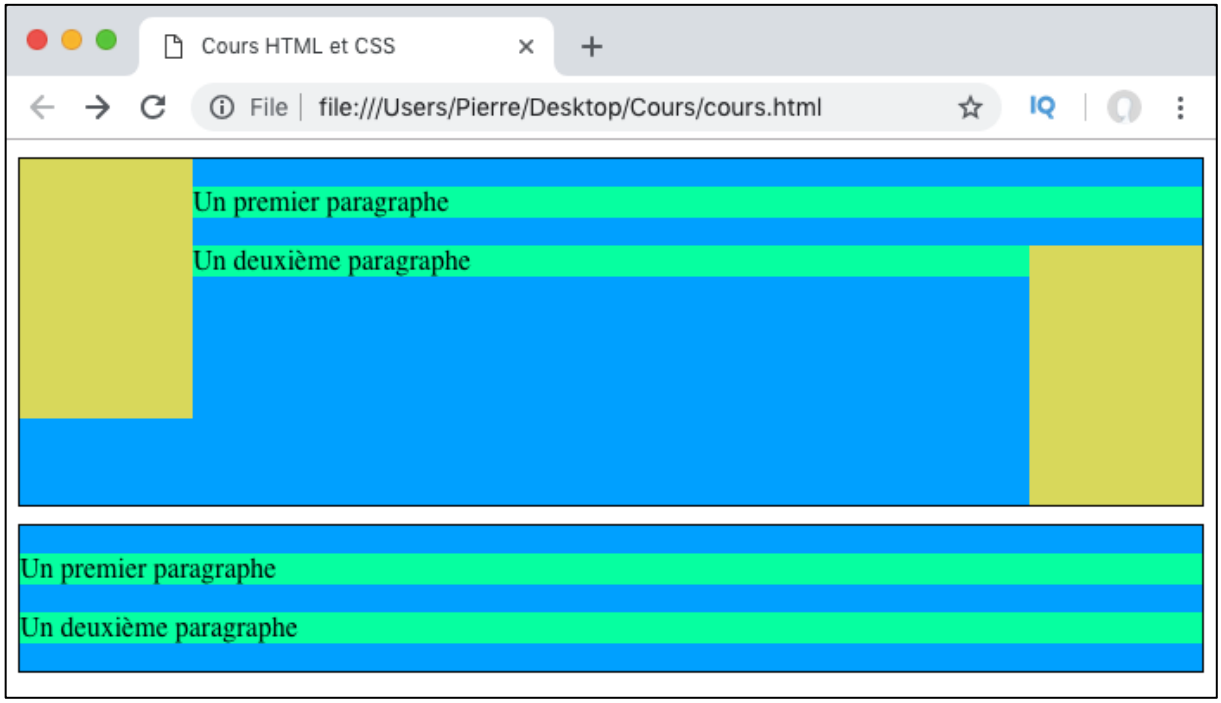
En effet, bien souvent, on voudra que les éléments flottants restent dans la limite de leur conteneur ou plus exactement que les conteneurs s'adaptent pour inclure les flottants dans leur taille.

Pour faire cela, on va utiliser ce qu'on appelle un « clearfix » qui est un hack bien connu en CSS utilisant le pseudo-élément `::after`. Attention : il faudra cette fois-ci l'appliquer au conteneur qui contient le flottant qui pose des problèmes de design pour qu'il fonctionne.

Ce clearfix est le suivant :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur clearfix">
            <div class="interne gauche w100 h150"></div>
            <p>Un premier paragraphe</p>
            <div class="interne droite w100 h150"></div>
            <p>Un deuxième paragraphe</p>
        </div>
        <div class="conteneur">
            <p>Un premier paragraphe</p>
            <p>Un deuxième paragraphe</p>
        </div>
    </body>
</html>
```

```css
p{
  background-color: #0FA;/*Vert*/
}
.conteneur{
  background-color: #0AF;/*Bleu*/
  border: 1px solid black;
  margin: 10px 0px;/*10px de marge haute et basse*/
}
.interne{
  background-color: #DD6;/*Jaune*/
}
.w100{width: 100px;}
.h150{height:150px;}

.gauche{
  float: left;
}
.droite{
  float: right;
}

.clearfix::after{
  content: "";
  display: table;
  clear: both;
}
```



Nous étudierons les pseudo-éléments plus tard dans ce cours. Cependant, je vais quand même essayer de vous expliquer brièvement comment fonctionne le clearfix. Le pseudo élément `::after` nous permet de créer un pseudo élément qui va être le dernier enfant de l'élément sélectionné.

L'idée est ici de l'utiliser avec la propriété `content` pour ajouter un contenu ou plus exactement une chaine de caractères vide en fin de l'élément.

Ensuite, nous allons appliquer le `clear` à ce pseudo élément. Pour que cela fonctionne, cependant, il va falloir lui définir un `display`. Le type d'affichage qui se prête le mieux à l'opération est ici `display : table`

La propriété float et la hauteur des conteneurs

Comme nous avons pu le voir et l'évoquer plus haut, un élément flottant est retiré du flux normal de la page.

Cela implique que l'élément qui contient le flottant ne va pas tenir compte de ce dernier dans le calcul de ses dimensions.

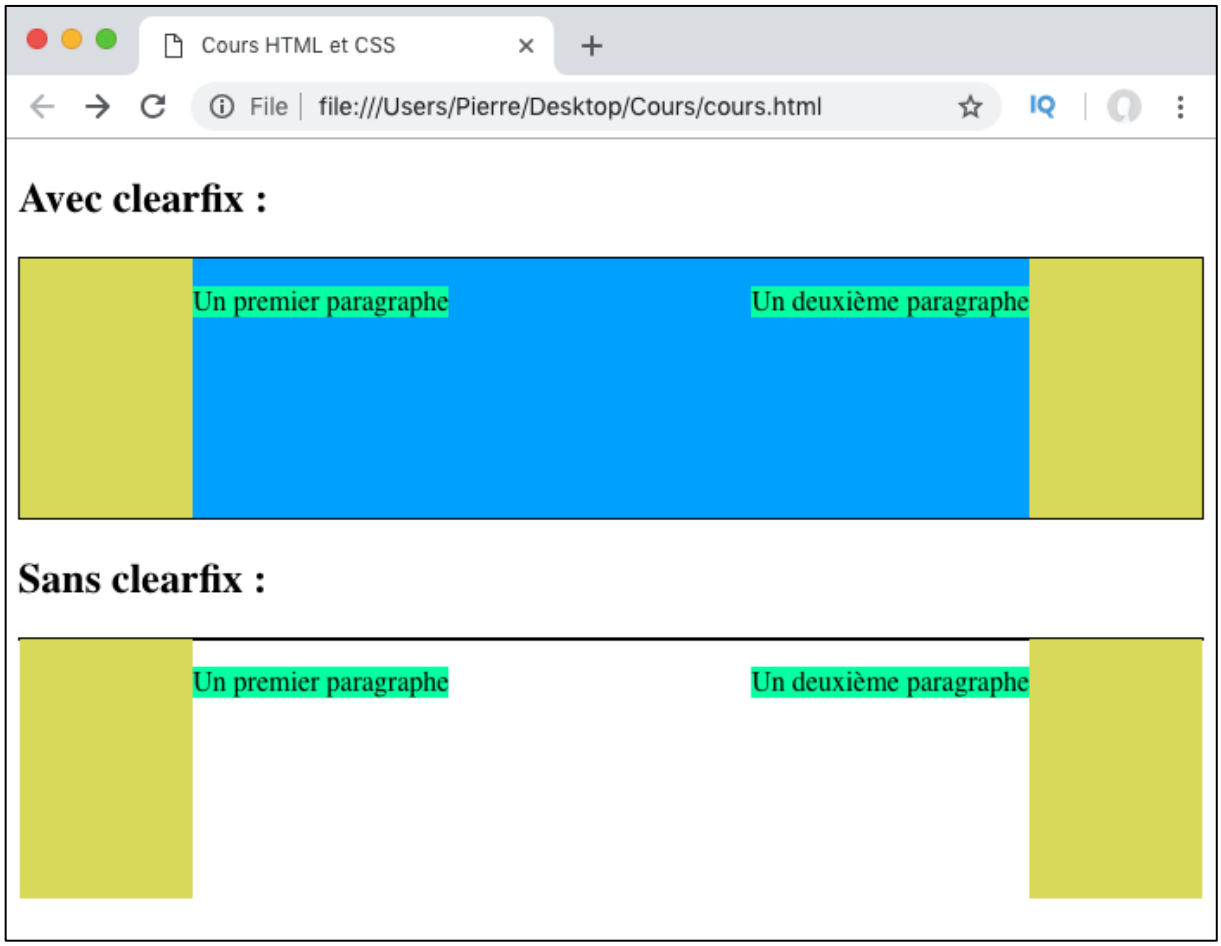
Cela peut donc mener à des problèmes de dépassement du flottant de son élément conteneur comme on a pu le voir précédemment.

Il y a un cas que nous n'avons cependant pas encore étudié : le cas où le conteneur ne contient que des éléments flottants. Dans ce cas-là, la hauteur du conteneur va être égale à la taille de ses bordures et de son padding et donc nulle si le conteneur ne possède ni bordures ni marges internes.

Ce comportement sera rarement souhaité. Pour rétablir la hauteur du conteneur, il suffit une nouvelle fois d'utiliser le clearfix vu dans le chapitre précédent.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h2>Avec clearfix : </h2>
        <div class="conteneur clearfix">
            <div class="interne gauche w100 h150"></div>
            <p class="gauche">Un premier paragraphe</p>
            <div class="interne droite w100 h150"></div>
            <p class="droite">Un deuxième paragraphe</p>
        </div>

        <h2>Sans clearfix :</h2>
        <div class="conteneur">
            <div class="interne gauche w100 h150"></div>
            <p class="gauche">Un premier paragraphe</p>
            <div class="interne droite w100 h150"></div>
            <p class="droite">Un deuxième paragraphe</p>
        </div>
    </body>
</html>
```

```css
p{
  background-color: #0FA;/*Vert*/
}
.conteneur{
  background-color: #0AF;/*Bleu*/
  border: 1px solid black;
  margin: 10px 0px;/*10px de marge haute et basse*/
}
.interne{
  background-color: #DD6;/*Jaune*/
}
.w100{width: 100px;}
.h150{height:150px;}

.gauche{
  float: left;
}
.droite{
  float: right;
}

.clearfix::after{
  content: "";
  display: table;
  clear: both;
}
```


Cours HTML et CSS
File | file:///Users/Pierre/Desktop/Cours/cours.html
Avec clearfix :
Un premier paragraphe
Un deuxième paragraphe
Sans clearfix :
Un premier paragraphe
Un deuxième paragraphe

Gestion des conflits entre display, position et float

En fonction des valeurs passées, la définition d'une propriété display, float ou position peut créer des conflits ou être incompatible avec chacune des deux autres si elles ont été également définies.

En effet, les propriétés display, position et float servent toutes les trois à modifier la disposition des éléments dans la page et vont pour cela modifier le flux normal de la page. Dans ces cas-là le CSS définira de nouvelles valeurs qui ne poseront pas de conflit pour l'une ou l'autre des propriétés. C'est ce qu'on appelle une « valeur calculée » par opposition à la « valeur spécifiée ».

Il va être très important de bien connaitre ces situations pour pouvoir prévoir le comportement de chaque propriété et pour pouvoir les définir correctement les unes par rapport aux autres et ainsi obtenir in-fine le design de page souhaité.

Gestion des conflits et valeurs calculées

Résumons ici ce qu'il se passe lorsqu'on utilise les propriétés display, position et float sur un même ensemble d'éléments en fonction des valeurs données.

Nous ne parlerons pas ici des valeurs des propriétés toujours en développement et ne faisant pas encore partie des recommandations officielles car leur impact n'est pas encore bien défini et car leur définition pourrait changer ou qu'elles pourraient être tout simplement abandonnées.

Voici les règles qui vont s'appliquer dans leur ordre d'application :

1. Si un élément possède un display : none, alors les propriétés float et position ne s'appliqueront évidemment pas ;
2. Si un élément est positionné avec position : absolute ou position : fixed, alors la propriété float ne s'appliquera pas (sa valeur calculée sera none et la valeur calculée du display sera celle du tableau ci-dessous ;
3. Si un élément N'est PAS positionné avec position : absolute ou position : fixed, la propriété floats'appliquera bien avec la valeur spécifiée. La valeur de display sera calculée selon le tableau ci-dessous ;
4. Si un élément N'est PAS positionné avec position : absolute ou position : fixed, qu'on NE lui applique PAS de float et que c'est l'élément racine du document, alors la valeur de displayappliquée sera la valeur calculée précisée dans le tableau ci-dessous ;
5. Pour tout autre élément NON positionné avec position : absolute ou position : fixed et auquel on N'applique PAS de float, la valeur de display appliquée sera la même que celle spécifiée.

Valeur spécifiée	Valeur calculée / appliquée
inline-table	table
inline, table-row-group, table-column, table-column-group, table-header-group, table-footer-group, table-row, table-cell, table-caption, inline-block	block
autre	identique à la valeur spécifiée

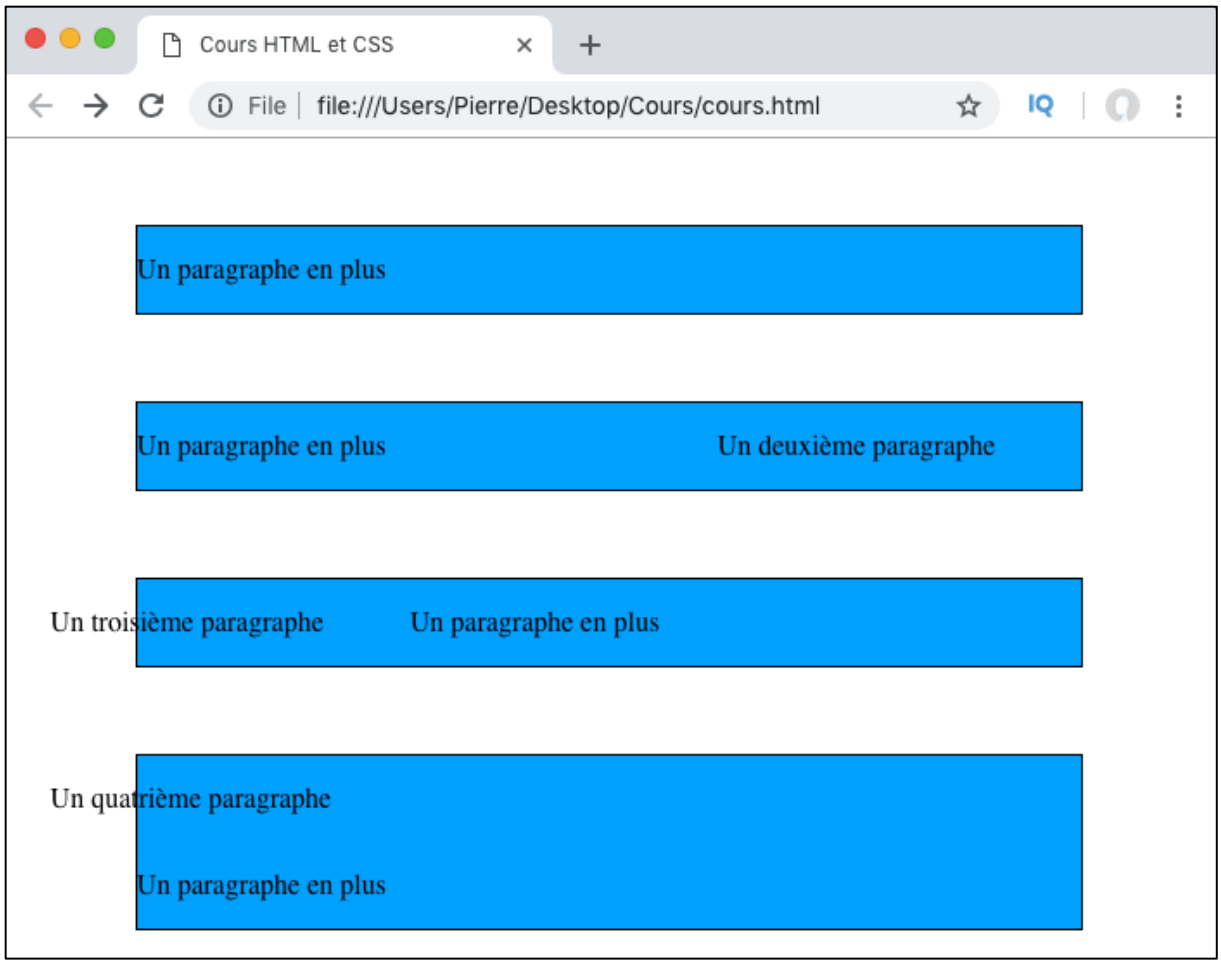
Illustration des règles de calcul des valeurs en cas de conflit

Les exemples suivants illustrent quelques situations conflictuelles entre les propriétés `display`, `position` et `float`.

Je vous laisse vous référer aux règles précédemment citées pour comprendre les valeurs appliquées !

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur">
            <p id="p1">Un premier paragraphe</p>
            <p>Un paragraphe en plus</p>
        </div>

        <div class="conteneur">
            <p id="p2">Un deuxième paragraphe</p>
            <p>Un paragraphe en plus</p>
        </div>

        <div class="conteneur">
            <p id="p3">Un troisième paragraphe</p>
            <p>Un paragraphe en plus</p>
        </div>
        <div class="conteneur">
            <p id="p4">Un quatrième paragraphe</p>
            <p>Un paragraphe en plus</p>
        </div>
    </body>
</html>
```

```css
.conteneur{
  width: 80%;
  border: 1px solid black;
  background-color: #0AF;
  margin: 50px auto;
  box-sizing: border-box;
  position: relative;
}
.conteneur::after{
  display: table;
  content: "";
  clear: both;
}

#p1{
  display: none;
  position: absolute;
  left: 100px;
  float: left;
}

#p2{
  display: inline-block;
  position: absolute;
  right: 50px;
  float: left;
}
#p3{
  display: inline-block;
  position: relative;
  right: 50px;
  float: left;
}
#p4{
  display: inline-block;
  position: relative;
  right: 50px;
}
```



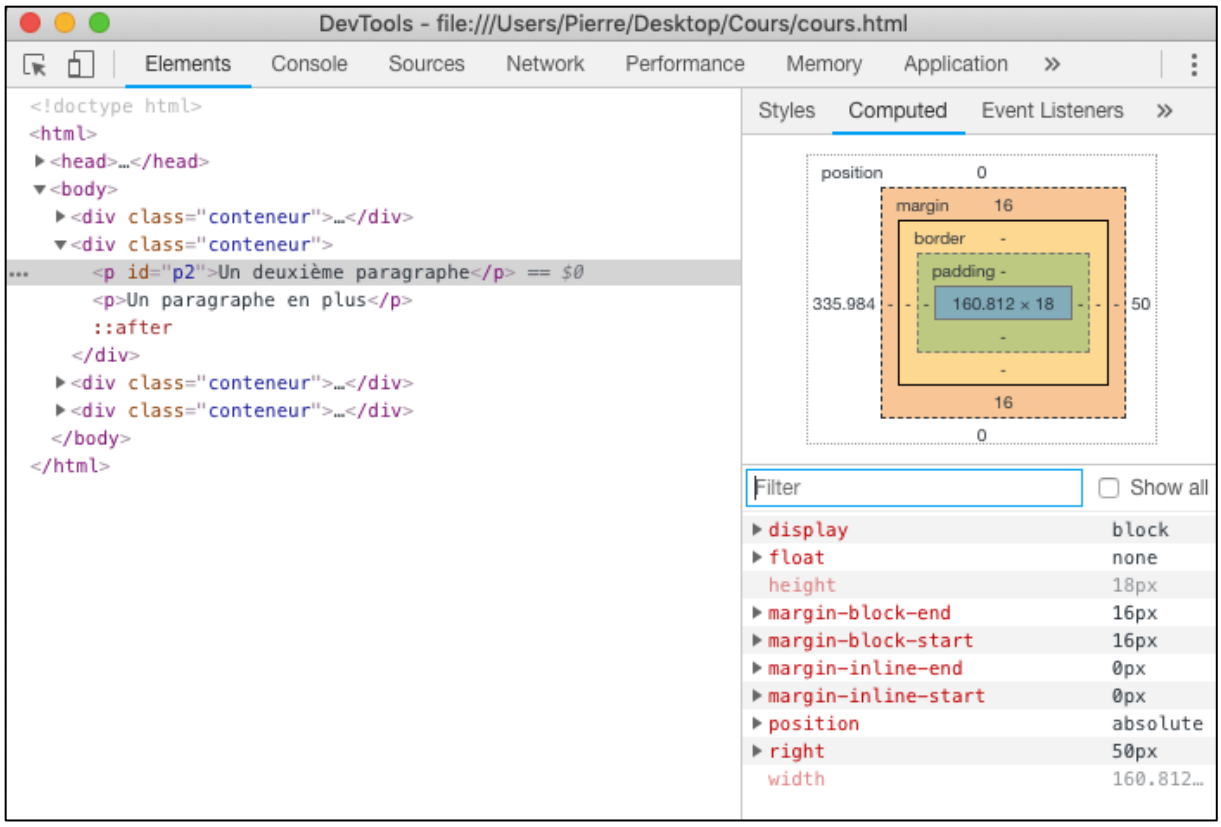
Ici, notre premier paragraphe `p id="p1"` possède un `display : none`. Les propriétés `position` (et `top`, `left`, etc.) et `float` vont donc être ignorées.

Le paragraphe `p id="p2"` possède une `position : absolute`. La valeur calculée de sa propriété `float` va donc être `none` et la valeur calculée du `display` va être `block`.

Notre paragraphe `p id="p3"` est positionné avec `position : relative` et non `display` n'est pas `none`. La propriété `float` va donc ici bien s'appliquer avec la propriété `right`. L'élément va donc flotter à gauche et être décalé de 50px vers la gauche par rapport à sa position d'origine, ce qui va le faire sortir de son conteneur. Comme l'élément flotte, la valeur calculée de son `display` va être `block`.

Finalement, notre paragraphe `p id="p4"` est positionné avec `position : relative` et ne possède pas de `float`. La valeur spécifiée du `display` va donc bien être appliquée.

Une astuce : en cas de doute sur les valeurs des propriétés appliquées à vos éléments, vous pouvez les inspecter (en faisant clic droit sur la page après avoir activé les outils pour développeur de votre navigateur) et aller voir les valeurs calculées (« computed » en anglais). Attention cependant aux inconsistances possibles entre les différents navigateurs.


DevTools - file:///Users/Pierre/Desktop/Cours/cours.html
Elements
Console
Sources
Network
Performance
Memory
Application
<!doctype html>
<html>
<head>…</head>
<body>
…

Un deuxième paragraphe == $0
Un paragraphe en plus
::after

…
…
</body>
</html>
Styles
Computed
Event Listeners
position 0
margin 16
border -
padding -
335.984
160.812 × 18
50
16
0
Filter
Show all
display block
float none
height 18px
margin-block-end 16px
margin-block-start 16px
margin-inline-end 0px
margin-inline-start 0px
position absolute
right 50px
width 160.812…

PARTIE VII

Les tableaux HTML

Créer des tableaux en HTML

Un tableau en HTML représente un ensemble organisé de données. Pour créer un tableau en HTML nous allons utiliser l'élément `table` qui signifie « tableau » en anglais.

Les tableaux sont une notion importante du HTML et il est important que vous sachiez comment les créer. Dans cette leçon, nous allons voir comment créer un tableau simple.

Définition et utilité des tableaux HTML

Les tableaux en HTML vont nous permettre de présenter des données de manière organisée et sous une certaine forme pour les structurer et les rendre compréhensibles pour les navigateurs, moteurs de recherche et utilisateurs.

Historiquement, de nombreux développeurs et web designer se sont servi à tort des tableaux pour mettre en forme de pages web. En effet, à l'époque, utiliser des tableaux semblait être un bon moyen de contrôler l'affichage et le placement des différents éléments d'une page en les plaçant dans des cellules.

Dans cette lignée, de nombreux attributs HTML permettant de modifier / mettre en forme un tableau avaient été créés.

Cependant, cette utilisation est à bannir aujourd'hui. En effet, je vous rappelle que le rôle du HTML est et a toujours été de structurer du contenu et de lui donner du sens, pas de mettre en forme ledit contenu (ce qui est le rôle du CSS).

Ainsi, vous comprendrez également que pour mettre en forme visuellement un tableau, nous n'utiliserons pas les attributs HTML (qui sont en grande majorité dépréciés) mais plutôt des propriétés CSS.

Les éléments constitutifs essentiels d'un tableau HTML

Un tableau est un ensemble de lignes et de colonnes. L'intersection entre une ligne et une colonne est une cellule de tableau.

Pour créer un tableau fonctionnel en HTML, nous allons devoir utiliser à minima 3 éléments :

- Un élément `table` (« tableau » en français) qui va définir le tableau en soi ;
- Des éléments `tr`, pour « table row » ou « ligne de tableau » en français qui vont nous permettre d'ajouter des lignes dans notre tableau ;
- Des éléments `td`, pour « table data » ou « donnée de tableau » en français qui vont nous permettre d'ajouter des cellules dans nos lignes et ainsi de créer automatiquement de nouvelles colonnes.

L'élément HTML `table` va représenter le tableau en soi. Cet élément est composé d'une paire de balises ouvrante `<table>` et fermante `</table>` au sein desquelles nous allons

placer les différents autres éléments de notre tableau. Les éléments `tr` et `td` sont également représentés sous la forme d'une paire de balises avec leur contenu entre les balises.

Créer un tableau simple en HTML

Pour créer un tableau en HTML, il y a une chose que vous devez bien comprendre qui est que les tableaux HTML vont être créés ligne par ligne.

A chaque fois que nous voudrons ajouter une ligne dans notre tableau, nous utiliserons un nouvel élément `tr`.

Nous allons ensuite pouvoir ajouter autant d'éléments `td` au sein de chacune de nos lignes. Chaque élément `td` va représenter une cellule dans le tableau.

Par exemple, pour créer un tableau en HTML contenant 3 lignes contenant chacune 4 cellules (c'est-à-dire un tableau de 3 lignes, 4 colonnes), nous utiliserons notre élément `table` qui va contenir 3 éléments `tr` et chaque élément `tr` contiendra 4 éléments `td` comme ceci (ne tenez pas compte de la mise en forme pour le moment, nous en reparlerons plus tard) :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table>
            <tr>
                <td>Nom</td>
                <td>Prénom</td>
                <td>Age</td>
                <td>Mail</td>
            </tr>
            <tr>
                <td>Giraud</td>
                <td>Pierre</td>
                <td>28</td>
                <td>pierre.giraud@edhec.com</td>
            </tr>
            <tr>
                <td>Joly</td>
                <td>Pauline</td>
                <td>27</td>
                <td>pjl@gmail.com</td>
            </tr>
        </table>
    </body>
</html>
```

Nom	Prénom	Age	Mail
Giraud	Pierre	28	pierre.giraud@edhec.com
Joly	Pauline	27	pjl@gmail.com

Nous venons ci-dessus de créer notre premier tableau entièrement en HTML. Retenez bien la syntaxe et particulièrement l'ordre d'imbrication des éléments avec nos éléments `td` à l'intérieur de nos éléments `tr` : on va utiliser un nouvel élément `td` dès que l'on va vouloir ajouter une cellule (et donc une colonne) au sein d'une même ligne.

Si vous essayez de ne recopier que le code HTML présenté ci-dessus et de l'afficher, vous risquez d'avoir un objet qui ne ressemble pas à l'idée que l'on se fait d'un tableau. C'est tout à fait normal puisque j'ai déjà ici mis sommairement notre premier tableau HTML en forme en utilisant des propriétés CSS.

Retenez cependant qu'au sens du HTML, et même sans mise en forme CSS, l'objet créé est bien un tableau. J'attire votre attention ici sur le fait que le rôle du HTML n'est que de structurer du contenu, c'est-à-dire le rendre intelligible pour les navigateurs et moteurs de recherche.

Notez par ailleurs que différentes lignes peuvent contenir un nombre différent de cellules même si cela est considéré comme une mauvaise pratique pour des raisons évidentes de sémantique et de lisibilité. Dans le cas où cela arriverait, la largeur de notre tableau HTML (i.e son nombre de colonnes) serait égale au nombre de cellules de la ligne contenant le plus de cellules.

Regardez plutôt l'exemple suivant où on a créé un tableau HTML de 3 lignes avec une première ligne contenant 4 cellules, une deuxième ligne contenant seulement 2 cellules, et la dernière ligne contenant 3 cellules :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <table>
            <tr>
                <td>Nom</td>
                <td>Prénom</td>
                <td>Age</td>
                <td>Mail</td>
            </tr>
            <tr>
                <td>Giraud</td>
                <td>Pierre</td>
            </tr>
            <tr>
                <td>Joly</td>
                <td>Pauline</td>
                <td>27</td>
            </tr>
        </table>
    </body>
</html>
```

Encore une fois, retenez que le fait de créer des lignes de tableau avec un nombre de cellules différent dans chacune est considéré comme une mauvaise pratique. Pour cela, nous attribuerons toujours un même nombre de `td` à nos différentes lignes. Si on souhaite laisser une cellule vide, nous nous contenterons d'écrire `<td> </td>` sans rien écrire entre les balises.

Imaginons par exemple que nous voulions laisser la cellule « age » vide pour la ligne relative à « Pierre » Pour cela, il suffit de créer un élément `td` pour son âge et de ne mettre aucun contenu dedans.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table>
            <tr>
                <td>Nom</td>
                <td>Prénom</td>
                <td>Age</td>
                <td>Mail</td>
            </tr>
            <tr>
                <td>Giraud</td>
                <td>Pierre</td>
                <td></td>
                <td>pierre.giraud@edhec.com</td>
            </tr>
            <tr>
                <td>Joly</td>
                <td>Pauline</td>
                <td>27</td>
            </tr>
        </table>
    </body>
</html>
```

Nom	Prénom	Age	Mail
Giraud	Pierre		pierre.giraud@edhec.com
Joly	Pauline	27	

Structurer un tableau HTML

Dans la leçon précédente, nous avons créé un premier tableau très simple. Ici, je vous rappelle que les tableaux sont des objets HTML qui servent à structurer et à donner du sens à des contenus en les organisant.

Souvent, lorsque nous choisirons d'organiser de l'information dans un tableau, cette information sera hiérarchisée et on voudra que notre tableau possède une ligne d'en-tête dans laquelle on va préciser le type de données attendues dans chaque colonne voire une légende, etc.

Dans cette nouvelle leçon, nous allons découvrir de nouveaux éléments et attributs qui vont nous servir à structurer nos tableaux HTML pour donner un maximum de sens à chaque donnée.

Ajouter une ligne d'en-tête à un tableau HTML

Très souvent, les tableaux vont posséder une ligne d'en-tête dans laquelle on va donner des informations au lecteur sur le type des données qui seront renseignées dans chaque colonne.

Cette ligne d'en-tête est différente des autres lignes puisqu'elle ne contient pas le même type d'informations que les autres lignes de notre tableau : elle sert simplement à indiquer quelles informations vont figurer dans le tableau. Nous allons ainsi pouvoir différencier cette ligne des autres en HTML à l'aide d'un élément particulier.

Pour créer une ligne d'en-tête en HTML, nous allons cette fois-ci devoir utiliser l'élément `th`, pour « table head » ou « en-tête du tableau » en français à la place de nos éléments `td` dans notre première ligne.

On peut appliquer un style particulier en CSS à cette ligne afin de bien la démarquer des autres. Par défaut, le texte au sein des éléments `th` s'affichera en gras dans la plupart des navigateurs.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table>
            <tr>
                <th>Nom</th>
                <th>Prénom</th>
                <th>Age</th>
                <th>Mail</th>
            </tr>
            <tr>
                <td>Giraud</td>
                <td>Pierre</td>
                <td>28</td>
                <td>pierre.giraud@edhec.com</td>
            </tr>
            <tr>
                <td>Joly</td>
                <td>Pauline</td>
                <td>27</td>
                <td>pjl@gmail.com</td>
            </tr>
        </table>
    </body>
</html>
```

Nom	Prénom	Age	Mail
Giraud	Pierre	28	pierre.giraud@edhec.com
Joly	Pauline	27	pjl@gmail.com

Si vous avez ajouté des bordures autour de chaque cellule de votre tableau en CSS, pensez bien ici à appliquer ces mêmes bordures aux éléments `th` en plus des éléments `td` pour avoir des bordures autour des cellules de la ligne d'en-tête également.

Création d'un tableau structuré en HTML

Si vous devez créer des tableaux assez longs, il peut être judicieux de commencer à les structurer en les subdivisant en plusieurs parties.

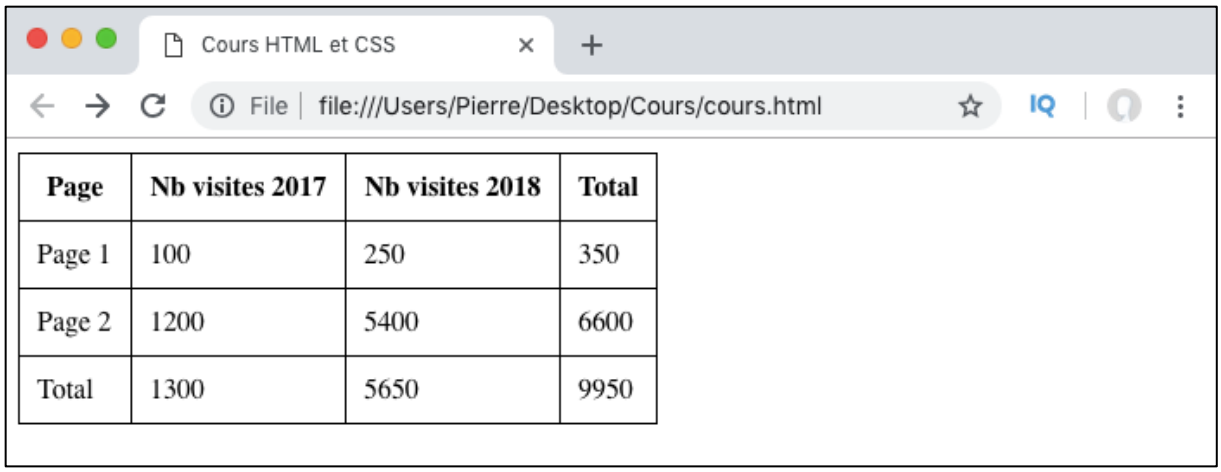
On peut distinguer trois parties dans un tableau : le haut du tableau contenant généralement la ligne d'en tête, le corps du tableau contenant les informations de notre tableau en soi et le pied du tableau servant à calculer des totaux ou à rappeler les informations d'en tête si votre tableau est vraiment long.

Pour définir chaque partie d'un tableau, nous disposons d'un élément HTML spécifique :

- `thead` pour la tête du tableau ;
- `tbody` pour le corps du tableau ;
- `tfoot` pour le pied du tableau.

Voyons immédiatement comment utiliser ces nouveaux éléments judicieusement.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table>
            <thead>
              <tr>
                <th>Page</th>
                <th>Nb visites 2017</th>
                <th>Nb visites 2018</th>
                <th>Total</th>
              </tr>
            </thead>
            <tbody>
              <tr>
                <td>Page 1</td>
                <td>100</td>
                <td>250</td>
                <td>350</td>
              </tr>
              <tr>
                <td>Page 2</td>
                <td>1200</td>
                <td>5400</td>
                <td>6600</td>
              </tr>
            </tbody>
            <tfoot>
              <tr>
                <td>Total</td>
                <td>1300</td>
                <td>5650</td>
                <td>9950</td>
              </tr>
            </tfoot>
        </table>
    </body>
</html>
```



Page	Nb visites 2017	Nb visites 2018	Total
Page 1	100	250	350
Page 2	1200	5400	6600
Total	1300	5650	9950

Vous pouvez par ailleurs noter ici une chose intéressante : il est strictement équivalent de mentionner un élément `tfoot` avant ou après un élément `tbody`. En effet, même si le `tfoot` est déclaré avant le `tbody`, son contenu s'affichera tout de même après dans le rendu final.

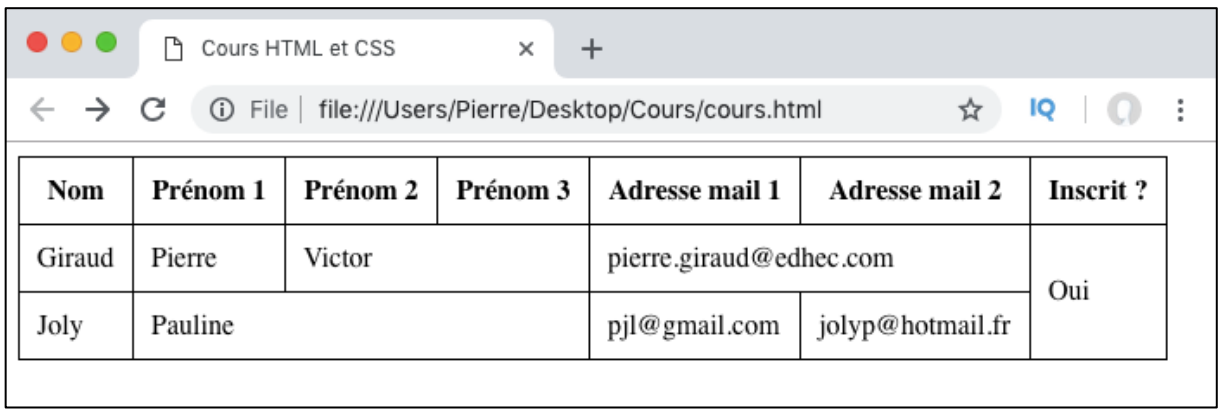
Cette curiosité nous vient du HTML4 dans lequel il était interdit de déclarer un élément `tfoot` après un élément `tbody` ; il fallait absolument le mentionner avant. Cette règle totalement contre intuitive a été supprimée avec le HTML5.

Fusionner des cellules entre elles avec colspan et rowspan

Le CSS, bien que puissant aujourd'hui, ne nous permet pas encore de tout faire et certains attributs HTML ne sont pas encore dépréciés. C'est notamment le cas des attributs `colspan` et `rowspan` qui vont nous permettre de fusionner plusieurs cellules adjacentes d'une même ligne ou d'une même colonne.

Ces attributs vont prendre en valeur le nombre de colonnes ou de lignes qu'une cellule doit couvrir, c'est-à-dire le nombre de colonnes ou de lignes qu'une cellule doit occuper ou encore sur lesquelles elle doit s'étendre. Essayons donc de fusionner plusieurs cellules entre elles dans un tableau HTML grâce à ces attributs :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table>
            <thead>
              <tr>
                <th>Nom</th>
                <th>Prénom 1</th>
                <th>Prénom 2</th>
                <th>Prénom 3</th>
                <th>Adresse mail 1</th>
                <th>Adresse mail 2</th>
                <th>Inscrit ?</th>
              </tr>
            </thead>
            <tbody>
              <tr>
                <td>Giraud</td>
                <td>Pierre</td>
                <td colspan="2">Victor</td>
                <td colspan="2">pierre.giraud@edhec.com</td>
                <td rowspan="2">Oui</td>
              </tr>
              <tr>
                <td>Joly</td>
                <td colspan="3">Pauline</td>
                <td>pjl@gmail.com</td>
                <td>jolyp@hotmail.fr</td>
              </tr>
            </tbody>
        </table>
    </body>
</html>
```



Nom	Prénom 1	Prénom 2	Prénom 3	Adresse mail 1	Adresse mail 2	Inscrit ?
Giraud	Pierre	Victor		pierre.giraud@edhec.com		Oui
Joly	Pauline			pjl@gmail.com	jolyp@hotmail.fr	

Notez bien ici qu'il faudra bien réfléchir à la construction de votre tableau lorsque vous utilisez ces attributs. Par exemple, si on définit qu'une cellule doit couvrir 3 lignes, il faudra omettre un élément `td` pour les deux lignes supplémentaires sur lesquelles la cellule s'étend et de même pour une cellule couvrant l'espace relatif à plusieurs colonnes.

Ajouter une légende à un tableau HTML avec caption

On va également pouvoir ajouter un titre ou une légende à notre tableau afin d'indiquer ce que contient notre tableau.

Pour ajouter une légende, nous allons utiliser l'élément HTML `caption`. Cet élément est très simple d'utilisation, mais il faut respecter une règle : il doit être inséré immédiatement après la balise ouvrante de l'élément `table`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table>
            <caption>Informations participants</caption>
            <thead>
              <tr>
                <th>Nom</th>
                <th>Prénom</th>
                <th>Adresse mail</th>
                <th>Inscrit ?</th>
              </tr>
            </thead>
            <tbody>
              <tr>
                <td>Giraud</td>
                <td>Pierre</td>
                <td>pierre.giraud@edhec.com</td>
                <td rowspan="2">Oui</td>
              </tr>
              <tr>
                <td>Joly</td>
                <td>Pauline</td>
                <td>pjl@gmail.com</td>
              </tr>
            </tbody>
        </table>
    </body>
</html>
```

Utiliser les éléments col et colgroup pour préparer la mise en forme des colonnes d'un tableau

Nous avons vu précédemment qu'en HTML nous créons les tableaux ligne par ligne avec des éléments `tr`. Cela fait qu'il est très simple de mettre en forme en CSS les différentes lignes d'un tableau HTML, en utilisant pas exemple une pseudo-classe `:nth-child()` que nous étudierons plus tard dans ce cours.

Cependant, cela est beaucoup plus compliqué lorsque l'on souhaite mettre en forme une colonne d'un tableau HTML puisque les cellules représentées par nos éléments `td` qui créent automatiquement ces colonnes sont dispersées au sein des différentes lignes.

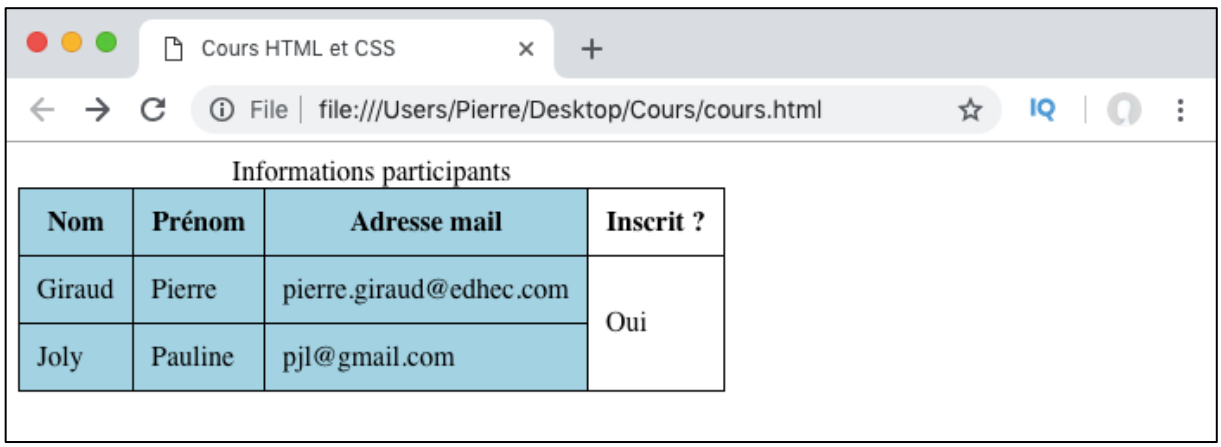
Pour régler ce problème de mise en forme, le HTML nous offre deux éléments qu'on va pouvoir intégrer dans nos tableaux : les éléments `col` et `colgroup`.

L'élément `colgroup` va représenter un groupe d'une ou plusieurs colonnes dans un tableau qu'on va pouvoir ensuite mettre en forme en CSS. Cet élément est représenté par une paire de balises et doit être un enfant direct de l'élément `table`, précédé éventuellement uniquement par un élément `caption`. Cela signifie qu'il doit être inséré avant tout élément `thead`, `tbody`, `tfoot`, and `tr` au sein de l'élément `table`.

Au sein d'un élément `colgroup`, nous allons pouvoir insérer autant d'éléments `col` qu'il y a de colonnes dans notre tableau. Chaque élément `col` représente une colonne de notre tableau. Cet élément ne sert également qu'à la mise en forme et est représenté par une unique balise orpheline `<col>`.

Pour représenter plusieurs colonnes de notre tableau avec un seul élément `col`, on peut utiliser un attribut `span` qui va prendre en valeur le nombre de colonnes de notre tableau que l'élément `col` doit représenter.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table>
            <caption>Informations participants</caption>
            <colgroup>
                <col span="3" class="info">
                <col>
            </colgroup>
            <thead>
              <tr>
                <th>Nom</th>
                <th>Prénom</th>
                <th>Adresse mail</th>
                <th>Inscrit ?</th>
              </tr>
            </thead>
            <tbody>
              <tr>
                <td>Giraud</td>
                <td>Pierre</td>
                <td>pierre.giraud@edhec.com</td>
                <td rowspan="2">Oui</td>
              </tr>
              <tr>
                <td>Joly</td>
                <td>Pauline</td>
                <td>pjl@gmail.com</td>
              </tr>
            </tbody>
        </table>
    </body>
</html>
```



Informations participants

Nom	Prénom	Adresse mail	Inscrit ?
Giraud	Pierre	pierre.giraud@edhec.com	Oui
Joly	Pauline	pjl@gmail.com	

Ici, notre tableau possède quatre colonnes. On veut appliquer une couleur de fond pour toutes les cellules des trois premières colonnes d'un coup.

On utilise donc un élément `colgroup` dans lequel on place deux éléments `col`. On passe un attribut `span ="3"` à notre premier élément `col` afin que cet élément représente les trois premières colonnes de notre tableau.

Nous n'avons alors plus qu'à appliquer une couleur de fond à cet élément `col` pour ajouter un fond à chacune des cellules des trois premières colonnes de notre tableau. Pour le cibler facilement, on lui attribue donc une `class`.

Mettre en forme un tableau HTML

Les tableaux sont des entités très particulières en HTML et des propriétés CSS ont donc été créées spécialement pour les mettre en forme et les personnaliser.

Dans cette leçon, nous allons découvrir ces différentes propriétés et apprendre à les utiliser.

Mettre en forme un tableau : attributs HTML ou CSS ?

S'il est vrai que le HTML sert à structurer le contenu et à lui donner du sens, nous créons des sites pour des visiteurs humains et donc nous voulons que ces sites soient compréhensibles pour eux et agréables à consulter.

Pour cela, nous devons généralement mettre en forme notre contenu HTML brut afin de mieux le présenter. Pour les tableaux, comme pour la majorité des éléments HTML, nous utilisions à l'époque des attributs de mise en forme HTML.

Les attributs les plus communs utilisés avec les tableaux étaient les suivants :

- `align` pour définir l'alignement du tableau dans la page ;
- `bgcolor` pour définir la couleur de fond du tableau ;
- `border` pour définir la taille d'une bordure autour du tableau ;
- `cellpadding` pour définir l'espace entre le contenu d'une cellule et sa bordure ;
- `cellspacing` pour définir l'espace entre deux cellules ;
- `char` pour aligner le contenu au sein d'une ligne à partir d'un certain caractère ;
- `charoff` pour définir le nombre de caractères à partir desquels le contenu doit être aligné depuis le caractère défini par l'attribut `char` ;
- `frame` pour définir les côtés du tableau sur lesquels tracer une bordure ;
- `rules` pour définir la manière dont les traits doivent apparaître au sein du tableau ;
- `summary` pour définir un texte servant à résumer le tableau au cas où celui-ci ne pourrait pas être affiché ;
- `valign` pour aligner verticalement le contenu des cellules au sein d'une ligne.

Ces attributs sont tous néanmoins tous dépréciés et vous ne devriez jamais les utiliser pour mettre en forme vos tableaux HTML. A la place, nous utiliserons plutôt le CSS qui nous offre aujourd'hui de très bonnes alternatives et un bon support pour la mise en forme de nos tableaux.

Ainsi, nous allons pouvoir par exemple ajouter des bordures autour de chaque cellule de notre tableau en appliquant la propriété `border` à nos éléments `td` puis fusionner les différentes bordures en une pour harmoniser l'ensemble en appliquant un `border-collapse: collapse` à notre élément `table`.

Nous allons encore pouvoir modifier la couleur de fond de notre tableau ou d'un élément particulier de notre tableau ou aligner son contenu en CSS.

Dans cette leçon, nous allons voir les propriétés CSS suivantes liées aux tableaux :

- border-collapse ;
- border-spacing ;
- caption-side ;
- empty-cells ;
- table-layout ;
- vertical-align.

Les propriétés border-collapse et border-spacing

La création de bordures dans les tableaux semble complexe à priori. En effet, en appliquant une propriété border à notre élément table, nous allons créer une bordure autour de notre tableau mais pas entre chaque cellule de celui-ci.

Au contraire, en appliquant une bordure à chaque élément th ou td, c'est-à-dire à chaque cellule de notre tableau, les bordures autour de chaque cellule ne vont pas se coller entre elles, ce qui n'est généralement pas le comportement voulu.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table class="t1">
            <tr><th>Nom</th><th>Prénom</th></tr>
            <tr><td>Giraud</td><td>Pierre</td></tr>
          </table>
          <br>
          <table class="t2">
            <tr><th>Nom</th><th>Prénom</th></tr>
            <tr><td>Joly</td><td>Pauline</td></tr>
          </table>
          <br>
          <table class="t1 t2">
            <tr><th>Nom</th><th>Prénom</th></tr>
            <tr><td>Giraud</td><td>Pierre</td></tr>
            <tr><td>Joly</td><td>Pauline</td></tr>
          </table>
    </body>
</html>
```

```
.t1{
  border : 2px solid black;
}
.t2 td, .t2 th{
  border: 1px solid orange;
}
```

Les propriétés `border-collapse` et `border-spacing` ont été créées pour répondre à ce problème.

La propriété `border-collapse` va nous permettre de choisir si on veut faire fusionner les bordures des différentes cellules de notre tableau ou pas. On va pouvoir lui passer l'une de ces deux valeurs :

- `separate` : Valeur par défaut ; les bordures seront séparées et distinctes ;
- `collapse` : Les bordures adjacentes vont être fusionnées.

Nous allons donc généralement appliquer `border` à nos éléments `th` et `td` ainsi que la propriété `border-collapse` avec sa valeur `collapse` à notre élément `table`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table>
            <tr>
                <th>Nom</th>
                <th>Prénom</th>
                <th>Mail</th>
            </tr>
            <tr>
                <td>Giraud</td>
                <td>Pierre</td>
                <td>pierre.giraud@edhec.com</td>
            </tr>
            <tr>
                <td>Joly</td>
                <td>Pauline</td>
                <td>pjl@gmail.com</td>
            </tr>
        </table>
    </body>
</html>
```

```css
table{
  border-collapse: collapse;
}

th, td{
  border: 1px solid black;
  padding: 10px;
}
```

Nom	Prénom	Mail
Giraud	Pierre	pierre.giraud@edhec.com
Joly	Pauline	pjl@gmail.com

Dans le cas où l'on souhaite conserver des bordures séparées, on va tout de même pouvoir gérer la distance entre chaque bordure grâce à la propriété `border-spacing`.

Cette propriété va prendre une distance en valeur (en `px`, `cm`, `em`, `rem`, etc.) qui représente l'espace entre deux bordures adjacentes. Il faudra à nouveau l'appliquer à l'élément `table`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table class="t1">
            <tr>
              <th>Nom</th>
              <th>Prénom</th>
              <th>Mail</th>
            </tr>
            <tr>
              <td>Giraud</td>
              <td>Pierre</td>
              <td>pierre.giraud@edhec.com</td>
            </tr>
          </table>

          <table class="t2">
            <tr>
              <th>Nom</th>
              <th>Prénom</th>
              <th>Mail</th>
            </tr>
            <tr>
              <td>Joly</td>
              <td>Pauline</td>
              <td>pjl@gmail.com</td>
            </tr>
        </table>
    </body>
</html>
```

```css
.t1{
  border-spacing: 15px;
}
.t2{
  border-spacing: 0px;
}
th, td{
  border: 2px solid black;
  padding: 10px;
}
```

Notez qu'indiquer un `border-spacing : 0px` ne va pas fusionner les bordures des cellules adjacentes comme le ferait `border-collapse : collapse` mais simplement coller les bordures des cellules adjacentes.

Contrôler la largeur des colonnes avec table-layout

La propriété `table-layout` va nous permettre de gérer la largeur des différentes colonnes de notre tableau ou plus exactement de choisir selon quel algorithme la largeur de chaque colonne doit être calculée.

On va pouvoir choisir parmi deux valeurs :

- `auto` : Valeur par défaut ; la largeur des différentes colonnes du tableau est calculée automatiquement afin que le tableau prenne le moins d'espace possible en hauteur. De plus, si une largeur a été définie avec `width` pour le tableau, alors celui-ci essaiera de la respecter tant que son contenu ne dépasse pas du tableau ;
- `fixed` : La largeur des colonnes va être déterminée à partir de la propriété `width` des éléments `table` ou `col` ou de la première ligne du tableau.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table class="t1">
            <tr>
              <th>Nom</th>
              <th>Prénom</th>
              <th>Mail</th>
            </tr>
            <tr>
              <td>Giraud</td>
              <td>Pierre</td>
              <td>pierre.giraud@edhec.com</td>
            </tr>
          </table>

          <table class="t2">
            <tr>
              <th>Nom</th>
              <th>Prénom</th>
              <th>Mail</th>
            </tr>
            <tr>
              <td>Giraud</td>
              <td>Pierre</td>
              <td>pierre.giraud@edhec.com</td>
            </tr>
        </table>
    </body>
</html>
```

```css
table{
  border-collapse: collapse;
  width: 300px;
  margin-bottom: 20px;
}
.t1{
  table-layout: fixed;
}

th, td{
  border: 1px solid black;
  padding: 10px;
}
```

Définir l'affichage des cellules vides d'un tableau avec empty-cells

La propriété `empty-cells` va nous permettre de choisir si l'arrière-plan et les bordures d'une cellule sans contenu doivent être affichés ou pas.

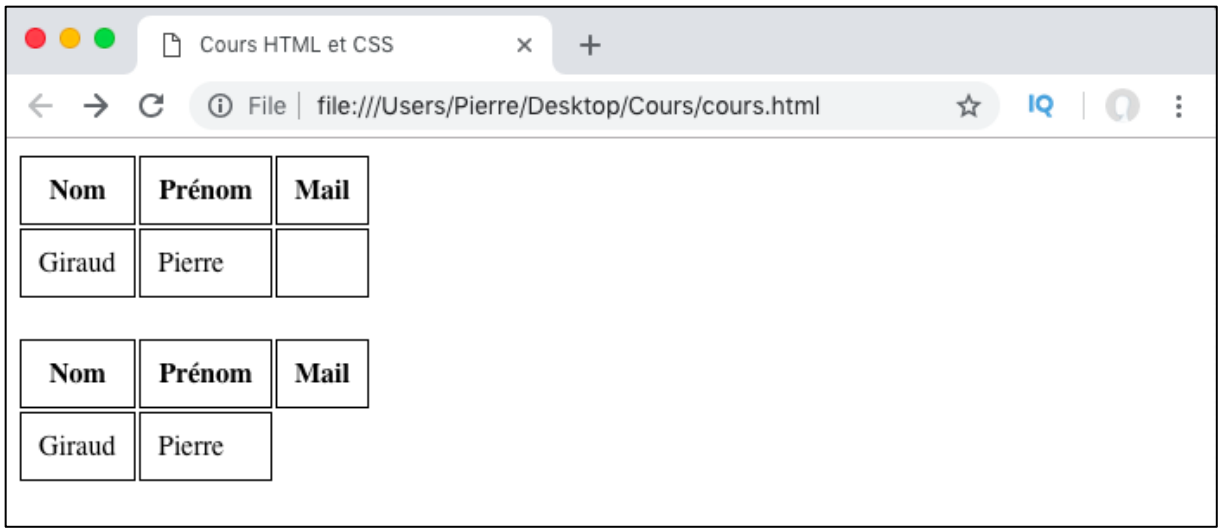
Notez que cette propriété ne va pouvoir s'appliquer que si la valeur de la propriété `border-collapse` pour le tableau est `separate`.

On va pouvoir choisir parmi deux valeurs :

- `show` : Valeur par défaut ; les bordures et l'arrière-plan des cellules vides du tableau seront bien visibles ;
- `hide` : Les bordures et l'arrière-plan des cellules vides du tableau ne sont pas affichés.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table class="t1">
            <tr>
              <th>Nom</th>
              <th>Prénom</th>
              <th>Mail</th>
            </tr>
            <tr>
              <td>Giraud</td>
              <td>Pierre</td>
              <td></td>
            </tr>
          </table>

          <table class="t2">
            <tr>
              <th>Nom</th>
              <th>Prénom</th>
              <th>Mail</th>
            </tr>
            <tr>
              <td>Giraud</td>
              <td>Pierre</td>
              <td></td>
            </tr>
        </table>
    </body>
</html>
```

```css
table{
  margin-bottom: 20px;
}
th, td{
  border: 1px solid black;
  padding: 10px;
}
.t2{
  empty-cells: hide;
}
```



Choisir où s'affiche la légende d'un tableau avec caption-side

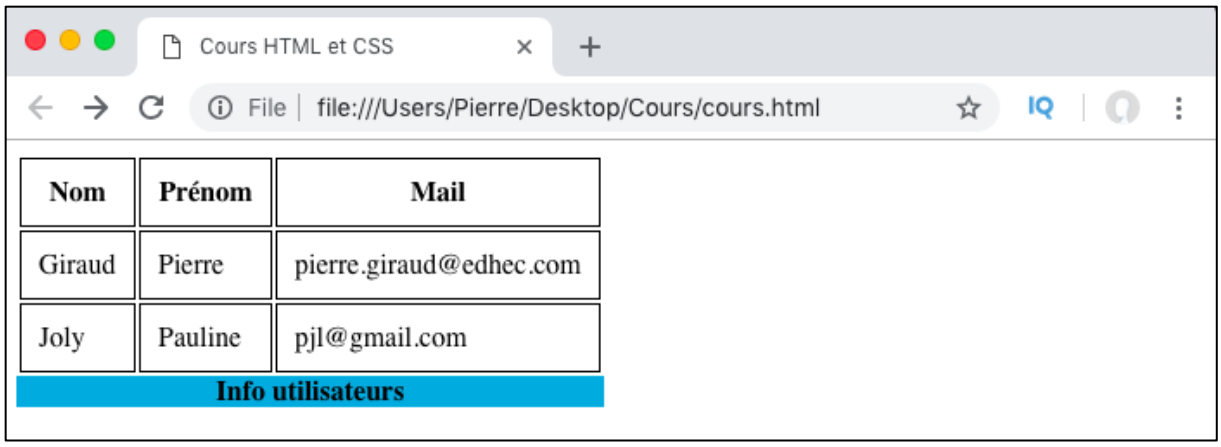
La propriété `caption-side` va nous permettre de choisir où doit s'afficher la légende d'un tableau, c'est-à-dire où doit s'afficher le texte présent dans l'élément `caption` du tableau.

Aujourd'hui, nous pouvons choisir entre deux valeurs qui représentent deux emplacements :

- `top` Valeur par défaut ; la légende sera positionnée au-dessus du tableau ;
- `bottom` : La légende sera positionnée en dessous du tableau.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table>
            <caption>Info utilisateurs</caption>
            <thead>
              <tr>
                <th>Nom</th>
                <th>Prénom</th>
                <th>Mail</th>
              </tr>
            </thead>
            <tbody>
              <tr>
                <td>Giraud</td>
                <td>Pierre</td>
                <td>pierre.giraud@edhec.com</td>
              </tr>
              <tr>
                <td>Joly</td>
                <td>Pauline</td>
                <td>pjl@gmail.com</td>
              </tr>
            </tbody>
        </table>
    </body>
</html>
```

```css
table{
  caption-side: bottom;
}
th, td{
  border: 1px solid black;
  padding: 10px;
}
caption{
  background-color: #0AD;
  font-weight: bold;
}
```



Aligner le contenu des cellules d'un tableau avec vertical-align

La propriété `vertical-align` sert à définir l'alignement vertical du contenu d'une cellule d'un tableau.

On va pouvoir lui passer une valeur de type longueur (en `px` ou en `em` par exemple), une valeur en pourcentage ou un mot clef à cette propriété.

Dans la grande majorité des cas, nous nous contenterons de choisir parmi l'un de ces 3 mots clefs :

- `top` : Le contenu de chaque cellule va se placer en haut de la cellule ;
- `middle` : Le contenu de chaque cellule va se placer au milieu de la cellule ;
- `bottom` : Le contenu de chaque cellule va se placer se bas de la cellule.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <table>
            <thead>
              <tr>
                <th>Nom</th>
                <th>Prénom</th>
                <th>Mail</th>
              </tr>
            </thead>
            <tbody>
              <tr>
                <td>Giraud</td>
                <td>Pierre Victor Raphael</td>
                <td>pierre.giraud@edhec.com</td>
              </tr>
              <tr>
                <td>Joly - Dechand</td>
                <td>Pauline</td>
                <td>pjl@gmail.com</td>
              </tr>
            </tbody>
        </table>
    </body>
</html>
```

```css
table{
  border-collapse: collapse;
  width: 400px;
}
th, td{
  border: 1px solid black;
  padding: 10px;
  vertical-align: top;
}
```

Nom	Prénom	Mail
Giraud	Pierre Victor Raphael	pierre.giraud@edhec.com
Joly - Dechand	Pauline	pjl@gmail.com

PARTIE VIII

Insérer des médias

Insérer des images en HTML

Nous allons pouvoir insérer des images au sein de nos fichiers HTML en utilisant l'élément HTML `img` représenté par une balise orpheline `<img>`.

Dans cette leçon, nous allons nous intéresser aux différents formats d'image (jpeg, png, etc.) et allons voir comment insérer une ou plusieurs images dans nos pages en HTML. Nous discuterons également de la notion d'accessibilité du web à tous ainsi que de la mise en forme des images.

Un point sur les différents formats d'image

Comme vous le savez certainement, vous pouvez enregistrer vos images sous différents formats. Les formats les plus utilisés sont :

- Le JPG ou JPEG ;
- Le PNG ;
- Le GIF ;
- Le BITMAP.

Chaque format possède ses propres spécificités et il faut donc faire bien attention au choix du format lorsqu'on enregistre une image.

Le format JPEG (pour Joint Photographic Expert Group), ou plus communément JPG est un format qui permet généralement de compresser le poids d'une image par dix tout en conservant une très bonne qualité. Généralement, nous choisirons ce format pour enregistrer des photos.

Le PNG (Portable Network Graphic) est un format qui a été créé à l'origine pour remplacer le format GIF. Le grand intérêt de ce format est qu'il gère la transparence. Celui-ci a un très bon taux de compression tout en conservant une bonne qualité d'image. Nous utiliserons généralement ce format pour enregistrer nos images qui ne sont pas des photographies.

Le GIF (Graphic Interchange Format) est un vieux format d'images que je ne recommande plus d'utiliser aujourd'hui, sauf dans le cas d'images animées.

Finalement, le BITMAP (ou BMP) est un format qui possède une très bonne prise en charge par tous les navigateurs et éditeurs. Cependant, la compression des images est assez mauvaise, ce qui donne au final des images très lourdes et donc longues à charger. Pour cette raison, je vous déconseille également d'utiliser le BITMAP pour enregistrer vos images.

Insérer des images en HTML

L'insertion d'images en HTML va se faire au moyen de l'élément HTML `img`. Cet élément est représenté par une balise orpheline.

Au sein de l'élément img, nous allons obligatoirement devoir préciser deux attributs : les attributs src et alt.

L'attribut src (pour source) va prendre comme valeur l'adresse de l'image (adresse relative ou absolue) tandis que l'attribut alt (pour alternative) va contenir un texte alternatif décrivant l'image. Ce texte va être affiché si l'image ne peut pas l'être pour une raison ou pour une autre, et est également très utile pour les non-voyants ou les mal voyants qui vont pouvoir « lire » notre image grâce à leurs lecteurs particuliers.

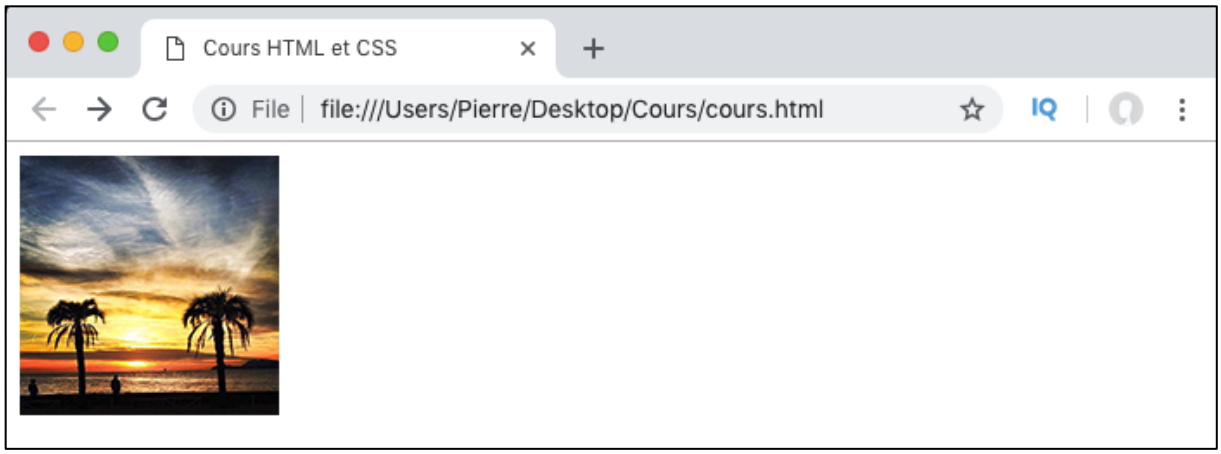
J'insiste ici sur l'importance de l'attribut alt : le web a été créé avec l'idée d'accessibilité à tous et il est donc de notre devoir de tout faire pour rendre chacune de nos pages lisibles pour tous et particulièrement pour les gens souffrant de déficiences.

Voyons maintenant comment fonctionne l'insertion d'images en pratique. On va commencer par enregistrer une image dans le même dossier que notre page HTML. Pour ma part, mon image s'appelle « sunset.png » et possède une largeur et une hauteur de 150 pixels.

Une nouvelle fois, choisissez un nom d'image sans espace ni caractère spécial (pas d'accent notamment). Cela évitera des problèmes potentiels.

L'attribut src va fonctionner de la même manière que l'attribut href pour les liens. Ainsi, si vous enregistrez votre image dans un dossier différent de votre page HTML, pensez bien à en tenir compte dans la valeur donnée à src.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <img src="sunset.png" alt="Coucher de soleil en bord de mer">
    </body>
</html>
```



Comme vous pouvez le voir, notre image s'affiche bien à sa taille d'origine, c'est-à-dire 150*150 pixels pour moi. Dans l'absolu, on essaiera de redimensionner l'image à priori (avant téléchargement sur serveur) pour éviter de consommer des ressources serveurs inutilement. En effet, une image plus grosse de base va demander plus de ressources et de temps à charger car celle-ci va être plus lourde.

Pour redimensionner une image à posteriori, nous avons deux façons de faire : soit en utilisant le CSS, soit en ajoutant des attributs `width` et `height` dans notre élément `img` en HTML.

Bien évidemment, nous préférerons toujours pour des raisons sémantiques et de maintenabilité du code dans la mesure du possible effectuer ces opérations en CSS. Cependant, dans certaines situations particulières, nous serons obligés de le faire en HTML et il est donc bon de savoir le faire.

Insérer des images provenant d'autres sites en HTML

Comme je vous l'ai précisé, on peut aussi préciser une adresse absolue comme valeur pour notre attribut `src`.

Par exemple, si je souhaite afficher une image provenant d'un autre site, j'utiliserai évidemment une adresse absolue, c'est-à-dire que j'indiquerai l'URL complète de la ressource image que je souhaite afficher en valeur de mon attribut `src`.

Vous pouvez essayer avec n'importe quelle image sur le web, cela fonctionnera. Faites cependant attention à bien récupérer l'URL complète de l'image ainsi qu'aux droits d'auteur ! Notez également qu'il est généralement déconseillé d'utiliser des images provenant d'autres sites car si ces sites les suppriment, elles ne s'afficheront plus non plus sur le vôtre.

Redimensionner une image en HTML

On va pouvoir ajouter des attributs facultatifs pour modifier l'affichage de la plupart des éléments HTML. Cependant, répétons-le, ce n'est jamais la façon recommandée de faire les choses pour des raisons de séparation de rôle des langages (raisons de sémantique) et de maintenabilité du code.

Le HTML est un langage de balisage qui ne doit normalement servir qu'à indiquer l'identité des différents contenus d'une page. Pour mettre en forme ces contenus, il faut utiliser dans la mesure du possible le CSS qui est là pour ça.

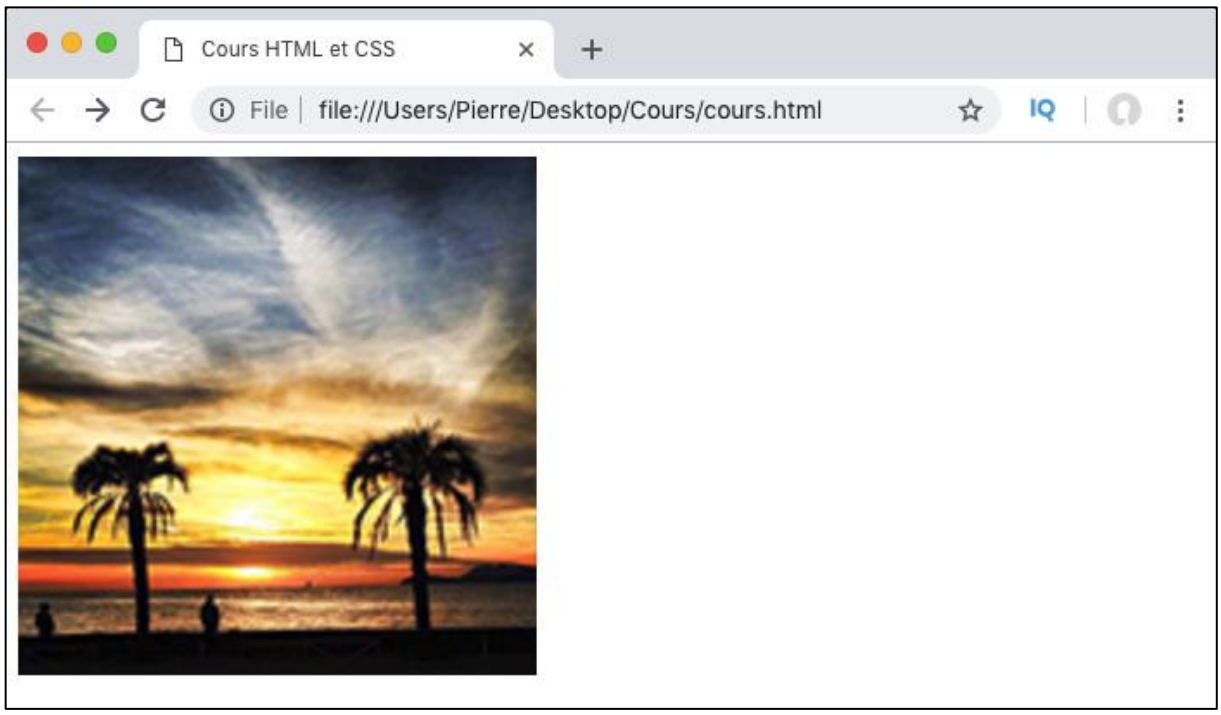
Cependant, dans certaines situations, nous n'allons pas avoir accès au CSS ou il va être beaucoup plus compliqué d'appliquer des styles à nos images en CSS qu'en HTML.

Dans ces cas-là, nous allons déjà pouvoir utiliser les attributs `width` (« largeur ») et `height` (« hauteur ») au sein de notre élément `img` pour modifier la largeur et la hauteur de l'image. Ces attributs peuvent prendre des valeurs absolues (en px généralement) ou relatives (en % dans la majorité des cas) ou des mots clefs comme auto.

Généralement, on ne modifiera que l'une des deux dimensions de l'image (largeur ou hauteur, mais plutôt la largeur car modifier la hauteur peut rapidement poser des problèmes d'ergonomie) afin que l'image se redimensionne d'elle-même et conserve ses proportions.

On pourra éventuellement préciser la valeur `auto` pour la deuxième dimension si on a peur qu'une taille ait déjà été définie quelque part dans le code.

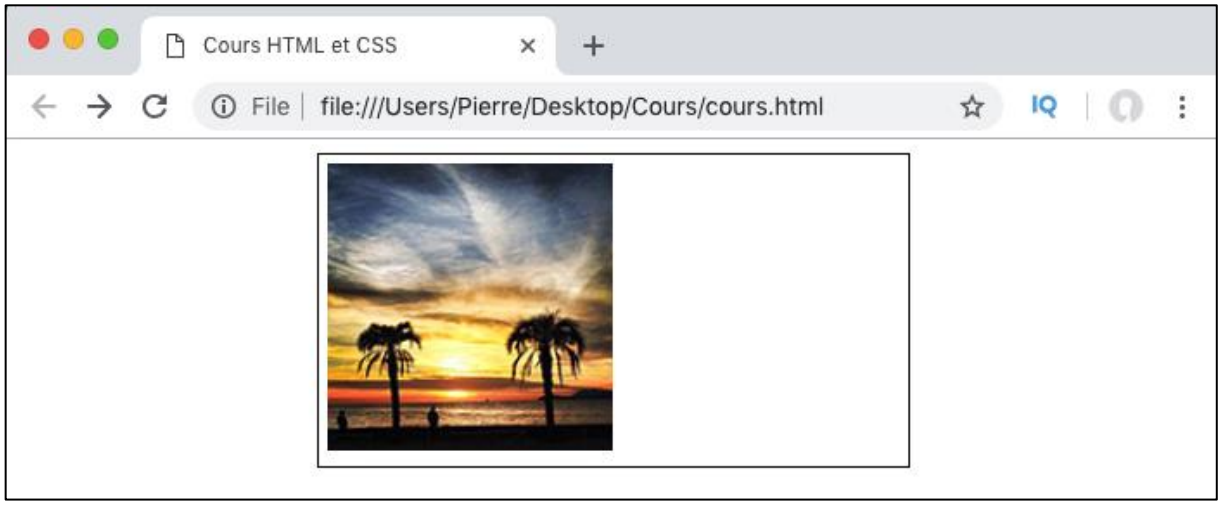
```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <img src="sunset.png"
             alt="Coucher de soleil en bord de mer"
             width="300px">
    </body>
</html>
```



Notez que dans le cas où on indique des valeurs en pourcentage, le pourcentage donné représente l'espace que l'image doit prendre par rapport à la taille de son conteneur (son élément parent le plus proche, c'est-à-dire l'élément dans lequel il est directement inclus dans le code) et n'est pas un pourcentage de la taille de base de l'image !

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur">
        <img src="sunset.png"
             alt="Coucher de soleil en bord de mer"
             width="50%">
        </div>
    </body>
</html>
```

```css
.conteneur{
  width: 50%;
  margin: 0 auto;
  border: 1px solid black;
  padding: 5px;
  box-sizing: border-box;
}
```



Mettre en forme nos images en CSS

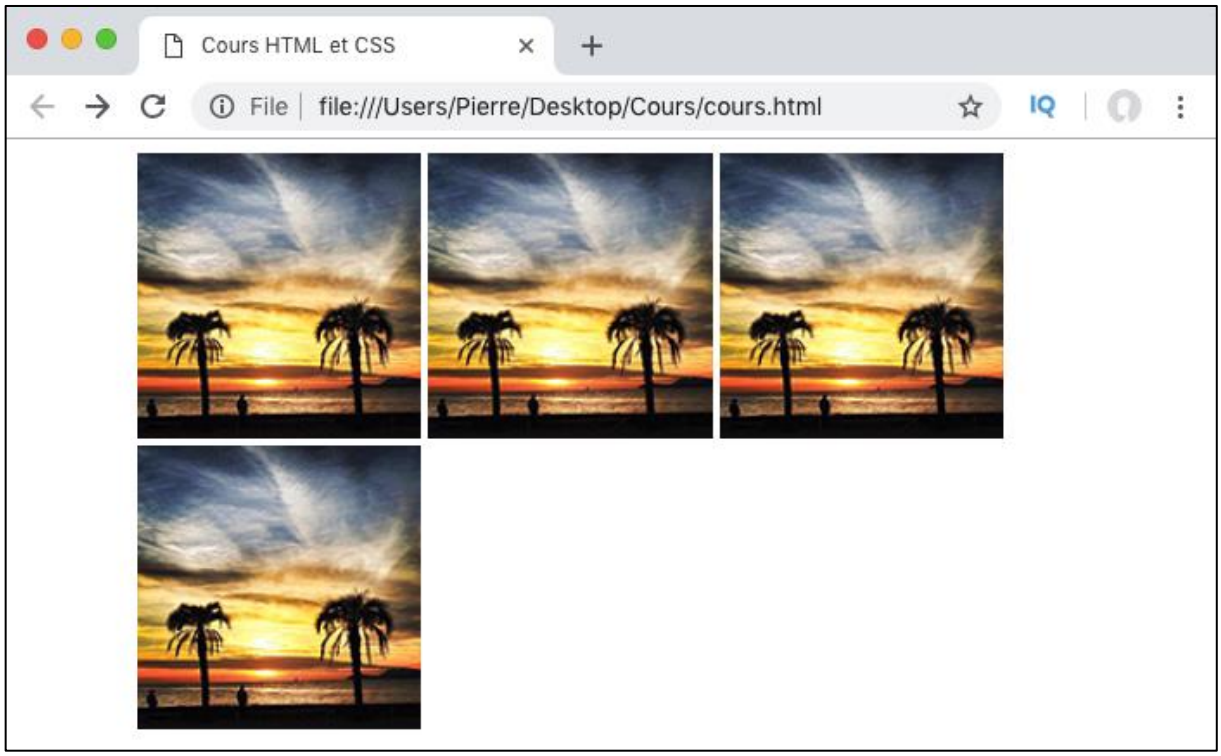
Nous utiliserons de préférence le CSS pour mettre en forme nos images, que ce soit pour les redimensionner ou pour les afficher d'une façon ou d'une autre ou à un endroit ou à un autre dans nos pages.

Redimensionner une image en CSS

Pour redimensionner une image en CSS, nous allons utiliser les propriétés `width` (pour la largeur) et `height` (pour la hauteur).

Ici, nous ne préciserons généralement que la valeur de l'une de ces deux propriétés. En procédant comme cela, l'autre dimension sera calculée automatiquement afin de ne pas casser le rapport largeur/hauteur de base de l'image.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur">
            <img class="sun" src="sunset.png" alt="Coucher de soleil">
            <img class="sun" src="sunset.png" alt="Coucher de soleil">
            <img class="sun" src="sunset.png" alt="Coucher de soleil">
            <img class="sun" src="sunset.png" alt="Coucher de soleil">
        </div>
    </body>
</html>
.conteneur{
  width: 80%;
  min-width: 400px;
  max-width: 800px;
  margin: 0 auto;
}

.sun{
  width: 30%;
}
```



Ici, je me contente d'insérer 4 fois la même image, ce qui est tout à fait autorisé puis je redimensionne mes images en CSS avec la propriété `width`, tout simplement.

Notez que des espaces se créent automatiquement entre chacune des images, à droite et en dessous. Cela est dû au fait que nos images sont des éléments de type `inline`.

L'espace à droite provient de mon code : les éléments `inline` vont conserver une espace s'il y en a au moins une dans le code. C'est le cas dans mon exemple puisque je suis allé à la ligne dans mon code à chaque nouvelle déclaration d'un élément `img`. Pour annuler cela, on peut par exemple faire flotter nos images ou les déclarer à la suite dans le code.

L'espace en bas provient du fait que les images sont alignées par défaut sur la baseline, c'est-à-dire sur une ligne imaginaire sur laquelle sont alignés les différents caractères de texte. Les navigateurs ajoutent toujours par défaut une espace sous la baseline pour laisser de la place pour les caractères qui passent dessous comme les « g », « p » ou « y » par exemple. Pour annuler cela, on peut utiliser la propriété `vertical-align` et définir un autre type d'alignement vertical.

Définir le type d'affichage et le positionnement des images en CSS

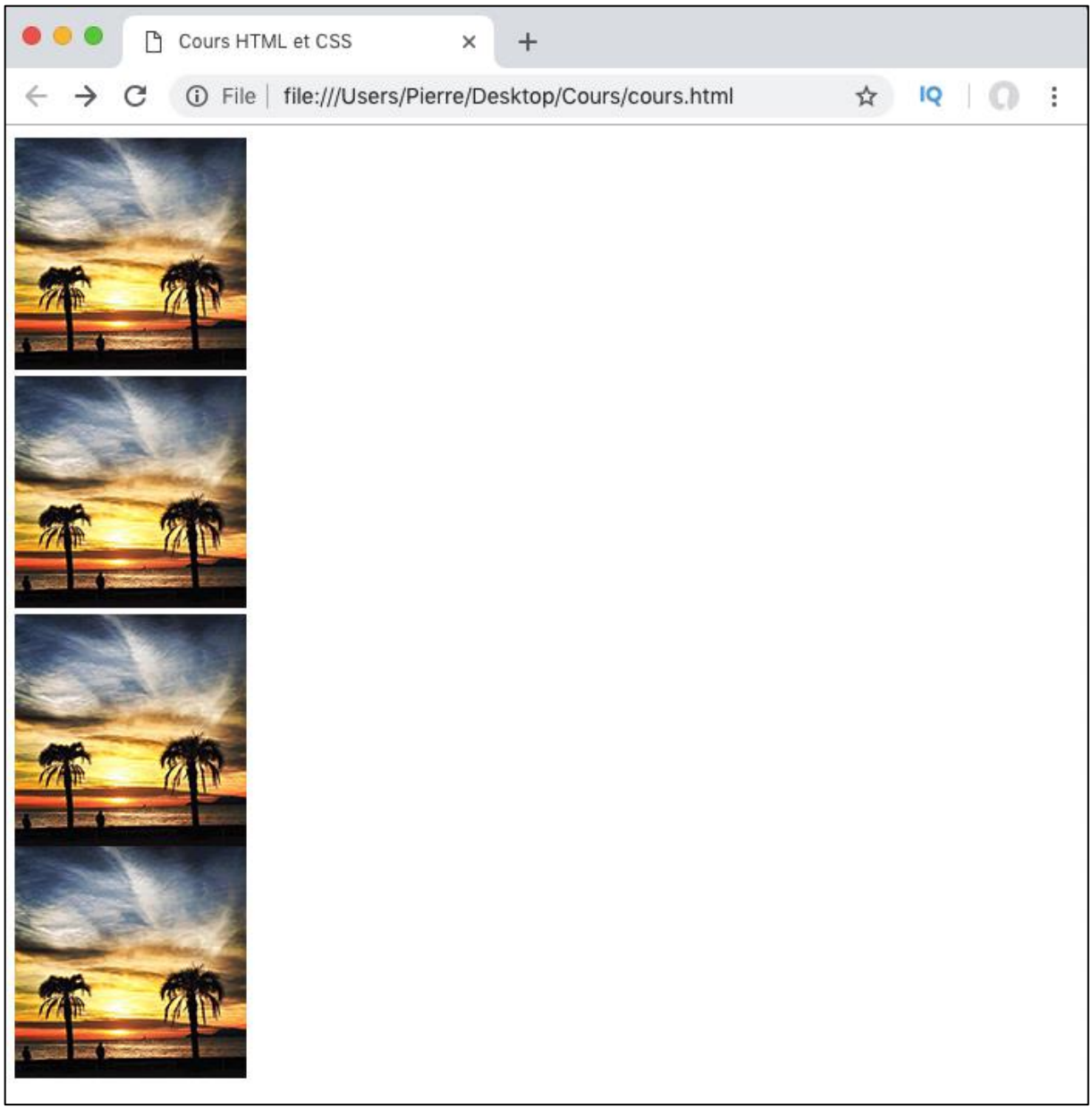
La plupart des navigateurs appliquent par défaut un `display : inline` aux images même si certains d'entre eux préfèrent appliquer un `display : inline-block`.

Ces deux types d'affichage font que nos images vont venir se positionner en ligne et se placer à côté d'autres contenus si elles en ont la place.

Nous avons différents moyens pour annuler ce comportement et faire que nos images s'affichent sur leur propre ligne. Les deux solutions les plus évidentes vont être soit d'enfermer chaque image à l'intérieur d'un élément de type `block` comme un élément `p` ou `div` par exemple ou d'appliquer directement un `display : block` en CSS à nos images.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur">
            <img class="sun" src="sunset.png" alt="Coucher de soleil">
        </div>
        <div class="conteneur">
            <img class="sun" src="sunset.png" alt="Coucher de soleil">
        </div>
        <img class="sun bloc" src="sunset.png" alt="Coucher de soleil">
        <img class="sun bloc" src="sunset.png" alt="Coucher de soleil">
    </body>
</html>
```

```css
.conteneur{
  width: 100%;
}

.bloc{
  display: block;
}
```



Poids des images et performances

Au début de cette leçon, j'ai beaucoup parlé de « poids des images ». Toute image possède un poids généralement exprimé de kilo-octets. Plus une image est lourde, plus elle va demander de ressources de la part du serveur et du navigateur pour être chargée.

En effet, lors de la mise en ligne de votre site web, vous allez louer un espace serveur chez un hébergeur afin d'y envoyer (« héberger ») toute l'architecture de votre site.

Le problème est que vous disposez d'un espace serveur et d'un débit limités. Il vous faudra donc déjà faire attention à ne pas le saturer avec des images inutilement lourdes.

Ensuite, selon la qualité de la connexion de vos utilisateurs et du navigateur utilisé, certaines images, si elles sont trop lourdes, risquent de mettre longtemps à s'afficher complètement.

Cela aura un effet négatif sur votre site puisque des visiteurs vont le quitter plutôt que de patienter. Cette problématique est véritablement à considérer aujourd'hui avec l'essor de l'Internet mobile car les mobiles disposent de moins de puissance que les ordinateurs et également d'une connexion plus lente.

Pour conserver des poids d'images minimaux tout en vous assurant des images d'une qualité convenable, on pensera déjà à les enregistrer au bon format et également à les recadrer à la bonne taille avant de les envoyer sur votre serveur. Ainsi, elles consommeront moins de ressources lors de leur affichage.

Media et sémantique : les éléments figure et figcaption

Les éléments `figure` et `figcaption` vont nous aider à marquer sémantiquement du contenu média comme des images, de l'audio ou des vidéos.

En effet, il est très difficile pour un navigateur de savoir « de quoi parle » ou « ce que représente » votre image, votre fichier audio ou vidéo sans plus de détail. On va utiliser l'élément HTML `figure` pour indiquer qu'une image, une piste audio ou vidéo n'est pas strictement décorative, mais sert à la compréhension générale de notre page web. On n'utilisera donc pas cet élément si nos contenus ne sont là que pour habiller la page.

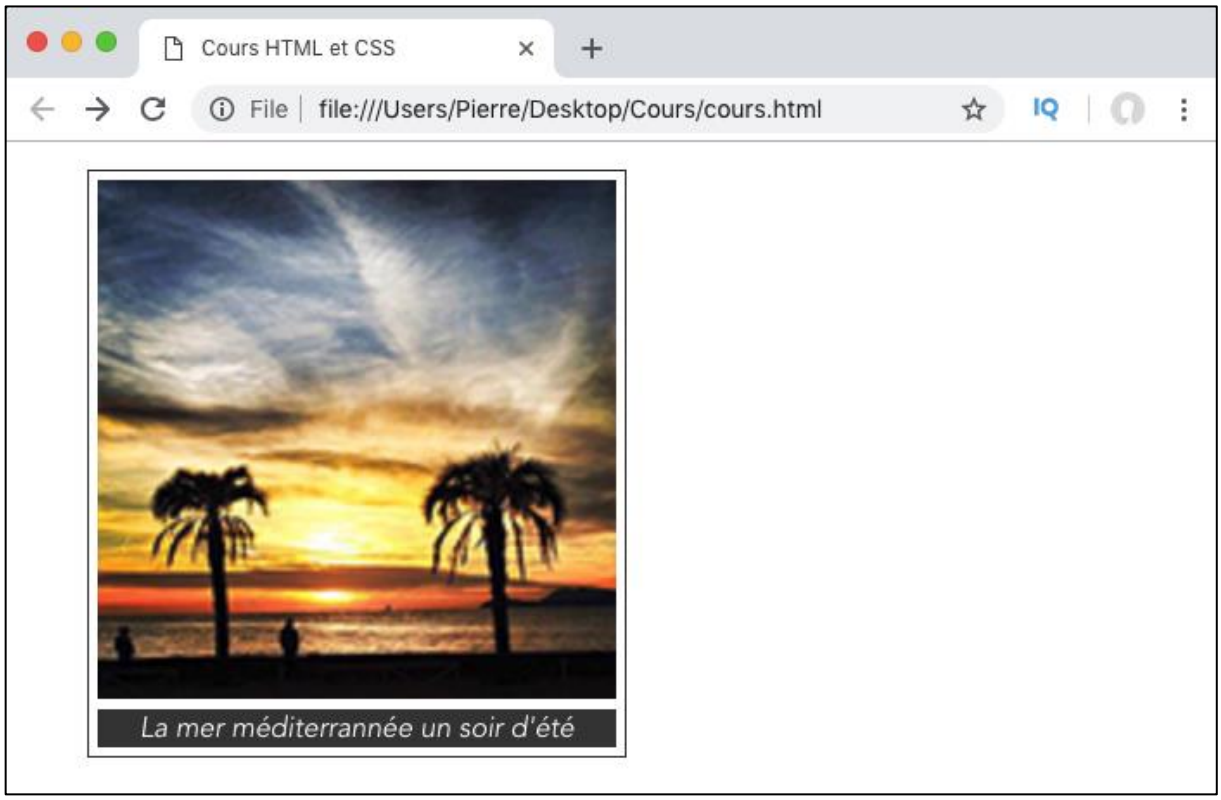
Notez qu'un élément `figure` peut englober plusieurs images ou d'autres types de contenus média (audio, vidéo, etc.).

On va ensuite utiliser l'élément `figcaption` à l'intérieur de `figure` pour accoler une légende au contenu de notre élément figure. Cette légende va être utile pour les personnes souffrant de déficiences et qui ne peuvent pas comprendre nos médias ainsi que pour les moteurs de recherche et navigateurs.

Par défaut, la légende va s'afficher sous les médias. Nous allons bien évidemment pouvoir mettre en forme les éléments `figure` et `figcaption` en CSS si on le souhaite.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <figure>
            <img src="sunset.png" alt="Coucher de soleil en bord de mer">
            <figcaption>La mer méditerrannée un soir d'été</figcaption>
        </figure>
    </body>
</html>
```

```css
*{
  font-family: avenir, sans-serif;
}

img{
  width: 300px;
}

figure {
    border: 1px solid #333;;
    padding: 5px;
    width: 300px;
}

figcaption {
    background-color: RGBa(0,0,0,0.8);
    color: #fff;
    text-align: center;
    font-style: italic;
}
```



Insérer de l'audio en HTML

Nous allons pouvoir intégrer un contenu audio dans un document avec l'élément `audio`. Dans cette nouvelle leçon, nous allons apprendre à l'utiliser et découvrir les subtilités des formats de codec audio.

Les formats d'audio et leur support

Il existe différents formats de fichiers audio de la même façon qu'on a pu voir précédemment qu'il existait différents formats d'image.

Jusqu'à très récemment, toutefois, l'insertion de fichiers audio était beaucoup plus complexe que l'insertion d'images pour la raison qu'aucun format de fichier audio n'était universellement reconnu : chaque navigateur possédait sa liste de formats audio qu'il était capable de lire.

La raison ici était une question de brevets déposés sur des éléments servant à constituer les différents formats audio ou sur les formats audio en soi, comme le brevet sur le MP3 effectif jusqu'en 2017. Ainsi, les navigateurs étaient soit obligés de posséder des licences soit de payer des royalties pour pouvoir utiliser tel ou tel format audio.

La situation s'est récemment beaucoup améliorée sur ce point-là et aujourd'hui certains formats audio sont bien reconnus par la plupart des navigateurs.

Cependant, comme certains problèmes peuvent toujours subsister, je vais vous montrer dans ce cours la méthode « historique » d'insertion d'audio qui consiste à contourner le problème en proposant plusieurs fichiers audio source de différents formats afin que chaque navigateur puisse choisir celui qui lui convient.

Les formats audio les plus courants généralement utilisés vont être les suivants :

- Le format audio Vorbis du conteneur WebM `audio/webm` ;
- Le format audio Vorbis du conteneur Ogg `audio/ogg` ;
- Le format audio MP3 `audio/mpeg` ;
- Le format audio PCM du conteneur WAVE `audio/wave`.

Pour information, voici les supports de ces différents formats par les navigateurs les plus utilisés dans leur version la plus récente :

Navigateur	**Vorbis**	**MP3**	**PCM (WAVE)**
Chrome	Supporté	Supporté	Supporté
Safari	Non supporté	Supporté	Supporté
Firefox	Supporté	Supporté	Supporté

Navigateur	Vorbis	MP3	PCM (WAVE)
Edge	Supporté	Supporté	Supporté
Opera	Supporté	Supporté	Supporté
Safari (iOS)	Non supporté	Supporté	Supporté
Android	Supporté	Supporté	Supporté
Chrome (Android)	Supporté	Supporté	Supporté

Insérer de l'audio dans un fichier HTML

Pour pouvoir insérer de l'audio dans nos pages, nous allons déjà devoir nous munir d'une piste audio qui devra être enregistrée sous différents formats.

Pour ma part, je vais utiliser un fichier audio qui s'appelle « perfect-time » et qui a été enregistré sous trois formats différents : `mp3`, `ogg` et `wav`.

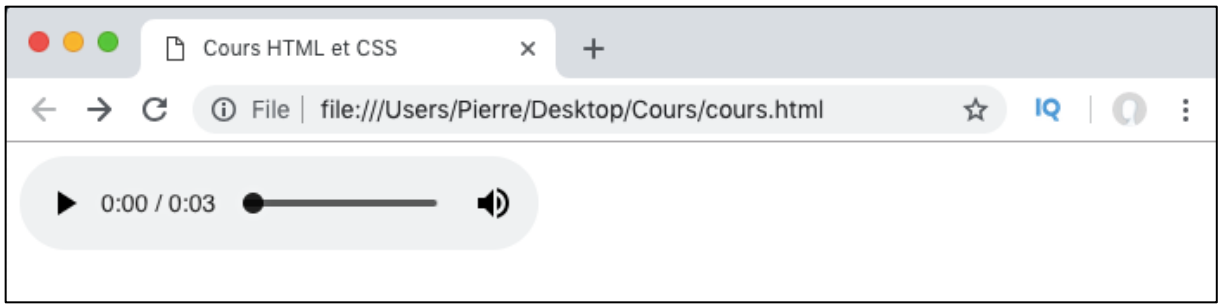
Munissez-vous de fichiers audio si vous voulez tester les codes suivants et placez ces fichiers dans le même dossier que vos fichiers de code.

La façon la plus simple d'insérer un fichier audio dans un fichier va être d'utiliser un élément `audio` avec deux attributs `src` et `controls`.

L'attribut `src` va indiquer le chemin vers la ressource à intégrer tandis que l'attribut `controls` va permettre de faire apparaitre les contrôles fournis par le navigateur (notamment un bouton de lecture / arrêt, un bouton pour choisir le niveau sonore, etc.).

Nous allons également renseigner un texte dans l'élément `audio` qui pourra être affiché dans certains anciennes versions de navigateurs dans le cas où le navigateur ne pourrait pas lire le fichier audio fourni.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <audio src="perfect-time.mp3" controls>
            Votre navigateur ne semble pas supporter ce fichier
        </audio>
    </body>
</html>
```

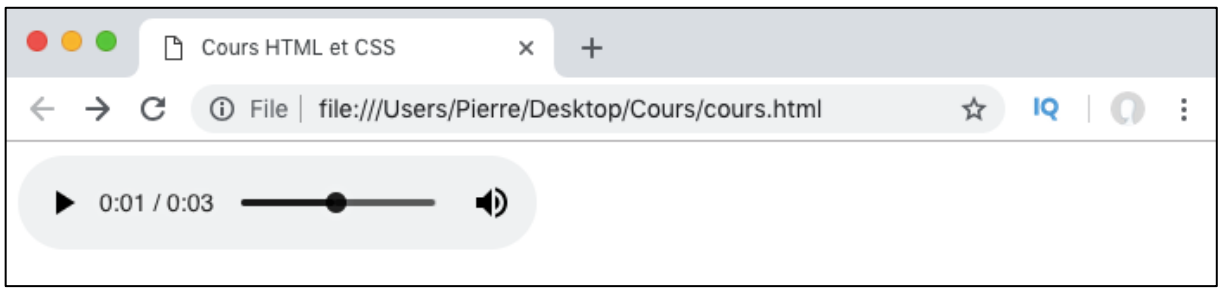


Le problème en utilisant l'élément audio de cette manière est qu'on ne va pas pouvoir fournir de format audio alternatif au cas où le navigateur ne puisse pas lire le format fourni.

Nous utiliserons donc généralement plutôt l'élément audio de concert avec des éléments source qui vont nous permettre d'intégrer différents fichiers parmi lesquels le navigateur fera son choix.

Dans l'élément source, nous allons préciser l'emplacement du fichier audio dans un attribut src et allons également facultativement pouvoir ajouter un attribut type qui va nous permettre d'indiquer rapidement au navigateur le type de codec audio utilisé dans notre fichier pour que le navigateur sache immédiatement s'il peut le lire ou pas sans même avoir à le tester. Cela optimisera les performances de notre page.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <audio controls>
            <source src="perfect-time.mp3" type="audio/mpeg">
            <source src="perfect-time.wav" type="audio/wav">
            <source src="perfect-time.mp3" type="audio/ogg; codecs=vorbis">
            Votre navigateur ne peut pas lire d'audio
        </audio>
    </body>
</html>
```



Ici, le navigateur va s'arrêter dès qu'il va rencontrer un format audio qu'il sait lire et ignorer les autres formats fournis en dessous. Cette technique permet de pouvoir lire un fichier audio sur (quasiment) tous les navigateurs.

Les attributs de l'élément audio

En plus de l'attribut `controls` qui est obligatoire pour des raisons évidentes d'ergonomie et d'accessibilité, nous allons pouvoir indiquer d'autres attributs facultatifs à notre élément `audio` qui vont nous permettre de mieux contrôler comment le fichier audio doit être lu.

Ces attributs sont les suivants :

- L'attribut `preload` permet d'indiquer au navigateur si le fichier doit être préchargé ou pas. On va pouvoir lui passer l'une des valeurs suivantes :
  - `none` : le fichier audio ne sera pas préchargé ;
  - `metadata` : seules les métadonnées seront préchargées ;
  - `auto` : le fichier sera préchargé.

- L'attribut `autoplay` va nous permettre de lancer automatiquement la lecture du fichier audio dès qu'il sera chargé. Il suffit de le renseigner (même sans valeur explicite) pour que le fichier audio se lance automatiquement. Notez que certains navigateurs (dont Chrome) peuvent bloquer cet attribut car celui-ci est jugé mauvais pour vos visiteurs.

- L'attribut `loop` permet de lire le fichier audio en boucle. Il suffit donc de le renseigner (même sans valeur explicite) pour que le fichier soit lu en boucle.

- L'attribut `muted` sert à définir si le son doit être initialement coupé. Il suffit de le renseigner (même sans valeur explicite) pour couper le son.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <audio controls preload="metadata" loop muted>
            <source src="perfect-time.mp3" type="audio/mpeg">
            <source src="perfect-time.wav" type="audio/wav">
            <source src="perfect-time.mp3" type="audio/ogg; codecs=vorbis">
            Votre navigateur ne peut pas lire d'audio
        </audio>
    </body>
</html>
```


Cours HTML et CSS
File | file:///Users/Pierre/Desktop/Cours/cours.html
0:00 / 0:03

Insérer des vidéos en HTML

Nous allons pouvoir intégrer un contenu vidéo dans un document avec l'élément `video`. Cet élément fonctionne de manière analogue à l'élément `audio` vu précédemment.

Les formats vidéo et leur support

Un fichier vidéo est relativement plus complexe qu'un simple fichier audio puisqu'une vidéo est va être composée à la fois d'une piste audio et d'une piste vidéo. Ces deux pistes vont être de formats différents et vont être regroupées dans un conteneur qui va se charger de les faire fonctionner ensemble.

En termes de support par les navigateurs, nous allons donc nous heurter aux mêmes problèmes qu'on a déjà vu avec l'intégration d'audio et notamment aux problèmes de brevet cette fois-ci non seulement sur les codecs des pistes audio mais également sur ceux des pistes vidéo et des conteneurs.

Comme pour l'élément `audio`, nous indiquerons donc généralement plusieurs formats de conteneurs dans notre élément `video` afin de s'assurer que la vidéo puisse être lue sur la plupart des navigateurs.

Les conteneurs les plus utilisés aujourd'hui sont les suivants :

- `WebM` : contient de l'audio Ogg Vorbis avec de la vidéo VP8/VP9 ;
- `MP4` : contient de l'audio AAC ou MP3 en audio avec de la vidéo H.264.

Notez par ailleurs que le MP4-HEVC utilisant le format vidéo H.265, annoncé pour la première fois en 2013, est censé succéder au MP4 utilisant de la vidéo H.264.

Le format AV1, annoncé en 2018 est lui censé succéder au WebM et concurrencer le HEVC. Ces deux formats n'ont cependant que peu de support par les navigateurs aujourd'hui : le HEVC n'est supporté que par Safari tandis que l'AV1 n'est supporté que par les dernières versions de Chrome (desktop).

Pour information, voici les supports de ces différents formats par les navigateurs les plus utilisés dans leur version la plus récente :

Navigateur	**MP4/H.264**	**WebM**
Chrome	Supporté	Supporté
Safari	Supporté	Non supporté
Firefox	Supporté	Supporté
Edge	Supporté	Supporté partiellement

Navigateur	MP4/H.264	WebM
Opera	Supporté	Supporté
Safari (iOS)	Supporté	Non supporté
Android	Supporté	Supporté
Chrome (Android)	Supporté	Supporté

Insérer de la vidéo dans un fichier HTML

Pour pouvoir insérer de la vidéo dans nos pages, nous allons déjà devoir nous munir d'un fichier vidéo disponible sous différents formats.

Ici, je vais utiliser un fichier vidéo qui s'appelle « tortue » qui est originellement un fichier MP4 et que j'ai converti également en WebM.

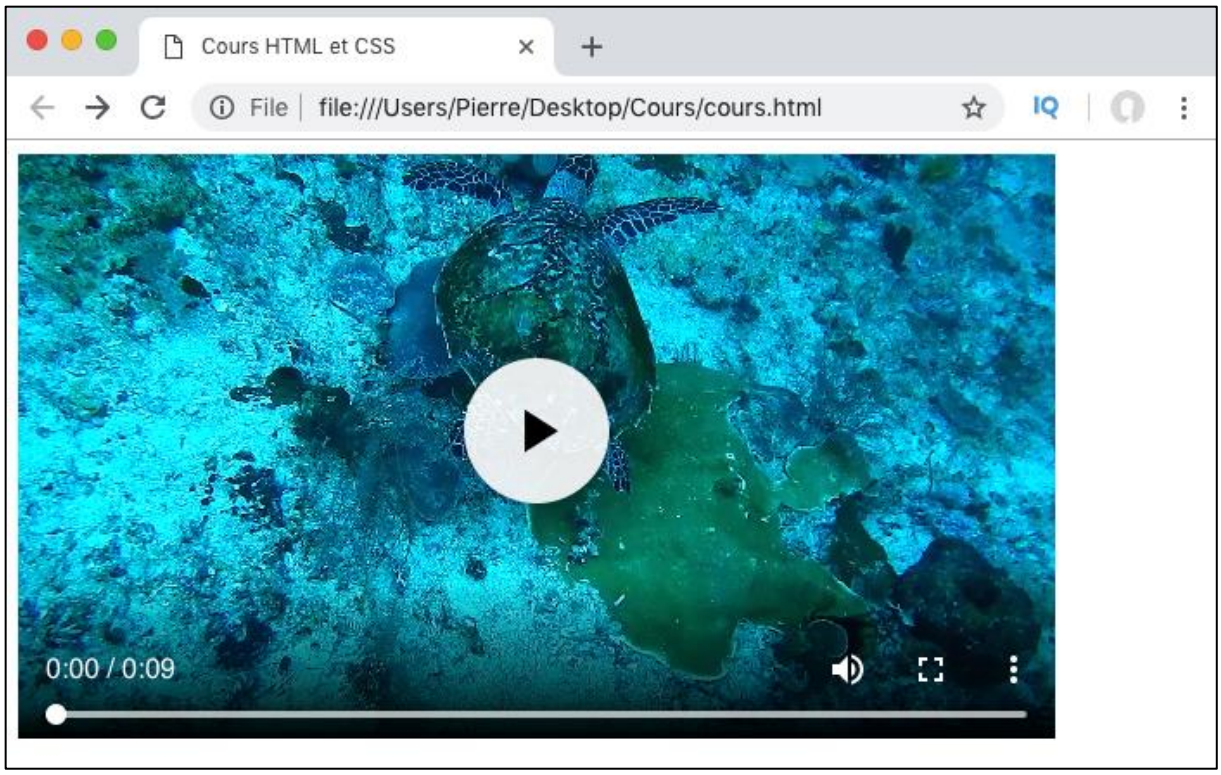
Pour reproduire les codes ci-dessous, munissez-vous de vos propres fichiers vidéo et placez ces fichiers dans le même dossier que vos fichiers de code.

La façon la plus simple d'insérer un fichier vidéo dans une page HTML va être d'utiliser un élément `video` avec deux attributs `src` et `controls`, exactement de la même façon que pour l'élément `audio`.

L'attribut `src` va indiquer le chemin vers la ressource à intégrer tandis que l'attribut `controls` va permettre de faire apparaitre les contrôles fournis par le navigateur (notamment un bouton de lecture / arrêt, un bouton pour choisir le niveau sonore, etc.).

Nous allons également renseigner un texte dans l'élément `video` qui pourra être affiché dans certains anciennes versions de navigateurs dans le cas où le navigateur ne pourrait pas lire le fichier vidéo fourni.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <video src="tortue.mp4" controls>
            Le ficiher vidéo ne peut pas être lu
        </video>
    </body>
</html>
```



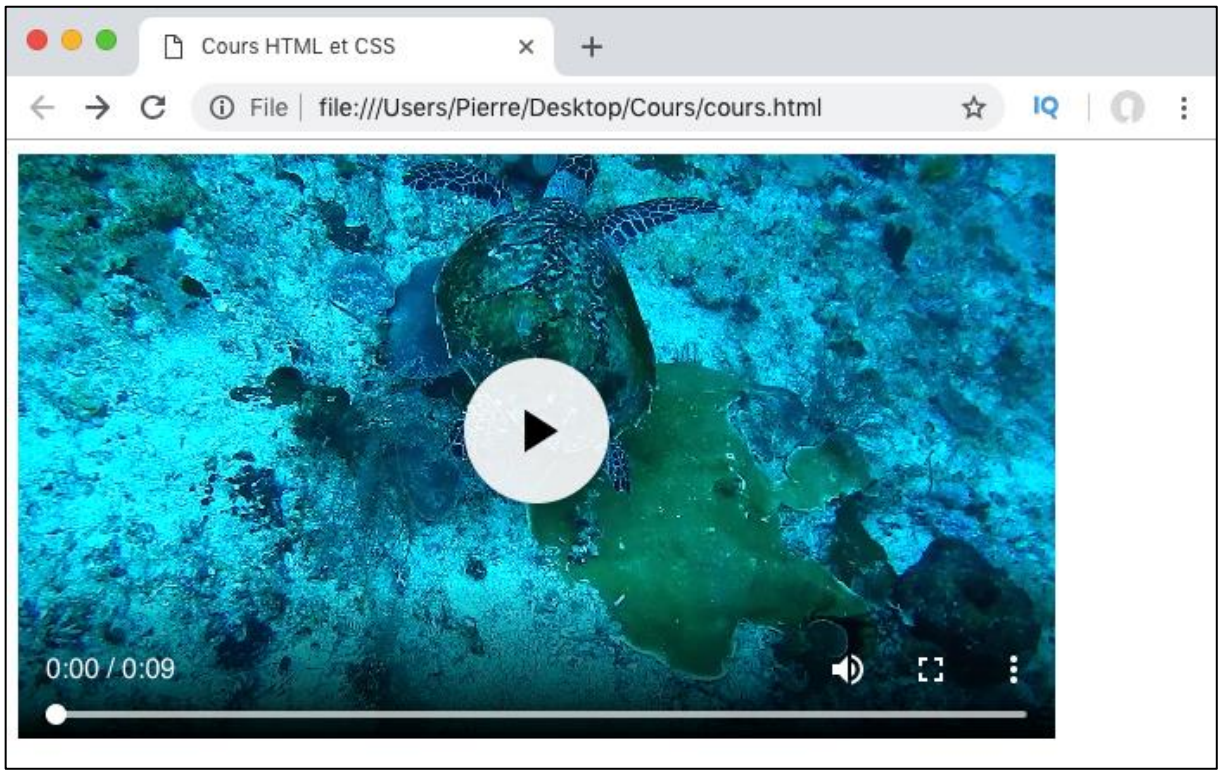
En pratique, toutefois, cette façon de faire n'est pas optimale puisqu'elle ne nous permet pas de proposer plusieurs formats de fichiers différents pour les navigateurs.

Nous utiliserons donc généralement plutôt l'élément `video` de concert avec des éléments `source` qui vont nous permettre d'intégrer différents formats de fichier parmi lesquels le navigateur fera son choix.

Dans l'élément `source`, nous allons devoir inclure un attribut `src` qui va nous permettre d'indiquer l'emplacement du fichier vidéo.

Nous allons également pouvoir, de manière facultative, ajouter un attribut `type` qui va nous permettre d'indiquer rapidement au navigateur le type de codecs utilisé dans notre fichier pour que le navigateur sache immédiatement s'il peut le lire ou pas sans même avoir à le tester. Cela optimisera les performances de notre page.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <video controls>
            <source src="tortue.mp4" type="video/mp4">
            <source src="tortue.webm" type="video/webm">
            Le fichier vidéo ne peut pas être lu
        </video>
    </body>
</html>
```



Ici, le navigateur va s'arrêter dès qu'il va rencontrer un format qu'il sait lire et ignorer les autres formats fournis en dessous.

Les attributs de l'élément video

L'élément `video` possède un attribut strictement obligatoire qui est l'attribut `controls`. Cet attribut permet d'afficher les contrôles de base pour l'utilisateur comme la lecture, la pause, le choix du volume, etc. Les différentes options de contrôle vont être fournies par le navigateur.

En plus de cet attribut `controls`, on va pouvoir passer d'autres attributs facultatifs à l'élément `video` afin de contrôler comment le fichier doit être lu et notamment :

- L'attribut `preload` permet d'indiquer au navigateur si le fichier doit être préchargé ou pas. On va pouvoir lui passer l'une des valeurs suivantes :
  - `none` : le fichier vidéo ne sera pas préchargé ;
  - `metadata` : seules les méta-données seront préchargées ;
  - `auto` : le fichier sera préchargé.

- L'attribut `autoplay` va nous permettre de lancer automatiquement la lecture du fichier vidéo dès qu'il sera chargé. Il suffit de le renseigner (même sans valeur explicite) pour que le fichier vidéo se lance automatiquement. Notez que certains navigateurs (dont Chrome) peuvent bloquer cet attribut car celui-ci est jugé mauvais pour vos visiteurs.

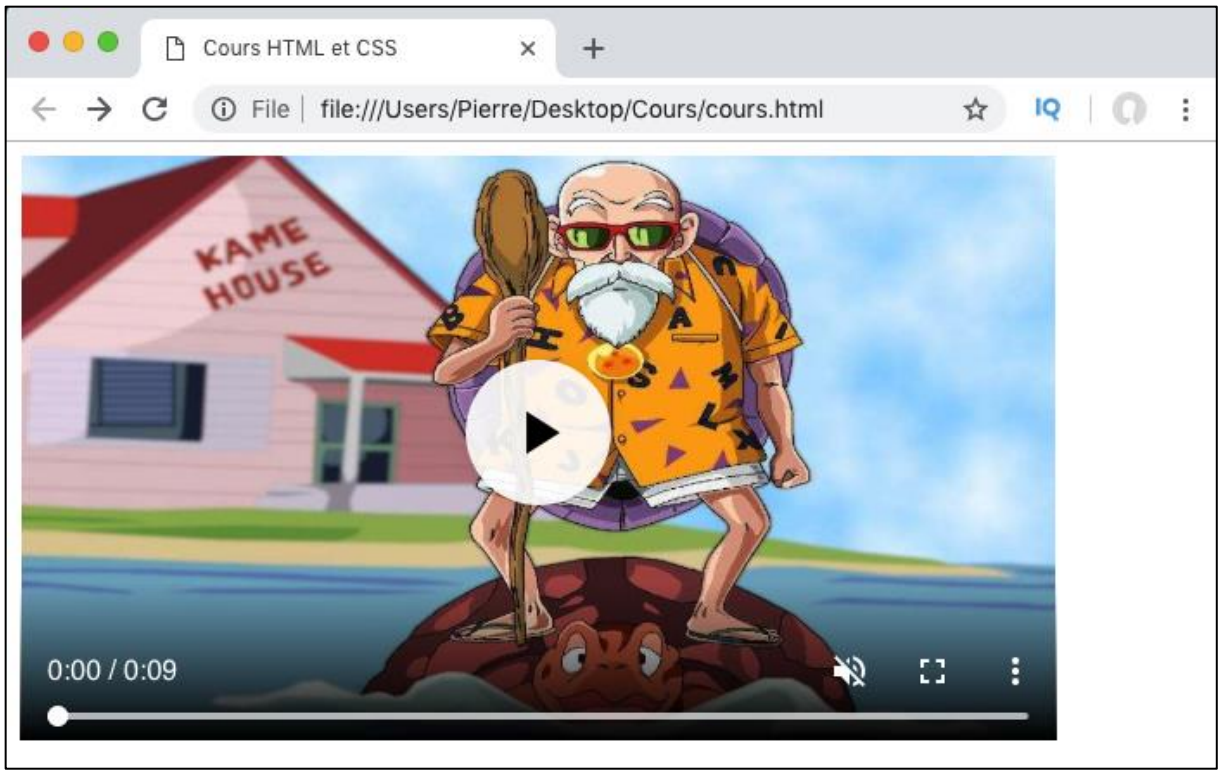
- L'attribut `loop` permet de lire le fichier vidéo en boucle. Il suffit donc de le renseigner (même sans valeur explicite) pour que le fichier soit lu en boucle.

- L'attribut `muted` sert à définir si le son doit être initialement coupé. Il suffit de le renseigner (même sans valeur explicite) pour couper le son.

- L'attribut `poster` va nous permettre de renseigner l'adresse d'une image qui devra être utilisée comme image d'illustration de fond de la vidéo avant que celle-ci ne soit chargée et lancée.

En plus de ces attributs, notez que nous allons pouvoir gérer la taille de nos vidéos grâce aux attributs HTML `width` et `height` ou idéalement avec les propriétés CSS du même nom. Pour des raisons évidentes de ratio, nous ne préciserons toujours que l'une de ces deux propriétés (généralement `width`) afin que la deuxième dimension soit calculée automatiquement.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <video controls muted poster="muten-roshi.jpg">
            <source src="tortue.mp4" type="video/mp4">
            <source src="tortue.webm" type="video/webm">
            Le fichier vidéo ne peut pas être lu
        </video>
    </body>
</html>
```



Ajouter des sous-titres à une vidéo

L'importance de l'ajout de sous-titres

Il est toujours très important lorsqu'on code en HTML de penser en termes d'accessibilité. L'idée ici est que votre code devrait toujours être accessible à tous, c'est-à-dire que vos pages devraient toujours pouvoir être comprises d'une façon ou d'une autre par chacun de vos visiteurs.

C'est là tout l'intérêt d'ajouter des descriptions sur nos images avec l'attribut `alt` par exemple. C'est la même logique qui se cache derrière l'ajout de sous-titres sur nos vidéos.

Pour ajouter des sous-titres à nos vidéos, nous allons déjà devoir les écrire dans un fichier séparé en respectant une certaine syntaxe. Le fichier sera enregistré sous le format WebVTT pour « Web Video Text Tracks ». Nous allons ensuite pouvoir intégrer ces sous titres dans notre vidéo en utilisant l'élément `track`.

L'écriture des sous-titres au format WebVTT

Nous allons écrire les sous-titres pour notre vidéo dans un fichier séparé au format WebVTT. Pour faire cela, nous pouvons utiliser notre éditeur ou un programme de bloc note et enregistrer le fichier avec l'extension `.vtt`.

Un fichier WebVTT doit respecter une certaine syntaxe pour pouvoir être lu convenablement :

- Tout fichier WebVTT doit commencer par le mot `WEBVTT`, qui peut être éventuellement suivi sur la même ligne par un texte d'en-tête ;
- La ligne sous la mention `WEBVTT` doit être vide ;
- Ensuite, nous allons placer des indications de temps sous la forme `00:01.000 --->00:05.000`suivies à chaque fois sur la ligne du dessous du sous-titre à afficher durant la période décrite.

Voici à quoi va pouvoir ressembler notre fichier :

```
WEBVTT

00:01.000 --> 00:05.000
La tortue nage tranquillement dans la mer

00:06.000 --> 00:09.000
Elle doit parcourir de grandes distances pour :
- Trouver à manger
- Se reproduire
```

L'intégration des sous-titres avec l'élément track et ses attributs

Une fois que nous avons écrit nos sous-titres et qu'on les a enregistrés au bon format, il va falloir les intégrer dans nos vidéos.

Pour cela, nous allons utiliser un élément `track` que nous allons placer à l'intérieur de notre élément `video`.

Cet élément `track` va devoir être obligatoirement accompagné d'un attribut `src` qui va nous permettre d'indiquer l'adresse du fichier contenant les sous titres ainsi que d'un attribut `srclang` qui va nous permettre de préciser la langue utilisée (`fr` pour français, `en` pour anglais, `es` pour espagnol, etc.).

Nous allons également pouvoir lui passer les attributs facultatifs suivants :

- `default` : Indique qu'on souhaite utiliser ce fichier de sous-titres par défaut (utile dans le cas où on dispose de plusieurs fichiers de sous-titrage pour différentes langues) ;

- `kind` : Cet attribut nous permet de préciser la nature du texte ajouté à la vidéo. Sa valeur par défaut est `subtitles` (« sous-titres ») mais si une mauvaise valeur est renseignée c'est la valeur `metadata` qui sera utilisée. Les valeurs possibles sont :
    - `subtitles` : Le texte passé correspond à des sous-titres. Les sous-titres sont pertinents pour des utilisateurs qui ne peuvent pas comprendre le contenu (cas des utilisateurs qui regardent des vidéos dans une autre langue que la leur) et ont également pour rôle de donner des informations contextuelles ;
    - `captions` : Le texte est une traduction de l'audio de la vidéo. Le type `captions` est pertinent pour des malentendants ou lorsque le son est désactivé par exemple ;
    - `descriptions` : Le texte est une description du contenu vidéo. Ce type est adapté pour les personnes malvoyantes ;

    - `chapters` : Ce type représente les titres des chapitres lorsque l'utilisateur navigue au sein du media ;
    - `metadata` : Le texte sera utilisé par des scripts mais sera invisible pour l'utilisateur.

- `label` : Permet d'indiquer la langue de chaque sous-titres comme « Français », « English » ou « Español. La liste de langues sera visible pour l'utilisateur et lui permettra de choisir le sous-titrage de son choix.

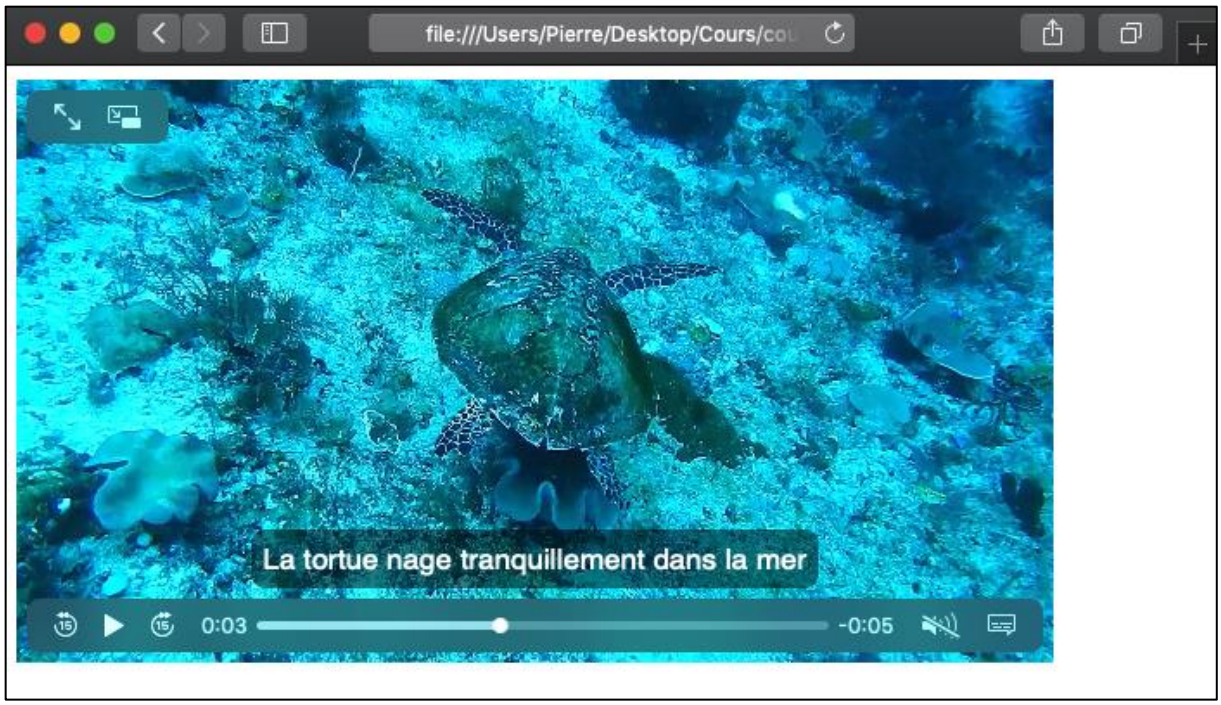
Notez que dans le cas où nous avons plusieurs langues de sous-titrage à proposer, il faudra enregistrer chaque sous-titres dans un fichier `.vtt` différent et utiliser un nouvel élément `track` pour chacun d'entre eux.

Nous pouvons également proposer plusieurs fichiers de texte de type (`kind`) différents dans un même langage.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <video controls muted poster="muten-roshi.jpg">
            <source src="tortue.mp4" type="video/mp4">
            <source src="tortue.webm" type="video/webm">
            <track kind="subtitles" src="tortue.vtt" srclang="fr">
            Le fichier vidéo ne peut pas être lu
        </video>
    </body>
</html>
```

Il ne nous reste plus qu'à définir les styles de notre texte en ciblant l'élément `track` en CSS.

```
video{
  width: 600px;
}

track{
  color: #fff;
  background-color: #000;
}
```



Notez ici que certains navigateurs ne vont pas afficher les sous-titres pour des fichiers en local (vs en production ou sur serveur). Cela va être le cas pour Chrome notamment. La capture d'écran ci-dessus a donc été faite avec Safari qui n'a pas cette limitation.

L’élément iframe

Nous savons désormais comment ajouter des images, de l’audio ou des vidéos dans nos pages en intégrant nos propres médias. Il nous reste donc une chose à voir : comment intégrer une autre page web dans un fichier HTML.

L’élément `iframe`, qui est l’objet de cette leçon, va nous permettre de faire cela et d’intégrer, entre autres, des vidéos YouTube directement dans nos pages.

L’élément iframe et ses attributs

L’élément `iframe` va nous permettre d’intégrer un fichier web entier dans un autre. Nous allons notamment utiliser cet élément pour intégrer des vidéos YouTube dans nos pages web ou des cartes Google Maps.

Pourquoi intégrer des contenus provenant d’autres sites web plutôt qu’héberger nos propres contenus ? Il y a deux raisons principales.

La première concerne l’économie des ressources : en hébergeant une vidéo sur YouTube, par exemple, et en l’intégrant après sur notre site on ne va pas utiliser notre espace serveur ni notre bande passante mais ceux de YouTube, ce qui peut se transformer en des économies substantielles de notre côté.

La deuxième raison est purement technique : il est souvent plus facile et sûr d’utiliser les services créés par d’autres entités reconnues plutôt que de créer nous-mêmes ces choses. Cela est le cas pour les cartes Google Maps par exemple : il est hors de question de se procurer une carte du monde détaillée et de recréer l’outil de notre côté pour afficher une carte quand on peut utiliser celles de Google.

De même pour l’intégration de vidéos YouTube : en passant par YouTube, on n’a pas à se soucier de tous les problèmes de compatibilité des différents codecs avec les différentes versions des navigateurs puisque YouTube se charge de faire ce travail de son côté lorsqu’on télécharge une vidéo sur la plateforme.

Les attributs généralement précisés lors de l’utilisation de l’élément `iframe` sont les suivants :

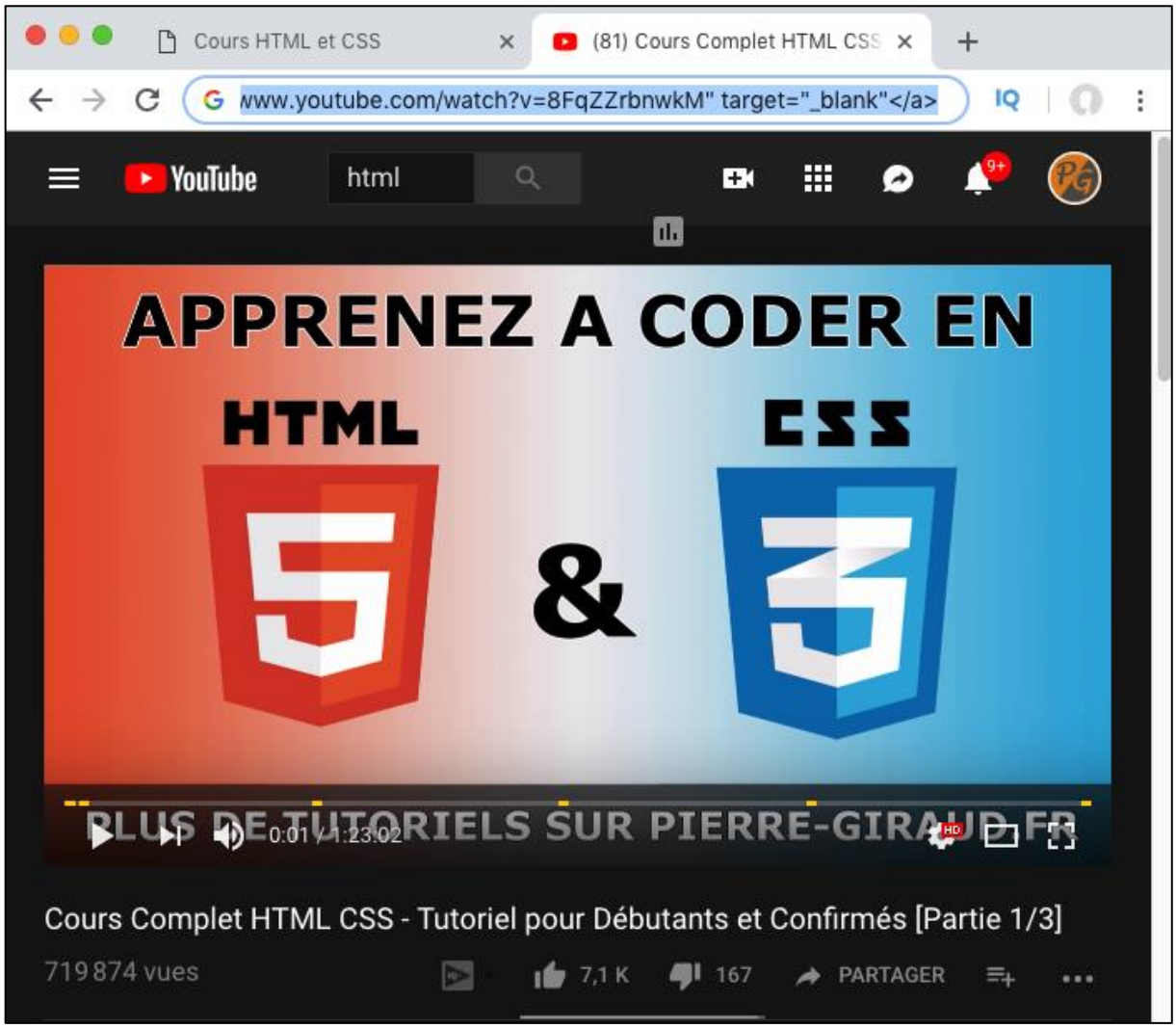
- `src` : Cet attribut va nous permettre d’indiquer l’adresse du document à intégrer
- `width` et `height` : Ces attributs permettent de définir les dimensions de notre élément `iframe`. On pourra les utiliser dans le cas où nous ne pouvons pas modifier les dimensions de l’iframe en CSS.
- `allow` : Cet attribut permet de définir une politique de fonctionnalité pour notre iframe. Le terme « politique de fonctionnalité » désigne le fait de choisir quelles fonctionnalités de l’iframe activer ou désactiver. Cet attribut est utilisé pour renforcer la sécurité du code.
- `sandbox` : Cet attribut est relativement récent. Il nous permet de limiter les permissions de notre iframe c’est-à-dire de limiter ses possibilités. Cet attribut est utilisé pour renforcer la sécurité de notre code.

Intégrer une vidéo YouTube avec l'élément iframe

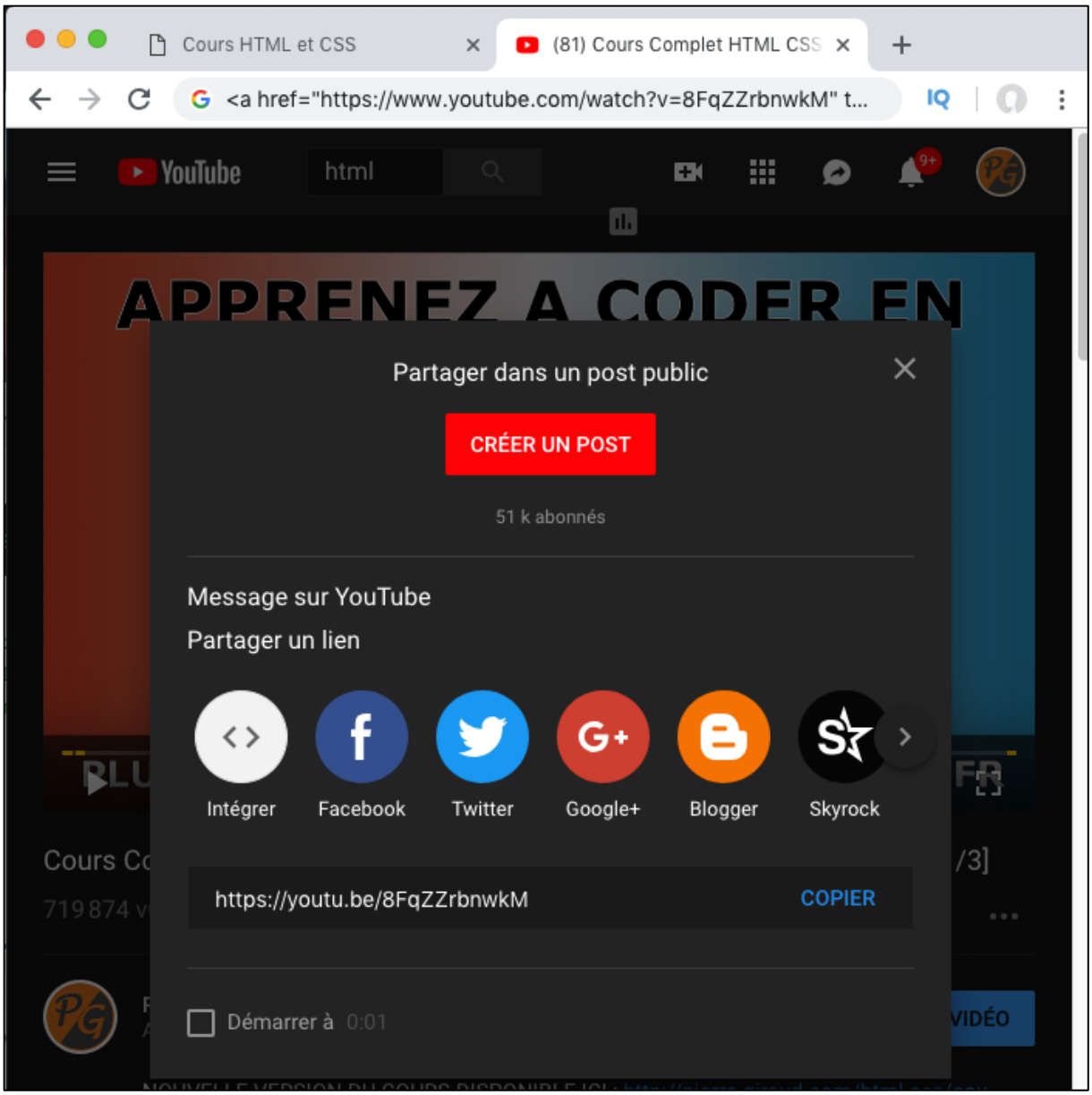
Voyons immédiatement comment utiliser l'élément `iframe` en pratique en intégrant une vidéo YouTube dans notre page.

Pour cela, je vais utiliser l'une de mes toutes premières vidéos (disponibles sur ma chaine YouTube Pierre Giraud).

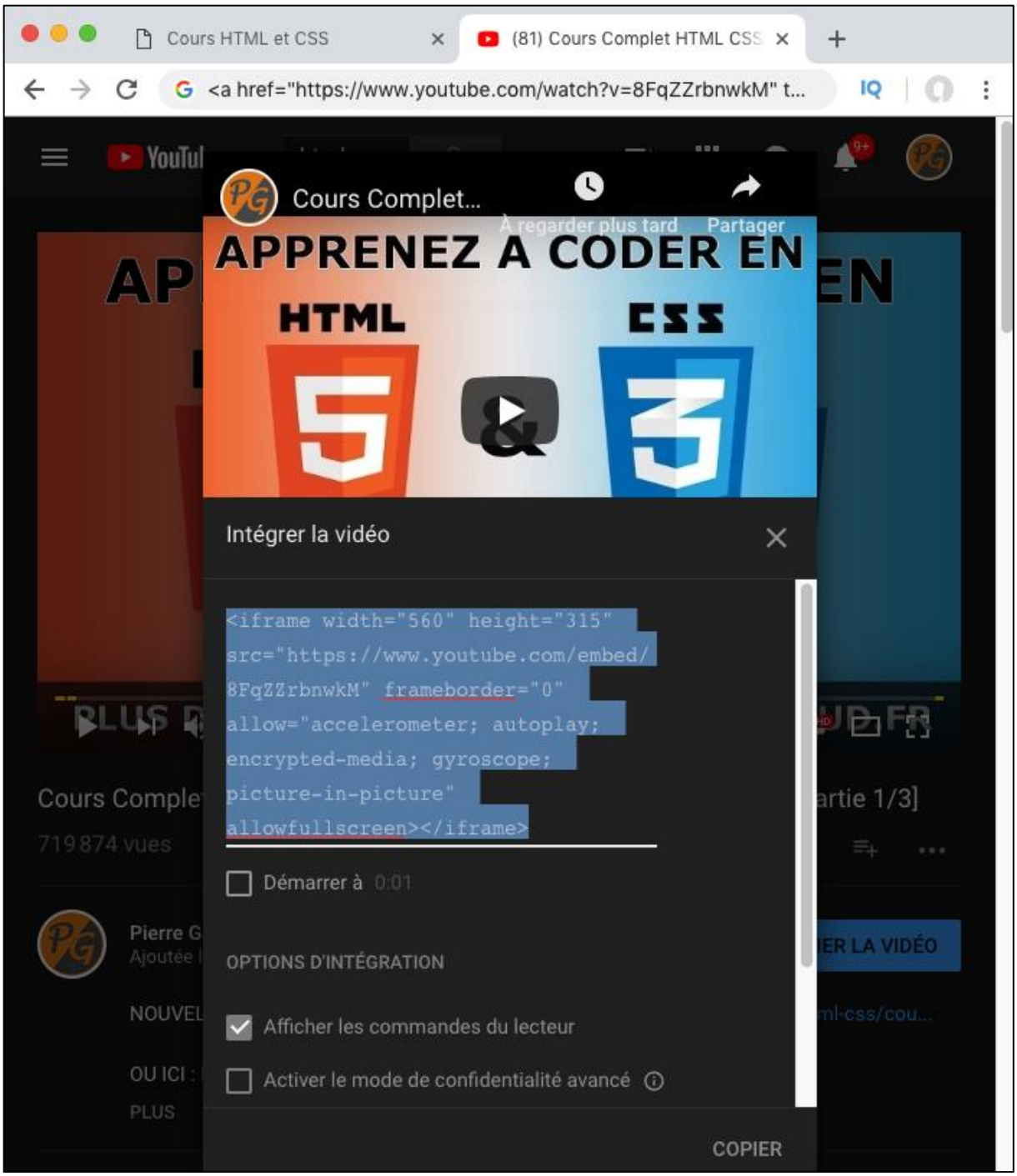
Ici, nous allons cliquer sur le bouton « partager » sous la vidéo…



…puis sur « intégrer » (« embed » en anglais) …



… et nous allons finalement copier le code d'intégration qui est bien un élément `iframe`



Nous n'avons ensuite plus qu'à coller ce code dans notre page HTML pour intégrer notre vidéo !

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <iframe
          width="560"
          height="315"
          src="https://www.youtube.com/embed/8FqZZrbnwkM"
          frameborder="0"
          allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture"
          allowfullscreen>
        </iframe>
    </body>
</html>
```

iframes et APIs : le cas de l'intégration de cartes Google Maps

L'intégration de cartes Google Maps va être plus complexe ou tout au moins plus longue que celles de vidéos YouTube.

Cela est dû au fait que l'API Google Maps est plus restrictive que celle de YouTube. Une API « Application Programming Interface » ou « Interface de Programmation Applicative » en français est une interface généralement complexe qui va permettre à différentes applications de communiquer entre elles et de s'échanger des données.

L'intégration d'iframes va souvent de pair avec l'utilisation d'API. Pour nous, vous pouvez considérer qu'une API nous permet d'accéder à une application sans avoir à se soucier des mécanismes d'arrière-plan permettant de la faire fonctionner.

D'un autre côté, les API servent également à limiter les intégrations « sauvages » en demandant à l'utilisateur de s'enregistrer et de générer une clef pour pouvoir utiliser une application en question sur son site.

Cela va être le cas avec l'intégration de cartes Google Maps : nous allons devoir passer une clef de sécurité (API Key ou API Token) en attribut de notre `iframe` pour pouvoir s'identifier auprès de Google et pouvoir utiliser le service de cartes.

Pour obtenir une clef d'API, il suffit généralement de s'inscrire sur le site proposant l'API. Dans le cas de Google Maps, cependant, cela n'est pas suffisant puisque l'utilisation de l'API Google Maps est payante depuis cette année. Il faudra donc également être prêt à payer pour intégrer des cartes Google Maps.

Pour cette raison, je ne vous présenterai pas comment intégrer des cartes Google Maps dans ce cours mais cet exemple me semblait intéressant car il m'a permis d'introduire le concept d'API et également de vous expliquer les différentes contraintes auxquelles nous pouvons être soumis en tant que développeurs.

iframe, sécurité et performance

Il convient de faire preuve de prudence lors de l'utilisation d'un élément `iframe` comme pour tout autre élément faisant appel à des données externes. En effet, ce type d'élément peut potentiellement se transformer en point d'entrée pour des personnes mal intentionnées (des « hackeurs ») qui peuvent exploiter certaines failles pour dérober des informations sensibles à nos utilisateurs ou pour rendre notre site non opérationnel.

Les problématiques de sécurité sont très complexes et ne sont pas à la portée de développeurs débutants et c'est la raison pour laquelle je n'en parlerai pas en détail dans ce cours. En effet, pour en comprendre les tenants et les aboutissants, il faut comprendre comment fonctionnent les sites Internet, les réseaux, ainsi que connaitre les langages comme le JavaScript et comprendre l'interaction entre les différents langages.

Je resterai donc très superficiel ici et vais simplement vous conseiller d'ajouter des attributs `allow` et >`sandbox` dans vos éléments `iframe` qui vont permettre de très largement limiter les menaces entrantes.

Chacun de ces deux attributs peut prendre de nombreuses valeurs. En les utilisant sans valeur, nous désactivons la plupart des fonctionnalités liées aux iframes. A chaque fois qu'on ajoute une valeur dans ces attributs, on indique qu'on souhaite réactiver la fonctionnalité liée à la valeur en question.

Pour information, on va pouvoir passer les valeurs suivantes (et donc activer les fonctionnalités correspondantes) à `allow` :

- `Accelerometer` ;
- `Ambient light sensor` ;

- Autoplay ;
- Camera ;
- Encrypted media ;
- Fullscreen ;
- Geolocation ;
- Gyroscope ;
- Lazyload ;
- Microphone ;
- Midi ;
- Payment Request ;
- Picture-in-picture ;
- Speaker ;
- USB ;
- VR / XR.

De même, l'attribut sandbox va nous permettre d'activer les fonctionnalités suivantes :

- allow-scripts : l'iframe peut exécuter des scripts ;
- allow-forms : l'envoi de formulaire dans l'iframe est permis ;
- allow-popups : l'iframe peut ouvrir des popup (fenêtres contextuelles) ;
- allow-modals : l'iframe peut ouvrir des fenêtres modales ;
- allow-orientation-lock ;
- allow-pointer-lock ;
- allow-popups-to-escape-sandbox ;
- allow-presentation ;
- allow-same-origin ;
- allow-top-navigation ;
- allow-top-navigation-by-user-activation.

Un autre souci lié à l'utilisation d'iframe est l'intégration du contenu de notre site sur d'autres sites. Nous pouvons appeler ça des menaces « sortantes » pour les distinguer des précédentes.

Les intentions liées à ce comportement sont généralement malveillantes : vol de contenu, tentative d'imitation de notre site ou épuisement de nos ressources.

En effet, vous devez bien comprendre que lorsque quelqu'un intègre nos contenus sur son site avec un élément iframe, l'affichage du contenu de l'iframe va puiser dans nos ressources serveur puisque le contenu va être demandé et chargé à partir de notre serveur.

Pour se prémunir contre cela, nous allons pouvoir interdire aux autres sites d'intégrer nos contenus via des éléments iframe. Pour faire cela, nous allons devoir définir une politique de sécurité de contenu ou CSP (Content Security Policy) adaptée.

A ce niveau, le sujet commence à devenir vraiment complexe puisqu'il va falloir configurer nos en-tête HTTP à partir de notre serveur. Je n'entrerai pas plus dans le détail ici car le sujet mériterait un cours à lui seul et serai bien trop complexe pour le moment.

Cependant, retenez tous ces termes pour pouvoir vous renseigner plus avant si un jour vous avez besoin d'utiliser ces objets.

Les autres éléments d'intégration

Pour être tout à fait exhaustif, je dois ici préciser que l'élément `iframe` n'est pas le seul élément d'intégration de ressources externes.

En effet, d'autres éléments comme `embed` et `object` existent et servent à intégrer des documents comment des PDF, des graphiques vectoriels (SVG) ou même des Flash.

Cependant, ces éléments sont aujourd'hui à mon sens complètement désuets et vous ne devriez jamais avoir à les utiliser. Nous ne les étudierons donc pas dans ce cours.

PARTIE IX

Fond, dégradés et ombres

Gestion de la couleur de fond en CSS

Dans cette nouvelle partie, nous allons nous intéresser à la personnalisation du fond de nos boites éléments en CSS et commençant ici avec l'ajout d'une couleur de fond qui va être possible avec la propriété `background-color` ou avec la notation `background` dont nous reparlerons dans la leçon suivante.

Le champ d'application de background-color

Tout élément HTML est avant tout une boite rectangulaire qui peut être composée d'un contenu, de marges internes, d'une bordure et de marges externes.

Lorsqu'on ajoute une couleur de fond à un élément, cette couleur va remplir l'espace pris par le contenu et les marges internes. En revanche, la couleur de fond ne s'appliquera pas aux marges externes qui sont considérées comme « en dehors » de l'élément.

Notez par ailleurs une chose intéressante ici : lorsqu'on donne une couleur de fond à un élément parent, on « dirait » que les éléments enfants en héritent. En réalité, ce n'est pas tout à fait correct : les éléments ont par défaut un `background-color : transparent` et c'est la raison pour laquelle la couleur de fond de l'élément parent va également remplir l'espace pris par les éléments enfants sauf si on spécifie une couleur de fond différente pour eux.

Les valeurs de couleur de background-color

La propriété CSS `background-color` va accepter les mêmes notations de couleurs que la propriété `color` c'est-à-dire toutes les formes de notations de couleurs en CSS. On va ainsi pouvoir lui passer :

- Un nom de couleur de anglais ;
- Une valeur hexadécimale ;
- Une valeur RGB (ou RGBa) ;
- Une valeur HSL (ou HSLa).

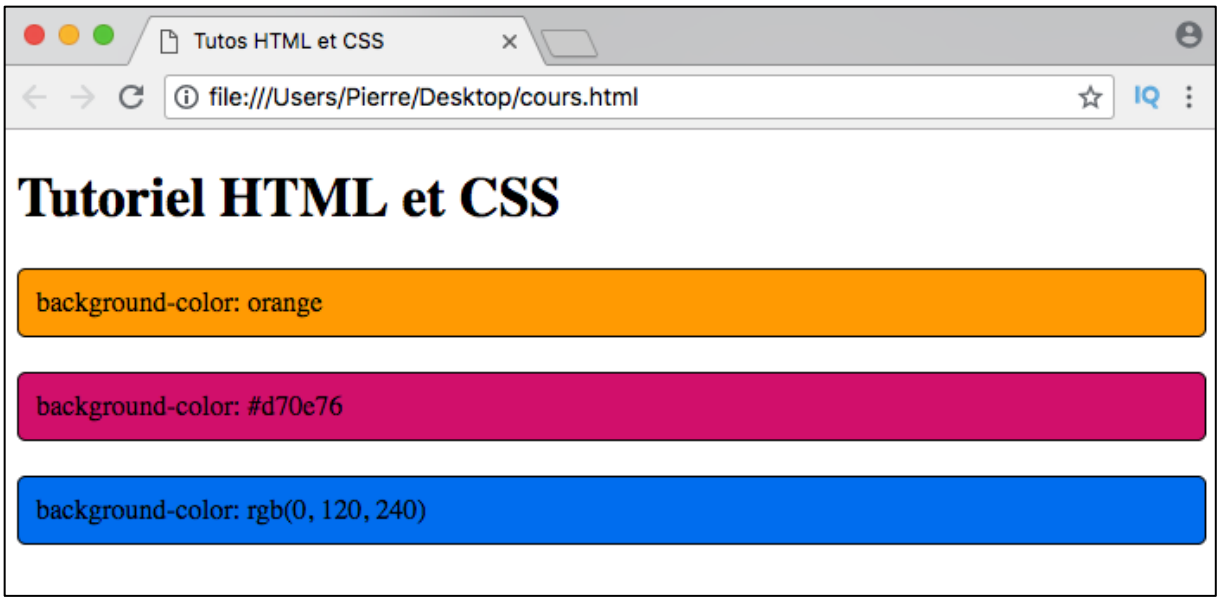
Exemples d'utilisation de background-color en CSS

Attribuer un background-color opaque à un élément

Nous allons pouvoir ajouter une couleur de fond opaque à un élément en utilisant soit une notation de couleur de type nom de couleur, soit une notation hexadécimale, soit une notation RGB ou HSL.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>

        <p id="ex1">background-color: orange</p>
        <p id="ex2">background-color: #d70e76</p>
        <p id="ex3">background-color: rgb(0, 120, 240)</p>
    </body>
</html>
```

```css
p{
    padding: 10px;
    border: 1px solid black;
    border-radius: 5px;
    margin-bottom: 20px;
}

#ex1{
    background-color: orange;
}
#ex2{
    background-color: #d70e76;
}
#ex3{
    background-color: rgb(0, 120, 240);
}
```



Ici, j'ai ajouté un padding c'est-à-dire des marges intérieures, une bordure et des marges externes aux différents paragraphes afin de rendre l'exemple plus visuel et que vous voyiez bien à quoi s'applique le background-color en CSS.

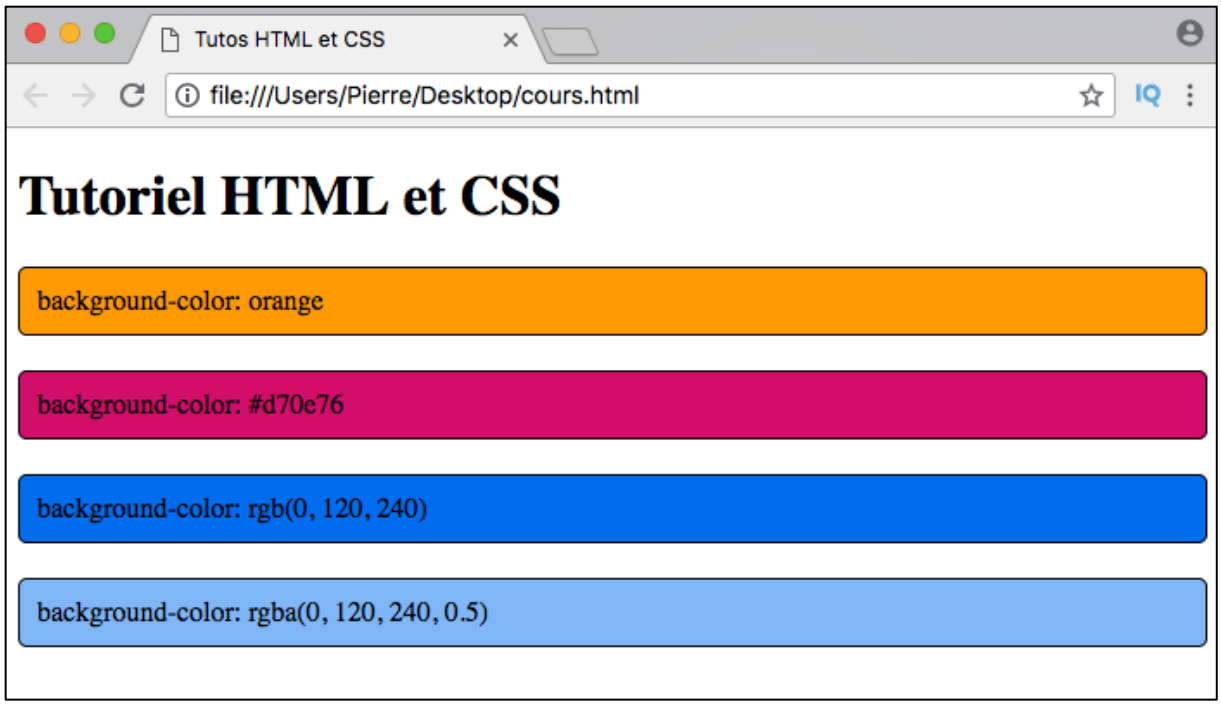
Ajouter un background-color semi transparent à un élément

Nous allons également pouvoir ajouter une couleur de fond semi transparente à un élément en utilisant cette fois-ci les notations RGBa.

Notez bien ici qu'utiliser la propriété opacity sur nos éléments ne produirait pas le comportement voulu puisque l'intégralité de nos éléments (contenu et bordure compris) serait alors semi transparent or nous voulons que seulement le fond de l'élément soit transparent.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>

        <p id="ex1">background-color: orange</p>
        <p id="ex2">background-color: #d70e76</p>
        <p id="ex3">background-color: rgb(0, 120, 240)</p>
        <p id="ex4">background-color: rgba(0, 120, 240, 0.5)</p>
    </body>
</html>
```

```css
p{
    padding: 10px;
    border: 1px solid black;
    border-radius: 5px;
    margin-bottom: 20px;
}

#ex1{
    background-color: orange;
}
#ex2{
    background-color: #d70e76;
}
#ex3{
    background-color: rgb(0, 120, 240);
}
#ex4{
    background-color: rgba(0, 120, 240, 0.5);
}
```

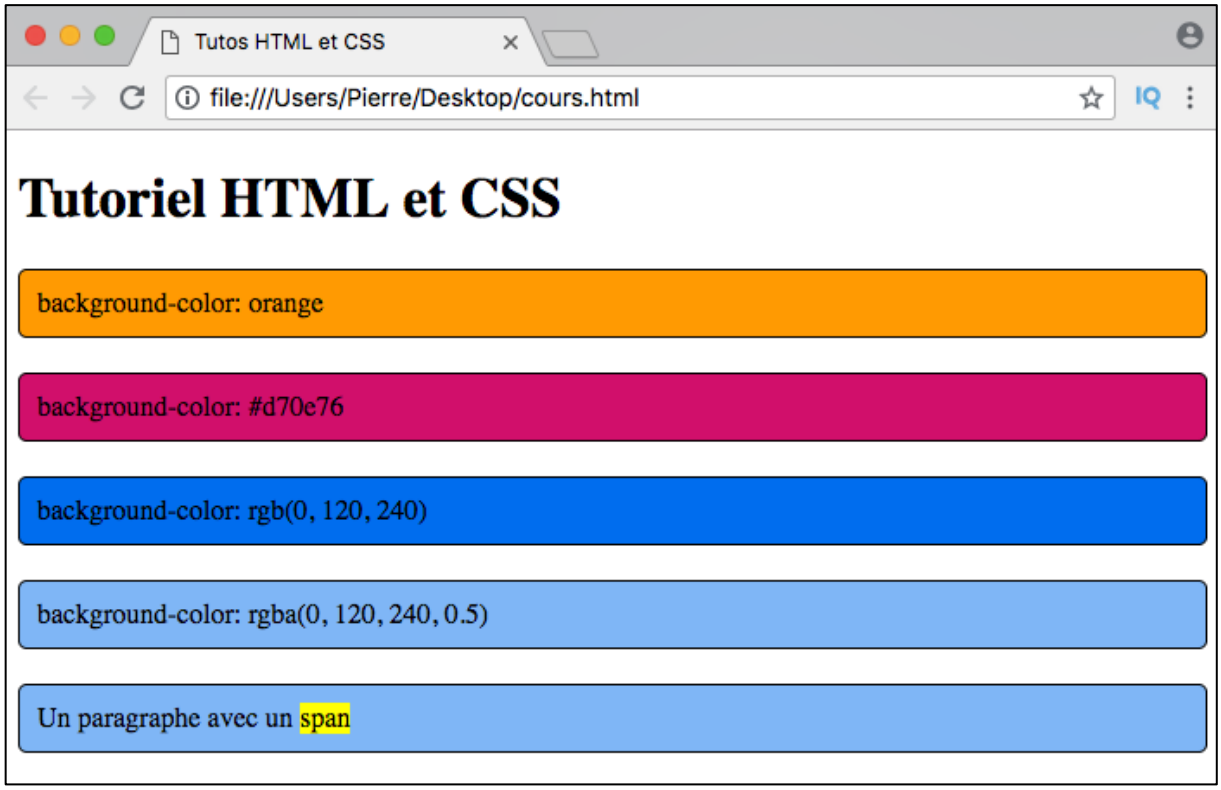


Ajouter différentes couleurs de fond à un élément parent et à ses enfants

Finalement, notez qu'on va tout à fait pouvoir ajouter différents `background-color` à différents éléments imbriqués. Dans ce cas-là, la couleur de fond définie pour chaque élément s'appliquera à l'élément en question.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>

        <p id="ex1">background-color: orange</p>
        <p id="ex2">background-color: #d70e76</p>
        <p id="ex3">background-color: rgb(0, 120, 240)</p>
        <p id="ex4">background-color: rgba(0, 120, 240, 0.5)</p>

        <p id="ex5">Un paragraphe avec un <span>span</span></p>
    </body>
</html>
```

```css
p{
    padding: 10px;
    border: 1px solid black;
    border-radius: 5px;
    margin-bottom: 20px;
}

#ex1{
    background-color: orange;
}
#ex2{
    background-color: #d70e76;
}
#ex3{
    background-color: rgb(0, 120, 240);
}
#ex4{
    background-color: rgba(0, 120, 240, 0.5);
}
#ex5{
    background-color: rgba(0, 120, 240, 0.5);
}
span{
    background-color: yellow;
}
```



Ici, vous pouvez noter que le background-color ne s'applique que sur l'arrière-plan du contenu de notre span ce qui est tout à fait normal puisqu'un élément span est un élément de type inline par défaut et ne prend donc que l'espace nécessaire à son contenu en largeur.

Ajouter des images de fond en CSS

Dans cette nouvelle leçon, nous allons apprendre à insérer une ou plusieurs images en fond d'un élément plutôt qu'une simple couleur. Nous allons pouvoir faire cela avec la propriété `background-image` ou avec la notation raccourcie `background`.

Notez que les dégradés sont considérés comme des images en CSS. Nous allons donc devoir utiliser `background-image` ou `background` pour définir un dégradé en fond d'un élément. Nous allons apprendre à créer des dégradés dans la leçon suivante.

Les couches ou « layers » constituant le fond d'un élément

Pour bien comprendre comment vont fonctionner et s'appliquer les propriétés `background-image` et `background-color`, vous devez avant tout savoir qu'un élément HTML (ou plus exactement la boite le représentant) peut avoir plusieurs couches depuis le CSS3.

Cela veut dire qu'on va pouvoir « empiler » différents fonds les uns au-dessus des autres pour un même élément, à une limitation près qui est qu'on ne peut déclarer qu'une seule valeur pour la propriété `background-color` pour un élément et donc qu'un élément ne peut avoir qu'une couche de couleur de fond avec cette propriété.

En revanche, on va tout à fait pouvoir empiler une ou plusieurs images les unes sur les autres et par-dessus une couleur de fond pour un même élément. Cela va pouvoir s'avérer très utile si une image de fond ne peut pas s'afficher pour une raison ou une autre par exemple.

Retenez bien ici l'ordre des couches : la première image déclarée va être la plus proche de l'utilisateur, c'est-à-dire l'image de fond qui sera visible par défaut. Si on spécifie également une couleur de fond, alors cette couche sera placée derrière les couches « images de fond ».

Note : J'ai dit plus haut qu'il était impossible d'ajouter plusieurs couches de couleurs de fond à un élément avec `background-color`. Cependant, rien ne nous empêche d'utiliser les dégradés ou gradient en anglais pour mixer plusieurs couleurs sur la même couche.

Utilisation de la propriété background-image

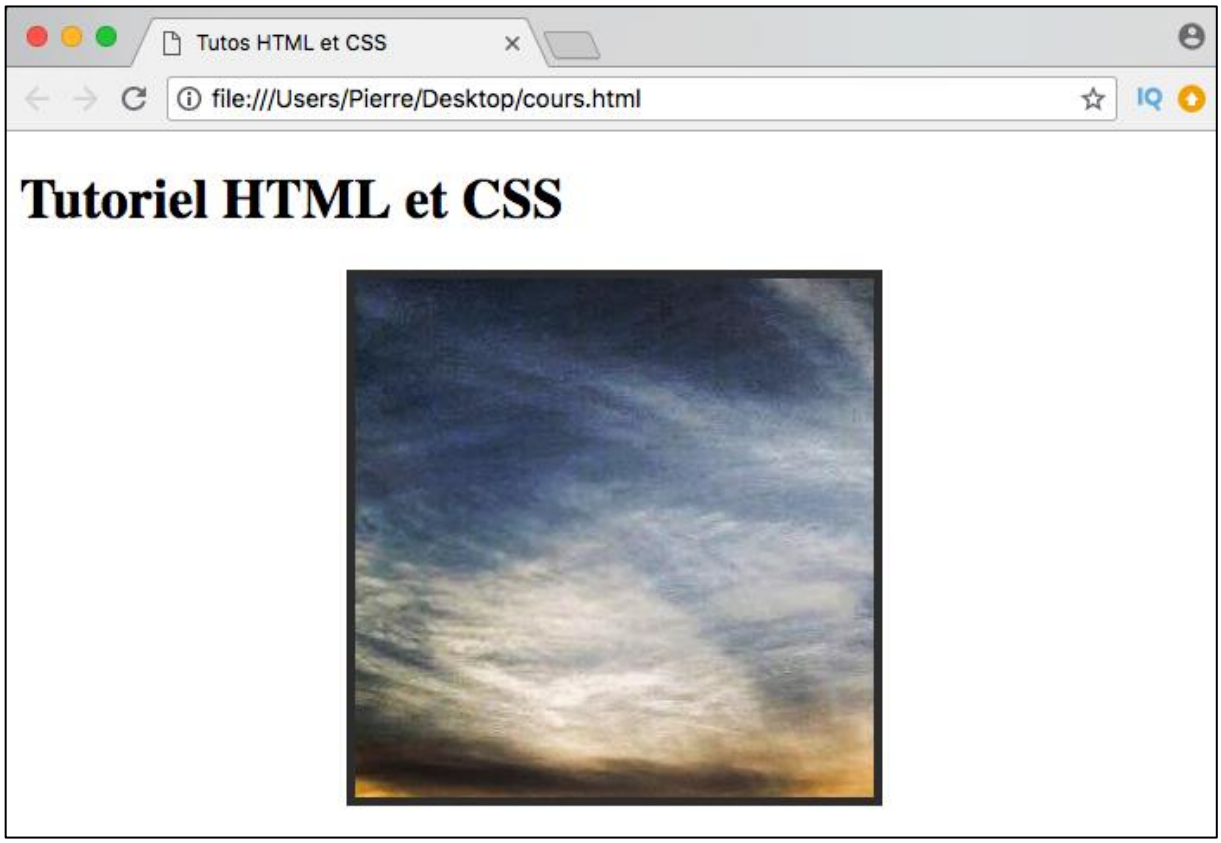
La propriété `background-image` va nous permettre d'affecter une ou plusieurs images de fond à un élément. La première image déclarée sera l'image visible par défaut et chacune des autres images possiblement déclarées sera une couche en dessous.

Ajouter une image de fond avec background-image

Voyons comment ajouter une image de fond :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
    </body>
</html>
```

```css
div{
    width: 300px;
    height: 300px;
    border: 5px solid #333;
    margin: 0px auto;
}

#ex1{
    background-image: url("sunset.jpg");
}
```



Ici, je me suis contenté d'attribuer une image de fond à mon `div id="ex1"`. Pour cela, je le cible en CSS et j'utilise ma propriété `background-image` en précisant l'adresse relative de mon image (qui est ici dans le même dossier que mes fichiers) au sein d'un paramètre URL.

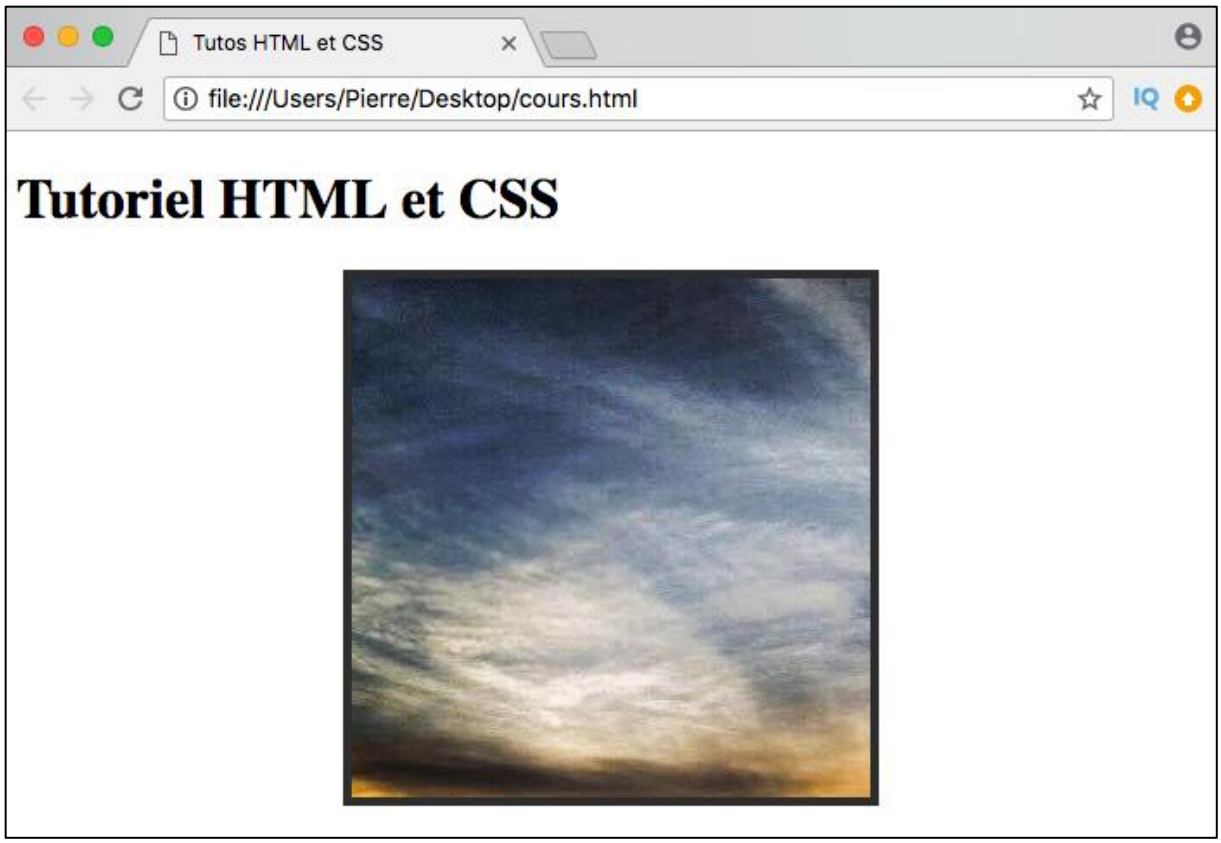
Notez qu'on va également tout à fait pouvoir passer une adresse absolue pour ajouter une image de fond (une adresse de type https://…). Cependant, pour des raisons évidentes, il est déconseillé d'utiliser une image hébergée sur un autre site comme image de fond : si le propriétaire du site supprime son image, votre image de fond n'apparaitra plus !

Ajouter plusieurs images de fond avec background-image

Pour ajouter plusieurs images de fond avec `background-image`, nous allons tout simplement séparer les différentes déclarations d'images par une virgule comme ceci :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
    </body>
</html>
```

```css
div{
    width: 300px;
    height: 300px;
    border: 5px solid #333;
    margin: 0px auto;
}

#ex1{
    background-image: url("sunset.jpg"), url("sunrise.jpg");
}
```



Comme vous pouvez le voir ici, seule la première image déclarée est visible. C'est un comportement tout à fait normal et cela signifie que la première image déclarée a bien été chargée. Cependant, la deuxième image a bien été placée en arrière-plan également, seulement elle est cachée sous la première.

Cette deuxième image va être visible dans deux cas : si la première image ne peut pas être affichée ou si la première image ne remplit pas le fond de l'élément (et que la deuxième image est plus grande que la première). Pas d'inquiétude, nous allons apprendre par la suite à modifier la position et la taille d'une image de fond.

Une bonne pratique : déclarer une couleur de fond avec une image de fond

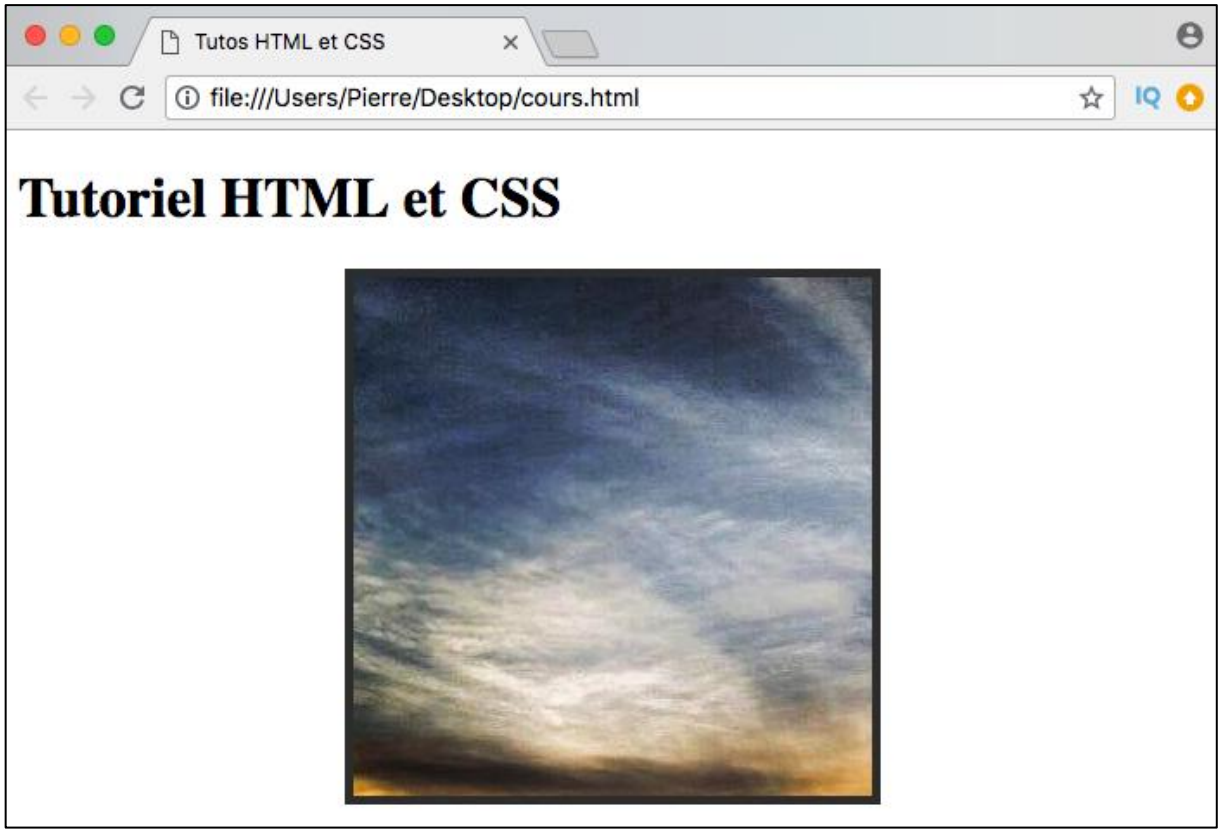
Par ailleurs, notez qu'il est considéré comme une bonne pratique de toujours déclarer une couleur de fond avec la propriété `background-color` en plus d'une ou de plusieurs images de fond au cas où celles-ci ne pourraient pas s'afficher.

Une nouvelle fois, si une image peut être chargée normalement, alors la propriété `background-color` n'aura aucun impact puisque la couche « couleur de fond » va se placer sous les couches « images de fond ».

La propriété `background-color` va accepter les mêmes valeurs que la propriété `color`, c'est à dire toutes les valeurs possibles de type « couleur » : nom de couleur, valeur hexadécimale, valeur RGB, RGBa, etc.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
    </body>
</html>
```

```css
div{
    width: 300px;
    height: 300px;
    border: 5px solid #333;
    margin: 0px auto;
}

#ex1{
    background-image: url("sunset.jpg"), url("sunrise.jpg");
    background-color : RGBa(0, 255, 0, 0.5);
}
```



Ajouter différentes images de fond à différents éléments

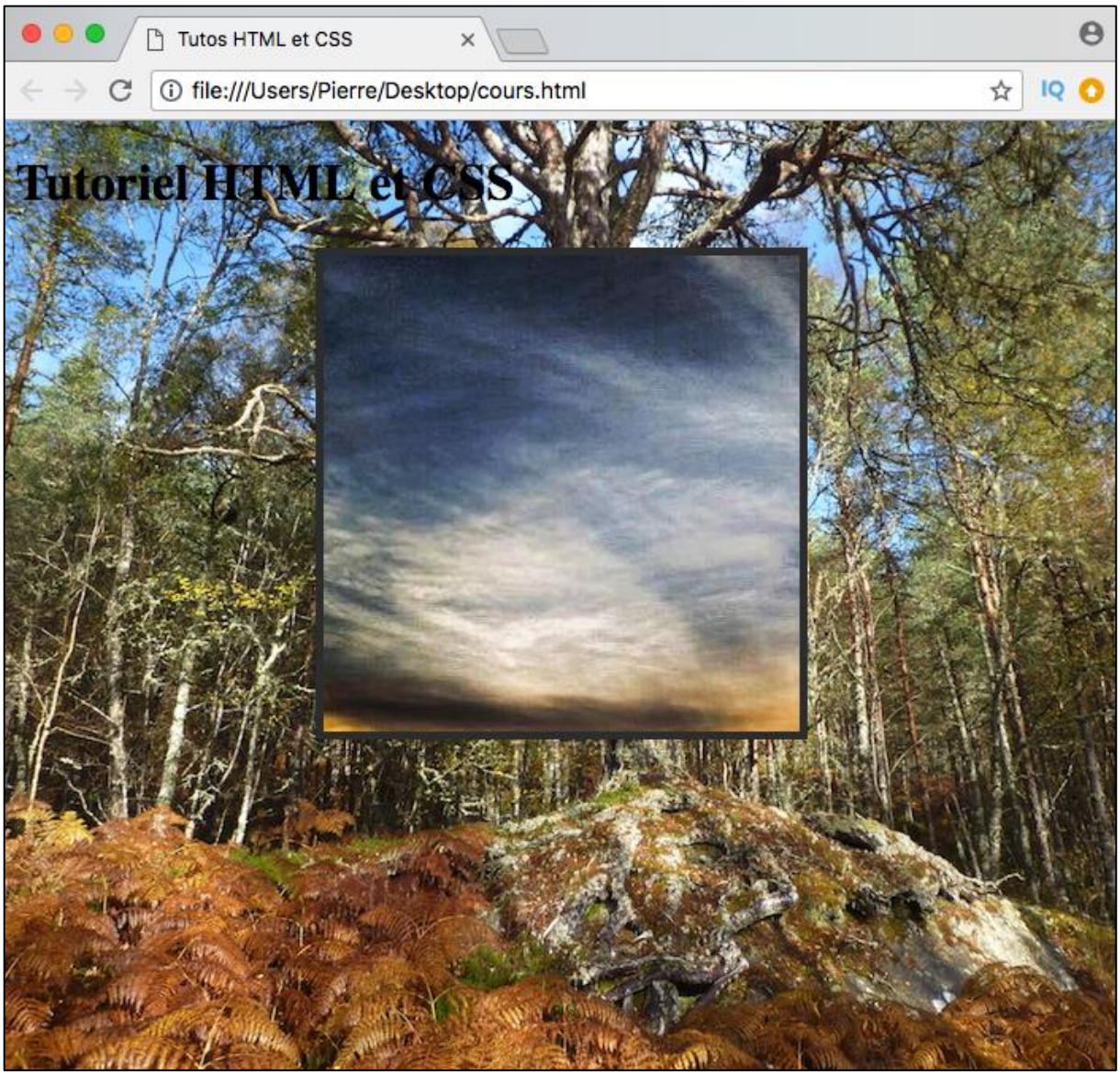
Bien évidemment, rien ne nous empêche d'attribuer différentes images de fond à différents éléments. Dans le cas où un élément est inclus dans un autre, l'image ou la couleur de fond qui lui a été attribuée restera au premier plan.

Ainsi, on va par exemple pouvoir ajouter une image de fond à l'élément représentant le contenu visible de notre page à savoir l'élément `body` afin d'ajouter une image de fond « à notre page » en plus de l'image de fond passée à notre `div`.

Notez cependant que pour des raisons évidentes de lisibilité et de clarté nous éviterons généralement de multiplier les images de fond et préférerons généralement des couleurs unies notamment pour le fond d'une page.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
    </body>
</html>
```

```
div{
    width: 300px;
    height: 300px;
    border: 5px solid #333;
    margin: 0px auto;
}
body{
    background-image: url("tree.jpg");
    background-color: #fafafa;
}

#ex1{
    background-image: url("sunset.jpg"), url("sunrise.jpg");
    background-color : RGBa(0, 255, 0, 0.5);
}
```



Les propriétés nous permettant de gérer le comportement du fond

Nous savons maintenant comment ajouter une ou plusieurs images de fond à nos éléments. Cependant, nous n'avons aucun contrôle sur leur affichage. Pour contrôler le comportement de nos images de fond, nous allons pouvoir utiliser les propriétés suivantes :

- La propriété `background-position` ;
- La propriété `background-size` ;
- La propriété `background-repeat` ;
- La propriété `background-origin` ;
- La propriété `background-clip` ;
- La propriété `background-attachment`.

Chacune des propriétés citées ci-dessous va nous permettre de préciser un comportement spécifique pour nos images de fond comme la taille, le comportement de répétition, etc.

Notez que toutes ces propriétés peuvent être déclarées d'un coup dans la notation raccourcie `background` que nous étudierons à la fin de cette leçon.

Gérer la répétition d'une image de fond avec background-repeat

La propriété `background-repeat` va nous permettre de définir si une image d'arrière-plan doit être répétée ou pas et selon quel(s) axe(s) (axe horizontal et / ou vertical).

Par défaut, une image de fond va être rognée à la taille de l'élément en largeur et / ou en hauteur si elle plus grande que celui-ci et au contraire être répétée horizontalement et verticalement si elle est plus petite que l'élément jusqu'à remplir la surface de fond définie. Regardez plutôt l'exemple suivant :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
    </body>
</html>
```

```css
div{
    width: 320px;
    height: 320px;
    border: 5px solid #333;
    margin: 0px auto;
}

#ex1{
    background-image: url("emoji-smile.png");
    background-color : RGBa(255, 255, 0, 0.2);
}
```

Ici, mon `div` fait 320px de large (ou 330px en comptant les bordures) et 320px de haut (ou 330px en comptant les bordures). Mon image « emoji-smile.png » possède quant-à-elle une dimension de 50px*50px et va donc être par défaut répétée jusqu'à remplir le fond du `div` à la fois en largeur et en hauteur.

Notez deux choses intéressantes ici :

- Par défaut, l'image ne va pas être redimensionnée pour apparaitre un nombre entier de fois : on voit bien ici que les dernières itérations en largeur et en hauteur sont coupées ;
- Mon image gère ici la transparence, c'est pour cela que vous pouvez voir la couleur de fond du `div` en arrière-plan dans les endroits non remplis par celle-ci !

Nous allons pouvoir annuler ce comportement de répétition horizontale et verticale par défaut et définir nos propres règles de répétitions grâce à la propriété `background-repeat`. Pour cela, nous allons passer une ou deux valeurs à la propriété `background-repeat` parmi les valeurs suivantes.

Si une seule valeur est passée, alors elle définira le comportement de répétition de l'image à la fois sur son axe horizontal et sur son axe vertical. Si deux valeurs sont passées, la première valeur définira le comportement de répétition horizontal tandis que la seconde déterminera le comportement de répétition vertical de l'image.

- `repeat` : valeur par défaut. L'image est répétée jusqu'à remplir le fond ;
- `no-repeat` : l'image n'est pas répétée et n'apparait qu'une fois ;

- `round` : l'image est répétée jusqu'à remplir le fond et est étirée si nécessaire de façon à n'avoir qu'un nombre de répétitions complet de l'image (l'image ne pourra pas être rognée);
- `space` : l'image est répétée jusqu'à remplir le fond un nombre complet de fois comme pour la valeur `round`. Cependant, ici, l'image n'est pas étirée : la première et la dernière répétition de l'image sont collées au bord de l'élément et l'espace va être distribué équitablement entre chaque répétition de l'image.

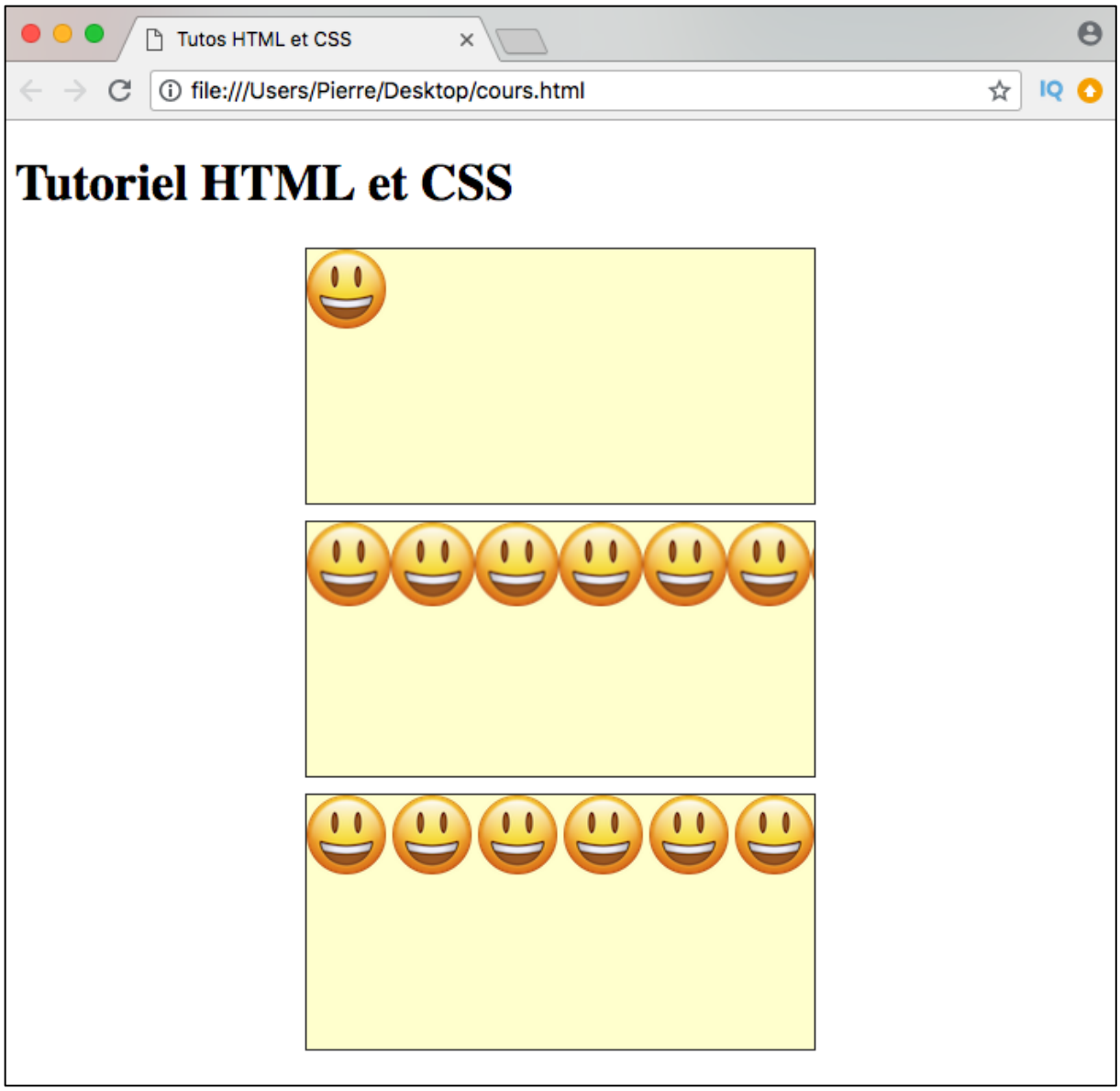
Attention cependant, les valeurs `round` et `space` ainsi que la syntaxe avec deux valeurs font partie des recommandations du CSS3 et ne sont donc à proprement parler pas encore officielles. Il est donc possible que vous n'ayez pas le comportement attendu selon le navigateur et la version utilisée.

En plus de ces valeurs, il existe également deux autres valeurs qui servent seules à définir les comportements de répétition à la fois horizontal et vertical de l'image :

- `repeat-x` : l'image est répétée horizontalement jusqu'à remplir le fond mais pas verticalement. L'équivalent avec deux valeurs va être `background-repeat : repeat no-repeat` ;
- `repeat-y` : l'image est répétée verticalement jusqu'à remplir le fond mais pas horizontalement. L'équivalent avec deux valeurs va être `background-repeat : no-repeat repeat`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
    </body>
</html>
```

```css
div{
    width: 320px;
    height: 160px;
    border: 1px solid #333;
    margin: 10px auto;
    background-color : RGBa(255, 255, 0, 0.2);
}

#ex1, #ex2, #ex3{
    background-image: url("emoji-smile.png");
}
#ex1{
    background-repeat: no-repeat;
}
#ex2{
    background-repeat: round no-repeat;
}
#ex3{
    background-repeat: space no-repeat;
}
```



Si plusieurs images de fond ont été déclarées avec background-image pour l'élément, alors on va pouvoir gérer le comportement de répétition de chacune d'entre elles exactement de la même façon : en passant plusieurs valeurs à background-repeat séparées par des virgules. Chaque valeur va s'appliquer à l'image de fond correspondante déclarée avec background-image.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
    </body>
</html>
```

```css
div{
    width: 320px;
    height: 160px;
    border: 1px solid #333;
    margin: 10px auto;
    background-color : RGBa(255, 255, 0, 0.2);
}

#ex1{
    background-image: url("emoji-smile.png"), url("emoji-think.png");
    background-repeat: repeat-x, repeat-y; /*repeat no-repeat, no-repeat repeat*/
}
```

Gérer la taille des images de fond avec background-size

La propriété CSS background-size va nous permettre de gérer la taille des images de fond d'un élément. Cette propriété va pouvoir prendre une ou deux valeurs. Les valeurs vont pouvoir être des mots clefs ou des dimensions absolues ou relatives. En cas de valeur relative, la dimension est exprimée en fonction de la taille du fond (du conteneur) et non pas de la taille originale de l'image.

Si on ne fournit qu'une valeur à background-size, alors cette valeur servira à déterminer la largeur de l'image de fond et la hauteur sera calculée automatiquement par rapport à la largeur fournie. En passant deux valeurs à background-size, la première servira à déterminer la largeur de l'image tandis que la seconde imposera la hauteur de celle-ci. Attention dans ce cas aux proportions de l'image dans le rendu final !

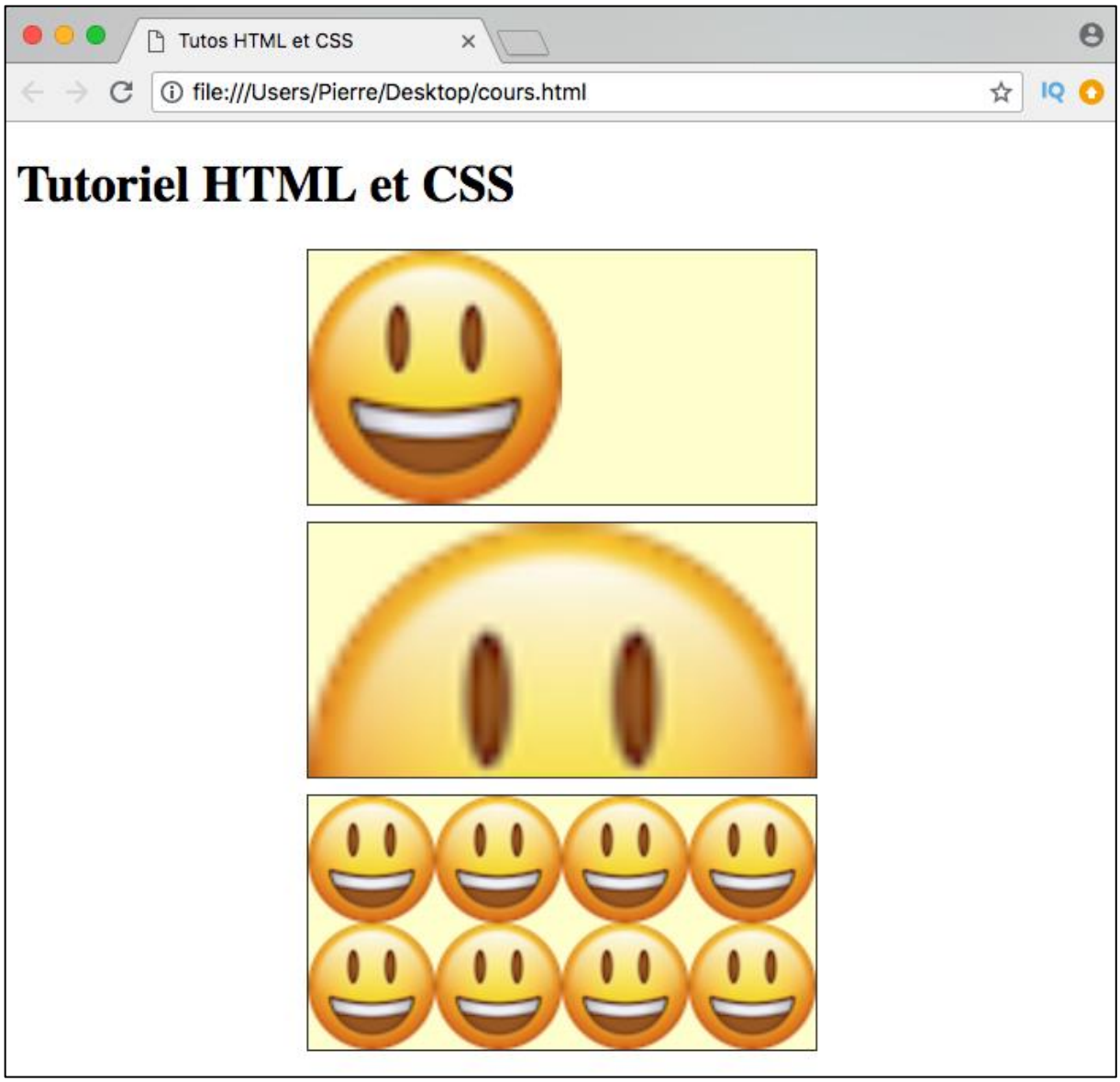
On va également pouvoir fournir l'un de ces deux mots clefs à background-size :

- contain : l'image va être redimensionnée afin qu'elle occupe le maximum de place dans le conteneur tout en conservant ses proportions de base et sans dépasser du conteneur. L'image va être contenue dans le conteneur ;

- cover : l'image va être redimensionnée afin de couvrir tout l'espace (en largeur) dans le conteneur tout en conservant ses proportions de base. L'image va pouvoir déborder du conteneur afin d'occuper toute la surface de fond. Les parties qui dépassent vont être rognées.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
    </body>
</html>
```

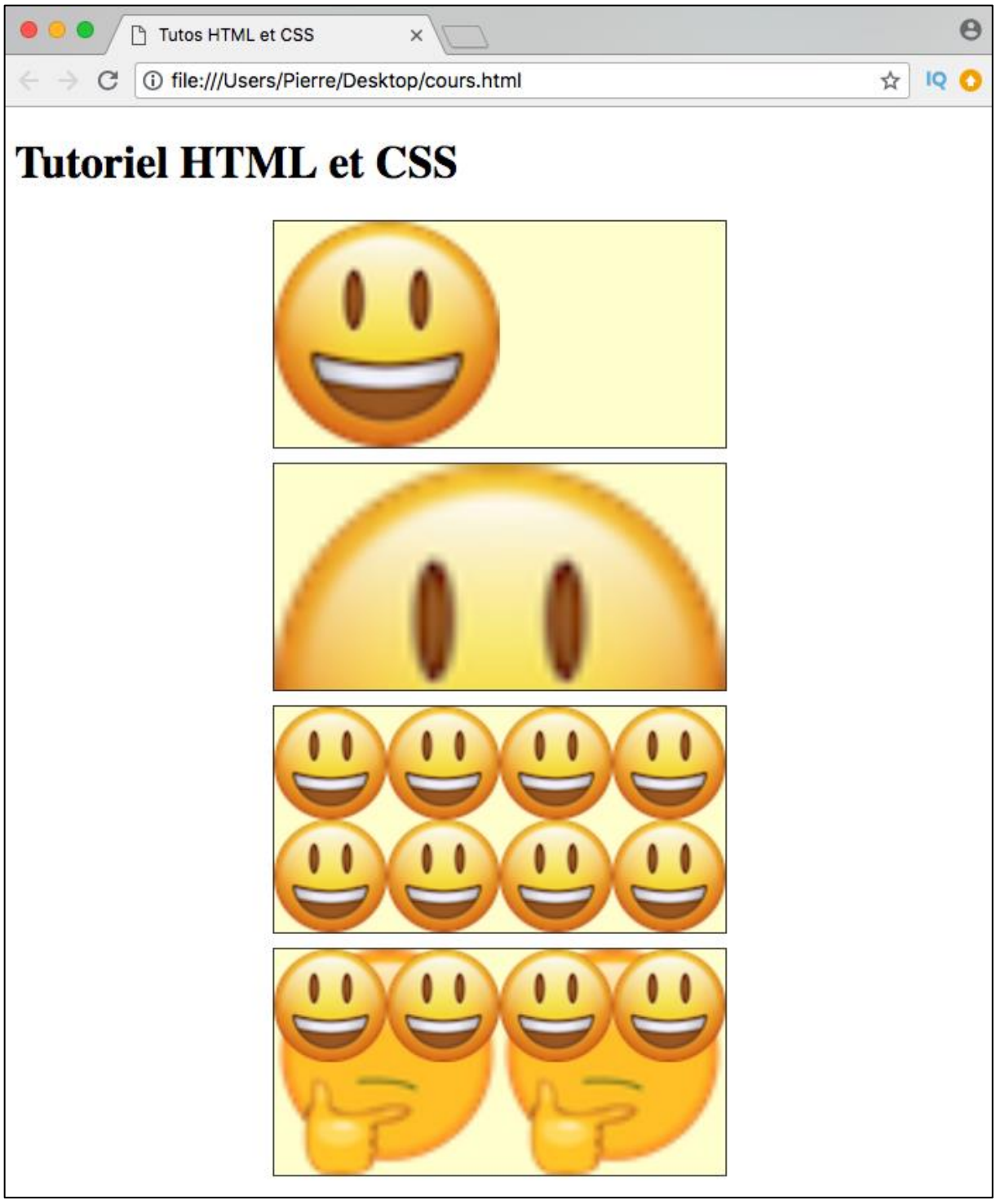
```css
div{
    width: 320px;
    height: 160px;
    border: 1px solid #333;
    margin: 10px auto;
    background-color : RGBa(255, 255, 0, 0.2);
}

#ex1, #ex2, #ex3{
    background-image: url("emoji-smile.png");
}
#ex1{
    background-size: contain;
    background-repeat: no-repeat;
}
#ex2{
    background-size: cover;
    background-repeat: no-repeat;
}
#ex3{
    background-size: 25% 50%; /*25% de la largeur du fond, 50% de sa hauteur*/
    background-repeat: repeat;
}
```



Dans le cas où plusieurs images de fond ont été attribuées avec `background-image`, et si l'on souhaite gérer la taille de chacune de ces images, alors il suffira une nouvelle fois d'indiquer les différentes valeurs de taille pour chaque image dans `background-size` en les séparant par des virgules.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
        <div id="ex4"></div>
    </body>
</html>
```

```css
div{
    width: 320px;
    height: 160px;
    border: 1px solid #333;
    margin: 10px auto;
    background-color : RGBa(255, 255, 0, 0.2);
}

#ex1, #ex2, #ex3{
    background-image: url("emoji-smile.png");
}
#ex1{
    background-size: contain;
    background-repeat: no-repeat;
}
#ex2{
    background-size: cover;
    background-repeat: no-repeat;
}
#ex3{
    background-size: 25% 50%; /*25% de la largeur du fond, 50% de sa hauteur*/
    background-repeat: repeat;
}
#ex4{
    background-image: url("emoji-smile.png"), url("emoji-think.png");
    background-size: 25% 50%, contain;
    background-repeat: repeat-x, repeat;
}
```



Attention : La définition et les valeurs de la propriété `background-size` ne sont pour le moment que candidates au statut de recommandation et ne sont donc pas encore tout à fait officielles. Cependant, elles disposent déjà d'une très bonne prise en charge par les versions récentes des navigateurs les plus populaires.

Gérer le défilement d'une image de fond avec background-attachment

La propriété background-attachment va nous permettre de définir si l'image de fond doit être fixe ou défiler dans son conteneur. Cette propriété ne va donc avoir d'impact que dans le cas où nous avons une barre de défilement ou de scrolling dans notre élément.

Nous allons pouvoir passer une valeur parmi les suivantes à background-attachment :

- scroll : valeur par défaut. L'image de fond ne défile pas avec le contenu de l'élément auquel elle est associée mais défile avec l'élément dans la page ;
- fixed : l'image de fond restera fixe par rapport à son conteneur quelle que soit la taille de l'élément auquel elle est associée et même si l'élément en soi possède une barre de défilement ;
- local : l'image de fond va se déplacer avec le contenu de l'élément auquel elle est associée et défiler dans la page avec l'élément auquel elle est associée.

Regardez plutôt l'exemple suivant pour bien comprendre le fonctionnement de cette propriété (pensez à bien jouer avec chacune des deux barres de défilement) :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div class="conteneur">
            <div id="ex1" class="ex">
                <p>Le fond ne défile pas <br>ni dans le conteneur
                <br>ni dans l'élément</p>
            </div>
        </div>
        <div class="conteneur">
            <div id="ex2" class="ex">
                <p>Défile <br>avec l'élément dans le conteneur mais
                <br>pas dans l'élément (pas avec le contenu)</p>
            </div>
        </div>
        <div class="conteneur">
            <div id="ex3" class="ex">
                <p>Défile <br>à la fois avec l'élément dans le
                conteneur et <br>aussi avec le contenu</p>
            </div>
        </div>
    </body>
</html>
```

```css
.conteneur{
    width: 400px;
    height: 200px;
    border: 1px solid #333;
    margin: 10px auto;
    background-color : RGBa(255, 255, 0, 0.2);
    overflow: scroll;
}
p{
    font-size: 50px;
    text-align:center;
    color: #000;
    text-shadow: 0px 0px 1px #fff;
    background-color: RGBa(255,255,255,0.5);
}

#ex1, #ex2, #ex3{
    background-image: url("emoji-smile.png");
    background-size: 100px;
    background-repeat: repeat;
    width: 385px;
    height: 300px;
    overflow: scroll;
}
#ex1{
    background-attachment: fixed;
}
#ex2{
    background-attachment: scroll;
}
#ex3{
    background-attachment: local;
}
```

Comme pour les propriétés précédentes, dans le cas où on a plusieurs images de fond, il suffira de donner autant de valeurs à `background-attachment` que d'images de fond déclarées et de séparer les différentes valeurs données par des virgules.

Déterminer la position du fond avec background-origin

La propriété `background-origin` nous permet d'ajuster la zone dans laquelle notre ou nos images de fond vont pouvoir se placer ou plus exactement de déterminer la position de l'origine (donc de son coin supérieur gauche par défaut) de l'image de fond dans son élément.

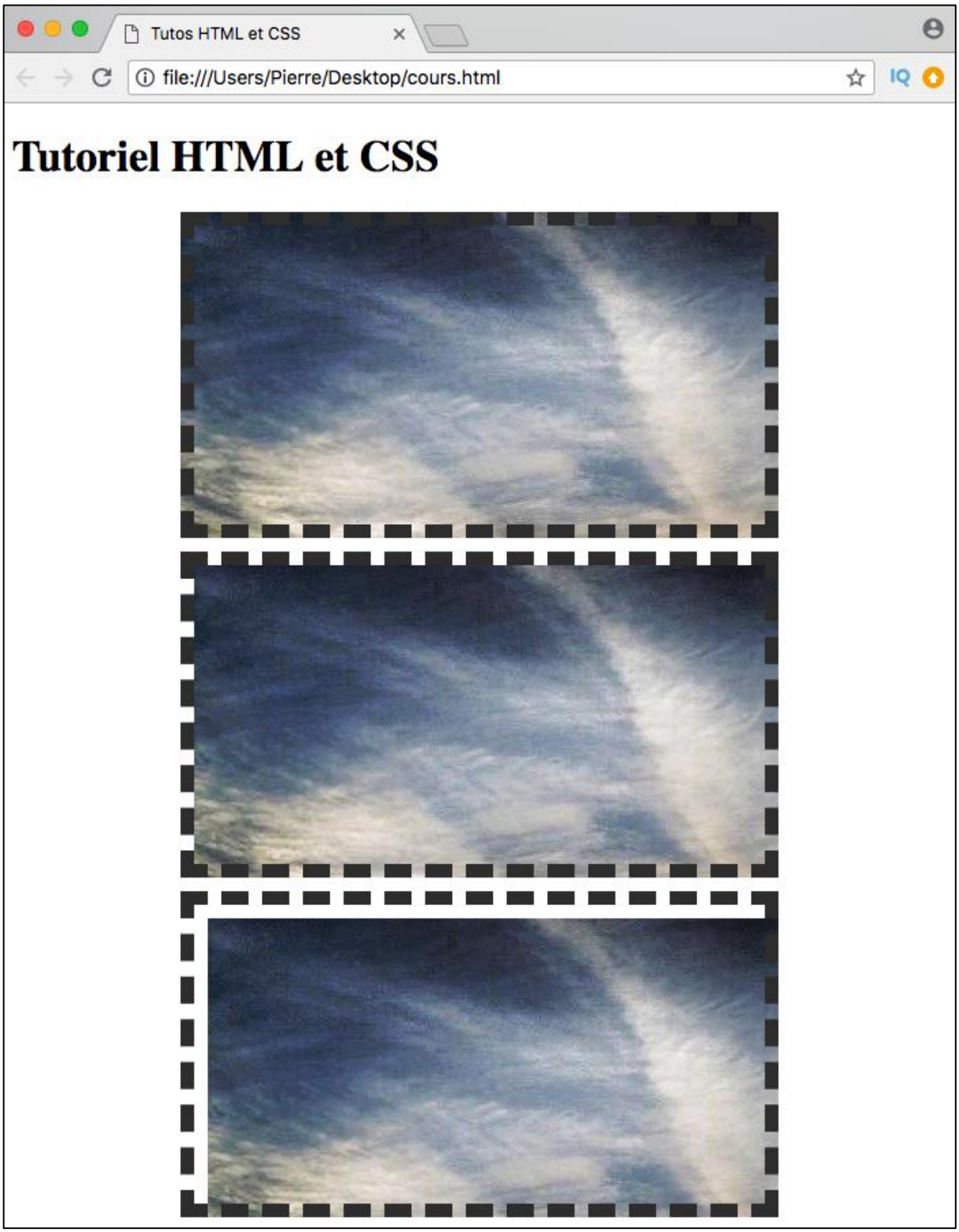
Notez que si on a appliqué un `background-attachment : fixed` à notre élément, alors la propriété `background-origin` sera tout simplement ignorée (l'image de fond sera fixe en toutes circonstances).

On va pouvoir passer l'une de ces trois valeurs à `background-origin` :

- `padding-box` : valeur par défaut. La position de l'origine de l'image va être déterminée par rapport à la boite représentant le contenu + marges internes de l'élément. Le point d'origine de l'image est donc à l'extérieur des bordures ;
- `border-box` : la position de l'origine de l'image de fond va être déterminée par rapport à la bordure de l'élément. Le coin supérieur gauche de l'image va donc se situer derrière la bordure supérieure gauche par défaut ;
- `content-box` : la position de l'origine de l'image va être déterminée par rapport à la boite représentant le contenu de l'élément. Elle ne va donc pas recouvrir l'espace supérieur gauche occupé par le `padding` par défaut.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
    </body>
</html>
```

```css
div{
    width: 400px;
    height: 200px;
    border: 10px dashed #333;
    padding: 10px;
    margin: 10px auto;
    background-color : RGBa(255, 255, 255, 0.2);
}

#ex1, #ex2, #ex3{
    background-image: url("sunset.jpg");

    background-repeat: no-repeat;
}
#ex1{
    background-origin: border-box;
}
#ex2{
    background-origin: padding-box;
}
#ex3{
    background-origin: content-box;
}
```



Comme précédemment, si plusieurs images de fond ont été déclarées avec background-image, il suffira de passer une valeur pour chaque image à background-origin en séparant les différentes valeurs par une virgule.

Positionner une image de fond avec background-position

La propriété CSS background-position va nous permettre de définir à partir de quelle position une image de fond doit s'afficher dans la surface de fond de l'élément associé et relativement à la surface de la zone de fond qu'elle peut occuper (qui a été définie avec background-origin).

Cette propriété va pouvoir prendre des mots clefs en valeur comme top, right, bottom, left ou center ou des distances relatives ou absolues comme 25% ou 50px ou un mélange des deux comme bottom 50pxqui indique que le bord bas de l'image de fond sera situé à 50px du « bas de l'élément » (selon la zone de fond définie avec background-origin une nouvelle fois).

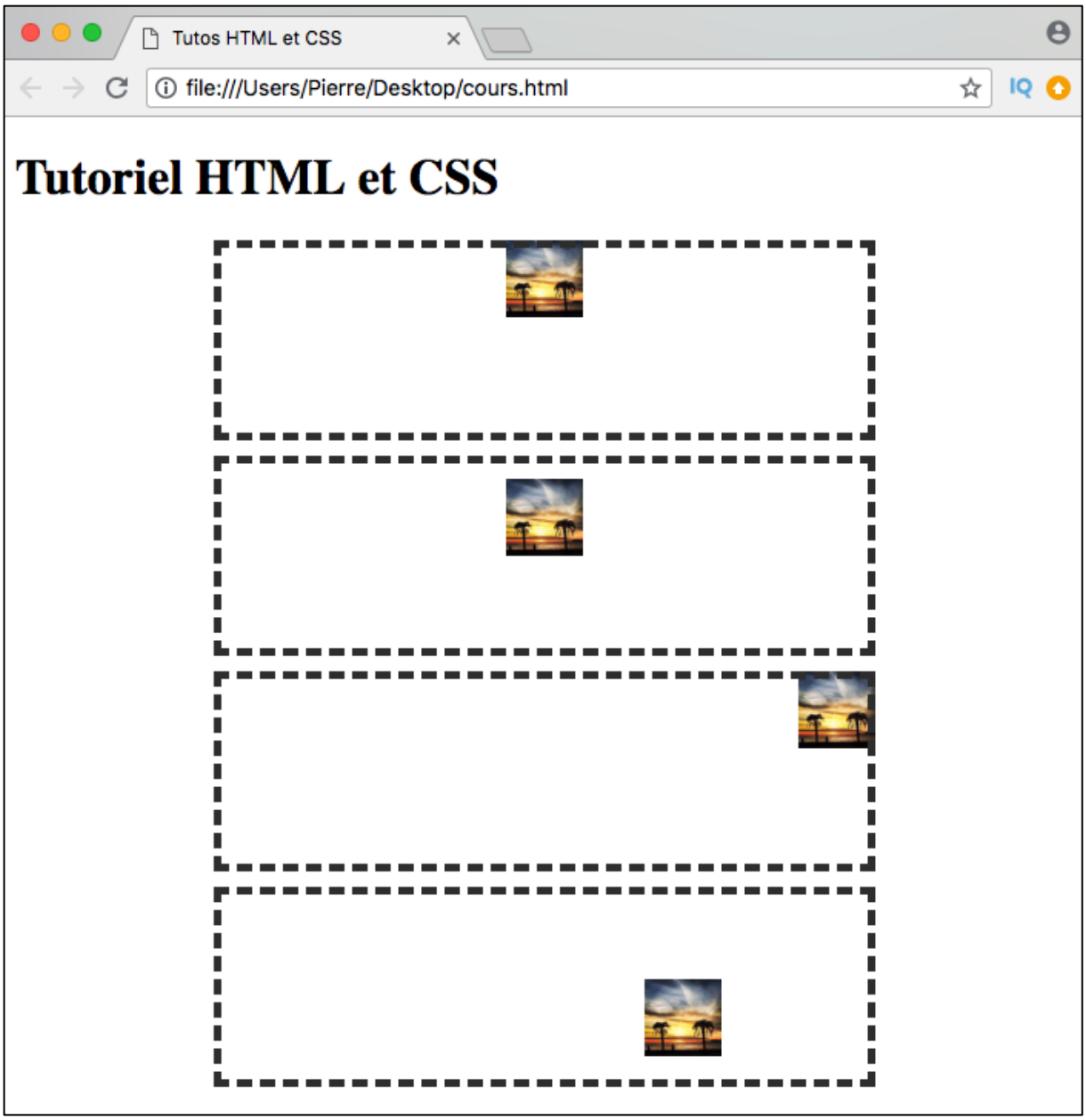
On va pouvoir passer une ou deux valeurs (simples comme bottom ou composées comme bottom 50px) à background-position. En ne passant qu'une valeur :

- background-position : center va centrer l'image de fond ;
- Les autres mots clefs vont coller l'image de fond au bord correspondant au mot clef et la deuxième position (horizontale ou verticale) sera par défaut déterminée à 50% de la taille de l'élément. Le point de départ de l'affichage de l'image de fond (dans le cas où elle serait plus grande que l'élément associé) sera déterminé par le mot clef ;
- Une longueur absolue ou relative va déterminer la position de l'image de fond par rapport au bord gauche de l'élément. La position verticale calculée sera 50% de la hauteur de l'élément. Le point de départ de l'image sera également son bord gauche.

En passant deux valeurs à background-position, la première valeur déterminera le point d'origine de l'image sur l'axe horizontal tandis que la seconde valeur déterminera le point d'origine de l'image dans l'axe vertical.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
        <div id="ex4"></div>
    </body>
</html>
```

```css
div{
    width: 400px;
    height: 100px;
    border: 5px dashed #333;
    padding: 10px;
    margin: 10px auto;
    background-color : RGBa(255, 255, 255, 0.2);
}

#ex1, #ex2, #ex3, #ex4{
    background-image: url("sunset.jpg");
    background-size: 50px;
    background-repeat: no-repeat;
}
#ex1{
    background-origin: border-box;
    background-position : top;
}
#ex2{
    background-origin: content-box;
    background-position : top;
}
#ex3{
    background-origin: border-box;
    background-position : top right;
}
#ex4{
    background-origin: border-box;
    background-position : bottom 20px right 100px;
}
```



Une nouvelle fois, dans le cas où l'image de fond est plus grande que son élément associé, le point de départ d'affichage de l'image sera son bord gauche si on passe une longueur ou sera déterminé par le mot clef passé.

Déterminer la surface que peut occuper notre fond avec background-clip

La propriété `background-clip` permet de définir la surface du fond de l'élément dans laquelle l'image de fond va être visible ou pas.

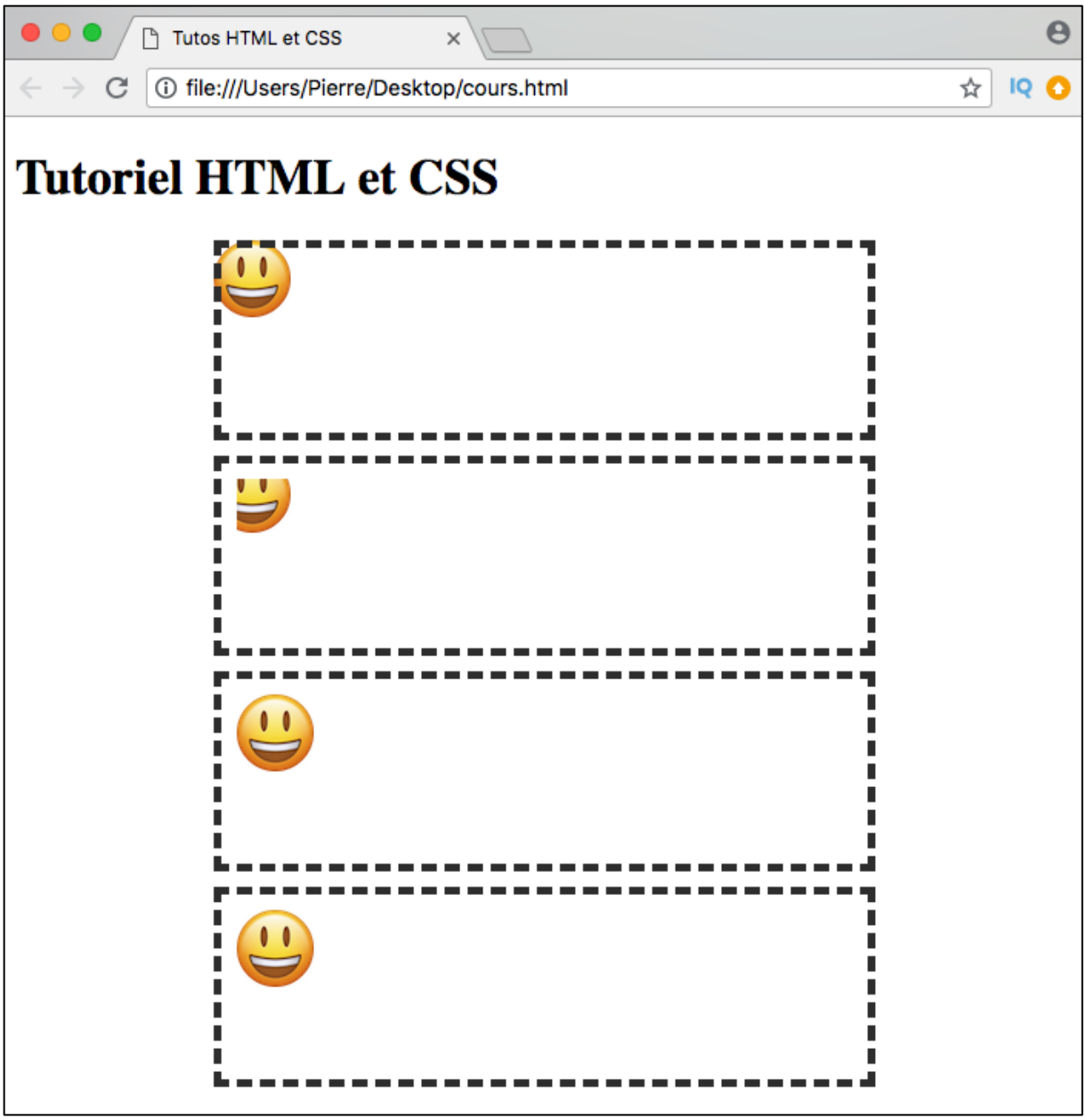
Cette propriété va pouvoir prendre les mêmes valeurs que `background-origin` à savoir :

- border-box : valeur par défaut. L'image de fond sera visible jusque sous les bordures de l'élément (on pourra la voir dans le cas d'une bordure en pointillés par exemple) ;
- padding-box : l'image de fond sera visible jusqu'à la limite de la marge intérieure (padding) mais la partie sous les bordures de l'élément ne le sera pas ;
- content-box : l'image de fond ne sera visible que dans l'espace relatif à la boite entourant le contenu de l'élément.

Il convient cependant de ne pas confondre les propriétés background-clip et background-origin : la propriété background-origin sert à déterminer le point d'origine de l'image de fond, c'est-à-dire à partir d'où celle-ci va être placée tandis que background-clip va nous permettre de définir quelle partie du fond va être affichée / visible.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
        <div id="ex4"></div>
    </body>
</html>
```

```css
div{
    width: 400px;
    height: 100px;
    border: 5px dashed #333;
    padding: 10px;
    margin: 10px auto;
    background-color : RGBa(255, 255, 255, 0.2);
}

#ex1, #ex2, #ex3, #ex4{
    background-image: url("emoji-smile.png");
    background-repeat: no-repeat;
}
#ex1{
    background-origin : border-box;
    background-clip : border-box;
}
#ex2{
    background-origin : border-box;
    background-clip : content-box;
}
#ex3{
    background-origin : content-box;
    background-clip : border-box;
}
#ex4{
    background-origin : content-box;
    background-clip : content-box;
}
```



Comme vous pouvez le voir ici, le point d'origine de notre image de fond défini par la propriété `background-origin` se situe à l'extrémité des bordures de l'élément pour les `div id="ex1"` et `div id="ex2"`. Pour le `div id="ex2"`, cependant, on impose le fait que la partie du fond située entre la boite relative au contenu et la l'extrémité de la bordure ne soit pas affichée avec `background-clip : content-box`.

La propriété CSS raccourcie background

La propriété CSS `background` est la notation short hand ou notation raccourcie des propriétés liées au fond. En CSS 2, la propriété `background` permettait de définir une couleur et / ou une image de fond pour un élément HTML ainsi que le comportement de répétition, le défilement et la position du fond.

La propriété background était donc une propriété condensée des propriétés background-color, background-image, background-repeat, background-attachment et background-position.

En CSS 3, ses fonctionnalités ont été étendues pour refléter les apports du CSS 3 en termes d'outil de gestion du fond et la propriété background dans sa nouvelle définition permet aujourd'hui également de gérer plusieurs fonds d'un coup ainsi que de définir les comportements relatifs aux propriétés background-size, background-origin et background-clip pour un fond.

Attention cependant : même si la plupart des navigateurs supportent parfaitement ces ajouts, je vous rappelle que ce module du CSS 3 n'est pas encore reconnu comme recommandation officielle par le W3C mais est seulement candidat au statut de recommandation et est donc potentiellement sujet à modifications (même si elles sont peu probables).

L'ordre de déclaration des valeurs de la propriété background va être le suivant :

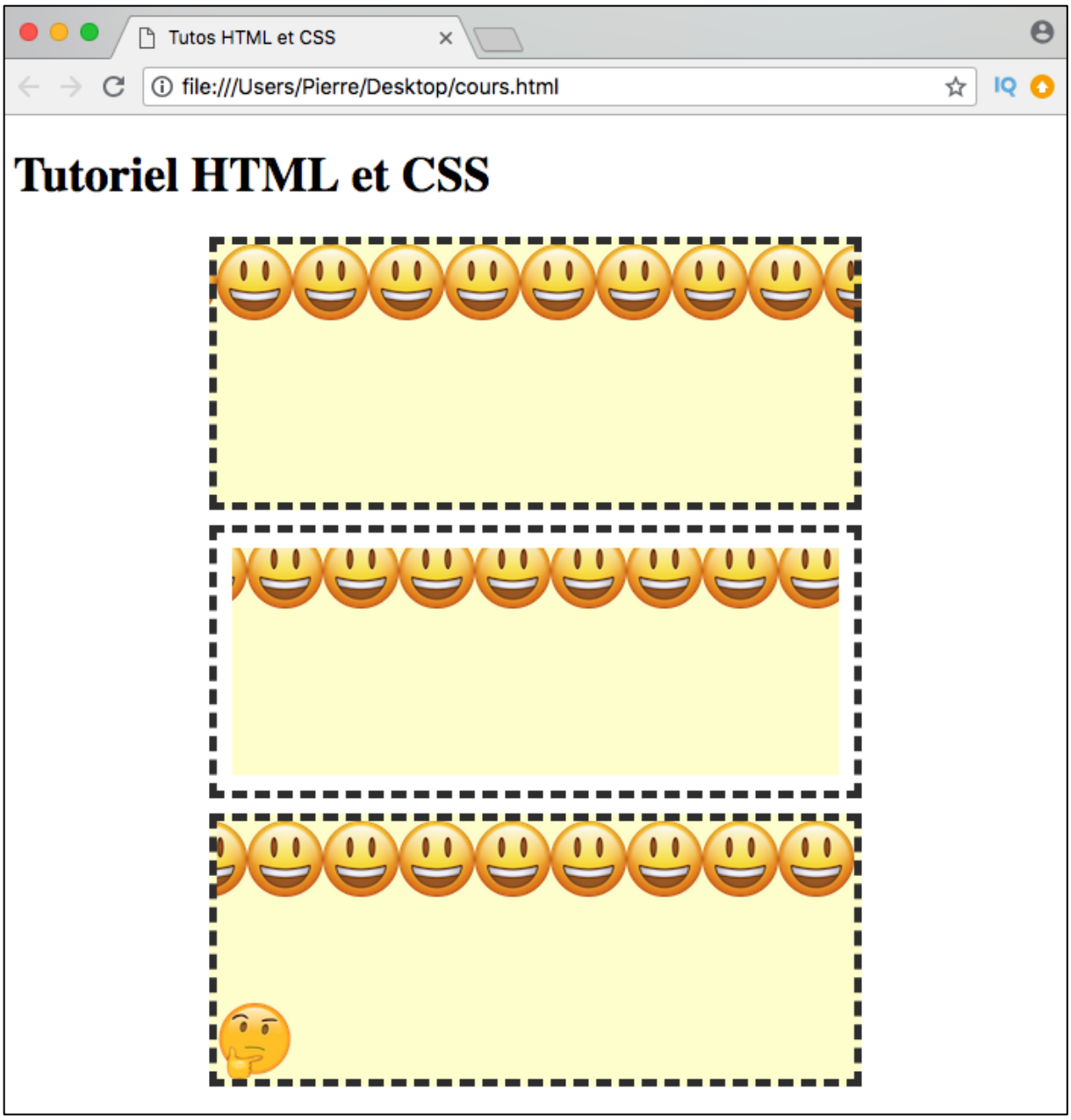
1. La valeur relative à la propriété background-image ;
2. Les valeurs relatives aux propriétés background-position / background-size (avec le slash entre les deux valeurs);
3. La valeur relative à la propriété background-repeat ;
4. La valeur relative à la propriété background-attachment ;
5. La valeur relative à la propriété background-origin ;
6. La valeur relative à la propriété background-clip ;
7. La valeur relative à la propriété background-color.

Notez que si certaines valeurs sont omises, alors les valeurs par défaut des différentes propriétés seront utilisées. Pour rappel, les valeurs par défaut de background vont être : background : none 0% 0% / auto auto repeat scroll padding-box padding-box transparent.

Pour déclarer plusieurs fonds, il suffira une nouvelle fois de séparer les déclarations complètes par une virgule. Attention cependant, pour que la propriété background fonctionne correctement avec des fonds multiples il faudra absolument réserver la valeur relative à la couleur de fond (background-color) pour le dernier fond déclaré.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
    </body>
</html>
```

```css
div{
    width: 400px;
    height: 150px;
    border: 5px dashed #333;
    padding: 10px;
    margin: 10px auto;
    background-color : RGBa(255, 255, 255, 0.2);
}

/*Une déclaration en omettant des valeurs*/
#ex1{
    background: url("emoji-smile.png") repeat-x RGBa(250,250,0,0.2);
}

/*Une déclaration complète*/
#ex2{
    background: url("emoji-smile.png") top right / 50px auto repeat-x padding-box content-box RGBa(250,250,0,0.2);
}

/*2 images de fond avec background*/
#ex3{
    background: url("emoji-smile.png") top right repeat-x padding-box,
                url("emoji-think.png") bottom left no-repeat RGBa(250,250,0,0.2);
}
```


Tutos HTML et CSS
file:///Users/Pierre/Desktop/cours.html
Tutoriel HTML et CSS

Créer des dégradés linéaires en CSS

En CSS, un dégradé est considéré comme une image qui va progressivement passer d'une couleur de base à une couleur d'arrivée.

Ainsi, nous allons pouvoir créer et utiliser les dégradés en CSS avec toutes les propriétés qui acceptent des images et notamment créer des dégradés en fond de nos éléments en utilisant la propriété `background`.

Il existe deux types de dégradés en CSS :

- Les dégradés linéaires ou `linear-gradient` en anglais dont le principe va être de passer progressivement d'une couleur à une autre de manière linéaire c'est-à-dire selon un axe donné ;
- Les dégradés radiaux ou `radial-gradient` en anglais pour lesquels le passage d'une couleur à une autre va se faire dans toutes les directions à partir d'un point central.

Dans cette leçon, nous allons nous intéresser à la création de dégradés linéaires seulement. Nous verrons comment créer des dégradés radiaux dans la leçon suivante.

Qu'est-ce qu'un dégradé linéaire ?

Un dégradé linéaire est un dégradé qui va se faire selon un axe ou une direction unique. Nous allons pouvoir créer un dégradé linéaire en CSS en utilisant la syntaxe `linear-gradient()` en valeur d'une propriété acceptant des images comme `background` par exemple.

Direction et couleurs du dégradé

Pour créer un dégradé linéaire de manière effective, il va falloir préciser un axe ou une direction pour notre dégradé ainsi qu'au moins deux couleurs et éventuellement définir des « color stops » qui vont nous permettre d'indiquer qu'à un certain point le dégradé doit passer par une couleur donnée.

Définition de la direction d'un dégradé linéaire

Nous allons pouvoir préciser la direction du dégradé de deux manières différentes : soit en utilisant une notation sous forme d'angle en degrés `deg`, soit avec les mots clefs `to top`, `to right`, `to bottom`, `to left` ou les combinaisons `to bottom right`, `to bottom left`, etc.

- Un angle de `0deg` indique que le dégradé se fera selon l'axe vertical à partir du bas vers le haut. L'équivalent de `0deg` sous forme de mot clef va être `to top` ;
- Un angle de `90deg` signifie que le dégradé se fera selon l'axe horizontal de la gauche vers la droite. L'équivalent de `90deg` sous forme de mot clef va être `to right` ;

- Un angle de `180deg` indique que le dégradé se fera selon l'axe vertical à partir du haut vers le bas. L'équivalent de `180deg` sous forme de mot clef va être `to bottom` ;
- Un angle de `270deg` signifie que le dégradé se fera selon l'axe horizontal de la droite vers la gauche. L'équivalent de `270deg` sous forme de mot clef va être `to left`.

De même, la valeur `to top right` va être équivalente à `45deg` : le dégradé partira d'en bas à gauche de l'élément pour aller en haut à droite et etc.

A noter : Utiliser des notations sous forme d'angles en degrés va nous permettre d'ajuster la direction de nos dégradés de manière beaucoup plus précise qu'en utilisant les mots clefs.

Notez que si on ne précise pas de direction pour un dégradé linéaire, alors la direction du dégradé sera par défaut verticale du haut vers le bas c'est-à-dire `to bottom` ou encore `180deg`.

Les "color stops" et l'avancement d'un dégradé

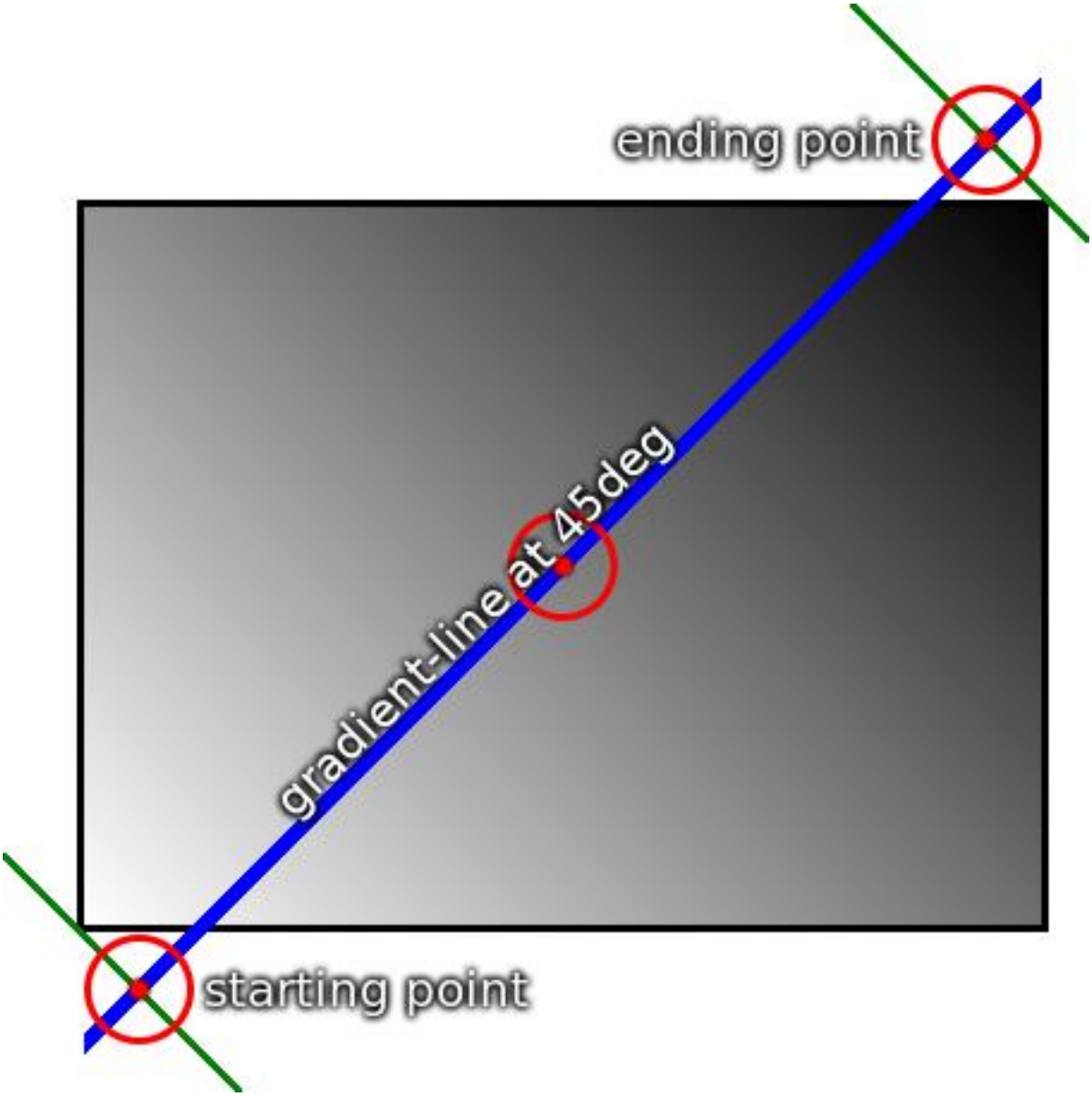
Les « color stops » vont être des points que nous allons définir au niveau desquels notre dégradé doit arriver à une couleur en particulier. Les « color stops » seront généralement situés entre le point de départ et le point d'arrivée de notre dégradé mais cela n'est pas obligatoire (on peut tout à fait définir un color stop en dehors de notre dégradé).

Pour définir un color stop, nous allons attribuer un pourcentage à une couleur. Ce pourcentage représente un niveau d'avancement du dégradé pour lequel il doit arriver à la couleur correspondante.

En effet, un dégradé linéaire se fait selon une direction. Vous pouvez imaginer cette direction sous forme d'une droite.

Le point de départ de notre dégradé linéaire ou le 0% va se trouver au point où la droite perpendiculaire à la direction du dégradé touche l'extrémité de la boite représentant l'élément dans lequel on crée notre dégradé et de manière identique pour le point d'arrivée de notre dégradé de l'autre côté de la boite.

Regardez plutôt le schéma suivant pour bien comprendre cela :



Exemples de création de dégradés linéaires

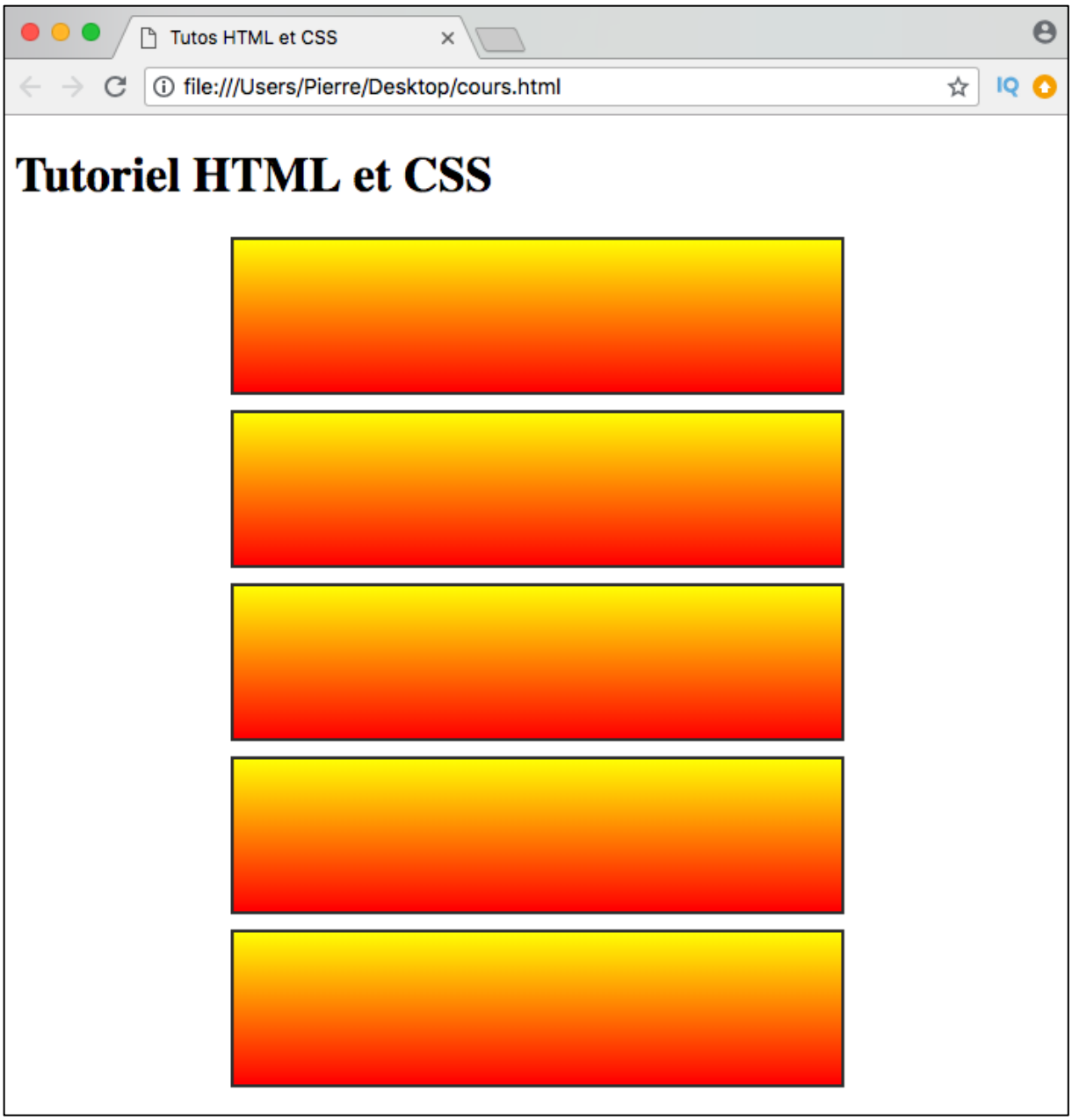
Maintenant que nous avons vu la théorie, il est temps de passer à la pratique et de créer nos premiers dégradés linéaires afin de voir comment cela va fonctionner en pratique. Dans les exemples suivants, je vais créer des dégradés linéaires en images de fond de mes éléments.

Création de dégradés linéaires simples

Commençons avec quelques exemples simples. Essayez de bien retenir la syntaxe des exemples suivants et notamment quand utiliser des virgules ou pas (pour faire simple, nous n'utilisons les virgules que pour séparer les couleurs entre elles et les séparer des autres valeurs à part des color stops qui leur sont associés).

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
        <div id="ex4"></div>
        <div id="ex5"></div>
    </body>
</html>
```

```css
div{
    width: 400px;
    height: 100px;
    border: 2px solid #333;
    margin: 10px auto;
}

#ex1{
    background:linear-gradient(yellow,red);
}
#ex2{
    background:linear-gradient(to bottom, yellow,red);
}
#ex3{
    background:linear-gradient(180deg, yellow,red);
}
#ex4{
    background:linear-gradient(to bottom, yellow 0%,red 100%);
}
#ex5{
    background:linear-gradient(to top, red, yellow);
}
```



Ici, mes cinq dégradés sont strictement identiques. En effet, je crée un dégradé en image de fond avec la propriété `background` et la notation `linear-gradient` pour mon premier `div id="ex1"` en renseignant une couleur de départ, le jaune (yellow), et une couleur d'arrivée, le rouge (red).

Dans ce premier exemple, je n'indique ni direction ni color stop. Les valeurs par défaut vont donc être utilisées à savoir `to bottom` ou `180deg` pour la direction et `0%` en color stop pour ma couleur de départ et `100%` pour ma couleur d'arrivée.

Dans les dégradés suivants, je me contente de renseigner des valeurs qui sont ici superflues puisque ces valeurs sont les valeurs par défaut.

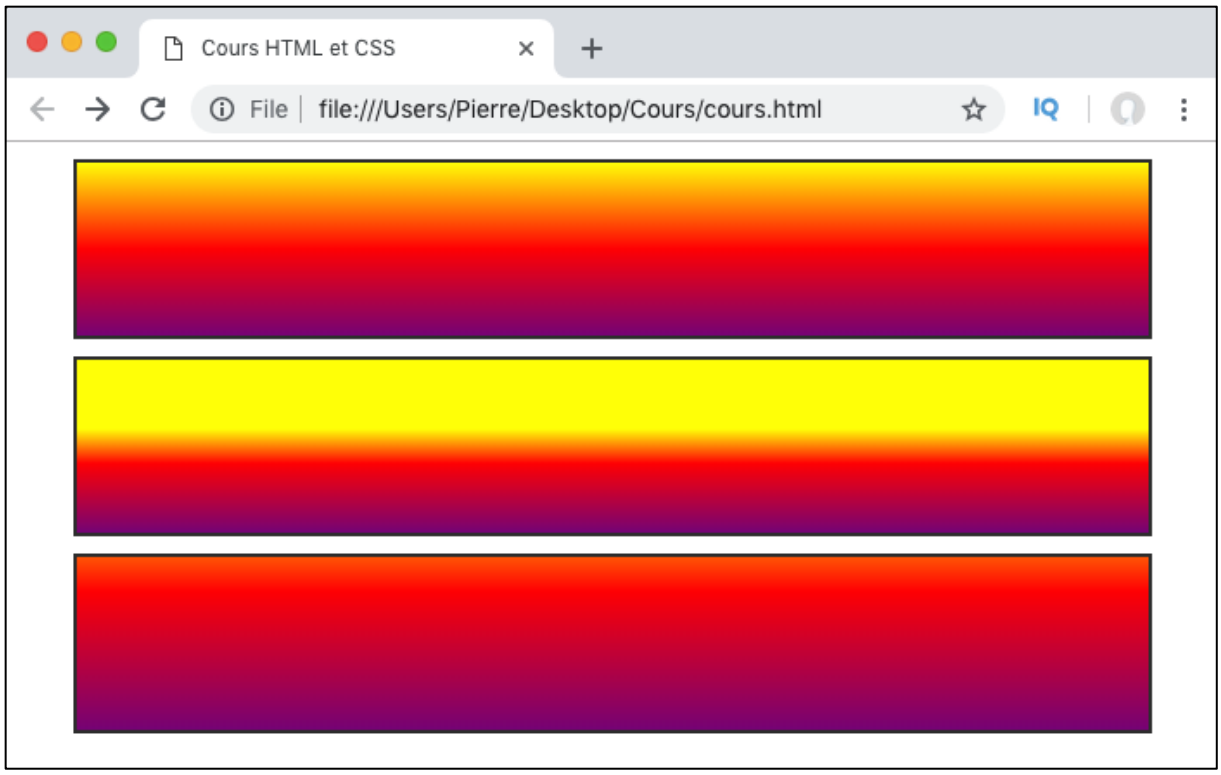
Notez que dans le cinquième exemple j'inverse à la fois la direction du dégradé et les couleurs de départ et d'arrivée ce qui fait que nous nous retrouvons une nouvelle fois avec le même dégradé !

Création de dégradés avec direction et color stops

Essayons maintenant de créer des dégradés linéaires avec plus de deux couleurs et en définissant des directions et color stops différents :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 100px;
    border: 2px solid #333;
    margin: 10px auto;
}

/*Un dégradé linéaire à 3 couleurs*/
#ex1{
    background:linear-gradient(yellow, red, purple);
}

/*Color stops personnalisés*/
#ex2{
    background:linear-gradient(yellow 40%, red 60%, purple);
}
#ex3{
    background:linear-gradient(yellow -40%, red 20%, purple);
}
```



Dans le premier exemple, on définit un dégradé linéaire à trois couleurs : du jaune vers le rouge puis vers le violet sans préciser de color stop. La couleur jaune sera ainsi par défaut atteinte à 0% du dégradé puis le rouge à 50% et enfin le violet à 100% puisque les color stops sont distribués équitablement par défaut (dans le cas de 4 couleurs, les color stops par défaut seraient 0% 33.33% 66.66% 100%, dans le cas d'un dégradé à 5 couleurs on aurait 0% 25% 50% 75% 100% et etc.).

Dans les deuxième et troisième exemples, en revanche, on définit des color stop personnalisées. Dans le deuxième exemple, on demande à ce que la couleur jaune soit atteinte à 40% du dégradé. Comme le jaune est également la couleur de départ, notre dégradé sera jaune entre 0 et 40%.

Ensuite, on demande à ce que la couleur rouge soit atteinte à 60%. Il va donc y avoir dégradé du jaune vers le rouge entre 40% et 60% de notre dégradé puis ensuite du rouge vers le violet entre 60% et 100%.

Dans le troisième exemple, on utilise un color stop en dehors de notre de la boite. On demande à ce que notre dégradé arrive à la couleur jaune à -40%. Cela signifie que si notre ligne de dégradé dans la boite de l'élément fait 100px de long, on souhaite atteindre le jaune 40px avant notre 0%.

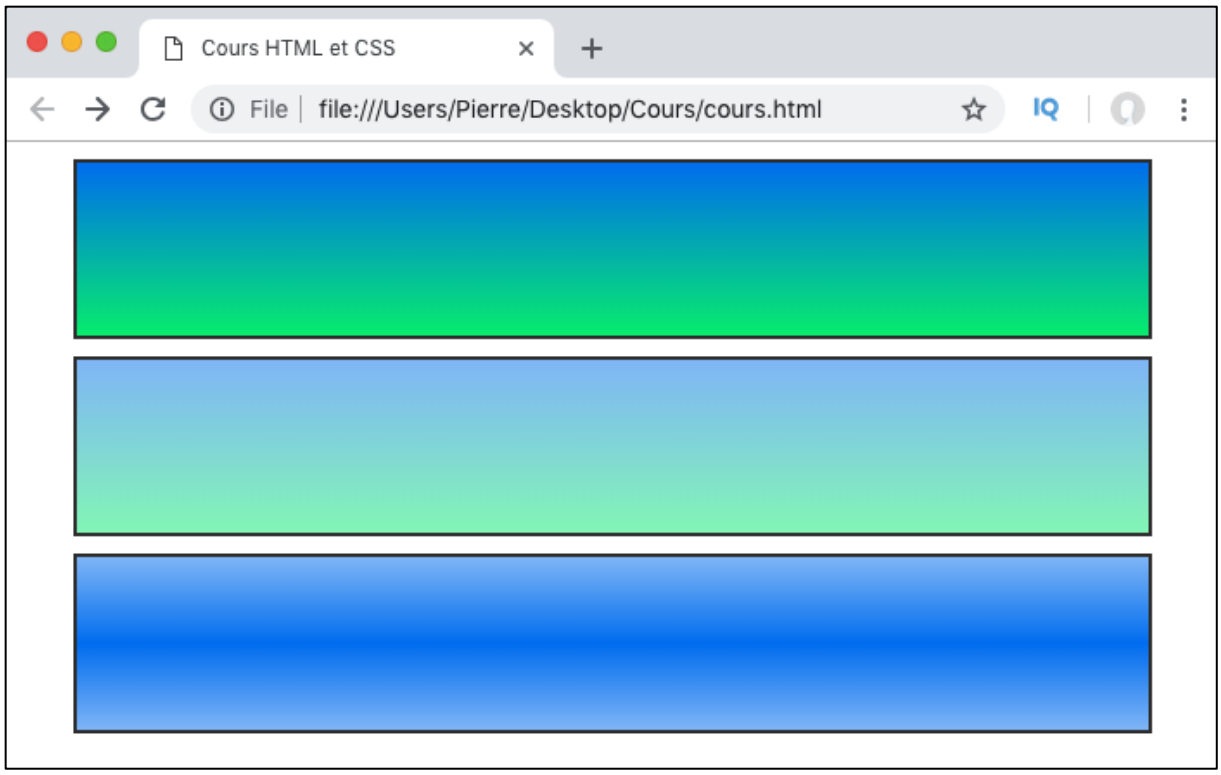
Ensuite on demande à arriver au rouge à 20%. La transition du jaune vers le rouge va donc se faire entre -40% et 20% et donc nous n'allons pas voir le jaune en soi puisque tous les color stops inférieurs à 0% ou supérieurs à 100% se situent en dehors de la boite représentant l'élément dans lequel on crée notre dégradé.

Création de dégradés semi-transparents

Nous pouvons tout à fait définir des dégradés semi-transparents en utilisant des notations RGBa ou HSLa pour préciser les couleurs de nos dégradés. Notez par ailleurs qu'on peut très bien définir le début d'un dégradé avec une couleur semi transparente et la fin de celui-ci avec une couleur opaque ou inversement.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 100px;
    border: 2px solid #333;
    margin: 10px auto;
}

/*Un dégradé linéaire bleu -> vert*/
#ex1{
  background:linear-gradient(RGB(0,120,240), RGB(0,240,120));
}

/*Un dégradé linéaire bleu -> vert semi transparent*/
#ex2{
    background:linear-gradient(RGBa(0,120,240,0.5), RGBa(0,240,120,0.5));
}

/*Dégradé bleu semi transparent au début/ opaque au milieu/ semi transparent à la fin*/
#ex3{
    background:linear-gradient(RGBa(0,120,240,0.5),RGB(0,120,240),RGBa(0,120,240,0.5));
}
```



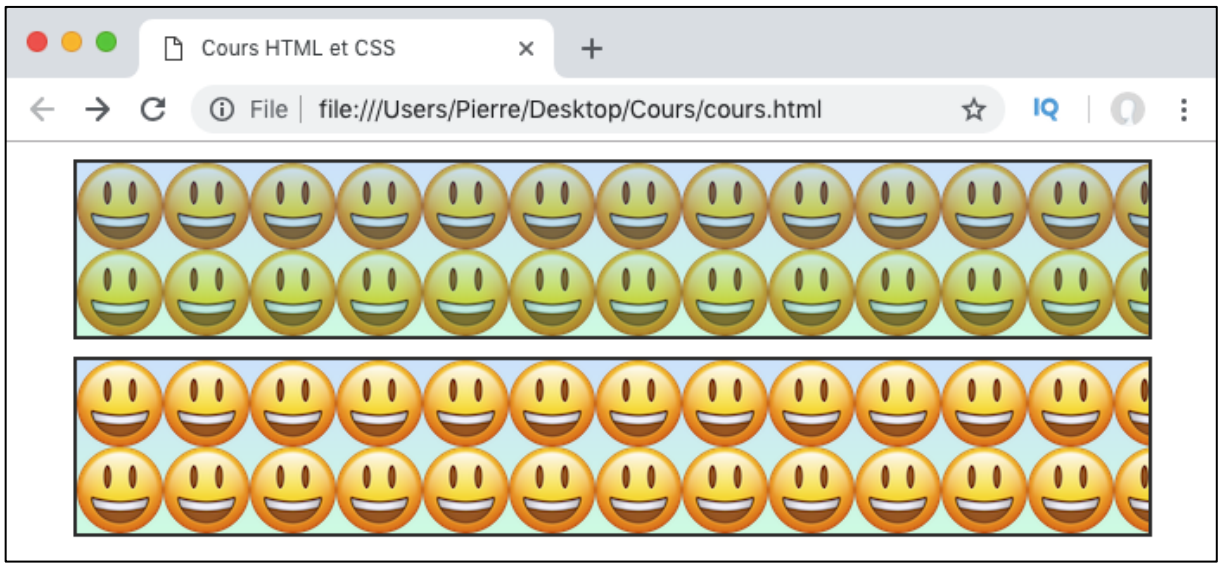
Dans l'exemple ci-dessus, on crée trois dégradés : un premier dégradé complètement opaque, un deuxième dégradé avec des couleurs semi transparentes en utilisant des notations RGBa et un troisième dégradé semi transparent au début, opaque au milieu puis à nouveau semi transparent à la fin.

Définir des dégradés semi-transparents va être véritablement intéressant dans le cas où on souhaite simplement ajouter un effet de couleur par-dessus une image de fond par exemple. En effet, rappelez-vous qu'un dégradé est considéré comme une image en CSS et que la propriété `background` supporte tout à fait plusieurs images de fond.

Ici, il va bien falloir faire attention à l'ordre des déclarations : je vous rappelle que le premier fond déclaré sera toujours au-dessus du deuxième, qui sera lui-même au-dessus du troisième et etc. Ainsi, en déclarant notre dégradé en fond avant notre image, nous allons pouvoir ajouter des couleurs devant notre image de fond.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div id="ex1"></div>
        <div id="ex2"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 100px;
    border: 2px solid #333;
    margin: 10px auto;
}

/*Un dégradé linéaire semi transparent devant une image de fond*/
#ex1{
  background:linear-gradient(RGBa(0,120,240,0.2), RGBa(0,240,120,0.2)), url("smile.png");
}

/*L'inverse ! (image de fond devant le dégradé)*/
#ex2{
  background:url("smile.png"),linear-gradient(RGBa(0,120,240,0.2), RGBa(0,240,120,0.2));
}
```



Créer plusieurs dégradés en fond d'un élément

Nous venons de voir qu'on pouvait tout à fait placer un dégradé et une image en fond d'un même élément. De la même façon, on va pouvoir déclarer plusieurs dégradés en fond d'un élément.

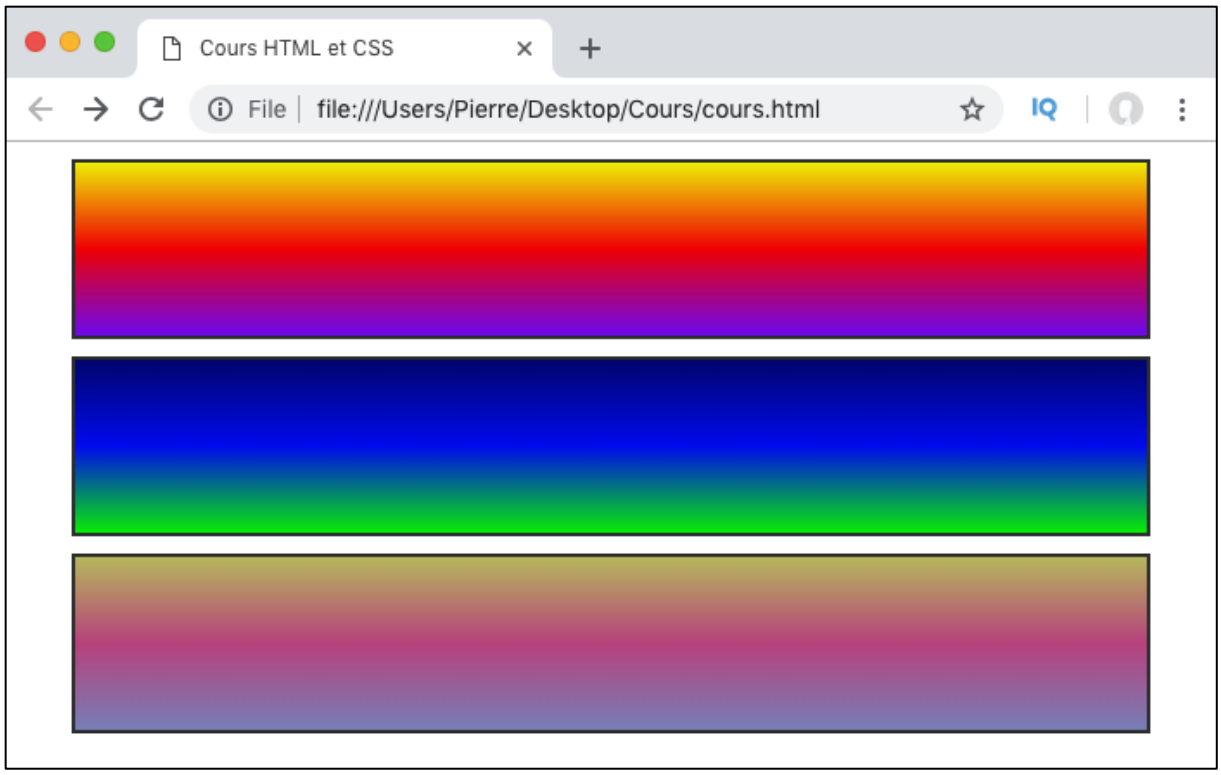
Pour affecter plusieurs dégradés linéaires à notre `background`, il va suffire de séparer les différentes déclarations par une virgule de la même façon qu'on séparait les images de fond.

Le premier dégradé déclaré sera au-dessus du deuxième qui sera lui-même au-dessus du troisième et etc., chaque nouvelle déclaration créant une nouvelle couche de fond pour l'élément concerné. Bien évidemment, si les dégradés sont opaques, alors seul le premier déclaré sera visible (il cachera totalement les autres).

En revanche, si le ou les dégradés du dessus sont semi transparents, alors on va pouvoir voir également les dégradés des couches plus profondes et les couleurs des différents dégradés vont se mélanger ou « fusionner » entre elles.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 100px;
    border: 2px solid #333;
    margin: 10px auto;
}

/*Dégradé n°1 : jaune-> rouge-> violet*/
#ex1{
    background:linear-gradient(RGB(240,240,0), RGB(240,0,0), RGB(120,0,240));
}

/*Dégradé n°2 : bleu foncé-> bleu-> vert*/
#ex2{
    background:linear-gradient(RGB(0,0,120), RGB(0,0,240), RGB(0,240,0));
}

/*Mélange des deux dégradés précédents*/
#ex3{
    background:
      linear-gradient(RGBa(240,240,0,0.5), RGBa(240,0,0,0.5), RGBa(120,0,240,0.5)),
      linear-gradient(RGBa(0,0,120,0.5), RGBa(0,0,240,0.5), RGBa(0,240,0,0.5));
}
```



La répétition des dégradés

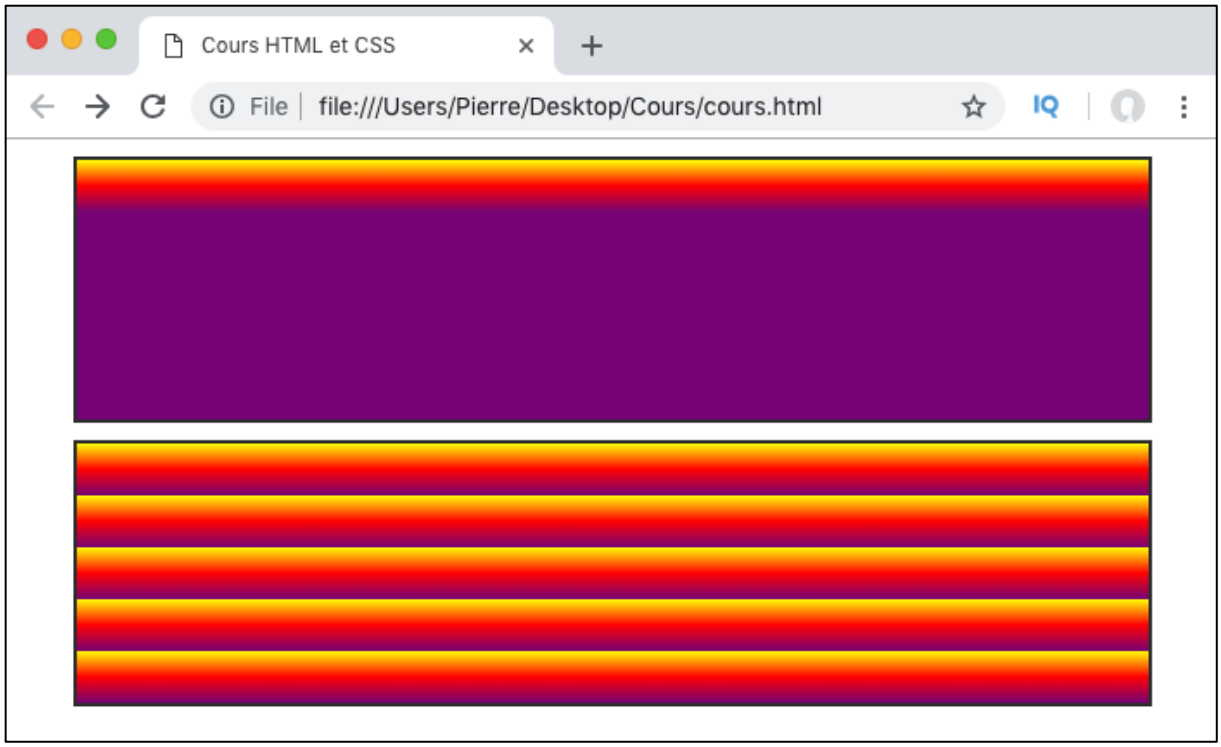
Nous avons vu comment créer des dégradés uniques linéaires avec la fonction `linear-gradient()`.

Le CSS nous offre également la possibilité de créer des dégradés qui vont pouvoir se répéter grâce en utilisant cette fois-ci plutôt la fonction `repeating-linear-gradient()`.

Bien évidemment, pour voir l'effet d'un dégradé répété il faudra régler les color stops de façon à ce que la première itération du dégradé n'occupe pas 100% de l'espace de base.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div id="ex1"></div>
        <div id="ex2"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 150px;
    border: 2px solid #333;
    margin: 10px auto;
}

#ex1{
    background: linear-gradient(yellow,red 10%,purple 20%);
}
#ex2{
    background: repeating-linear-gradient(yellow,red 10%,purple 20%);
}
```



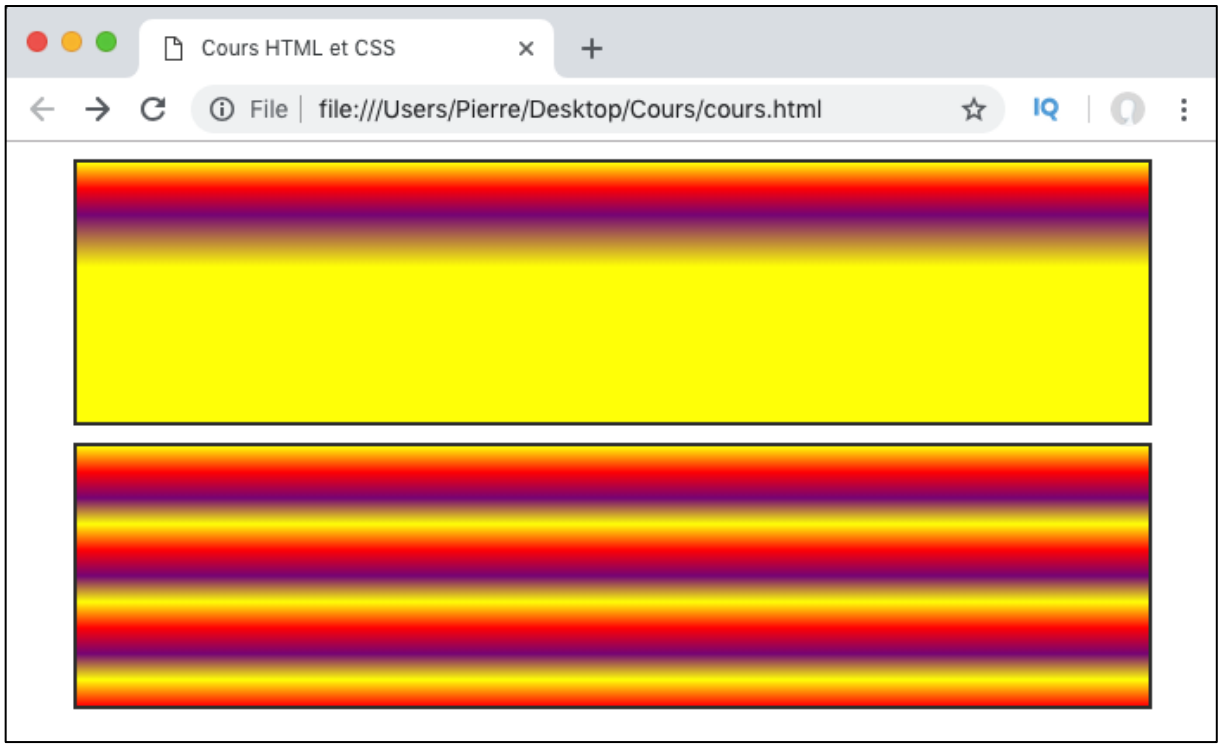
Ici, vous pouvez voir que la transition entre la première et la deuxième itération du dégradé est brutale lors de l'utilisation de `repeating-linear-gradient()`.

Ce comportement est tout à fait normal puisqu'ici le color stop relatif au violet est en fin de dégradé et le color stop du jaune est au tout début. On va donc passer directement du violet au jaune sans transition fluide.

Si on souhaite créer une transition plus fluide entre les différentes itérations de notre dégradé, il faudra créer le dégradé de manière à ce qu'il boucle sur lui-même c'est-à-dire qu'il faudra répéter la couleur de début de dégradé en fin de dégradé.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div id="ex1"></div>
        <div id="ex2"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 150px;
    border: 2px solid #333;
    margin: 10px auto;
}

#ex1{
    background: linear-gradient(yellow,red 10%,purple 20%,yellow 40%);
}
#ex2{
    background: repeating-linear-gradient(yellow,red 10%,purple 20%,yellow 30%);
}
```



Créer des dégradés radiaux en CSS

En CSS, un dégradé est considéré comme une image qui va progressivement passer d'une couleur de base à une couleur d'arrivée.

Ainsi, nous allons pouvoir créer et utiliser les dégradés en CSS avec toutes les propriétés qui acceptent des images et notamment créer des dégradés en fond de nos éléments en utilisant la propriété `background`.

Nous allons pouvoir créer deux types de dégradés en CSS : des dégradés linéaires et des dégradés radiaux. Dans cette nouvelle leçon, nous allons apprendre à créer des dégradés radiaux.

Qu'est-ce qu'un dégradé radial ?

Un dégradé radial est un dégradé qui va partir d'un point central et se propager dans toutes les directions à partir de ce point.

Pour créer un dégradé radial, nous allons utiliser la notation `radial-gradient()` en valeur d'une propriété CSS acceptant des images comme `background` par exemple.

Comment se construit un dégradé radial en CSS

Pour créer un dégradé radial ou `radial-gradient`, nous allons devoir définir le point central ou point de départ de notre dégradé, les couleurs du dégradé ainsi que la taille de la forme finale de celui-ci. Nous allons également pouvoir fournir des color stops de la même façon qu'avec nos dégradés linéaires.

Le point d'origine ou point central du dégradé

Nous avons deux méthodes pour définir le point d'origine ou point central de notre `radial-gradient` : nous allons soit pouvoir utiliser des mots clefs, soit des valeurs absolues ou relatives. A noter que si on omet de préciser la position de départ pour notre dégradé radial, alors la position par défaut `center` sera utilisée.

Les mots clefs vont pouvoir être les mêmes qu'avec nos dégradés linéaires, à savoir `top`, `right`, `bottom`, `left` et `center` mais devrons être cette fois-ci être précédés du mot `at`.

Pour préciser les coordonnées d'un point, il faut normalement une valeur dans l'axe horizontal et une autre dans l'axe vertical et c'est la raison pour laquelle nous utiliserons souvent des mots clefs composées comme `at top left`. Si on ne précise que la valeur d'un axe (`at top` ou `at left` par exemple), alors la deuxième valeur calculée sera par défaut `center`.

Si on choisit d'utiliser des valeurs absolues ou relatives pour déterminer le point de départ de notre dégradé radial, alors il suffit de se rappeler que ces valeurs vont représenter

l'écartement du point central du dégradé par rapport au coin supérieur gauche de son élément. Dans le cas où une seule valeur est passée, elle représentera l'écartement par rapport au bord gauche et par rapport au bord supérieur.

Si deux valeurs sont passées, la première représente l'écartement du point central par rapport au bord gauche tandis que la seconde représente l'écartement du point central du dégradé radial par rapport au coin supérieur de l'élément dans lequel on le crée.

La forme finale du dégradé

Pour définir la forme finale d'un dégradé radial, nous allons pouvoir utiliser les mots clefs `circle` (le dégradé aura une forme de cercle) soit `ellipse` (le dégradé sera une ellipse). Si aucune valeur n'est précisée, le dégradé sera un cercle dans le cas où une seule dimension a été précisée pour sa taille ou une ellipse dans le cas contraire.

La taille du dégradé

Nous allons également pouvoir indiquer la **taille** totale de notre dégradé radial soit de manière explicite en utilisant des valeurs soit en utilisant des mots clefs. La « taille du dégradé » correspond à la distance (ou aux distances) entre son point central et l'endroit où l'on va arriver à la dernière couleur déclarée dans le dégradé. Le reste de l'élément si le dégradé en soi possède une taille plus petite que lui sera rempli avec cette dernière couleur.

En ne passant qu'une valeur explicite, notre dégradé aura de fait la forme d'un cercle et la valeur passée sera la taille du rayon du dégradé. Si deux valeurs sont passées, la première représentera le « rayon » de l'ellipse dans l'axe horizontal tandis que la seconde représentera le « rayon » de l'ellipse dans l'axe vertical. Bien évidemment, on évitera de déclarer notre dégradé comme `circle` et de passer deux valeurs différentes en taille puisque cela n'aurait aucun sens.

Les mots clefs relatifs à la gestion de la taille d'un dégradé radial à notre disposition sont les suivants :

- `closest-side` : Le dégradé va stopper dès qu'il va rencontrer le premier bord (le bord le plus proche) de l'élément auquel il est appliqué si il a une forme de cercle. S'il a une forme d'ellipse, alors il s'arrêtera au moment où il va toucher les deux côtés les plus proches dans les deux axes horizontal et vertical (il les touchera toujours en même temps par définition) ;
- `farthest-side` : Le principe est le même que pour `closest-side`, à la différence que le dégradé va stopper lorsqu'il touchera le bord le plus lointain de l'élément auquel on l'applique cette fois-ci ;
- `closest-corner` : Le dégradé va stopper dès qu'il va rencontrer le premier coin (le coin le plus proche) de l'élément auquel il est appliqué ;
- `farthest-corner` : Même principe que pour `closest-corner` à la différence que la dégradé ne stoppera qu'une fois qu'il aura atteint l'angle le plus lointain de l'élément auquel on l'applique.

Notez que si on ne précise pas de taille pour notre dégradé radial alors c'est la valeur `farthest-corner` qui sera utilisée par défaut.

Couleurs et color stops du dégradé

Finalement, par rapport à la position des color stops, notre `0%` correspond au point central ou point d'origine du dégradé radial tandis que `100%` va correspondre à l'endroit où le dégradé se termine.

Exemples de création de dégradés radiaux

Dans les exemples suivants, je vais créer des dégradés radiaux que je vais placer en fond de mes éléments. Avant tout, je vous rappelle les valeurs par défaut qui seront appliquées au dégradé si elles sont omises :

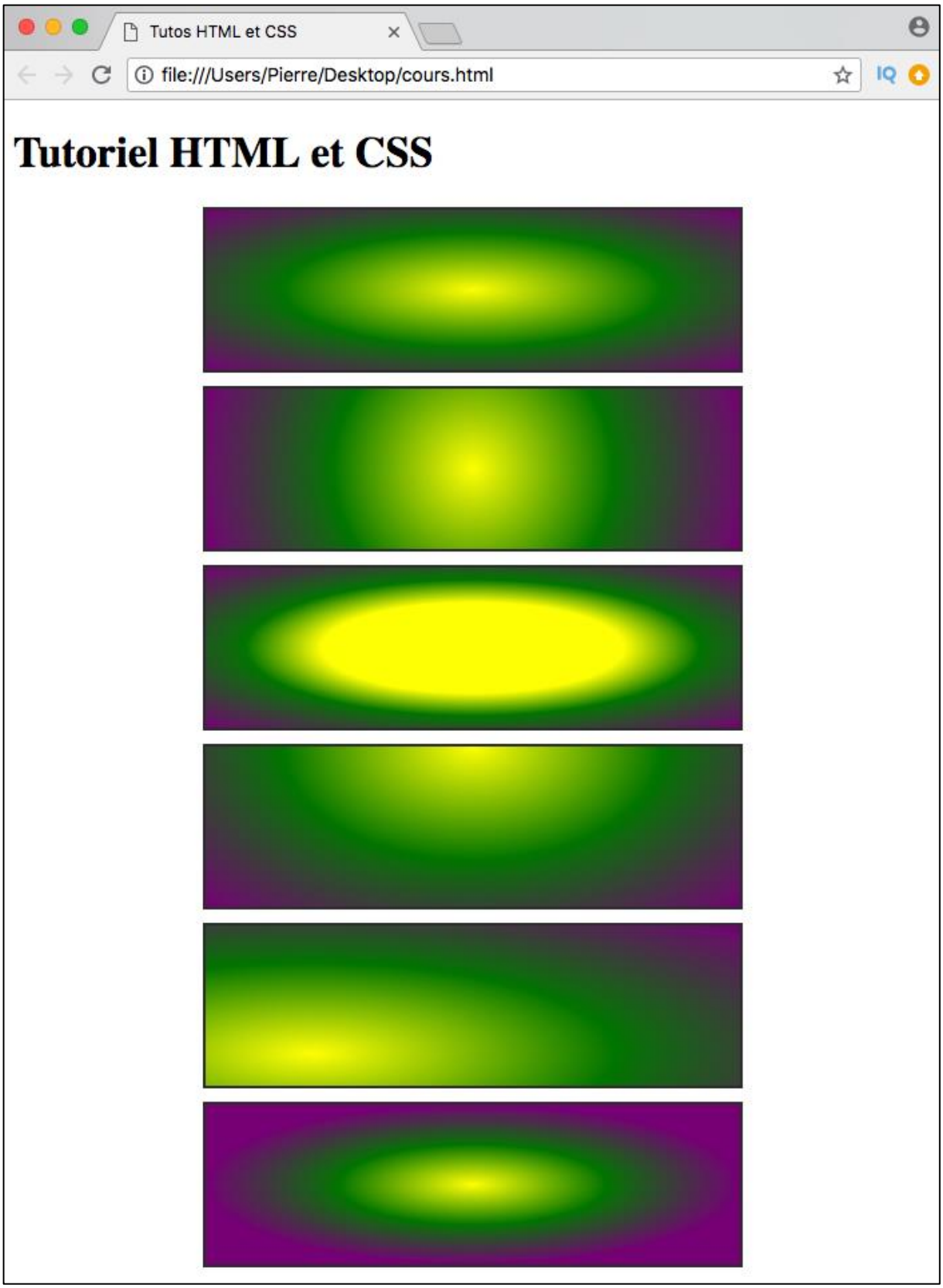
- Forme du dégradé : ellipse par défaut sauf si une seule valeur (ou deux valeurs identiques) ont été précisées en taille du dégradé ;
- Position du point central : center (centré) dans l'élément par défaut ;
- Taille du dégradé : farthest-corner par défaut, ce qui signifie que la fin du dégradé va venir toucher le coin de l'élément le plus éloigné de son centre.

Création de dégradés radiaux simples

Commençons avec quelques exemples simples de création de dégradés radiaux avec `radial-gradient`. Prenez le temps une nouvelle fois de faire l'effort de retenir la syntaxe qui ressemble à celle déjà utilisée avec les dégradés linéaires.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
        <div id="ex4"></div>
        <div id="ex5"></div>
        <div id="ex6"></div>
    </body>
</html>
```

```css
div{
    width: 400px;
    height: 120px;
    border: 2px solid #333;
    margin: 10px auto;
}

#ex1{
    background:radial-gradient(yellow, green, purple);
}
#ex2{
    background:radial-gradient(circle, yellow, green, purple);
}
#ex3{
    background:radial-gradient(yellow 40%, green 60%, purple);
}
#ex4{
    background:radial-gradient(at top, yellow, green, purple);
}
#ex5{
    background:radial-gradient(at 20% 80%, yellow, green, purple);
}
#ex6{
    background:radial-gradient(closest-side, yellow, green, purple);
}
```



Dans le premier exemple, on crée un dégradé radial d'arrière-plan avec `background : radial-gradient(yellow, green, purple)` pour notre premier `div`. Comme on ne précise que les couleurs du dégradé, notre dégradé sera par défaut une ellipse dont le point central

est centré dans l'élément et qui va aller toucher le coin le plus éloigné de l'élément (les quatre coins en l'occurrence vu que notre dégradé est centré).

Dans le deuxième exemple, on utilise le mot clef `circle` pour que notre dégradé ait une forme de cercle. Comme la taille de celui-ci est toujours réglée sur `farthest-corner` et comme notre élément est plus large que haut, les parties haute et basse du dégradé vont être rognées.

Dans notre troisième exemple, on utilise des color stops juste pour vous montrer que ceux-ci vont fonctionner exactement de la même façon qu'avez les dégradés linéaires à la différence qu'ici notre dégradé a une forme d'ellipse. Pour bien calculer les color stops, il va falloir visualiser le « rayon » du dégradé. Dans le cas présent vous pouvez imaginer une ligne partant du centre de l'élément (puisque notre dégradé est centré par défaut) et allant toucher un de ses coins (puisque la taille du dégradé est par défaut `farthest-side` et qu'il est centré).

Dans nos quatrième et cinquième exemples, on définit des points de départ ou point centraux pour nos dégradés personnalisés. Dans le quatrième exemple, on indique simplement `top`. Le point central du dégradé va donc se trouver contre la bordure supérieure de l'élément et être centré par défaut en largeur. Dans le cinquième exemple, on précise deux coordonnés pour définir un point d'origine totalement personnalisé avec `at 20% 80%`. Le point central du dégradé sera donc situé à une distance de 20% de la bordure gauche de l'élément et à 80% de la bordure haute. Les pourcentages donnés ici sont exprimés en fonction de la taille de la boite représentant l'élément.

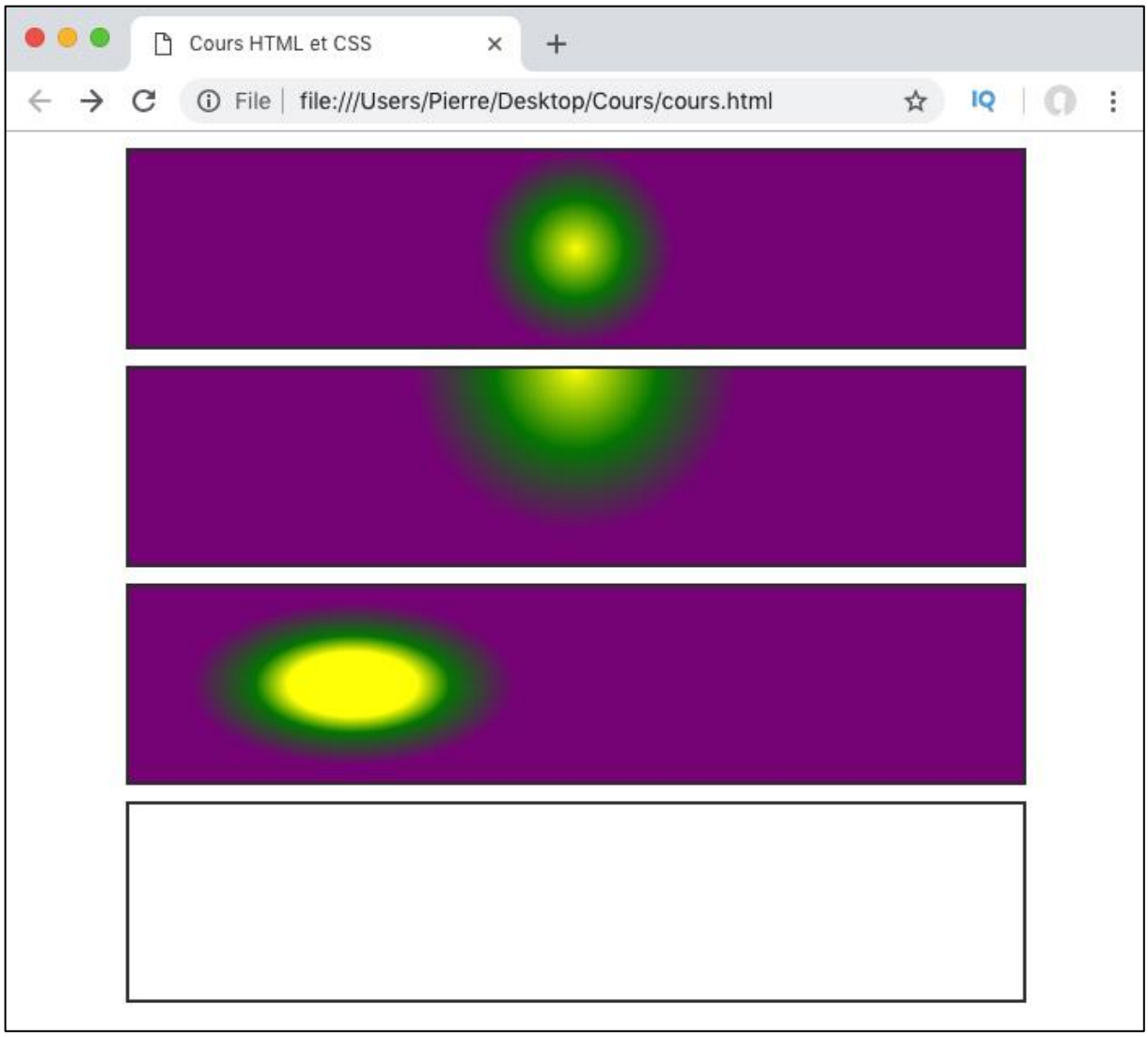
Dans le dernier exemple, on indique la valeur `closest-side` en taille de notre dégradé ce qui va le contraindre dans notre boite puisque dès qu'il va toucher un bord de l'élément il va s'arrêter.

Création de dégradés radiaux complexes

Nous allons également bien évidemment pouvoir créer des dégradés radiaux plus complexes et complets en définissant à la fois leur taille totale, la position de leur point central, la forme de ceux-ci etc. Faites juste attention ici à ne pas écrire de déclarations aberrantes (par exemple tenter d'imposer une forme de cercle à un dégradé radial tout en lui passant deux valeurs de taille différentes).

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
        <div id="ex4"></div>
    </body>
</html>
```

```css
div{
    width: 80%;
    height: 120px;
    border: 2px solid #333;
    margin: 10px auto;
}

#ex1{
    background:radial-gradient(closest-side circle, yellow, green, purple);
}
#ex2{
    background:radial-gradient(100px circle at top, yellow, green, purple);
}
#ex3{
    background:radial-gradient(100px 50px at 25% 50%, yellow 40%, green 60%, purple);
}

/*Déclaration aberrante*/
#ex4{
    background:radial-gradient(100px 50px circle at 25% 50%, yellow, green, purple);
}
```



Ici, notre premier dégradé radial a une forme de cercle et un rayon d'une taille égale à la distance entre le point central du dégradé et le bord de l'élément le plus proche de ce point central.

Notre deuxième dégradé a un rayon de 100px et on place le point central du dégradé contre le bord supérieur de l'élément. Comme on ne précise pas de position dans l'axe horizontal, le dégradé sera centré horizontalement par défaut.

On donne deux dimensions (largeur / hauteur du « rayon ») différentes à notre troisième dégradé qui va donc de fait prendre la forme d'une ellipse. On place également le point central à une distance de la bordure gauche de l'élément égale à 25% de la largeur de la boite de l'élément et à une distance de la bordure haute de l'élément égale à 50% de la hauteur de la boite de l'élément.

Dans notre dernier `div`, on tente de créer un dégradé avec un « rayon » de 100px de large pour 50px de haut tout en lui imposant une forme de cercle. Cela n'est évidemment pas possible et va résulter en une erreur et le dégradé ne sera pas affiché.

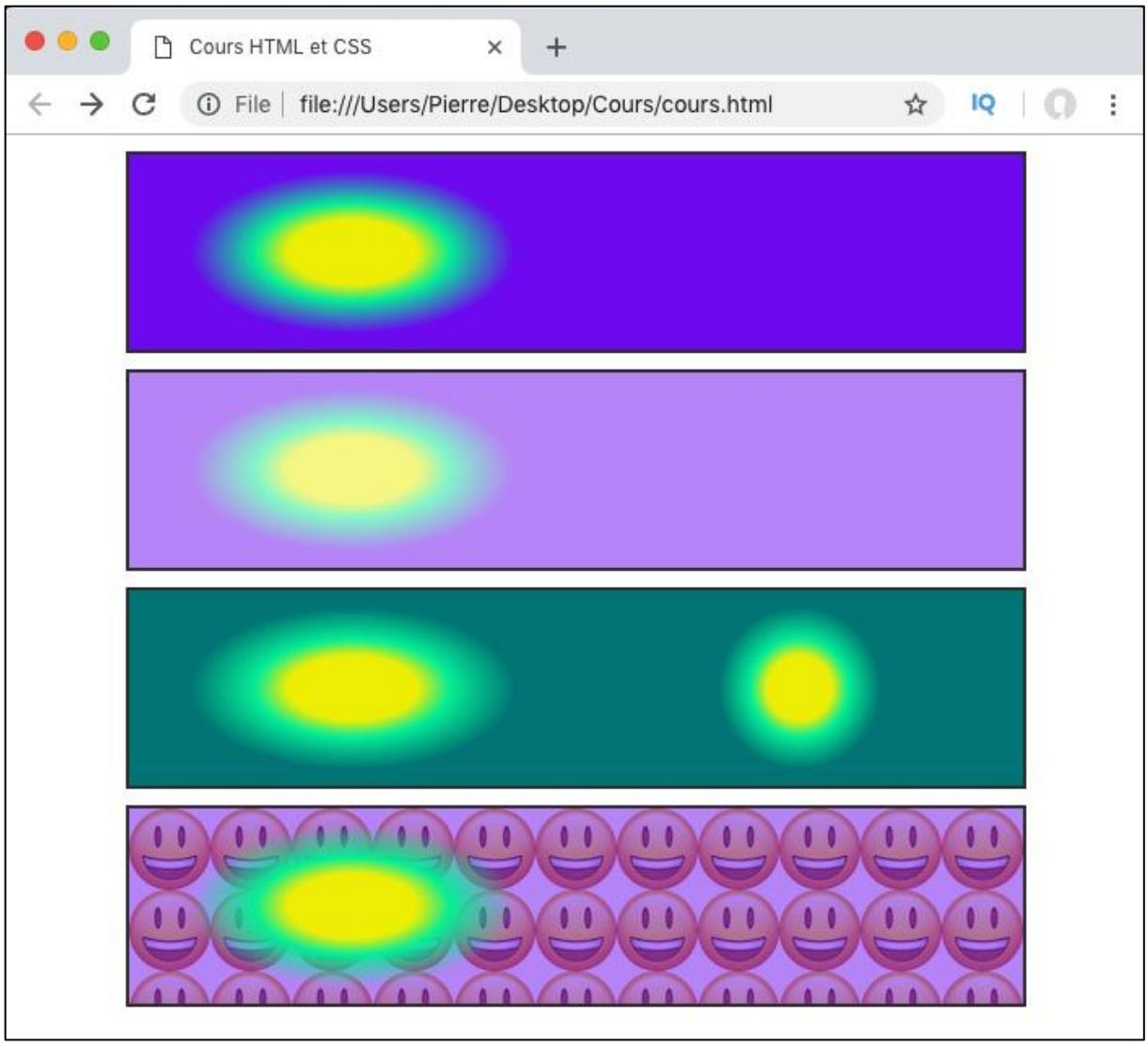
Dégradés radiaux multiples et semi-transparents

La propriété `background` est tout à fait capable de gérer des fonds multiples. Nous allons donc pouvoir placer plusieurs dégradés en fond de nos éléments et les faire fusionner entre eux en utilisant des notations `RGBa` ou `HSLa()` qui vont nous permettre de rendre nos dégradés semi transparents.

Nous allons également de la même manière pouvoir placer un dégradé radial et une image en fond d'un élément. Je vous rappelle ici que le premier élément de fond déclaré en CSS sera au-dessus du second et etc.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
        <div id="ex4"></div>
    </body>
</html>
```

```
div{
    width: 80%;
    height: 120px;
    border: 2px solid #333;
    margin: 10px auto;
}

#ex1{
  background:radial-gradient(100px 50px at 25% 50%,
                             RGB(240,240,0) 40%,
                             RGB(0,240,160) 60%,
                             RGB(120,0,240));
}
#ex2{
  background:radial-gradient(100px 50px at 25% 50%,
                             RGBa(240,240,0,0.5) 40%,
                             RGBa(0,240,160,0.5) 60%,
                             RGBa(120,0,240,0.5));
}
#ex3{
    background:radial-gradient(100px 50px at 25% 50%,
                               RGB(240,240,0) 40%,
                               RGB(0,240,160) 60%,
                               RGBa(120,0,240,0)),
               radial-gradient(50px at 75% 50%,
                               RGB(240,240,0) 40%,
                               RGB(0,240,160) 60%,
                               teal);
}
#ex4{
    background:radial-gradient(100px 50px at 25% 50%,
                               RGB(240,240,0) 40%,
                               RGB(0,240,160) 60%,
                               RGBa(120,0,240,0.5)),
               url("smile.png");
}
```



Notre premier dégradé est ici totalement opaque. Le dégradé de notre deuxième `div` utilise les mêmes paramètres que celui du premier à la différence que nous utilisons des notations `RGBa()` et qu'on lui attribue des couleurs semi transparentes.

Ensuite, on place deux dégradés radiaux en fond de notre troisième `div`. Afin qu'on puisse voir les deux, ici, j'opte pour un premier dégradé d'une taille relativement petite et utilise les notations `RGBa()` en déclarant une couleur d'arrivée de mon dégradé totalement transparente pour ne pas « polluer » le dégradé du dessous.

Je décale suffisamment mon deuxième dégradé dans mon élément afin qu'il ne se trouve pas au même endroit que le premier. C'est la couleur de fin du deuxième dégradé qui va remplir le `div`.

Finalement, on utilise un dégradé radial en semi-transparence avec une image de fond pour notre quatrième `div`.

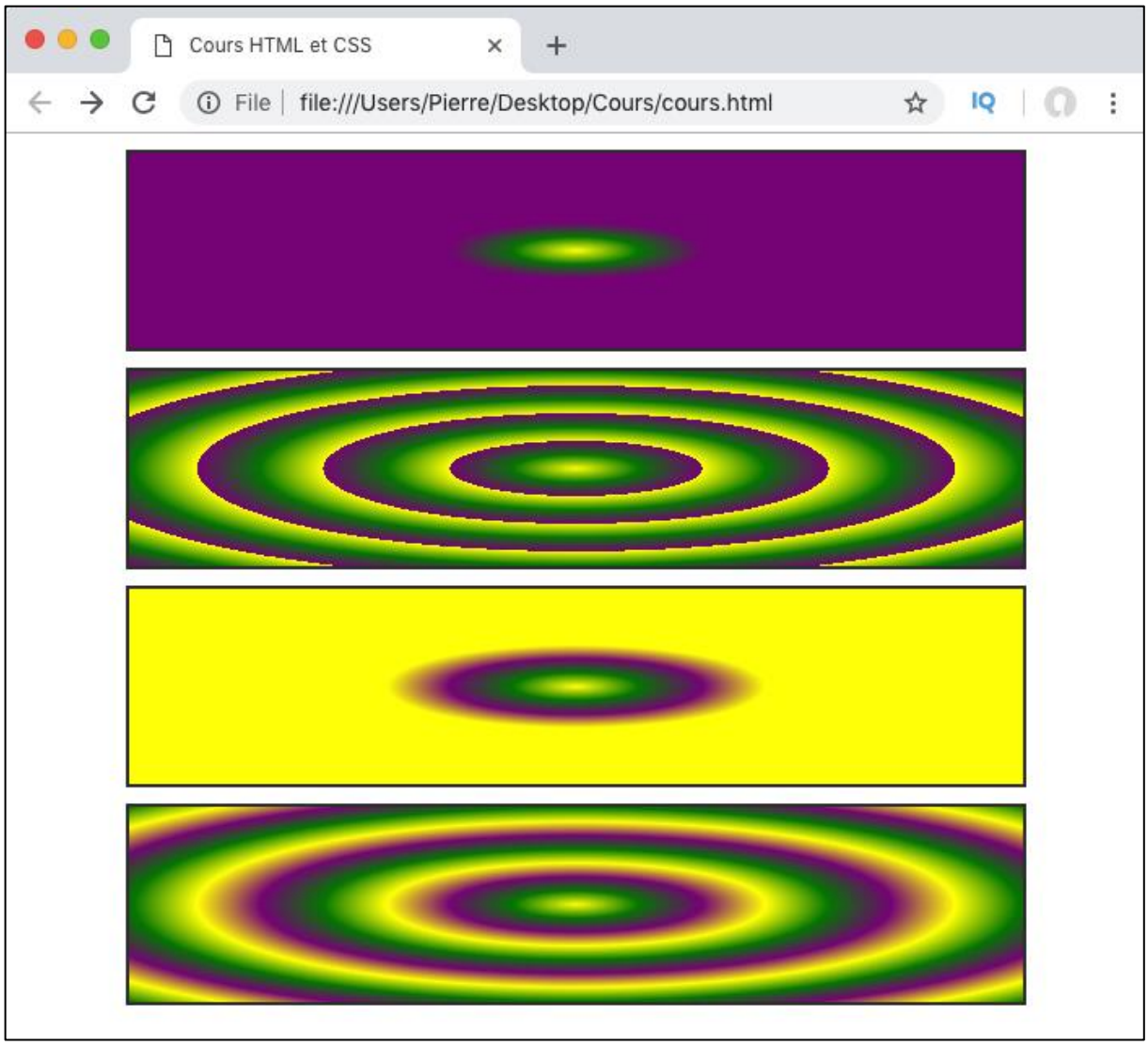
La répétition de dégradés

Nous avons vu dans la leçon précédente qu'on pouvait créer des dégradés linéaires qui vont se répéter avec la fonction repeating-linear-gradient(). Nous allons avoir accès à la même fonctionnalité avec les dégradés radiaux en utilisant cette fois-ci plutôt repeating-radial-gradient().

Cette fonction va s'utiliser de manière analogue à sa sœur servant à créer des dégradés linéaires. Nous allons une nouvelle fois pouvoir créer des transitions fluides en faisant boucler un dégradé radial sur lui-même, c'est-à-dire en lui donnant la même couleur de départ et d'arrivée.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div id="ex1"></div>
        <div id="ex2"></div>
        <div id="ex3"></div>
        <div id="ex4"></div>
    </body>
</html>
```

```css
div{
    width: 80%;
    height: 120px;
    border: 2px solid #333;
    margin: 10px auto;
}

#ex1{
    background: radial-gradient(yellow, green 10%, purple 20%);
}
#ex2{
    background: repeating-radial-gradient(yellow, green 10%, purple 20%);
}
#ex3{
    background: radial-gradient(yellow, green 10%, purple 20%, yellow 30%);
}
#ex4{
    background: repeating-radial-gradient(yellow, green 10%, purple 20%, yellow 30%);
}
```


Cours HTML et CSS
File | file:///Users/Pierre/Desktop/Cours/cours.html

Ajouter des ombres autour des éléments

La propriété CSS box-shadow va nous permettre de créer des ombres dites « portées » autour de nos éléments (ou plus exactement des boîtes qui représentent les éléments). Nous allons pouvoir projeter l'ombre dans n'importe quelle direction et même vers l'intérieur.

Il convient de ne pas confondre box-shadow et text-shadow même si ces deux propriétés vont s'utiliser de façon similaire : la propriété text-shadow va nous permettre de créer des ombres derrière des textes tandis que box-shadow sert à créer des ombres autour de la boite de l'élément.

Définition d'une ombre avec box-shadow

Pour générer une ombre portée, nous allons pouvoir passer jusqu'à 6 valeurs à la propriété box-shadow. Cependant, seules deux de ces valeurs vont être strictement obligatoires : il s'agit des valeurs définissant le décalage horizontal et du décalage vertical de l'ombre.

La liste complète des valeurs que va pouvoir accepter box-shadow est la suivante (valeurs dans leur ordre d'écriture) :

1. **L'inset** : si on précise le mot clef inset, alors l'ombre sera tournée vers l'intérieur de la boite c'est à dire interne à l'élément. Si on omet la valeur (cas par défaut), alors l'ombre sera externe ;
2. **Le décalage horizontal de l'ombre** (valeur requise). Cette valeur est une longueur qu'on va donc pouvoir préciser en unités absolues (px par exemple) ou relatives (em par exemple ; attention les pourcentages ne sont pas acceptés). Si la valeur passée est positive, l'ombre sera projetée sur une distance égale à la valeur passée vers la droite en partant de l'extrémité droite de la boite. Dans le cas contraire, l'ombre sera projetée à gauche de la boite en partant de son extrémité gauche ;
3. **Le décalage vertical de l'ombre** (valeur requise). Cette valeur est également une longueur et va pouvoir prendre les mêmes unités que le décalage horizontal de l'ombre. Si la valeur passée est positive, l'ombre sera projetée sur une distance égale à la valeur passée vers le bas en partant du bas de la boite. Dans le cas contraire, l'ombre sera projetée vers le haut de la boite en partant de son extrémité supérieure ;
4. **Le rayon de flou**. Cette valeur est encore une longueur et va nous permettre de rendre notre ombre floue en lui appliquant un flou Gaussien. La formule de création d'un flou Gaussien est assez complexe ; retenez simplement qu'une moyenne entre les couleurs des pixels environnants va être établie en mixant ces couleurs ensemble. Plus la valeur du flou va être grande, plus le flou de l'ombre sera diffus et l'ombre étalée. La valeur du flou vient s'ajouter aux décalage horizontal et vertical de l'ombre ;
5. **Le rayon d'étendue de l'ombre**. Cette valeur est à nouveau une longueur et va servir à agrandir ou à rétrécir la taille de l'ombre. Par défaut, sa valeur est 0 et cela correspond à une ombre de même taille que la boîte. Passer une valeur positive

augmentera d'autant la taille de l'ombre tandis que passer une valeur négative la rétrécira d'autant ;

6. **La couleur de l'ombre**. Nous allons pouvoir définir la couleur de notre ombre. Toutes les valeurs de couleurs sont ici acceptées : nom de couleur, hex, RGB, RGBa…

Avant de passer à la création d'ombres avec `box-shadow`, j'aimerais attirer votre attention sur un point qui me semble essentiel de connaitre pour bien comprendre ensuite le comportement des ombres créées : la propriété `box-shadow` à elle seule va suffire à créer une ombre qui va par défaut faire la taille de la boite et être centrée derrière la boite. Cette ombre va donc être éclipsée par la boite et être invisible mais elle est bien présente.

Les valeurs données à `box-shadow` ne vont que nous permettre d'agrandir ou de rétrécir cette ombre, de la décaler par rapport à la boite ou de lui appliquer un effet de flou afin d'obtenir une ombre visible autour de notre boite. Savoir cela va être très important notamment pour comprendre le résultat lié à la définition d'un rayon de flou pour une ombre.

Exemples d'utilisation de box-shadow et de création d'ombres en CSS

Voyons immédiatement comment utiliser la propriété `box-shadow` en pratique. Dans les exemples suivants, nous allons commencer par ajouter des ombres très simples puis nous créerons des ombres de plus en plus complexes avec `box-shadow`.

Création d'ombres simples

Nous allons pouvoir créer des ombres très basiques autour de nos boites en ne renseignant que deux valeurs à la propriété box-shadow : la distance de projection ou de décalage de l'ombre sur le plan horizontal et sa distance de projection sur l'axe vertical.

En donnant une valeur positive au paramètre « décalage horizontal », l'ombre sera projetée à droite de la boite. Dans le cas contraire, elle sera projetée à gauche.

En donnant une valeur positive au paramètre « décalage vertical », l'ombre sera projetée en dessous de la boite. Dans le cas contraire, elle sera projetée au-dessus.

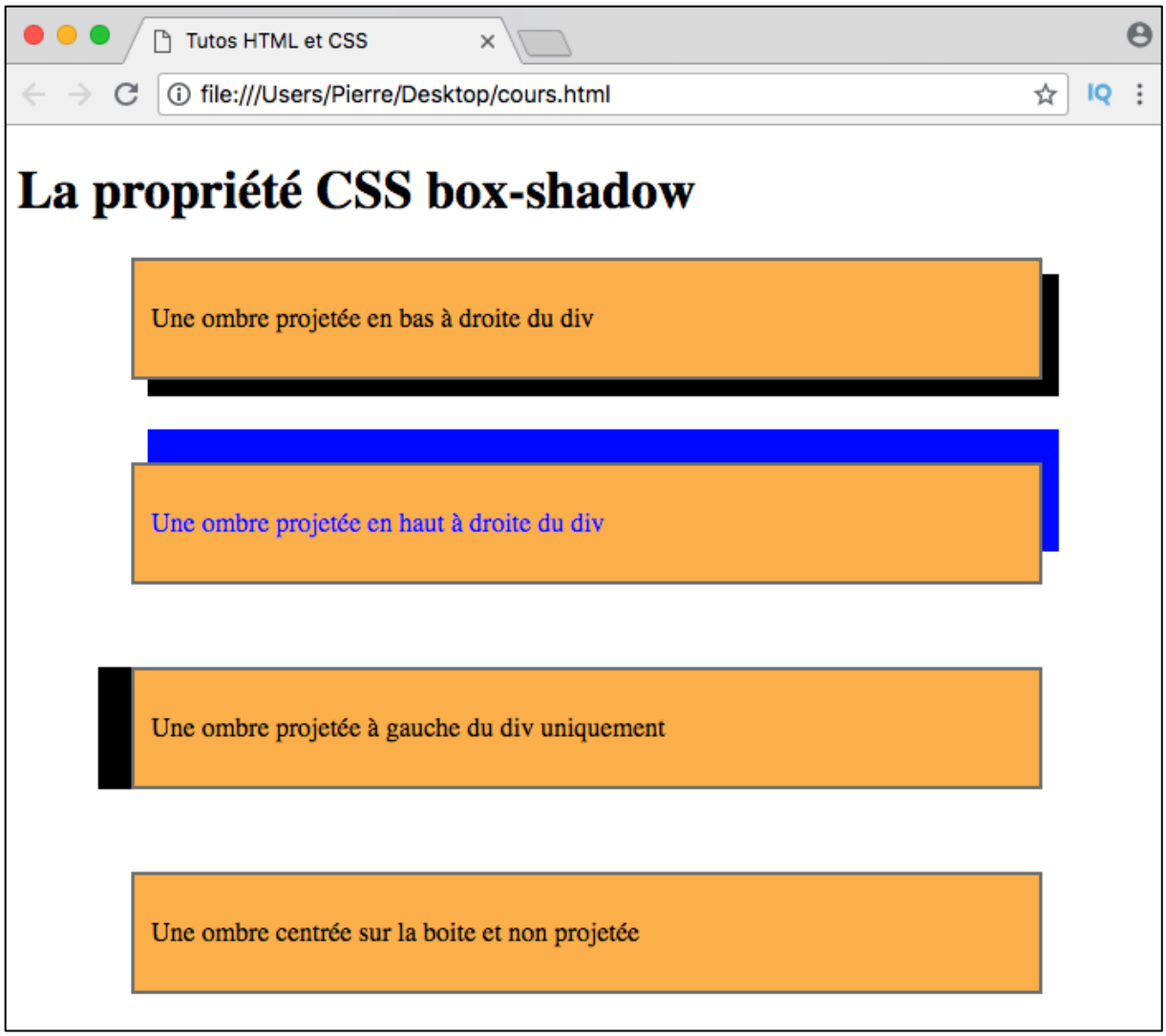
En passant des valeurs nulles, l'ombre ne sera pas projetée et restera derrière la boite. Attention : l'ombre existe quand même, elle n'est juste pas visible.

Notez que si on ne donne pas explicitement une couleur à l'ombre, alors elle sera de la même couleur que le texte de la boite.

Regardez plutôt les exemples suivants (les styles appliqués aux différents `div` ne sont qu'esthétiques, préoccupez-vous seulement des ombres définies avec `box-shadow` :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>La propriété CSS box-shadow</h1>

        <div id="ombre1">
            <p>Une ombre projetée en bas à droite du div</p>
        </div>

        <div id="ombre2">
            <p>Une ombre projetée en haut à droite du div</p>
        </div>

        <div id="ombre3">
            <p>Une ombre projetée à gauche du div uniquement</p>
        </div>

        <div id="ombre4">
            <p>Une ombre centrée sur la boite et non projetée</p>
        </div>
    </body>
</html>
```

```css
div{
    border: 2px solid #777; /*Bordure de 2px*/
    width: 80%; /*Largeur des divs = 80% de la page*/
    margin: 0 auto; /*Centre nos div dans la page*/
    background-color: #fbb854; /*Fond orangé*/
    margin-bottom : 50px ;/*Marge externe basse de 50px*/
    padding: 10px; /*Marges internes de 10px*/
    box-sizing: border-box; /*Contenu + padding + border = 80%*/
}

/*Projection : 10px à droite, 10px vers le bas*/
#ombre1{
    box-shadow: 10px 10px;
}

/*Projection : 10px à droite, pas de projection verticale*/
/*Le texte de la boite est bleu, l'ombre aussi par défaut*/
#ombre2{
    box-shadow: 10px -20px;
    color: blue;
}

/*Projection : 20px à gauche, 5px vers le haut*/
#ombre3{
    box-shadow: -20px 0px;
}

/*Projection : 0px (pas de projection)*/
#ombre4{
    box-shadow: 0px 0px;
}
```

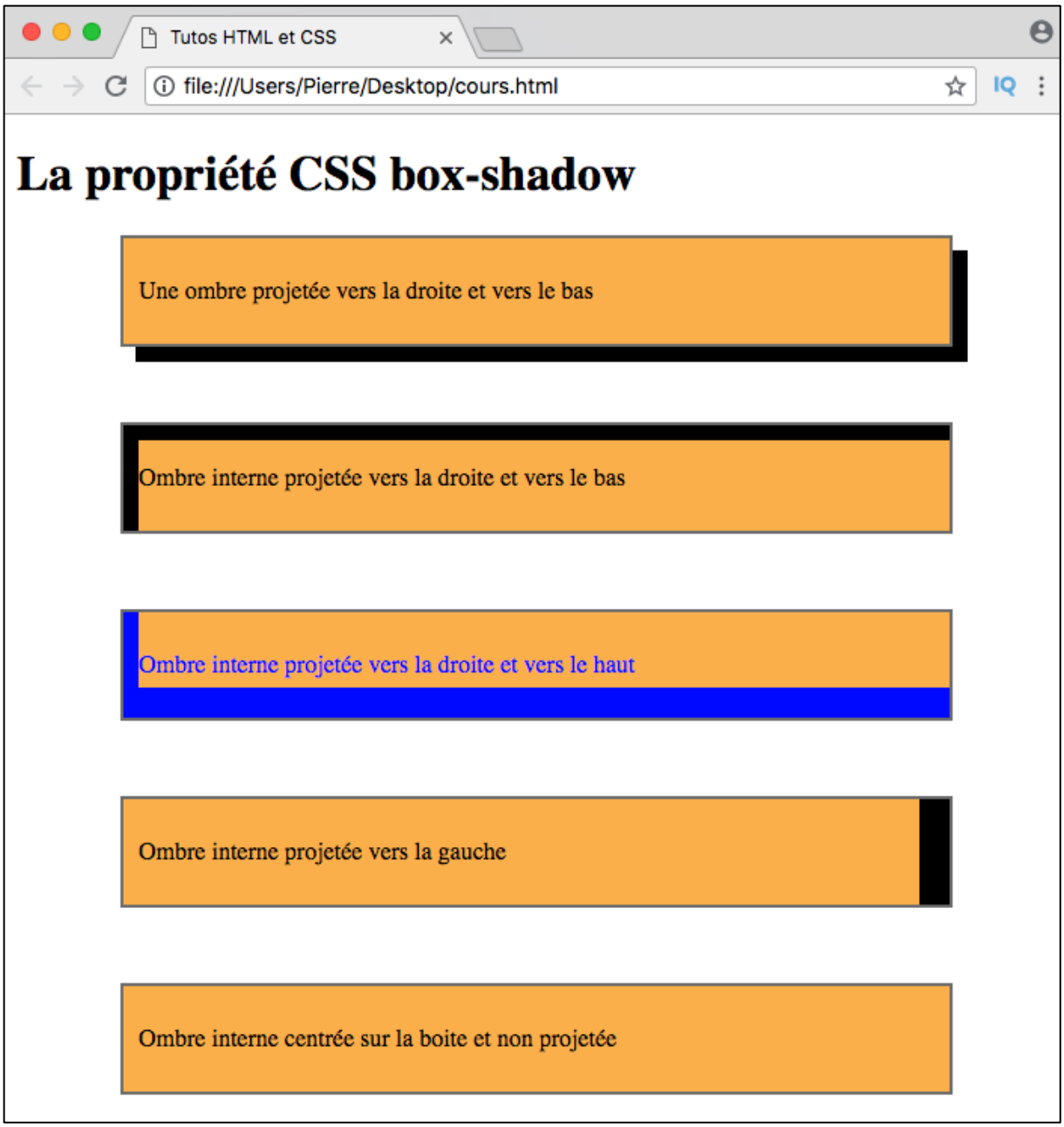


Ajout de la valeur inset pour créer une ombre interne

La propriété box-shadow va également nous permettre de créer des ombres internes à nos boites, c'est-à-dire de projeter l'ombre à l'intérieur de l'élément plutôt qu'à l'extérieur en lui ajoutant la valeur inset en première valeur comme cela :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>La propriété CSS box-shadow</h1>

        <div id="ombre1">
            <p>Une ombre projetée vers la droite et vers le bas</p>
        </div>

        <div id="ombre1b">
            <p>Ombre interne projetée vers la droite et vers le bas</p>
        </div>

        <div id="ombre2b">
            <p>Ombre interne projetée vers la droite et vers le haut</p>
        </div>

        <div id="ombre3b">
            <p>Ombre interne projetée vers la gauche</p>
        </div>

        <div id="ombre4b">
            <p>Ombre interne centrée sur la boite et non projetée</p>
        </div>
    </body>
</html>
```

```css
div{
    border: 2px solid #777; /*Bordure de 2px*/
    width: 80%; /*Largeur des divs = 80% de la page*/
    margin: 0 auto; /*Centre nos div dans la page*/
    background-color: #fbb854; /*Fond orangé*/
    margin-bottom : 50px ;/*Marge externe basse de 50px*/
    padding: 10px; /*Marges internes de 10px*/
    box-sizing: border-box; /*Contenu + padding + border = 80%*/
}

/*Projection : 10px à droite, 10px vers le bas*/
#ombre1{
    box-shadow: 10px 10px;
}
#ombre1b{
    box-shadow: inset 10px 10px;
}

/*Projection : 10px à droite, pas de projection verticale*/
/*Le texte de la boite est bleu, l'ombre aussi par défaut*/
#ombre2b{
    box-shadow: inset 10px -20px;
    color: blue;
}

/*Projection : 20px à gauche, 5px vers le haut*/
#ombre3b{
    box-shadow: inset -20px 0px;
}

/*Projection : 0px (pas de projection)*/
#ombre4b{
    box-shadow: inset 0px 0px;
}
```



Notez bien ici que la direction des ombres est toujours la même. Cependant, comme on demande à ce que l'ombre soit projetée à l'intérieur de la boite, l'ombre va être visible de l'autre côté de la boite par rapport à une ombre externe définie de la même façon. C'est simplement le « point de départ » de l'ombre qui change.

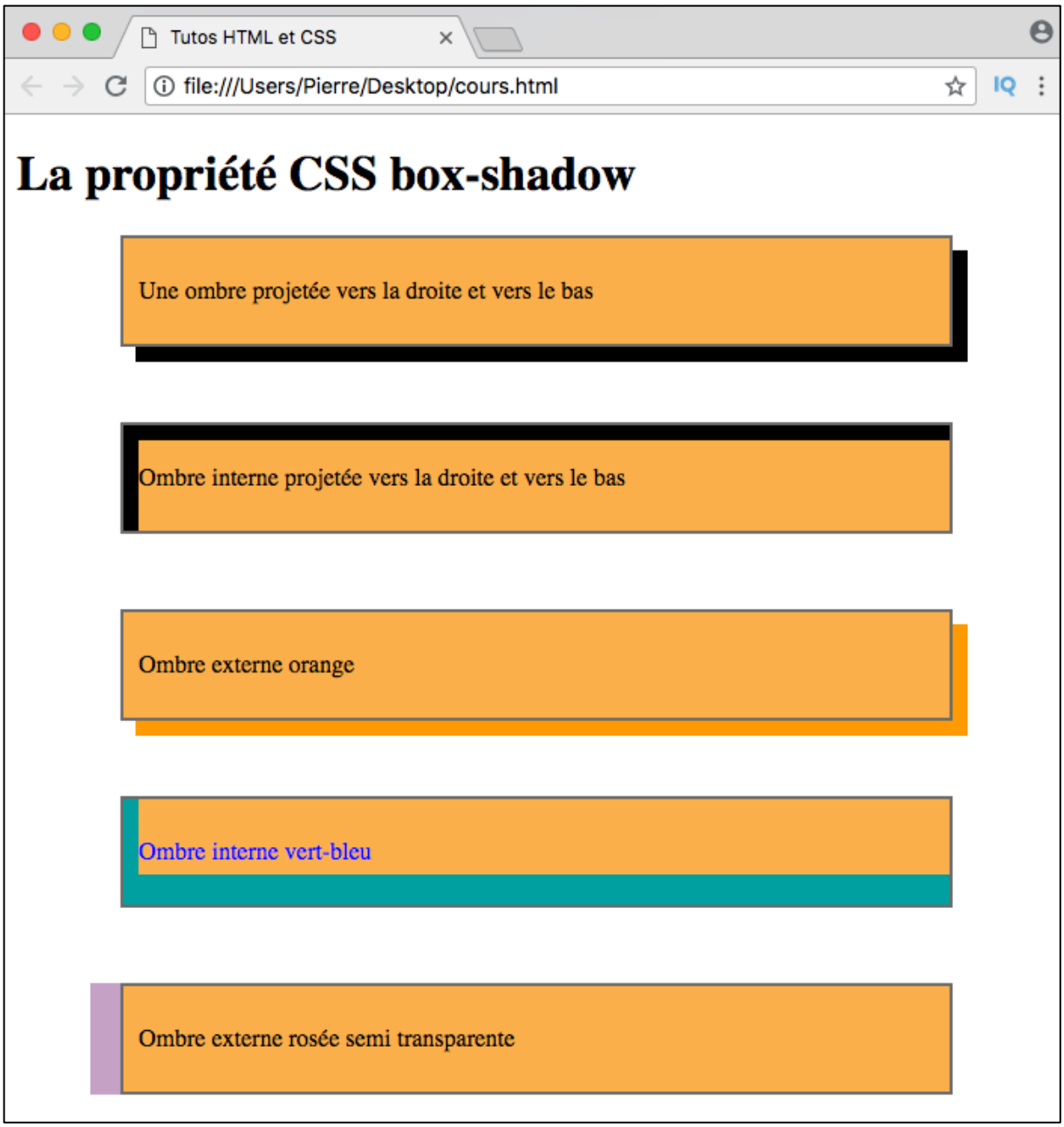
Création d'ombres colorées et / ou semi transparentes

Par défaut, l'ombre créée par `box-shadow` va être de même couleur que le texte de la boite à laquelle on applique l'ombre.

Nous allons cependant pouvoir choisir la couleur de notre ombre. Pour cela, il suffira de la renseigner en valeur de `box-shadow`. Cette propriété accepte toutes les notations de couleurs, que ce soit une couleur nommée, une notation RGB, hexadécimale ou même une notation RGBa qui va nous permettre de créer des ombres semi transparentes.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>La propriété CSS box-shadow</h1>

        <div id="ombre1">
            <p>Une ombre projetée vers la droite et vers le bas</p>
        </div>

        <div id="ombre1b">
            <p>Ombre interne projetée vers la droite et vers le bas</p>
        </div>
        <div id="ombre1c">
            <p>Ombre externe orange</p>
        </div>

        <div id="ombre2c">
            <p>Ombre interne vert-bleu</p>
        </div>

        <div id="ombre3c">
            <p>Ombre externe rosée semi transparente</p>
        </div>
    </body>
</html>
```

```css
div{
    border: 2px solid #777; /*Bordure de 2px*/
    width: 80%; /*Largeur des divs = 80% de la page*/
    margin: 0 auto; /*Centre nos div dans la page*/
    background-color: #fbb854; /*Fond orangé*/
    margin-bottom : 50px ;/*Marge externe basse de 50px*/
    padding: 10px; /*Marges internes de 10px*/
    box-sizing: border-box; /*Contenu + padding + border = 80%*/
}

#ombre1{
    box-shadow: 10px 10px;
}
#ombre1b{
    box-shadow: inset 10px 10px;
}
#ombre1c{
    box-shadow: 10px 10px orange;
}

#ombre2c{
    box-shadow: inset 10px -20px #0AA; /*Ombre vert-bleu*/
    color: blue;
}

#ombre3c{
    box-shadow: -20px 0px RGBa(150,80,150,0.5); /*Rose ?*/
}
```



Note : La valeur « couleur » de l'ombre sera la dernière valeur à indiquer dans box-shadow en CSS.

Ajout d'un flou et ombre centrée autour de la boite

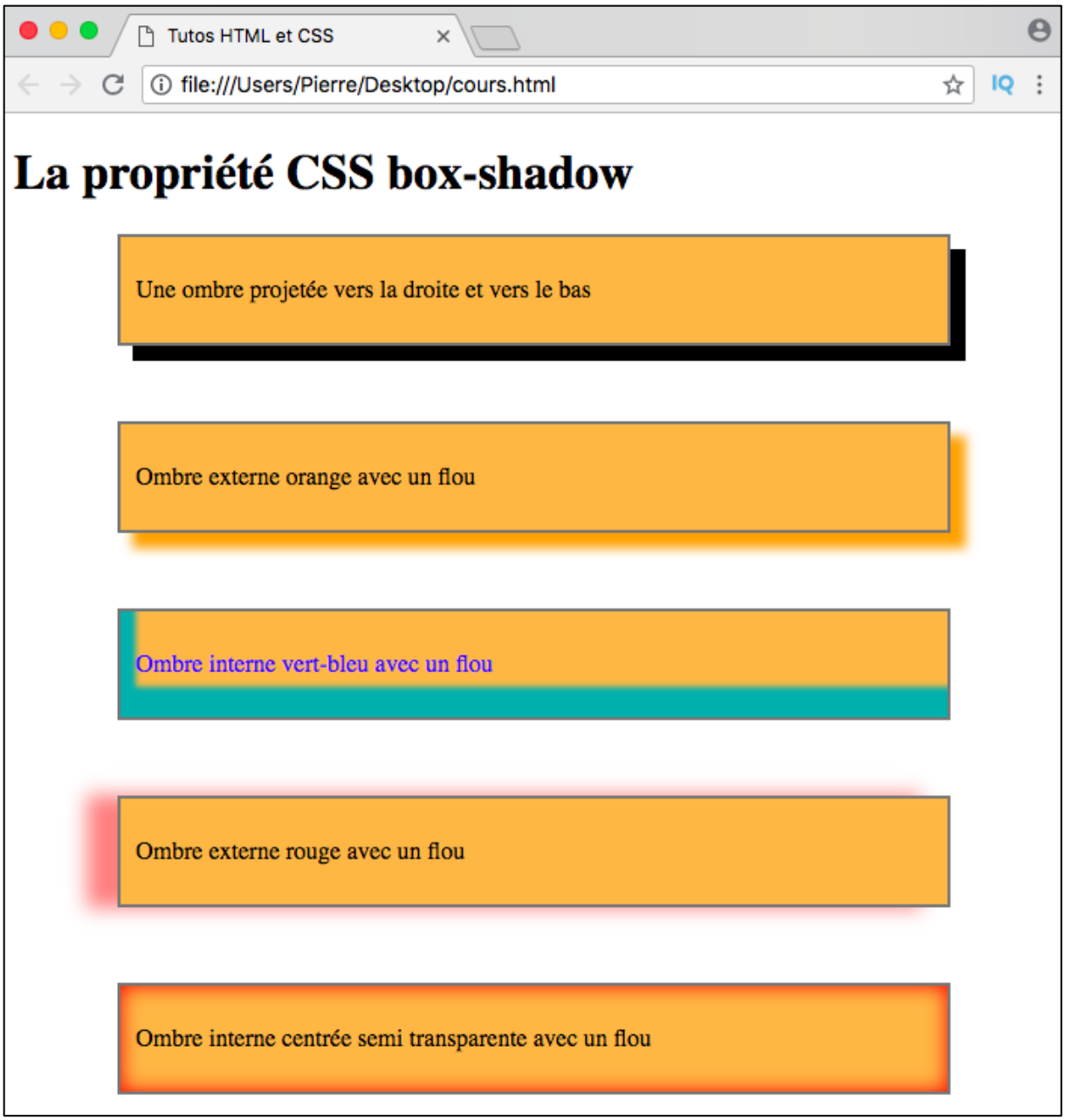
Nous avons vu que pour centrer une ombre par rapport à la boite, il suffisait d'indiquer des décalages horizontal et vertical de 0px lors de la définition de l'ombre avec box-shadow. Cela fait sens : par défaut, l'ombre d'une boite se trouve derrière la boite et la propriété box-shadow va nous permettre, entre autres, de la décaler.

Jusqu'à présent, nos ombres centrées n'étaient pas visibles puisqu'elles faisaient exactement la taille de notre boite. Nous allons cependant pouvoir les faire apparaitre en ajoutant un flou à notre ombre (la « taille » du flou va s'ajouter à la taille de l'ombre ou en modifiant la taille de celle-ci.

Essayons déjà d'ajouter un flou à nos ombres en ajoutant une nouvelle valeur à box-shadow et regardons les résultats ensemble. La valeur du flou va se placer après les valeurs liées aux décalages dans la propriété CSS.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>La propriété CSS box-shadow</h1>

        <div id="ombre1">
            <p>Une ombre projetée vers la droite et vers le bas</p>
        </div>

        <div id="ombre1d">
            <p>Ombre externe orange avec un flou</p>
        </div>

        <div id="ombre2d">
            <p>Ombre interne vert-bleu avec un flou</p>
        </div>

        <div id="ombre3d">
            <p>Ombre externe rouge semi transparente avec un flou</p>
        </div>

        <div id="ombre4d">
            <p>Ombre interne centrée rouge avec un flou</p>
        </div>
    </body>
</html>
```

```css
div{
    border: 2px solid #777; /*Bordure de 2px*/
    width: 80%; /*Largeur des divs = 80% de la page*/
    margin: 0 auto; /*Centre nos div dans la page*/
    background-color: #fbb854; /*Fond orangé*/
    margin-bottom : 50px ;/*Marge externe basse de 50px*/
    padding: 10px; /*Marges internes de 10px*/
    box-sizing: border-box; /*Contenu + padding + border = 80%*/
}

#ombre1{
    box-shadow: 10px 10px;
}

#ombre1d{
    box-shadow: 10px 10px 10px orange;
}

#ombre2d{
    box-shadow: inset 10px -20px 5px #0AA; /*Ombre vert-bleu*/
    color: blue;
}

#ombre3d{
    box-shadow: -20px 0px 20px RGBa(255,0,0,0.5);
}

#ombre4d{
    box-shadow: inset 0px 0px 20px red; /*Rose ?*/
}
```



Pour créer cet effet de flou, une moyenne entre les couleurs des pixels environnants va être établie en mixant ces couleurs ensemble. Plus la valeur du flou va être grande, plus le flou de l'ombre sera diffus et l'ombre étalée. Notez que le flou s'applique tout autour de l'ombre.

Ici, vous pouvez observer que notre la partie floue de notre ombre apparait également au-dessus et en dessous de notre avant dernière boite qui possède une ombre centrée verticalement par rapport à la boite. Ce comportement est dû au fait que la valeur du flou vient s'ajouter à la taille de l'ombre.

Comme l'ombre fait exactement la taille de la boite par défaut, si celle-ci est centrée et si on ajoute un flou autour de l'ombre, alors la partie floue de l'ombre dépassera de la boite et sera visible. C'est la même chose qui se passe dans notre dernier exemple : vous pouvez remarque que seule la partie floue de l'ombre est visible.

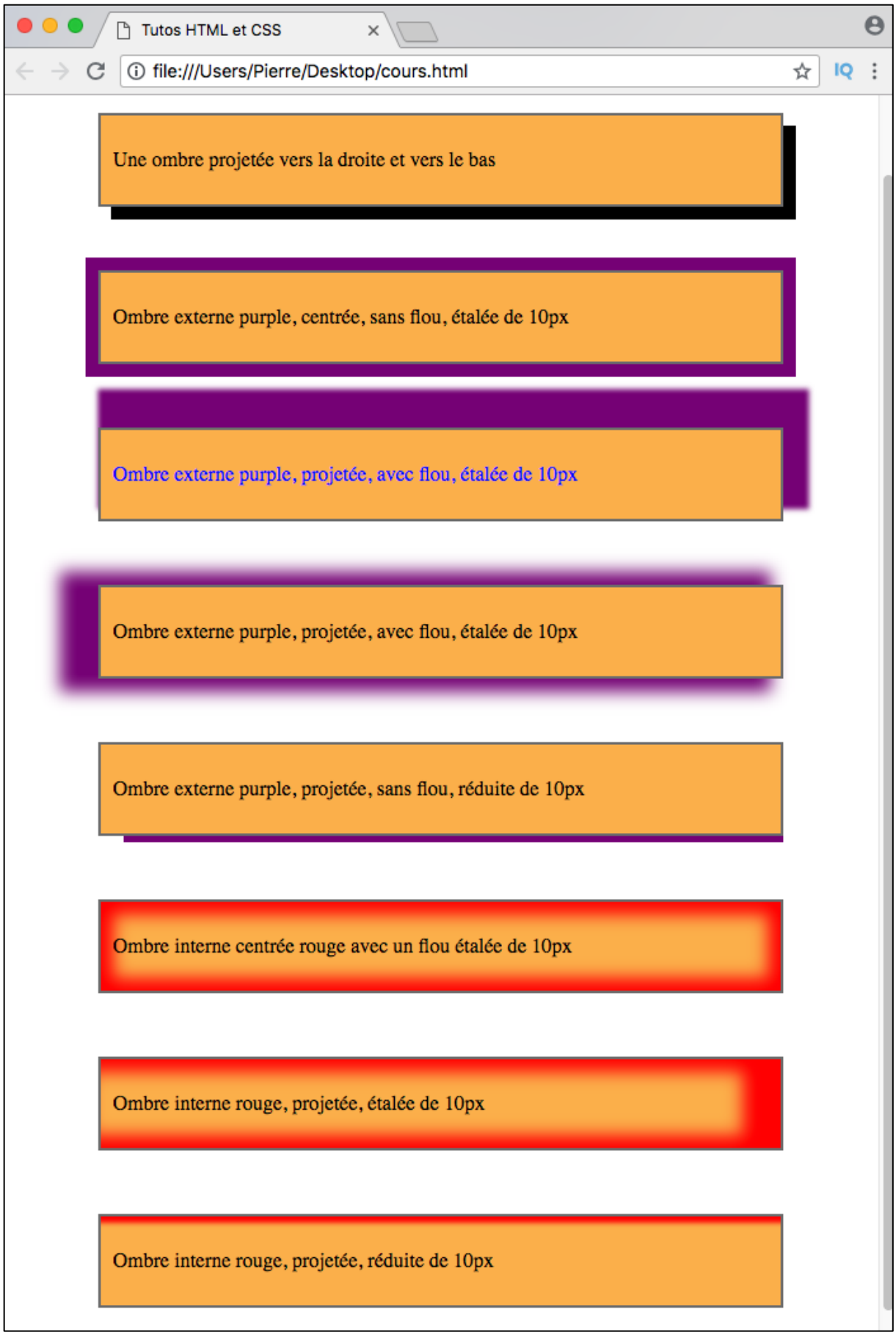
Une ombre plus grande ou plus petite que notre boite

Finalement, nous allons pouvoir modifier la taille de départ de notre ombre pour que celle-ci dépasse par défaut de la boite ou, au contraire, soit plus petite qu'elle.

Les valeurs positives agrandiront (ou étaleront) l'ombre et les valeurs négatives rétréciront l'ombre. Nous allons renseigner cette valeur correspondant au rayon d'étalement de l'ombre après la valeur du flou dans `box-shadow`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>La propriété CSS box-shadow</h1>

        <div id="ombre1">
            <p>Une ombre projetée vers la droite et vers le bas</p>
        </div>

        <div id="ombre1e">
            <p>Ombre externe purple, centrée, sans flou, étalée de 10px</p>
        </div>

        <div id="ombre2e">
            <p>Ombre externe purple, projetée, avec flou, étalée de 10px</p>
        </div>

        <div id="ombre3e">
            <p>Ombre externe purple, projetée, avec flou, étalée de 10px</p>
        </div>

        <div id="ombre4e">
            <p>Ombre externe purple, projetée, sans flou, réduite de 10px</p>
        </div>

        <div id="ombre5e">
            <p>Ombre interne centrée rouge avec un flou étalée de 10px</p>
        </div>

        <div id="ombre6e">
            <p>Ombre interne rouge, projetée, étalée de 10px</p>
        </div>

        <div id="ombre7e">
            <p>Ombre interne rouge, projetée, réduite de 10px</p>
        </div>
    </body>
</html>
```

```css
div{
    border: 2px solid #777; /*Bordure de 2px*/
    width: 80%; /*Largeur des divs = 80% de la page*/
    margin: 0 auto; /*Centre nos div dans la page*/
    background-color: #fbb854; /*Fond orangé*/
    margin-bottom : 50px ;/*Marge externe basse de 50px*/
    padding: 10px; /*Marges internes de 10px*/
    box-sizing: border-box; /*Contenu + padding + border = 80%*/
}

#ombre1{
    box-shadow: 10px 10px;
}

#ombre1e{
    box-shadow: 0px 0px 0px 10px purple;
}

#ombre2e{
    box-shadow: 10px -20px 5px 10px purple;
    color: blue;
}

#ombre3e{
    box-shadow: -20px 0px 20px 10px purple;
}

#ombre4e{
    box-shadow: 10px 15px 0px -10px purple;
}

#ombre5e{
    box-shadow: inset 0px 0px 20px 10px red;
}

#ombre6e{
    box-shadow: inset -20px 0px 20px 10px red;
}

#ombre7e{
    box-shadow: inset 5px 15px 5px -10px red;
}
```


Tutos HTML et CSS
file:///Users/Pierre/Desktop/cours.html
Une ombre projetée vers la droite et vers le bas
Ombre externe purple, centrée, sans flou, étalée de 10px
Ombre externe purple, projetée, avec flou, étalée de 10px
Ombre externe purple, projetée, avec flou, étalée de 10px
Ombre externe purple, projetée, sans flou, réduite de 10px
Ombre interne centrée rouge avec un flou étalée de 10px
Ombre interne rouge, projetée, étalée de 10px
Ombre interne rouge, projetée, réduite de 10px

Nous pouvons voir nettement dans le premier exemple que la taille de notre ombre a été agrandie puisque celle-ci est centrée et ne possède pas de flou mais dépasse bien de la boite.

Parmi ces exemples, j'attire votre attention sur les 4è et dernier, c'est à dire sur les deux ombres qu'on a rétrécit. En effet, ces deux ombres sont censées être décalées vers le bas et vers la droite. Or, on ne voit pas le décalage droit. Pour comprendre pourquoi, il suffit d'analyser le contenu de notre propriété box-shadow dans chacun des cas.

Dans le 4è exemple, l'ombre possède un décalage vers la droite de 10px et pas de flou. Or, on demande également à ce que l'ombre soit rétrécie de 10px. Ainsi, l'ombre va bien être décalée de 10px vers la droite mais comme elle fait désormais 10px de moins que la boite, elle ne va pas dépasser.

Dans notre dernier exemple, on définit un décalage vers la droite de 5px pour notre ombre. De plus, on lui ajoute un flou de 5px qui vient s'ajouter à la taille de l'ombre. Notre ombre devrait donc à nouveau dépasser de 10px. Cependant, ici aussi, on demande à ce que l'ombre soit rétrécie de 10px. L'ombre va donc bien à nouveau être décalée de 5px et un flou de 5px va bien être appliqué mais tout cela ne sera pas visible car compensé par le rétrécissement de l'ombre.

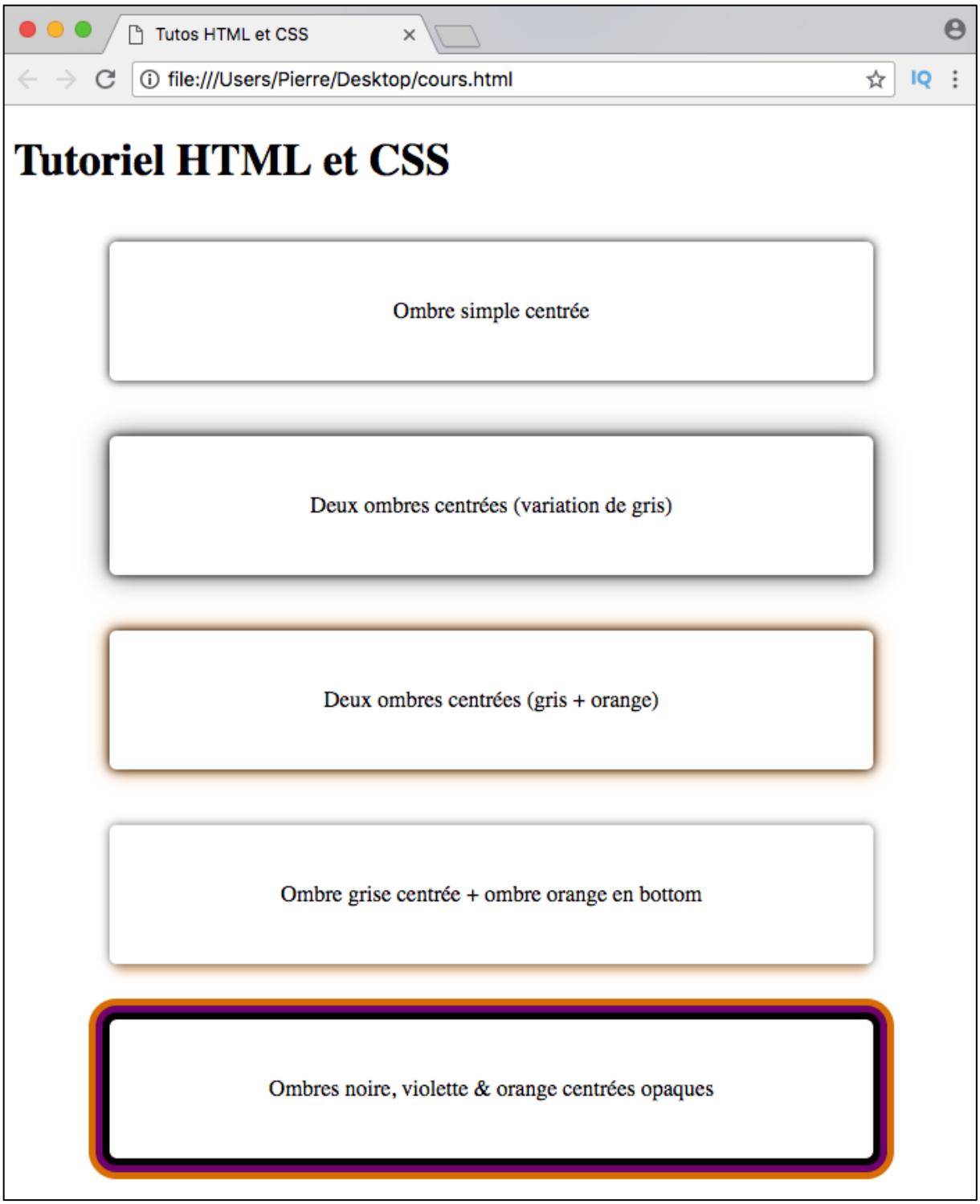
Création d'ombres multiples autour d'un élément

Nous allons tout à fait pouvoir définir plusieurs ombres autour de nos éléments en utilisant box-shadow. Pour cela, il suffit de séparer nos différentes ombres par des virgules lors de leur déclaration avec box-shadow.

Appliquer plusieurs ombres d'un coup à un élément peut nous permettre de créer différents niveaux d'opacité d'ombre et de créer des effets 3D intéressants.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tutos HTML et CSS</title>
        <meta charset='utf-8'>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Tutoriel HTML et CSS</h1>

        <div id="d1">Ombre simple centrée</div>
        <div id="d2">Deux ombres centrées (variation de gris)</div>
        <div id="d3">Deux ombres centrées (gris + orange)</div>
        <div id="d4">Ombre grise centrée + ombre orange en bottom</div>
        <div id="d5">Ombres noire, violette & orange centrées opaques</div>
    </body>
</html>
```

```css
div{
    width: 80%;
    height: 100px;
    margin: 40px auto;
    border-radius: 5px;
    text-align: center;
    line-height: 100px;
}

#d1{
    box-shadow: 0px 0px 10px #000;
}
#d2{
    box-shadow: 0px 0px 10px #000,
                0px 0px 30px #777;
}
#d3{
    box-shadow: 0px 0px 10px #000,
                0px 0px 20px RGB(220,120,0);
}
#d4{
    box-shadow: 0px 0px 10px #777,
                0px 10px 10px -5px RGBa(220,120,0,0.5);
}
#d5{
    box-shadow: 0px 0px 0px 5px #000,
                0px 0px 0px 10px RGB(120,0,120),
                0px 0px 0px 15px RGB(220,120,0);
}
```



Ici, nous réutilisons simplement ce que nous avons vu jusqu'à présent pour créer des ombres. Simplement, nous déclarons plusieurs ombres d'aspects et de tailles différents autour de nos éléments `div`.

PARTIE X

Sélecteurs CSS complexes

Les sélecteurs CSS d'attributs

L'une des grandes forces du CSS réside dans le fait qu'on va pouvoir cibler très précisément tel ou tel élément HTML grâce à la grande variété de ces sélecteurs.

Cette partie est dédiée à l'étude de trois types de sélecteurs que nous n'avons pas encore étudiés : les sélecteurs d'attributs, les pseudo-classes et les pseudo-éléments.

Dans cette leçon, nous allons déjà apprendre à manipuler les sélecteurs d'attributs qui vont nous permettre de sélectionner un élément HTML selon qu'il possède un attribut avec une certaine valeur ou pas.

Sélectionner un élément selon qu'il possède un attribut

Pour commencer, nous allons en CSS pouvoir cibler un élément HTML selon qu'il possède un certain attribut comme un attribut `href`, `src`, `target`, `class`, etc.

Pour faire cela, nous allons utiliser la syntaxe suivante : `E[foo]` où « E » représente un nom d'élément et « foo » représente un nom d'attribut.

Par exemple, on va pouvoir cibler tous les éléments `a` qui possèdent un attribut `target` en écrivant `a[target]` ou encore tous les éléments `p` qui possèdent un attribut `class` avec `p[class]`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un <a href="#" target="_blank">lien</a> vide</p>
        <p class="test">Un autre <a href="#">lien</a> vide</p>
        <p>Un troisième <a href="#">lien</a> vide</p>
    </body>
</html>
```

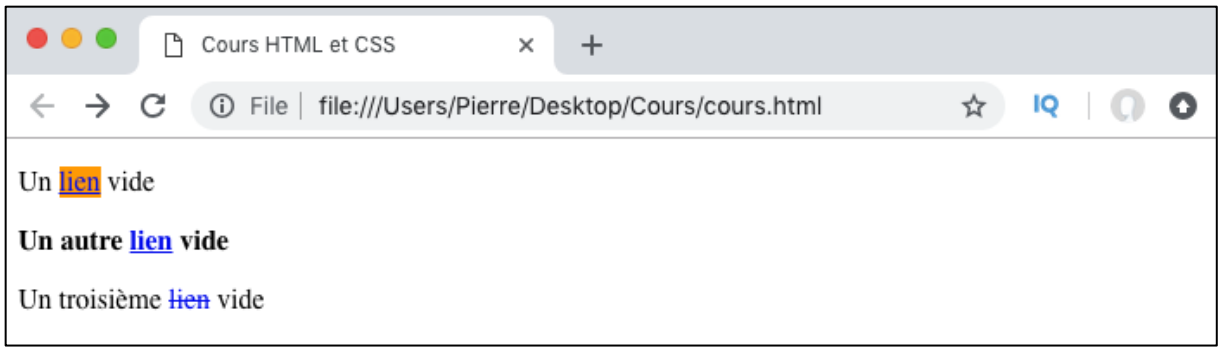
```css
/*Cible les éléments a qui possèdent un attribut target*/
a[target]{
    background-color: orange;
}

/*Cible les éléments p qui possèdent un attribut class*/
p[class]{
    font-weight: bold;
}
```

Sélectionner un élément selon qu'il possède un attribut avec une valeur exactement

Nous allons également pouvoir sélectionner de façon plus précise les éléments HTML possédant un attribut auquel on a attribué une valeur en particulier en CSS. Par exemple, nous allons pouvoir sélectionner tous les éléments `a` possédant un attribut `rel` avec une valeur exactement égale à `nofollow` en utilisant la syntaxe `a[rel="nofollow"]`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un <a href="#" target="_blank">lien</a> vide</p>
        <p class="test">Un autre <a href="#" rel="follow">lien</a> vide</p>
        <p>Un troisième <a href="#" rel="nofollow">lien</a> vide</p>
    </body>
</html>
```

```
/*Cible les éléments a qui possèdent un attribut target*/
a[target]{
    background-color: orange;
}

/*Cible les éléments p qui possèdent un attribut class*/
p[class]{
    font-weight: bold;
}
 /*Cible les éléments a avec un attribut rel de valeur nofollow*/
a[rel="nofollow"]{
    text-decoration: line-through;
}
```



Note : l'attribut `rel` sert à indiquer la relation entre le document de départ et le document lié. Ici, la valeur `nofollow` permet d'indiquer aux robots des moteurs de recherche qu'il n'est pas nécessaire de suivre le lien. Cet attribut est souvent utilisé à des fins d'optimisation du référencement.

Sélectionner un élément selon qu'il possède un attribut contenant une sous valeur distincte (séparée du reste par un espace)

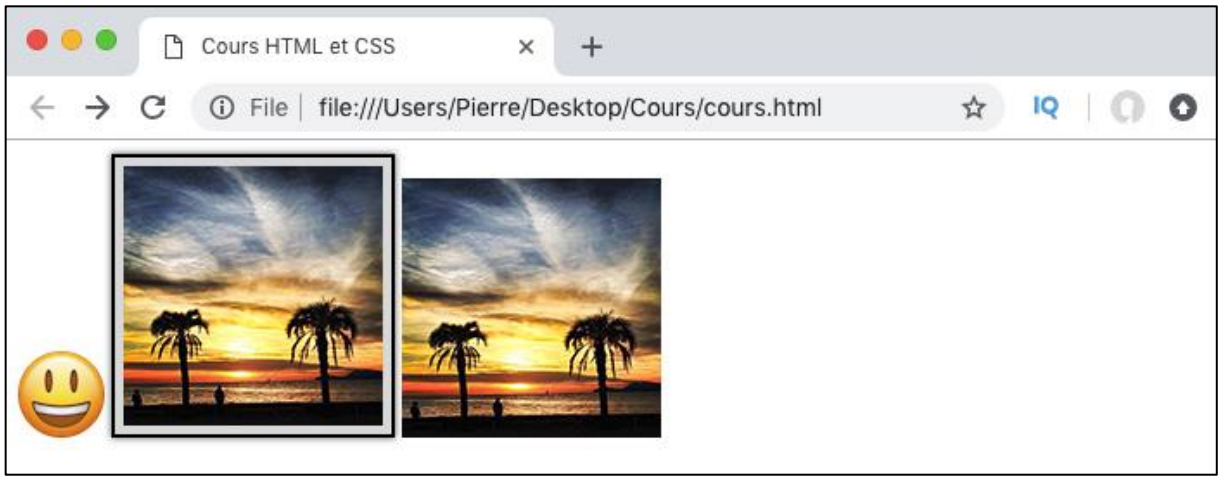
Nous allons encore pouvoir sélectionner en CSS un élément HTML possédant un attribut qui contient une certaine sous valeur distincte des autres, c'est-à-dire séparée des autres par une espace.

Ici, je parle de « sous valeur » pour désigner une partie de la valeur d'un attribut, c'est-à-dire pour désigner un ou plusieurs caractères inclus dans la valeur passée à l'attribut.

Par exemple, on va pouvoir cibler tous les éléments `img` d'une page dont l'attribut `alt` contient le texte « soleil » grâce à la syntaxe suivante : `img[alt~="soleil"]`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <img src="smile.png" alt="Un emoji content">
        <img src="sunset.png" alt="Un coucher de soleil">
        <img src="sunset.png" alt="Coucher-de-soleil">
    </body>
</html>
```

```css
/*Cible les éléments img qui possèdent un attribut alt avec " soleil "*/
img[alt~="soleil"]{
    display: inline-block;
    padding: 5px;
    border: 2px solid black;
    background-color: #ddd;
    box-shadow: 0px 0px 5px #777
}
```



Sélectionner un élément selon qu'il possède un attribut commençant par une certaine sous valeur

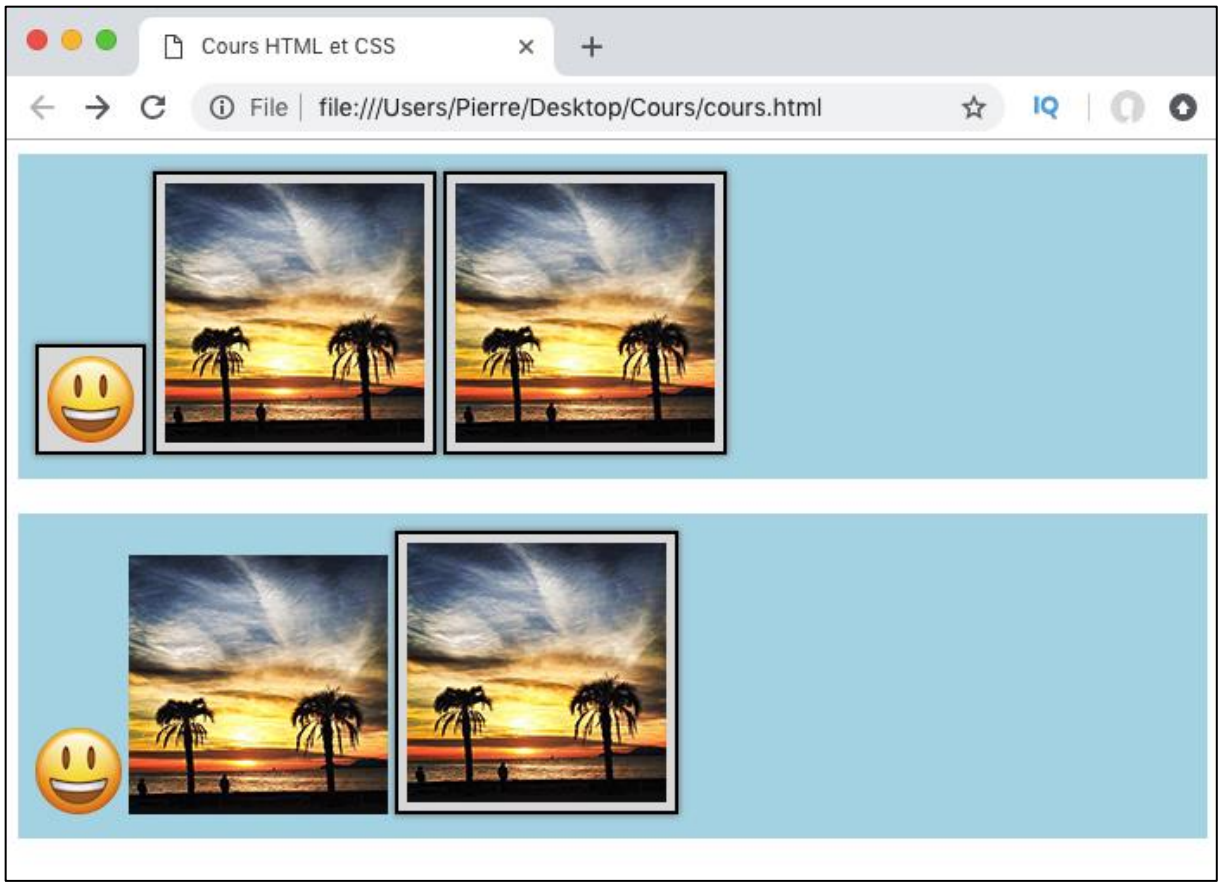
Pour sélectionner un élément `E` selon qu'il possède un attribut `foo` commençant par la sous valeur `val`, nous utiliserons la syntaxe suivante : `E[foo^=val]`.

En utilisant ce sélecteur, on n'impose pas plus de contrainte sur la sous valeur `val`. Celle-ci peut donc faire partie d'un mot plus grand par exemple.

On va également pouvoir utiliser le sélecteur `E[foo|=val]` dans le cas où l'on souhaite cibler un élément possédant un attribut `foo` dont la valeur est exactement `val` ou commence par `val` suivi d'un tiret `-`(hyphen) en anglais).

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="set1">
            <img src="smile.png" alt="Une emoticone content">
            <img src="sunset.png" alt="Un coucher de soleil">
            <img src="sunset.png" alt="Un-Coucher-de-soleil">
        </div>
        <div class="set2">
            <img src="smile.png" alt="Une emoticone content">
            <img src="sunset.png" alt="Un coucher de soleil">
            <img src="sunset.png" alt="Un-Coucher-de-soleil">
        </div>
    </body>
</html>
```

```css
div{
    width: 100%;
    padding: 10px;
    box-sizing: border-box;
    margin-bottom: 20px;
    background-color: lightBlue;
}

/*Cible les éléments img qui possèdent un attribut alt commençant par "Un"*/
.set1 img[alt^="Un"]{
    display: inline-block;
    padding: 5px;
    border: 2px solid black;
    background-color: #ddd;
    box-shadow: 0px 0px 5px #777
}

/*Cible les éléments img qui possèdent un attribut alt avec la valeur "Un"
 *exactement ou qui commence avec "Un-"*/
.set2 img[alt|="Un"]{
    display: inline-block;
    padding: 5px;
    border: 2px solid black;
    background-color: #ddd;
    box-shadow: 0px 0px 5px #777
}
```



Sélectionner un élément selon qu'il possède un attribut se finissant par une certaine sous valeur

Nous allons encore pouvoir en CSS cibler des éléments HTML possédant un attribut dont la valeur se termine par une certaine valeur, sans plus de restriction que cela.

Nous allons ainsi par exemple pouvoir cibler tous nos éléments `a` possédant un attribut `href` dont la valeur se termine par « .com » avec la syntaxe suivante : `a[href$=".com"]`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Lien vers <a href="https://www.pierrre-giraud.fr">mon site .fr</a></p>
        <p>Lien vers <a href="https://www.pierrre-giraud.com">mon site .com</a></p>
        <p>Lien vers <a href="https://fr.wikipedia.org/">wikipedia.org</a></p>
    </body>
</html>
```

```css
/*Cible les éléments a possédant un attribut href dont la
 *valeur se termine par ".com"*/
a[href$=".com"]{
    background-color: orange;
}
```

Sélectionner un élément selon qu'il possède un attribut possédant une valeur parmi d'autres

Finalement, on va pouvoir cibler en CSS un élément HTML qui possède un attribut contenant lui-même une certaine sous valeur sans aucune restriction en utilisant la syntaxe `E[foo*="val"]` (où `E` représente un élément, `foo` représente un attribut et `val` une sous valeur.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Un <a href="#">lien</a> vide</p>
        <p>Un autre <a href="#" rel="follow">lien</a> vide</p>
        <p>Un troisième <a href="#" rel="nofollow">lien</a> vide</p>
    </body>
</html>
```

```css
/*Cible les éléments a possédant un attribut rel contenant la valeur follow*/
a[rel*="follow"]{
    background-color: orange;
}
```

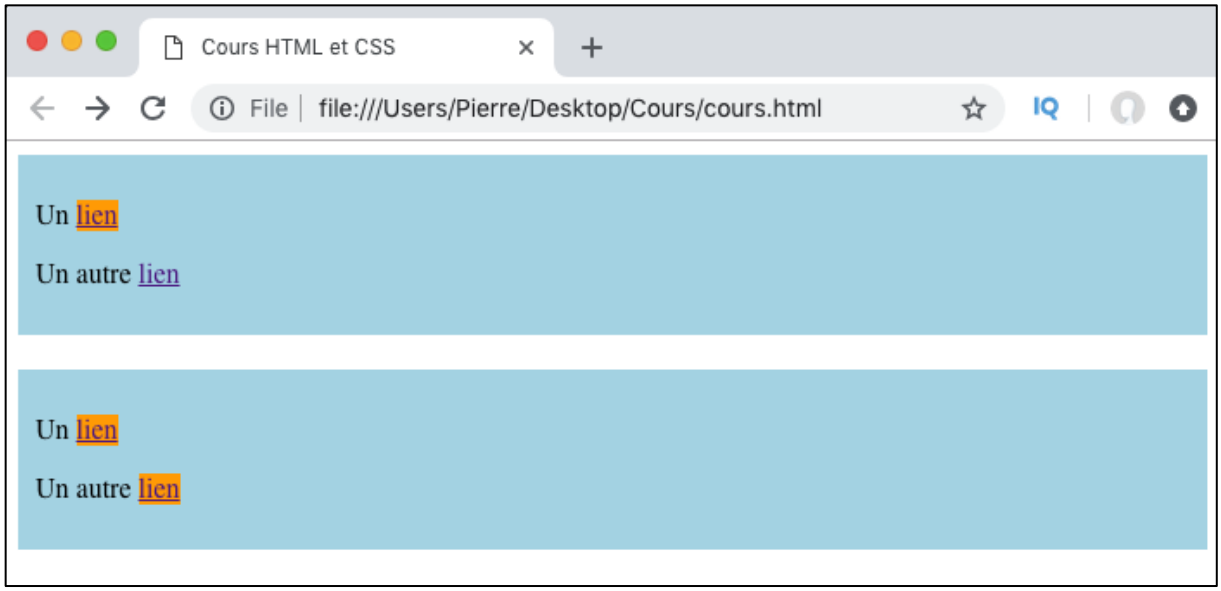
Rendre ses sélecteurs d'attributs insensibles à la casse

Par défaut, les sélecteurs d'attributs vont être sensibles à la casse c'est-à-dire faire la différence entre un caractère en majuscule et en minuscule. Ainsi, les valeurs « un », « UN », « Un » et « uN » vont être considérées comme différentes.

On va pouvoir changer ce comportement et rendre nos sélecteurs insensibles à la casse en ajoutant la lettre i (I minuscule) à la fin de nos sélecteurs d'attributs.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="set1">
            <p>Un <a href="https://www.pierre-giraud.com">lien</a></p>
            <p>Un autre <a href="https://www.Pierre-giraud.com">lien</a></p>
        </div>
        <div class="set2">
            <p>Un <a href="https://www.pierre-giraud.com">lien</a></p>
            <p>Un autre <a href="https://www.Pierre-giraud.com">lien</a></p>
        </div>
    </body>
</html>
```

```css
div{
    width: 100%;
    padding: 10px;
    box-sizing: border-box;
    margin-bottom: 20px;
    background-color: lightBlue;
}

/*Sélectionne les éléments a avec un attribut href contenant "pierre"*/
.set1 a[href*="pierre"]{
     background-color: orange;
}

/*Sélectionne les éléments a avec un attribut href contenant
 *"pierre", "Pierre", "PiErRe", "PIERRE", etc.*/
.set2 a[href*="pierre" i]{
     background-color: orange;
}
```



Attention cependant : cette notation est récente est n'est à l'heure actuelle par encore une recommandation et n'est pas supportée par les navigateurs Edge ni d'Internet Explorer.

Tableau récapitulatif des différents sélecteurs d'attributs

Vous pourrez retrouver ci-dessous les différents sélecteurs d'attributs avec leur définition et la version CSS où ils ont été passés en recommandation.

Sélecteur	Description	Recommandation CSS
E[foo]	Sélectionne tout élément E possédant un attribut foo	CSS2

Sélecteur	Description	Recommandation CSS
`E[foo="bar"]`	Sélectionne tout élément E possédant un attribut `foo` dont la valeur est exactement « bar »	CSS2
`E[foo~="bar"]`	Sélectionne tout élément E possédant un attribut `foo` dont la valeur contient distinctement « bar » (c'est-à-dire dont la valeur contient le mot « bar » séparé du reste par des espaces)	CSS2
`E[foo|="en"]`	Sélectionne tout élément E possédant un attribut `foo` dont la valeur commence par « en » séparé du reste par un tiret (ou hyphen en anglais)	CSS2
`E[foo^="bar"]`	Sélectionne tout élément E possédant un attribut `foo` dont la valeur commence exactement par « bar »	CSS3
`E[foo$="bar"]`	Sélectionne tout élément E possédant un attribut `foo` dont la valeur se termine exactement par « bar »	CSS3
`E[foo*="bar"]`	Sélectionne tout élément E possédant un attribut `foo` dont la valeur contient la valeur « bar »	CSS3
`E[foo{~|^$*}="bar" i]`	L'ajout d'un `i` à la fin des sélecteurs précédents rend la sélection « case insensitive » c'est-à-dire « insensible à la casse ». La valeur recherchée ne tiendra pas compte de la casse, c'est-à-dire de l'utilisation de majuscules et de minuscules.	CSS4

Les pseudo-classes CSS

Les pseudo-classes vont nous permettre d'appliquer des styles à des éléments HTML uniquement lorsque ceux-ci sont dans un certain état (cliqués, activés, etc.).

Il existe de nombreuses pseudo-classes en CSS. Dans cette leçon, nous n'allons étudier en détail que les plus utilisées car toutes les pseudo-classes s'utilisent de manière similaire (la syntaxe sera toujours la même).

Ainsi, il vous suffira de comprendre comment appliquer une pseudo-classe pour savoir toutes les utiliser, à condition bien évidemment de savoir à quoi correspond chaque pseudo-classe.

Pour cela, je vais également vous fournir un tableau récapitulatif de toutes les pseudo-classes actuellement disponibles afin que vous puissiez piocher dedans selon vos besoins.

Définition et syntaxe des pseudo-classes

Les pseudo-classes vont nous permettre de cibler des éléments HTML en fonction de leur état, ou plus exactement d'appliquer des règles CSS à des éléments HTML uniquement dans un certain contexte (lorsqu'ils sont dans un certain état).

Ainsi, nous allons pouvoir modifier les styles d'un lien HTML selon que le curseur de la souris d'un visiteur soit dessus (état `hover`) ou qu'il ait été déjà cliqué (état `visited`).

Pour utiliser une pseudo classe en CSS, il faudra commencer avec un sélecteur « classique » (simple ou complexe) qui sera suivi du caractère `:` puis du nom de la pseudo-classe ou de l'état à appliquer.

Par exemple, on va pouvoir changer la couleur de fond des éléments `div` d'une page en orange lors du survol du curseur de la souris uniquement en écrivant en CSS `div:hover{background-color : orange ;}`.

Les pseudo-classes :hover, :visited, :active et :link

La pseudo-classe `:hover` va nous permettre d'appliquer des styles à un élément HTML uniquement lorsque celui-ci est survolé par la souris d'un utilisateur. Cette pseudo-classe peut être utilisée avec la majorité des éléments HTML mais on l'utilisera généralement pour appliquer des styles différents à des liens lorsqu'un utilisateur passe sa souris dessus.

La pseudo-classe CSS `:visited` va nous permettre d'appliquer des styles à un élément déjà visité, c'est-à-dire déjà cliqué. En pratique, cette pseudo-classe va une nouvelle fois surtout être utilisée avec des éléments de lien. Ainsi, on va pouvoir changer l'apparence d'un lien après que celui-ci ait été cliqué, lorsque l'utilisateur revient sur notre page de départ.

La pseudo-classe CSS :active va elle nous permettre de modifier les styles d'un élément HTML lors d'un état très particulier qui est le moment où l'on clique sur l'élément. Pour bien visualiser cet état, je vous conseille de rester cliqué sur l'élément en question le temps de voir les changements de style. Une nouvelle fois, en pratique, cette pseudo-classe va généralement être utilisée pour modifier l'apparence des liens au moment du clic.

La pseudo-classe :link va elle nous permettre au contraire de cibler tous les liens non visités et de leur appliquer des styles particuliers en CSS.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div><p>Passez sur moi !</p></div>
        <p>Un paragraphe avec un <a href="#">lien</a></a></p>
    </body>
</html>
```

```css
div{
    height: 100px;
    border: 1px solid black;
    width: 100%;
    box-sizing: border-box;
    text-align: center;
}

div:hover{
    background-color: lightBlue;
}

a:link{
    color: black;
    text-decoration: none;
}
a:hover{
    color: orange;
    text-decoration: underline;
}
a:visited{
    color: #0AF;/*Bleu*/
}
a:active{
    background-color: #0FA;/*Vert*/
}
```

Notez qu'en cas d'application de plusieurs des pseudo-classes ci-dessus à un élément on respectera l'ordre d'écriture suivant en CSS : d'abord les déclarations liées à :link puis :visited puis :hover et enfin :active.

Cela va permettre d'avoir le comportement le plus logique en cas de conflit : les styles liés à :active s'appliqueront uniquement lors du clic sur l'élément, ceux liés à :hover s'appliqueront lors du survol sans clic et etc.

Les pseudo-classes :first-child, :first-of-type, :last-child et :last-of-type

Les pseudo-classes :first-child et :first-of-type vont nous permettre de sélectionner respectivement un élément qui va être le premier enfant de son parent et un élément qui va être le premier élément de ce type de son parent.

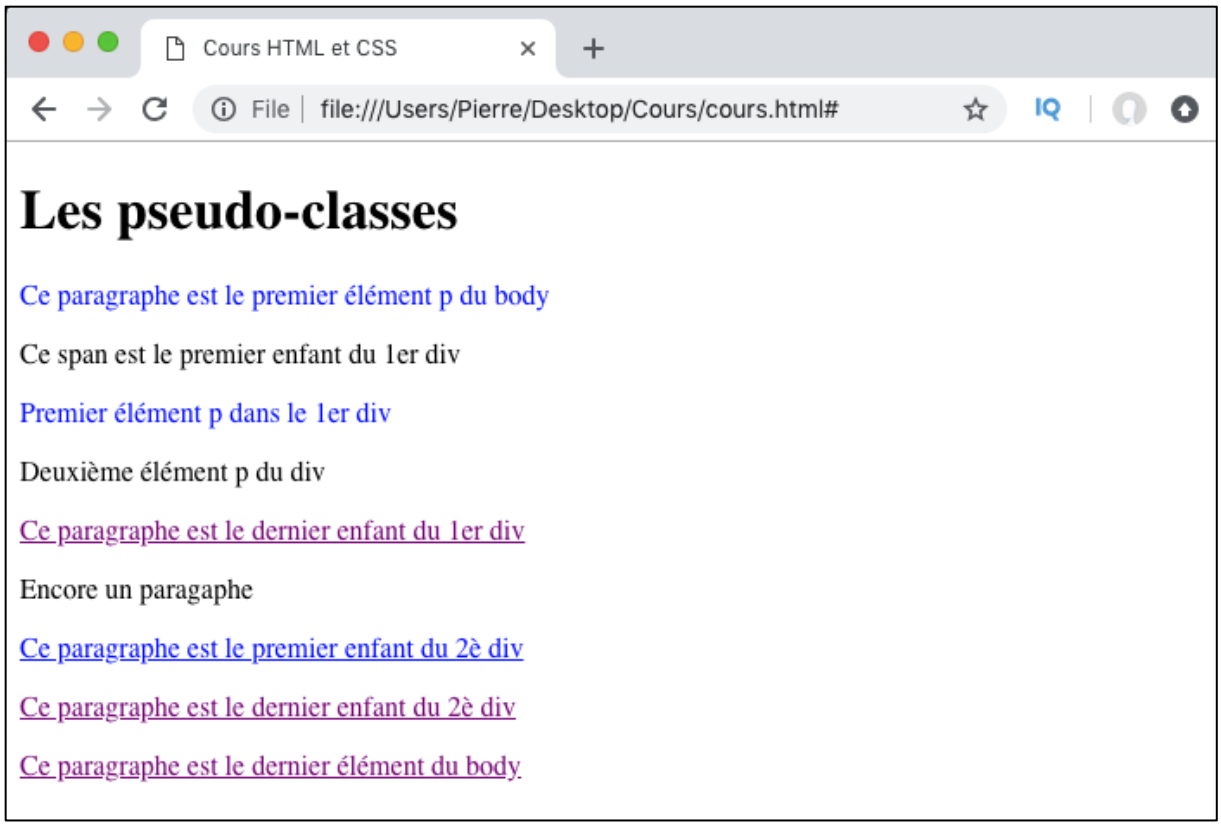
Par exemple, le sélecteur p:first-child va sélectionner tous les éléments p d'une page qui sont les premiers enfant de leur parent tandis que le sélecteur CSS p:first-of-type va nous permettre de sélectionner tous les éléments p qui sont le premier élément p enfant de leur parent.

A l'inverse, les pseudo-classes CSS :last-child et :last-of-type vont nous permettre de sélectionner respectivement un élément qui va être le dernier enfant de son parent et un élément qui va être le dernier élément de ce type de son parent.

Ainsi, le sélecteur CSS last-child va sélectionner tous les éléments p d'une page qui sont les derniers enfants de leur parent tandis qu'on va pouvoir sélectionner tous les éléments p qui sont le dernier élément p enfant de leur parent grâce au sélecteur CSS last-of-type.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Les pseudo-classes</h1>
        <p>Ce paragraphe est le premier élément p du body</p>

        <div>
            <span>Ce span est le premier enfant du 1er div</span>
            <p>Premier élément p dans le 1er div</p>
            <p>Deuxième élément p du div</p>
            <p>Ce paragraphe est le dernier enfant du 1er div</p>
        </div>

        <p>Encore un paragaphe</p>

        <div>
            <p>Ce paragraphe est le premier enfant du 2è div</p>
            <p>Ce paragraphe est le dernier enfant du 2è div</p>
        </div>
        <p>Ce paragraphe est le dernier élément du body</p>
    </body>
</html>
```

```css
p:first-child{
    color: orange;
    text-decoration: underline;
}
p:first-of-type{
    color: blue;
}
p:last-child{
    color: green;
    text-decoration: underline;
}
p:last-of-type{
    color: purple;
}
```



Notez que dans si plusieurs pseudo-classes ciblent un même élément, alors les styles appliqués en cas de conflit à l'élément seront une nouvelle fois ceux de la dernière pseudo-classe déclarée en CSS.

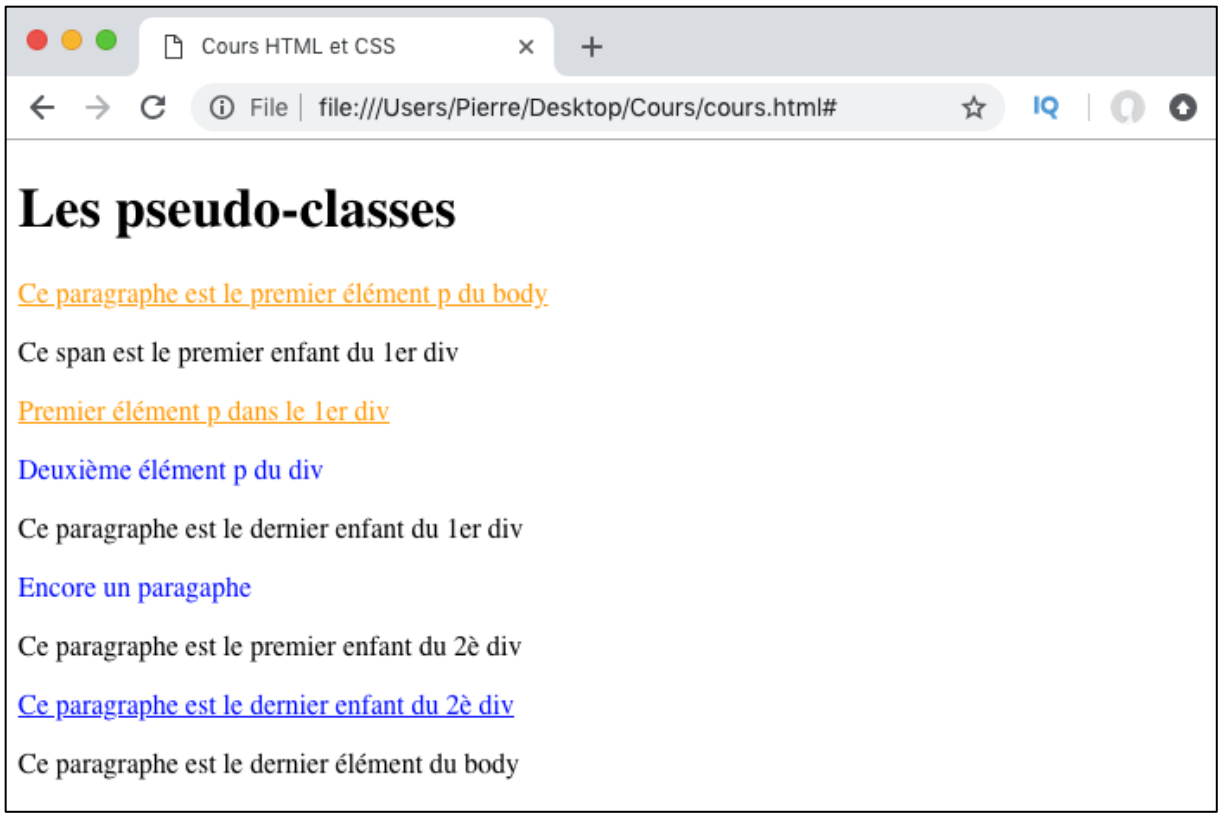
Les pseudo-classes :nth-child() et :nth-of-type()

Pour pouvoir cibler n'importe quel élément HTML en fonction de sa place dans le document, nous allons également pouvoir utiliser les sélecteurs CSS `:nth-child(n)` et `nth-of-type(n)` qui vont nous permettre de sélectionner un élément HTML qui est le énième enfant de son parent ou un élément qui est le énième enfant de ce type de son parent.

Par exemple, le sélecteur CSS `p:nth-child(2)` va nous permettre de sélectionner tous les éléments `p` qui sont les deuxièmes enfants de leur parent tandis que le sélecteur `p:nth-of-type(2)` sélectionne tous les paragraphes qui sont les deuxièmes enfants de type `p` de leur parent.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Les pseudo-classes</h1>
        <p>Ce paragraphe est le premier élément p du body</p>

        <div>
            <span>Ce span est le premier enfant du 1er div</span>
            <p>Premier élément p dans le 1er div</p>
            <p>Deuxième élément p du div</p>
            <p>Ce paragraphe est le dernier enfant du 1er div</p>
        </div>

        <p>Encore un paragaphe</p>

        <div>
            <p>Ce paragraphe est le premier enfant du 2è div</p>
            <p>Ce paragraphe est le dernier enfant du 2è div</p>
        </div>
        <p>Ce paragraphe est le dernier élément du body</p>
    </body>
</html>
```

```css
p:nth-child(2){
    color: orange;
    text-decoration: underline;
}

p:nth-of-type(2){
    color: blue;
}
```



Là encore, si plusieurs pseudo-classes ciblent le même élément (que ce soit :first-child, :nth-child(), :nth-of-type() ou :last-child, etc. alors en cas de conflit les styles de la dernière pseudo-classe déclarée s'appliqueront.

Liste complète des pseudo-classes et définition

Vous pourrez trouver ci-dessous toutes les pseudo-classes avec la description de leur comportement et la version CSS dans laquelle elles sont passées comme recommandation pour référence.

Les pseudo-classes dont la version est « 3/4 » sont celles dont la définition initiale a été posée avec le CSS3 mais dont la définition va certainement être modifiée avec la prochaine version du CSS.

De plus, notez que certaines pseudo-classes ont été créées pour ne fonctionner qu'avec certains éléments en particulier et notamment avec des éléments de formulaire que nous étudierons plus tard.

Sélecteur	Description	Version CSS
E:link	Sélectionne tout élément E représentant l'ancre d'un lien non visité jusqu'à présent	CSS1

Sélecteur	Description	Version CSS
E:visited	Sélectionne tout élément E représentant l'ancre d'un lien déjà visité	CSS1
E:active	Sélectionne un élément E au moment où il est cliqué	CSS1
E:hover	Sélectionne un élément E lorsque le curseur de la souris passe dessus	CSS2
E:focus	Sélectionne un élément E qui a le focus (dans lequel le curseur de la souris est placé)	CSS2
E:first-child	Sélectionne tout élément E étant le premier enfant de son parent	CSS2
E:lang(fr)	Sélectionne tout élément E dont l'attribut langage possède la valeur « fr »	CSS2
E:target	Sélectionne un élément E contenant une ancre qui vient d'être cliquée à partir d'un lien ancre	CSS3
E:enabled	Sélectionne tout élément E avec lequel l'utilisateur peut interagir et qui est activé	CSS3
E:disabled	Sélectionne tout élément E avec lequel l'utilisateur peut interagir et qui est désactivé	CSS3
E:root	Sélectionne un élément E racine du document	CSS3
E:empty	Sélectionne tout élément E qui ne possède pas d'enfant (ni de nœud de type texte)	CSS3
E:nth-child(n)	Sélectionne tout élément E étant le n-ième enfant de son parent	CSS3
E:nth-last-child(n)	Sélectionne tout élément E étant le n-ième enfant de son parent en comptant les enfants à partir du dernier	CSS3
E:checked	Sélectionne tout élément E de type input coché au sens large (checked ou selected)	CSS3
E:last-child	Sélectionne tout élément E étant le dernier enfant de son parent	CSS3
E:only-child	Sélectionne tout élément E qui est le seul enfant de son parent	CSS3

Sélecteur	Description	Version CSS
E:nth-of-type(n)	Sélectionne tout élément E étant le n-ième enfant d'un certain type par rapport à son parent	CSS3
E:nth-last-of-type(n)	Sélectionne tout élément E étant le n-ième enfant d'un certain type par rapport à son parent en comptant à partir de la fin	CSS3
E:first-of-type	Sélectionne tout élément E premier enfant de son type par rapport à son parent	CSS3
E:last-of-type	Sélectionne tout élément E dernier enfant de son type par rapport à son parent	CSS3
E:only-of-type	Sélectionne tout élément E seul enfant de son type par rapport à son parent	CSS3
E:read-write	Sélectionne tout élément E de type input avec lequel l'utilisateur peut interagir (comme un champ dans lequel il peut écrire par exemple)	3-UI/4
E:read-only	Sélectionne tout élément E de type input avec lequel l'utilisateur ne peut pas interagir (éléments possédant un attribut disabled par exemple)	3-UI/4
E:placeholder-shown	Sélectionne tout élément E qui affiche actuellement la valeur de son attribut placeholder	3-UI/4
E:default	Sélectionne un élément E dans une liste ou un groupe qui est l'élément défini par défaut	3-UI/4
E:valid	Sélectionne tout élément E de type input dont la valeur est évaluée comme valide (dont la valeur possède une forme correspondant à ce qui est attendu)	3-UI/4
E:invalid	Sélectionne tout élément E de type input dont la valeur est évaluée comme invalide (valeur ne correspondant pas à ce qui est attendu)	3-UI/4
E:in-range	Sélectionne tout élément E de type input dont la valeur fournie se situe dans une fourchette de valeurs prédéfinies	3-UI/4
E:out-of-range	Sélectionne tout élément E de type input dont la valeur fournie se situe en dehors d'une certaine fourchette de valeurs prédéfinies	3-UI/4

Sélecteur	Description	Version CSS
E:required	Sélectionne tout élément E de type input dont la valeur doit être renseignée (élément possédant un attribut required)	3-UI/4
E:optional	Sélectionne tout élément E de type input dont la valeur ne doit pas obligatoirement être renseignée	3-UI/4
E:not(E1, .c1)	Sélectionne tout élément E qui n'est pas de type E1 et qui ne possède pas d'attribut class= »c1″	3/4

Notez qu'il existe encore d'autres pseudo-classes très particulières comme :first (pour gérer l'impression de la première page d'un document) ou qui ne sont pas encore au statut de recommandation mais qui sont pour le moment en développement comme dir(), scope, drop, indeterminate, etc.

Les pseudo-éléments CSS

Les pseudo-éléments CSS vont nous permettre de ne cibler qu'une partie d'un élément HTML pour lui appliquer des styles. Il convient de ne pas les confondre avec les pseudo-classes qui elles servent à cibler un élément selon son état.

Les pseudo-éléments ont une syntaxe particulière puisqu'il faudra utiliser un double deux-points `::` entre le sélecteur élément et le mot clef désignant la partie de l'élément qui va être ciblé.

Il existe aujourd'hui 4 pseudo-éléments qui sont des recommandations du W3C :

- `::first-letter` ;
- `::first-line` ;
- `::before` ;
- `::after`.

Il existe également un cinquième pseudo-élément `::selection` qu'il convient de connaitre car celui-ci devait faire partie des recommandations du CSS3 puis a finalement été abandonné et est de nouveau candidat aujourd'hui.

Insérer du contenu avant ou après le contenu d'un élément HTML

Les pseudo-éléments `::before` et `::after` vont nous permettre respectivement de cibler l'emplacement avant et après le contenu d'un élément HTML.

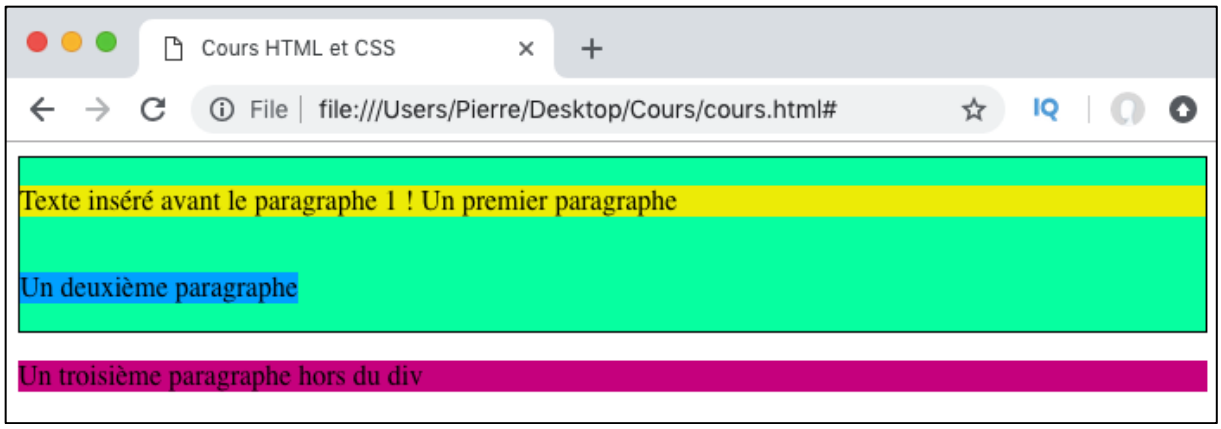
Dans l'immense majorité des cas nous allons utiliser ces pseudo-éléments de concert avec la propriété CSS `content` pour ajouter du texte avant ou après le contenu de l'élément.

Le pseudo-élément `::after` va également nous permettre de réaliser le clearfix CSS pour qu'un élément parent contienne ses enfants flottants. En effet, en appliquant `::after` à un élément contenant des éléments flottants, on va créer un pseudo-élément qui sera le dernier enfant du parent.

Il suffit ensuite d'appliquer un `clear` à ce pseudo élément tout en lui attribuant une boite de contenu avec `content` et en lui définissant un type d'affichage avec `display`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur">
            <p class="p1">Un premier paragraphe</p>
            <p class="p2">Un deuxième paragraphe</p>
        </div>

        <p class="p3">Un troisième paragraphe hors du div</p>
    </body>
</html>
```

```css
.conteneur{
  border: 1px solid black;
  box-sizing: border-box;
  background-color: #0FA;
}
.p1{
  background-color: #EE0;
}
.p2{
  float: left;
  background-color: #0AF;
}
.p3{
  background-color: #C08;
}
.p1::before{
  content: "Texte inséré avant le paragraphe 1 ! ";
}
.conteneur::after{
  display: table;
  content: "";
  clear: both;
}
```



Sélectionner la première lettre d'un élément en CSS

Le pseudo-élément `::first-letter` va nous permettre de sélectionner uniquement la première lettre du contenu d'éléments HTML.

Nous allons ensuite pouvoir appliquer cette lettre toutes sortes de mise en forme. Par exemple, pour sélectionner la première lettre de tous les paragraphes d'une page, nous utiliserons le sélecteur CSS `p::first-letter`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Le jour de ses onze ans, Harry Potter, un orphelin élevé...</p>
    </body>
</html>
```

```
p::first-letter{
    font-size: 40px;
    font-weight: bold;
    color: gold;
    font-family: Harrington, Zapfino;
}
```

Sélectionner la première ligne d'un élément en CSS

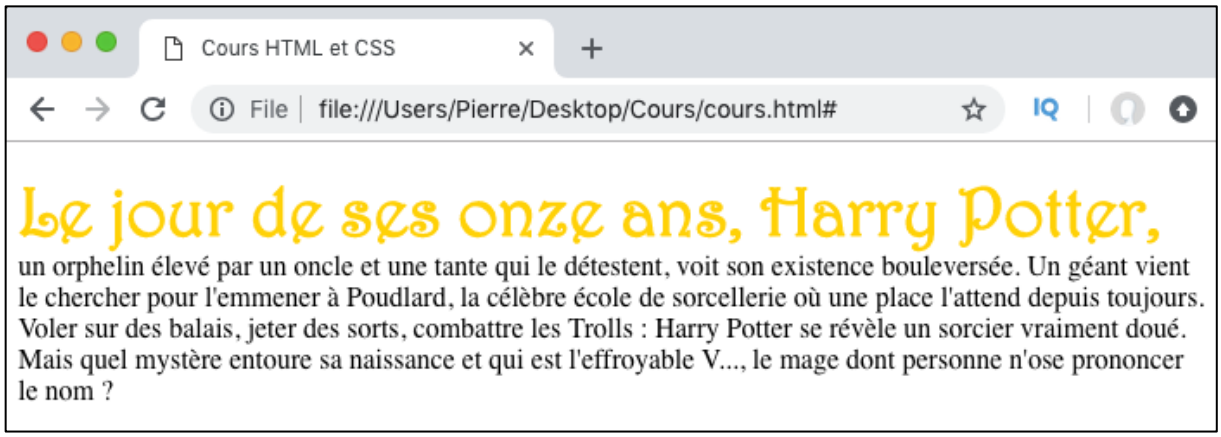
De manière analogue au pseudo-élément ::first-letter, nous allons pouvoir sélectionner uniquement la première ligne d'un élément HTML grâce au pseudo-élément ::first-line pour lui appliquer des styles.

Attention, la mise en forme va ici être dépendante de l'écran de chacun de vos visiteurs puisque la « première ligne » correspond ici à la première ligne dans le rendu visuel de la page de vos visiteurs.

Ainsi, pour sélectionner la première ligne de tous les paragraphes de notre page nous utiliserons le sélecteur CSS p::first-line.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Le jour de ses onze ans, Harry Potter, un orphelin élevé par un oncle
        et une tante qui le détestent, voit son existence bouleversée. Un géant
        vient le chercher pour l'emmener à Poudlard, la célèbre école de
        sorcellerie où une place l'attend depuis toujours. Voler sur des
        balais, jeter des sorts, combattre les Trolls : Harry Potter se révèle
        un sorcier vraiment doué. Mais quel mystère entoure sa naissance et qui
        est l'effroyable V..., le mage dont personne n'ose prononcer le nom ?</p>
    </body>
</html>
```

```css
p::first-line{
    font-size: 40px;
    font-weight: bold;
    color: gold;
    font-family: Harrington, Zapfino;
}
```



Cibler la partie d'un élément déjà sélectionnée par l'utilisateur

Finalement, nous allons pouvoir appliquer des styles particuliers à certaines parties de nos éléments HTML en fonction de ce qui est déjà présélectionné par l'utilisateur grâce au pseudo-élément `::selection`.

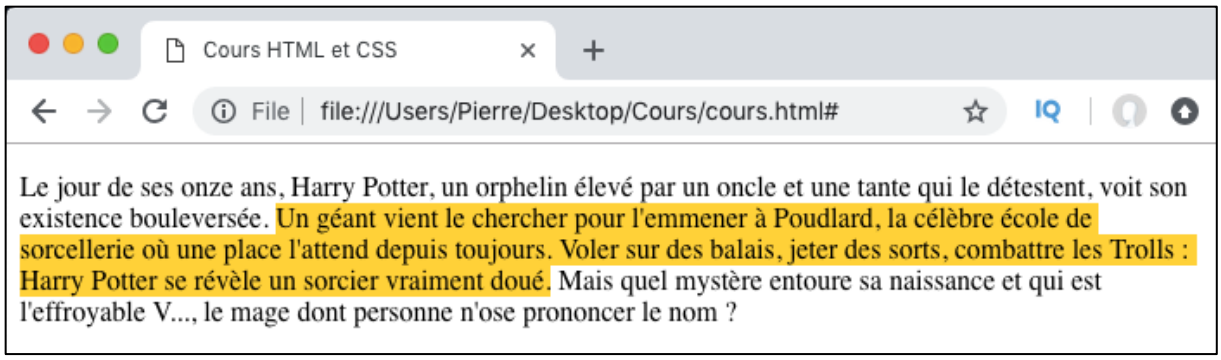
Les styles n'apparaitront que sur la partie de l'élément sélectionnée par un utilisateur. Pour sélectionner et appliquer des mises en forme particulières à la partie sélectionnée de tous les paragraphes d'une page, nous utiliserons donc le sélecteur CSS `p::selection`.

Attention toutefois à la compatibilité et à l'utilisation de ce sélecteur : celui-ci devait être intégré à la sortie du CSS3 mais a finalement été abandonné. Il fera normalement parti des recommandations pour le CSS4.

Je vous en parle ici car le fait que ce pseudo-élément devait faire partie des recommandations du CSS3 fait que la plupart des navigateurs le supportent déjà bien.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <p>Le jour de ses onze ans, Harry Potter, un orphelin élevé par un oncle
        et une tante qui le détestent, voit son existence bouleversée. Un géant
        vient le chercher pour l'emmener à Poudlard, la célèbre école de
        sorcellerie où une place l'attend depuis toujours. Voler sur des
        balais, jeter des sorts, combattre les Trolls : Harry Potter se révèle
        un sorcier vraiment doué. Mais quel mystère entoure sa naissance et qui
        est l'effroyable V..., le mage dont personne n'ose prononcer le nom ?</p>
    </body>
</html>
```

```css
p::selection{
    background-color: gold;
}
```


Cours HTML et CSS
File | file:///Users/Pierre/Desktop/Cours/cours.html#
Le jour de ses onze ans, Harry Potter, un orphelin élevé par un oncle et une tante qui le détestent, voit son existence bouleversée. Un géant vient le chercher pour l'emmener à Poudlard, la célèbre école de sorcellerie où une place l'attend depuis toujours. Voler sur des balais, jeter des sorts, combattre les Trolls : Harry Potter se révèle un sorcier vraiment doué. Mais quel mystère entoure sa naissance et qui est l'effroyable V..., le mage dont personne n'ose prononcer le nom ?

EXERCICE #1 : Création d'un menu horizontal sticky

Au cours des dernières leçons, nous avons vu beaucoup de nouveaux concepts HTML et CSS. Il est maintenant temps de mettre ce qu'on a vu en application en créant un menu simple horizontal entièrement en HTML et en CSS.

Étant donné que c'est notre premier « vrai » exercice, nous allons progresser ensemble. Commençons avec un conseil d'ordre général : lorsqu'on se lance dans un projet de code, il est rarement bénéfique de commencer immédiatement à coder. Au contraire, on commencera généralement par clarifier ce qu'on souhaite obtenir et par définir les différents langages, objets, etc. que nous allons devoir utiliser pour arriver au résultat voulu.

Procéder comme cela limite les risques d'avoir à revenir en arrière et à réécrire son code au milieu du projet car on n'avait pas pensé à ceci ou cela et fait au final gagner beaucoup de temps !

Ici, le projet est très simple puisqu'on souhaite simplement créer un menu horizontal simple. Pour créer des menus, nous allons utiliser des éléments de liste `ul` que nous allons ensuite mettre en forme en CSS.

Squelette HTML du menu

Nous allons commencer par créer le squelette de notre menu en HTML avant de le mettre en forme en CSS.

Ici, nous allons utiliser une liste `ul` qui va représenter notre menu. Chaque élément de liste `li` va représenter un onglet de notre menu.

Comme un menu est utilisé pour naviguer entre les pages d'un site, il va falloir que nos onglets de menu soient cliquables. Nous allons donc ajouter des éléments `a` dans nos éléments de liste. Pour cet exercice, je laisserai mes liens vides en écrivant `href="#"`.

En pratique, nous devrons également placer notre menu principal de navigation dans un élément structurant `nav` qui sert à indiquer aux navigateurs et moteurs de recherche que ce qui est dans l'élément `nav` est notre menu principal de navigation.

Voici donc le code HTML de notre menu et le résultat dans le navigateur :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <nav>
            <ul>
                <li><a href="#">Cours Complets</a></li>
                <li><a href="#">Articles</a></li>
                <li><a href="#">Contact</a></li>
                <li><a href="#">A propos</a></li>
            </ul>
        </nav>
    </body>
</html>
```

Pour le moment, il s'agit visuellement d'une simple liste non ordonnée et cela ne ressemble pas à un menu. C'est tout à fait normal : je vous rappelle que la mise en page est du ressort du CSS.

Mise en forme du menu en CSS

Nous allons maintenant transformer notre liste HTML en quelque chose qui ressemble visuellement à un menu en CSS.

Ici, nous allons déjà commencer par effectuer un reset des marges intérieures et extérieures des différents éléments de notre page pour nous assurer d'avoir un affichage cohérent d'un navigateur à l'autre. Profitons-en également pour définir une liste de polices d'écriture pour notre page.

```css
/*Reset CSS*/
*{
    margin: 0px;
    padding: 0px;
    font-family: Avenir, sans-serif;
}
```

Ensuite, nous allons enlever les puces devant nos éléments de liste. Nous allons pouvoir faire cela en utilisant la propriété `list-style-type` avec sa valeur `none`.

Ici, nous allons cibler uniquement la liste de notre menu avec le sélecteur `nav ul` afin de ne pas enlever les puces des autres listes qui pourraient potentiellement être affichées dans notre page.

```
nav ul{
    list-style-type: none;
}
```

A partir d'ici, nous avons plusieurs solutions en CSS pour mettre nos éléments de liste sur la même ligne. Les deux solutions fonctionnant le mieux ici vont être d'ajouter soit un `display : inline-block`, soit un `float : left` à nos éléments de liste.

Ici, je vais plutôt opter pour un `float : left` pour n'avoir aucune espace entre mes éléments de liste et faciliter la mise en forme.

En effet, je vous rappelle qu'une espace va se créer entre différents éléments possédant un `display : inline-block` si ces éléments ne sont pas collés dans votre code.

Cette espace, généralement de `4px`, est l'équivalent d'une espace insécable. On peut le supprimer en collant les différents éléments dans le code.

Comme on applique un `float : left` à tous nos éléments de liste, nous allons également utiliser le clearfix avec le pseudo-élément `::after` que nous allons appliquer à la liste en soi pour éviter que sa hauteur ne soit nulle.

```
nav li{
    float: left;
}

nav ul::after{
    content: "";
    display: table;
    clear: both;
}
```

C'était l'étape la plus complexe du menu. Nous allons ensuite espacer nos différents onglets de menu. Pour cela, on va attribuer une largeur égale à 100% divisée par le nombre d'éléments de menu à chaque élément de menu.

Nous allons également pouvoir centrer le contenu textuel de chacun de nos éléments de menu dans l'élément de menu en appliquant un `text-align : center` aux différents éléments.

```
nav li{
    float: left;
    width: 25%;
    text-align: center;
}
```

Note : Ici, notre élément nav occupe 100% de la largeur disponible par défaut et on dirait donc que le menu est centré dans la page. Cependant, ce n'est pas le cas (il suffit de lui attribuer une largeur plus petite pour s'en apercevoir). Pour centrer le menu, on pourra appliquer une largeur explicite à l'élément nav ainsi que margin : 0 auto.

```
nav{
    width: 100%;
    margin: 0 auto;
}
```

Une fois arrivé ici, on s'aperçoit d'un problème d'ergonomie : seul le texte est cliquable et non pas tout l'espace dans chaque élément de menu. Cela est dû au fait que l'élément a est un élément inline. Changeons donc ce comportement par défaut et profitons-en pour enlever le trait de soulignement et pour changer la couleur de nos textes.

```
nav a{
    display: block;
    text-decoration: none;
    color: black;
}
```

Ensuite, nous allons vouloir mettre en relief un élément de menu lorsque celui-ci est survolé par l'utilisateur. Nous allons pouvoir faire cela en utilisant la pseudo-classe :hover. On va par exemple pouvoir changer la couleur du texte et ajouter une bordure basse lorsqu'un utilisateur passe sa souris sur un élément de menu.

```
nav a:hover{
    color: orange;
    border-bottom: 2px solid gold;
}
```

Ici, on se retrouve face à un autre problème d'ergonomie : en effet, en ajoutant une bordure durant l'état :hover, la hauteur de l'élément est modifiée de la taille de la bordure et cela va faire bouger les différents contenus sous le menu.

La façon la plus simple de résoudre ce problème est d'ajouter une bordure de même taille mais de couleur transparente à nos éléments a dans leur état normal. Ainsi, les éléments auront toujours la même hauteur.

```
nav a{
    display: block;
    text-decoration: none;
    color: black;
    border-bottom: 2px solid transparent;
}
```

Finalement, on va pouvoir aérer notre menu en hauteur. Ici, je vous propose d'ajouter des padding haut et bas de 10px à nos éléments a. Cela aura également pour effet de séparer la bordure basse de ces éléments de leur contenu pour un meilleur rendu visuel.

```css
nav a{
    display: block;
    text-decoration: none;
    color: black;
    border-bottom: 2px solid transparent;
    padding: 10px 0px;
}

nav a:hover{
    color: orange;
    border-bottom: 2px solid gold;
}
```

Pour rendre enfin notre menu sticky, il va suffire d'ajouter une `position :sticky` à notre élément `nav` avec une propriété `top :0px` si on souhaite que le menu reste collé en haut de la page. Nous allons également en profiter pour ajouter une couleur de fond au menu.

Notez que pour constater l'effet du positionnement sticky il va falloir pouvoir descendre dans la page. N'hésitez donc pas à ajouter un conteneur `div` sous le menu et à lui donner une hauteur de 2000px par exemple.

```css
nav{
    width: 100%;
    margin: 0 auto;
    background-color: white;
    position: sticky;
    top: 0px;
}
```

Voilà tout pour ce menu simple créé en HTML et en CSS. N'hésitez pas à modifier ou à ajouter des styles à ce menu ! Voici le résultat auquel vous devriez arriver :


Cours HTML et CSS
File | /Users/Pierre/Desktop/cours.html#ancre
Menu horizontal sticky HTML et CSS
Cours Complets
Articles
Contact
A propos
Du contenu sous le menu

EXERCICE #2 : Création d’un menu déroulant

Dans l’exercice précédent, nous avons réussi à créer un menu horizontal simple en HTML et en CSS. Je vous propose maintenant de transformer ce menu pour créer un menu déroulant, c’est-à-dire un menu comportant plusieurs niveaux.

Créer un menu déroulant en HTML et en CSS va s’avérer un peu plus complexe que de créer un menu simple à cause notamment des questions de positionnement qu’il va falloir régler.

Dans cet exercice, nous allons rester simple et nous contenter seulement d’un deuxième niveau de menu. Notez que de manière générale il est déconseillé d’aller plus loin que cela pour des raisons d’ergonomie de votre site.

L’idée ici va donc être de créer des sous-menus. Chaque sous-menu va être lié à un onglet du menu principal et ne va devoir apparaitre que lorsqu’un utilisateur passe sa souris sur l’onglet en question.

Squelette HTML du menu déroulant

Nous allons commencer par récupérer le code HTML de notre menu simple vu à la leçon précédente et lui rajouter des sous-menus.

Pour représenter nos sous-menus, nous allons simplement imbriquer une nouvelle liste dans les différents éléments de notre liste principale.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <nav>
            <ul>
                <li class="deroulant"><a href="#">Cours Complets  </a>
                    <ul class="sous">
                        <li><a href="#">Cours HTML et CSS</a></li>
                        <li><a href="#">Cours JavaScript</a></li>
                        <li><a href="#">Cours PHP et MySQL</a></li>
                    </ul>
                </li>
                <li class="deroulant"><a href="#">Articles  </a>
                    <ul class="sous">
                        <li><a href="#">CSS display</a></li>
                        <li><a href="#">CSS position</a></li>
                        <li><a href="#">CSS float</a></li>
                    </ul>
                </li>
                <li><a href="#">Contact</a></li>
                <li><a href="#">A propos</a></li>
            </ul>
        </nav>
    </body>
</html>
```

Ici, nous créons un deuxième niveau de menu sous nos deux premiers onglets de menu. Notez que j'ai rajouté des attributs `class` et des entités HTML ` ` dans le code (qui servent à créer un espace double). Cela va nous être utile plus tard en CSS.

Mise en forme du menu déroulant en CSS

Ici, nous allons essayer de nous appuyer un maximum sur les styles du menu horizontal simple créé dans l'exercice précédent en les complétant. Vous pouvez donc déjà commencer par récupérer les styles CSS du menu créé précédemment :

```css
/*Reset CSS*/
*{
    margin: 0px;
    padding: 0px;
    font-family: Avenir, sans-serif;
}

nav{
    width: 100%;
    margin: 0 auto;
    background-color: white;
    position: sticky;
    top: 0px;
}

nav ul{
    list-style-type: none;
}

nav ul li{
    float: left;
    width: 25%;
    text-align: center;
}

nav ul::after{
    content: "";
    display: table;
    clear: both;
}

nav a{
    display: block;
    text-decoration: none;
    color: black;
    border-bottom: 2px solid transparent;
    padding: 10px 0px;
}

nav a:hover{
    color: orange;
    border-bottom: 2px solid gold;
}
```

On va donc déjà vouloir par défaut cacher les sous menus et ne les afficher que lorsqu'un utilisateur passe sa souris par-dessus l'onglet de menu correspondant. Nous allons donc appliquer un `display : none` par défaut à nos sous-menus et un `display : block` lorsqu'un utilisateur passe sa souris sur l'onglet du menu principal correspondant.

On va également en profiter pour ajouter une couleur de fond aux sous-menus ainsi qu'une ombre légère autour des sous-menus pour qu'ils se distinguent du reste de la page avec `box-shadow`.

```css
.sous{
    display: none;
    box-shadow: 0px 1px 2px #CCC;
    background-color: white;
}
nav > ul li:hover .sous{
    display: block;
}
```

Ici, notez qu'on peut placer les styles généraux de mes sous-menus soit dans le sélecteur `nav > ul li:hover .sous`, soit dans le sélecteur `.sous`. Par défaut, on préfèrera placer les styles dans le sélecteur le plus simple et général.

Le sélecteur `nav > ul li:hover .sous` peut sembler complexe à première vue. Il sert à cibler un sous menu `.sous` lorsqu'un utilisateur passe sa souris sur un éléments `li` de la liste représentant notre menu principal.

Ensuite, on va vouloir que les éléments de nos sous-menus s'affichent cette fois-ci les uns en dessous des autres, occupent tout l'espace disponible dans leur conteneur et que le texte soit aligné à gauche. On va donc annuler le flottement hérité des onglets du menu principal.

```css
.sous li{
    float: none;
    width: 100%;
    text-align: left;
}
```

On va également déjà en profiter pour mettre en forme les onglets de nos sous-menus et s'appuyant une nouvelle fois sur les styles déjà créées. Ici, on va se contenter de supprimer la bordure basse héritée du menu principal, d'ajouter une marge interne et une couleur de fond lorsqu'on passe la souris sur un élément.

```css
.sous a{
    padding: 10px;
    border-bottom: none;
}
.sous a:hover{
    border-bottom: none;
    background-color: RGBa(200,200,200,0.1);
}
```

On va aussi montrer aux utilisateurs que notre menu est déroulant en ajoutant une petite flèche à côté des textes des onglets de menu qui contiennent des sous-menus. Pour cela, on va insérer le symbole ▼après le texte en utilisant le pseudo-élément `::after`, ce qui va également nous permettre de définir la taille du symbole.

```css
.deroulant > a::after{
    content:" ▼";
    font-size: 12px;
}
```

Une fois arrivé ici, il nous reste un détail technique à régler qui est la partie complexe de ce menu : pour le moment, lorsqu'on affiche un sous-menu, le contenu situé sous le menu est poussé vers le bas ce qui n'est pas le comportement souhaité.

On va pouvoir régler cela en appliquant une `position : absolute` à nos sous-menus, ce qui va avoir pour effet de les retirer du flux normal de la page. Les sous-menus n'impacteront plus le contenu suivant le menu.

Pour que les sous-menus ne passent pas derrière le contenu qui suit le menu, nous allons également définir un `z-index` élevé. Nous allons également définir la taille des sous-menus à 100%.

```
.sous{
    display: none;
    box-shadow: 0px 1px 2px #CCC;
    background-color: white;
    position: absolute;
    width: 100%;
    z-index: 1000;
}
```

Ensuite, pour replacer les sous-menus correctement sous leur onglet correspondant et pour qu'ils fassent bien la même taille, on va également appliquer une `position : relative` aux éléments du menu principal.

```
nav ul li{
    float: left;
    width: 25%;
    text-align: center;
    position: relative;
}
```

Voilà tout pour cet exercice ! Nous disposons maintenant d'un menu déroulant totalement fonctionnel. Vous devriez parvenir au résultat suivant :

PARTIE XI

Formulaires HTML

Présentation des formulaires HTML

Nous allons aborder dans cette nouvelle partie une partie essentielle du HTML qui va consister en la création de formulaires.

Nous allons donc définir ensemble ce qu'est un formulaire HTML et à quoi vont servir les formulaires puis nous allons apprendre à créer des formulaires plus ou moins complexes.

Définition et rôle des formulaires HTML

Les formulaires HTML vont permettre à nos visiteurs de nous envoyer des données que nous allons ensuite pouvoir manipuler et / ou stocker.

Nous allons utiliser les formulaires pour permettre à des utilisateurs de s'inscrire sur notre site (formulaires d'inscription), de se connecter (formulaire de connexion), de nous envoyer des messages (formulaire de contact), de laisser des commentaires, etc.

Vous l'aurez donc compris : les formulaires se révèlent indispensables et incontournables dans de nombreuses situations.

Les formulaires HTML vont pouvoir être composés de champs de texte (cas d'un champ de formulaire demandant à un utilisateur de renseigner son adresse mail pour se connecter ou pour s'inscrire sur le site par exemple), de listes d'options (choix d'un pays dans une liste de pays par exemple), de cases à cocher, etc.

Cependant, vous devez bien comprendre ici qu'on touche avec les formulaires aux limites du langage HTML. En effet, nous allons tout à fait pouvoir construire nos formulaires en HTML, mais le HTML ne va nous permettre ni de recueillir les données envoyées par nos visiteurs, ni de les manipuler, ni de les stocker.

Pour réaliser ces différentes opérations, il faudra utiliser d'autres types de langages comme le PHP (pour la manipulation des données) et le MySQL (pour le stockage) par exemple qui vont s'exécuter côté serveur.

L'élément form et ses attributs

Pour créer un formulaire, nous allons utiliser l'élément HTML `form`. Cet élément `form` va avoir besoin de deux attributs pour fonctionner normalement : les attributs `method` et `action`.

L'attribut `method` va indiquer comment doivent être envoyées les données saisies par l'utilisateur. Cet attribut peut prendre deux valeurs : `get` et `post`.

Que signifient ces deux valeurs et laquelle choisir ? Les valeurs `get` et `post` vont déterminer la méthode de transit des données du formulaire. En choisissant `get`, on indique que les données doivent transiter via l'URL (sous forme de paramètres) tandis

qu'en choisissant la valeur `post` on indique qu'on veut envoyer les données du formulaire via transaction post HTTP.

Concrètement, si l'on choisit l'envoi via l'URL (avec la valeur `get`), nous serons limités dans la quantité de données pouvant être envoyées et surtout les données vont être envoyées en clair. Évitez donc absolument d'utiliser cette méthode si vous demandez des mots de passe ou toute information sensible dans votre formulaire.

Cette méthode de transit est généralement utilisée lors de l'affichage de produits triés sur un site e-commerce (car oui, les options de tris sont des éléments de formulaires avant tout !). Regardez bien les URL la prochaine fois que vous allez sur un site e-commerce après avoir fait un tri : si vous retrouvez des éléments de votre tri dedans c'est qu'un `get` a été utilisé.

En choisissant l'envoi de données via post transaction HTTP (avec la valeur `post`), nous ne sommes plus limités dans la quantité de données pouvant être envoyées et les données ne sont visibles par personne. C'est donc généralement la méthode que nous utiliserons pour l'envoi véritablement de données que l'on souhaite stocker (création d'un compte client, etc.).

L'attribut `action` va lui nous servir à préciser l'adresse de la page qui va traiter les données. Je vous rappelle ici qu'on ne va pas pouvoir manipuler les données des formulaires en HTML mais que nous allons être obligé d'utiliser un autre langage comme le PHP pour cela.

Généralement, nous enverrons les données reçues vers un fichier PHP dont le but sera de traiter ces données justement. Nous n'allons bien évidemment pas créer ce fichier dans ce cours, mais je vous propose pour les exemples suivants de faire « comme si » ce fichier existait. Nous pourrons l'appeler `form.php`.

Création d'un premier formulaire simple en HTML

Il est temps de passer à la pratique et de créer un premier formulaire simple en HTML. Dans ce formulaire, nous allons demander trois choses à l'utilisateur :

- Un pseudonyme à l'utilisateur ;
- Un mot de passe ;
- Une adresse mail.

Création des champs de formulaire avec l'élément input

Pour faire cela, nous allons avoir besoin de créer trois champs de formulaires dans lesquels l'utilisateur pourra remplir les informations demandées. Pour créer ces champs, nous allons utiliser des éléments HTML `input`.

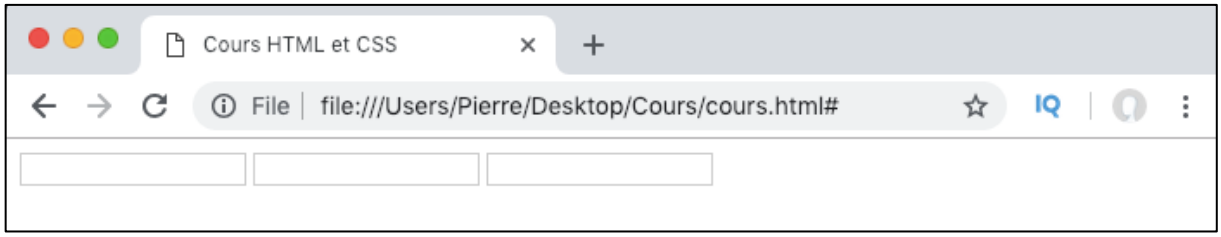
L'élément HTML `input` est un élément qui va permettre à l'utilisateur d'envoyer des données. Il se présente sous la forme d'une balise orpheline et va obligatoirement posséder un attribut `type` auquel on va pouvoir donner de nombreuses valeurs.

La valeur passée à l'attribut `type` va déterminer le type de données que l'utilisateur va pouvoir envoyer : un texte avec `input type="text"`, une adresse mail avec `input type="email"`, une date avec `input type="date"`, etc.

Si le format de données entrées par l'utilisateur ne correspond pas à ce qui est attendu, alors il ne pourra pas envoyer le formulaire et un message d'erreur s'affichera selon le navigateur utilisé.

Notez également que les navigateurs proposent aujourd'hui des présentations différentes en fonction du type d'`input` et notamment sur mobile : pour un `input type="date"`, par exemple, un calendrier sera souvent affiché pour aider l'utilisateur à choisir une date.
La base de notre formulaire va donc ressembler à cela :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <form method="post" action="form.php">
            <input type="text">
            <input type="password">
            <input type="email">
        </form>
    </body>
</html>
```



Comme vous pouvez le voir, nos trois champs de formulaires sont bien créés. Si vous placez le curseur de votre souris dans un champ et que vous commencez à écrire, vous devriez voir que le navigateur comprend bien le type de données attendu dans chaque champ : par exemple, le texte inscrit dans le champ demandant un mot de passe par exemple devrait s'afficher sous forme d'étoiles.

De plus, il est également possible que votre navigateur vous propose une auto-complétion des champs si vous avez déjà rempli des formulaires demandant des types de données similaires sur un autre site précédemment.

En l'état, notre formulaire n'est cependant pas exploitable pour la simple et bonne raison que l'utilisateur ne possède pas d'indication sur ce qu'il doit rentrer dans chaque champ et également car pour le moment notre formulaire ne possède pas de bouton d'envoi des données !

Ajout d'indications sur les données attendues avec l'élément label

Pour donner des indications sur les données attendues dans chaque champ aux utilisateurs, nous allons utiliser des éléments `label`. Cet élément va permettre d'attribuer un libellé (c'est-à-dire une étiquette ou une description) à chaque champ de notre formulaire.

Pour des raisons d'ergonomie et de cohérence, il est considéré comme une bonne pratique de lier chaque `label` à son `input` correspondant. Ainsi, lorsque l'utilisateur va cliquer sur le `label`, le curseur de la souris va automatiquement être placé dans l'`input` correspondant.

Pour lier un `label` et un `input` ensemble, nous allons attribuer un attribut `id` unique à chacun de nos éléments `input` pour les identifier de manière formelle et allons également rajouter un attribut `for` à chacun de nos `label` qui devra avoir la même valeur que l'`id` de l'`input` auquel il doit être lié.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <form method="post" action="form.php">
            <div class="champ">
              <label for="pseudo">Entrez un pseudonyme : </label>
              <input type="text" id="pseudo">
            </div>
            <div class="champ">
              <label for="pass">Entrez un mot de passe :</label>
              <input type="password" id="pass">
            </div>
            <div class="champ">
              <label for="mail">Entrez un mail :</label>
              <input type="email" id="mail">
            </div>
        </form>
    </body>
</html>
```

Ici, j'en ai profité pour placer mes couples `label` + `input` dans des `div` afin que chacun d'entre eux soit sur une ligne à eux et j'ai également passé une `margin-bottom : 10px` aux `div`.

Nous en avons fini avec les champs de notre premier formulaire HTML en soi et il ne nous reste donc plus qu'à nous préoccuper de l'envoi des données.

Ajout du bouton d'envoi de données

Pour permettre l'envoi de ces données vers notre page `form.php`, il va nous falloir créer un bouton de soumission de formulaire. Pour cela, nous allons à nouveau tout simplement utiliser un élément `input` mais cette fois-ci de type `input type="submit"`.

Cet `input` un peu spécial va se présenter sous forme de « bouton » et nous n'allons pas avoir besoin de lui ajouter de `label`. A la place, nous allons utiliser un attribut `value` et lui

passer en valeur le texte que doit contenir notre bouton (« soumettre », « valider », ou « envoyer » par exemple).

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <form method="post" action="form.php">
            <div class="champ">
              <label for="pseudo">Entrez un pseudonyme : </label>
              <input type="text" id="pseudo">
            </div>
            <div class="champ">
              <label for="pass">Entrez un mot de passe :</label>
              <input type="password" id="pass">
            </div>
            <div class="champ">
              <label for="mail">Entrez un mail :</label>
              <input type="email" id="mail">
            </div>
            <div class="champ">
                <input type="submit" value="Envoyer">
            </div>
        </form>
    </body>
</html>
```

Identification des données du formulaire

En l'état, notre formulaire n'est pas encore tout à fait utilisable. En effet, celui-ci fonctionne côté « façade » et va bien permettre à un utilisateur de remplir des données et de les envoyer.

Cependant, pour le moment, nous ne pourrions pas manipuler les données envoyées car elles n'ont pas été nommées : les données vont bien être envoyées mais il va être impossible pour nous de définir à quel champ correspond chaque donnée envoyée.

Pour identifier les données et pouvoir ensuite les manipuler en PHP ou autre, nous allons devoir ajouter un attribut `name` qui doit également posséder une valeur unique à chaque élément de formulaire demandant des données à l'utilisateur.

Notez bien ici que chaque attribut `name` doit posséder une valeur unique par rapport aux autres attributs `name` mais rien ne nous empêche de choisir la même valeur pour un attribut `name` et un attribut `id` par exemple puisqu'il n'y a pas de risque de confusion étant donné que ce sont deux attributs totalement différents.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <form method="post" action="form.php">
            <div class="champ">
              <label for="pseudo">Entrez un pseudonyme : </label>
              <input type="text" id="pseudo" name="pseudo">
            </div>
            <div class="champ">
              <label for="pass">Entrez un mot de passe :</label>
              <input type="password" id="pass" name="pass">
            </div>
            <div class="champ">
              <label for="mail">Entrez un mail :</label>
              <input type="email" id="mail" name="mail">
            </div>
            <div class="champ">
              <input type="submit" value="Envoyer">
             </div>
        </form>
    </body>
</html>
```

Ça y est, notre premier formulaire HTML est maintenant tout à fait fonctionnel. Expliquons rapidement la logique derrière ce formulaire, c'est-à-dire le fonctionnement de notre formulaire HTML.

Lorsqu'un utilisateur arrive sur notre page, il va remplir les différents champs de formulaire créés avec des éléments `input`.

Ensuite, il va cliquer sur le bouton d'envoi du formulaire. Les données du formulaire vont alors être envoyées via transaction HTTP de type `post` vers la page `form.php`.

Dans cette page `form.php`, nous allons ensuite pouvoir manipuler les différentes données du formulaire qu'on va pouvoir identifier grâce aux attributs `name` donnés à chaque `input`.

Encore une fois, l’objet de ce tutoriel n’est pas de présenter la manipulation des données en PHP ni le stockage mais bien de nous concentrer sur la création des formulaires en soi c’est-à-dire sur la partie HTML des formulaires.

Les éléments des formulaires

En HTML, nous avons de nombreux éléments spécifiques aux formulaires qui vont nous permettre de définir le formulaire en soi, de créer des zones de saisie de texte courtes ou longues, de proposer des zones d'options à nos utilisateurs, etc.

Dans la leçon précédente, nous nous sommes concentrés sur l'éléments `input` qui est de loin le plus utilisé car son attribut `type` permet de créer toutes sortes de champs et le rend très puissant mais il en existe bien d'autres.

L'objet de cette leçon va être de vous présenter la plupart des éléments de formulaire ainsi que les valeurs les plus courantes qu'on va pouvoir passer à l'attribut `type` de l'élément `input`.

Les éléments de formulaires HTML

Commençons avec la liste des éléments que l'on va pouvoir utiliser dans nos formulaires HTML ainsi que leur rôle :

Elément	Définition
form	Définit un formulaire
input	Définit un champ de données pour l'utilisateur
label	Définit une légende pour un élément input
textarea	Définit un champ de texte long
select	Définit une liste de choix
optgroup	Définit un groupe d'options dans une liste
option	Définit une option dans une liste
fieldset	Permet de regrouper les éléments d'un formulaire en différentes parties
legend	Ajoute une légende à un élément fieldset

L'élément input et les valeurs de l'attribut type

Commençons la présentation des éléments de formulaire avec un élément que nous connaissons : l'élément `input`.

Cet élément est l'élément à connaitre dans le contexte de la création de formulaires HTML puisqu'il va nous permettre de créer tous types de champs pour récolter des données utilisateurs.
Le type de champ va être défini par la valeur que l'on va donner à son attribut `type`. L'attribut `type` peut prendre de nombreuses valeurs. Voici les valeurs les plus utiles et les plus courantes :

Type d'input	Définition
text	Définit un champ de saisie monoligne qui accepte du texte
number	Définit un champ qui accepte un nombre décimal
email	Crée un champ qui permet de renseigner une adresse mail
date	Permet à l'utilisateur d'envoyer une date
password	Créer un champ de saisie monoligne dont la valeur va être cachée
checkbox	Permet de créer une case à cocher. L'utilisateur peut cocher une ou plusieurs cases d'un coup
radio	Permet de créer un bouton radio. Par définition, un seul bouton radio peut être coché dans un ensemble
url	Crée un champ qui accepte une URL
file	Permet à un utilisateur de télécharger un fichier
submit	Crée un bouton d'envoi des données du formulaire

L'élément textarea

Pour créer un champ de saisie classique dans nos formulaires, le premier réflexe va être d'utiliser un `input type="text"` et cela va marcher pour toutes les données de type « texte court ».

Le souci ici est qu'on ne va plus pouvoir utiliser d'`input type="text"` si on veut par exemple laisser la possibilité à un utilisateur de commenter quelque chose ou si on lui demande de se décrire car le texte qu'on va pouvoir placer dans un `input type="text"` est limité en taille.

Dans le cas où on souhaite laisser la possibilité à un utilisateur d'écrire un texte long, on utilisera plutôt un élément `textarea` qui définit un champ de texte long.

Les éléments select, option et optgroup

L'élément `select` va nous permettre de créer une liste d'options déroulante. Dans cette liste, nous allons placer autant d'éléments `option` que l'on souhaite donner de choix aux utilisateurs.

Les listes d'options forcent l'utilisateur à faire un choix unique dans la liste et sont généralement utilisées lorsqu'on souhaite proposer de nombreux choix, comme par exemple dans le cas où on demande à l'utilisateur de choisir son pays de résidence.

L'élément `optgroup` va nous permettre d'ordonner notre liste d'options et groupant des options. Par exemple, dans une liste de choix de pays, on pourrait grouper les différents pays par continent.

Les éléments fieldset et legend

Les éléments `fieldset` et `legend` vont nous permettre de structurer et d'améliorer la sémantique d'un formulaire.

L'élément `fieldset` va nous permettre de grouper plusieurs champs dans un formulaire pour l'organiser en parties.

L'élément `legend` va nous permettre d'ajouter une légende à une partie de notre formulaire déterminée par `fieldset` pour expliquer son but par exemple.

Création d'un formulaire HTML complet

Continuons à pratiquer et utilisons certaines des valeurs vues précédemment pour créer un formulaire HTML plus complet que le premier formulaire que nous avons créé.

Nous allons cette fois-ci demander à l'utilisateur de nous transmettre les informations suivantes via un formulaire qui va être divisé en 3 sections :

Section 1 :

- Son nom de famille ;
- Son prénom ;
- Son adresse mail ;
- Son âge ;
- Son sexe ;
- Son pays de résidence.

Section 2 :

- Ses compétences parmi une liste de choix ;
- Une description d'une expérience professionnelle passée / d'un problème résolu.

Section 3 :

- Un mot de passe – obligatoire.

Commençons donc par définir les sections de notre formulaire avec des éléments `fieldset` et donnons-leur une légende avec l'élément `legend`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <form method="post" action="form.php">
            <fieldset>
                <legend>Informations personnelles :</legend>
            </fieldset>
            <fieldset>
                <legend>Compétences / expérience :</legend>
            </fieldset>
            <fieldset>
                <legend>Validation :</legend>
            </fieldset>
          </form>
    </body>
</html>
```

Ensuite, nous allons créer les différents champs section par section et nous nous occuperons de la mise en forme à la toute fin. Notre première section de formulaire contient 6 champs :

- Un premier champ demandant le nom de famille. On utilisera donc un `input type="text"` ;
- Un deuxième champ demandant le prénom, obligatoire également. On utilisera à nouveau un `input type="text"`
- Un troisième champ demandant une adresse mail. On utilisera ici un `input type="email"` ;
- Un quatrième champ demandant un âge. On utilisera un `input type="number"` ;
- Un cinquième champ qui va demander le sexe de l'utilisateur. Ici, on va proposer un choix parmi deux boutons radio « homme » et « femme ». On utilisera ici des boutons radio plutôt qu'un `input type="text"` pour recevoir des données bien formatés (choix 1 ou choix 2) plutôt que des résultats comme « homme », « masculin », « garçon », « mec », « fille », « femme », etc. qui seraient très difficile à classer par la suite ;
- Un sixième champ qui va permettre à l'utilisateur d'indiquer son pays de résidence. Ici, nous allons proposer une liste d'options avec une liste préconstruite de pays parmi lesquels l'utilisateur devra choisir. Nous allons également classer les pays par continent.

```
<fieldset>
  <legend>Informations personnelles :</legend>
  <div class="champ">
    <label for="nom">Nom de famille :</label>
    <input type="text" id="nom" name="nom">
  </div>
  <div class="champ">
    <label for="prenom">Prénom :</label>
    <input type="text" id="prenom" name="prenom">
  </div>
  <div class="champ">
    <label for="mail">Adresse mail :</label>
    <input type="email" id="mail" name="mail">
   </div>
   <div class="champ">
     <label for="age">Age :</label>
     <input type="number" id="age" name="age">
   </div>
  <div class="champ">
    <input type="radio" id="h" name="sexe" value="homme">
    <label for="h">Homme</label>
    <input type="radio" id="f" name="sexe" value="femme">
    <label for="f">Femme</label>
   </div>
  <div class="champ">
    <label for="pays">Pays de résidence :</label>
    <select id="pays" name="pays">
      <optgroup label="Europe">
        <option value="france">France</option>
        <option value="belgique">Belgique</option>
        <option value="suisse">Suisse</option>
       </optgroup>
      <optgroup label="Afrique">
        <option value="algerie">Algérie</option>
        <option value="tunisie">Tunisie</option>
        <option value="maroc">Maroc</option>
        <option value="madagascar">Madagascar</option>
        <option value="benin">Bénin</option>
        <option value="togo">Togo</option>
       </optgroup>
      <optgroup label="Amerique">
        <option value="canada">Canada</option>
      </optgroup>
    </select>
  </div>
</fieldset>
```

Regardons de plus près le code ci-dessus pour noter les choses intéressantes. Tout d'abord, vous pouvez remarquer que nos deux boutons radio possèdent des attributs `name` qui partagent la même valeur.

C'est tout à fait normal ici puisque l'attribut `name` nous sert dans le cas des boutons radio à grouper les options d'un groupe d'options ensemble (cela nous permet de distinguer

d'une autre liste de boutons radio qu'il pourrait y avoir dans le formulaire). C'est également ce qui fait qu'on a besoin d'un attribut `value` supplémentaire pour ensuite pouvoir savoir quelle option a été cochée par l'utilisateur lors du traitement des données.

Ici, on place également le `label` après l'élément `input` pour des raisons d'esthétisme tout simplement.

Ensuite, vous pouvez également noter comment se construit une liste d'options de type `select`. Dans une liste d'options, chaque option est placée dans un élément `option`.

On utilise à nouveau un attribut `value` qui va nous permettre de savoir quelle option a été choisie par l'utilisateur lors du traitement des données.

On peut également classer les options par groupe pour structurer notre formulaire grâce à des éléments `optgroup`.

Les éléments `input type="radio"` et `select` servent le même but. De manière générale, pour des raisons d'ergonomie, nous utiliserons les boutons radio pour une liste de choix courte et des listes d'options pour les listes de choix longues.

Dans notre deuxième section de formulaire, nous demandons aux utilisateurs de cocher leurs compétences dans une liste sans limite de nombre. Nous allons donc utiliser des `input type="checkbox"` pour créer cette liste.

Nous voulons également que l'utilisateur nous raconte une de ses expériences professionnelles. Pour cela, nous allons utiliser un élément `textarea`.

```
<fieldset>
  <legend>Compétences / expérience :</legend>
  <div class="champ">
    <input type="checkbox" id="html" name="competences" value="html">
    <label for="html">HTML</label>
    <input type="checkbox" id="css" name="competences" value="css">
    <label for="css">CSS</label>
    <input type="checkbox" id="js" name="competences" value="javascript">
    <label for="js">JavaScript</label>
    <input type="checkbox" id="php" name="competences" value="php">
    <label for="php">PHP</label>
    <input type="checkbox" id="sql" name="competences" value="sql">
    <label for="sql">SQL</label>
    <input type="checkbox" id="seo" name="competences" value="seo">
    <label for="seo">SEO</label>
  </div>
  <div class="champ">
    <textarea name="exp" placeholder="Décrivez une expérience pro"></textarea>
   </div>
</fieldset>
```

Les `input type="checkbox"` vont posséder une syntaxe relativement similaire aux `input type="radio"` : les différentes cases à cocher d'un même groupe vont partager un même attribut `name` et nous allons pouvoir distinguer quelles cases ont bien été cochées grâce à l'attribut `value`.

La grande différence entre les `input type="checkbox"` (cases à cocher) et les `input type="radio"` (boutons radios) est qu'avec les cases à cocher un utilisateur va pouvoir cocher plusieurs options.

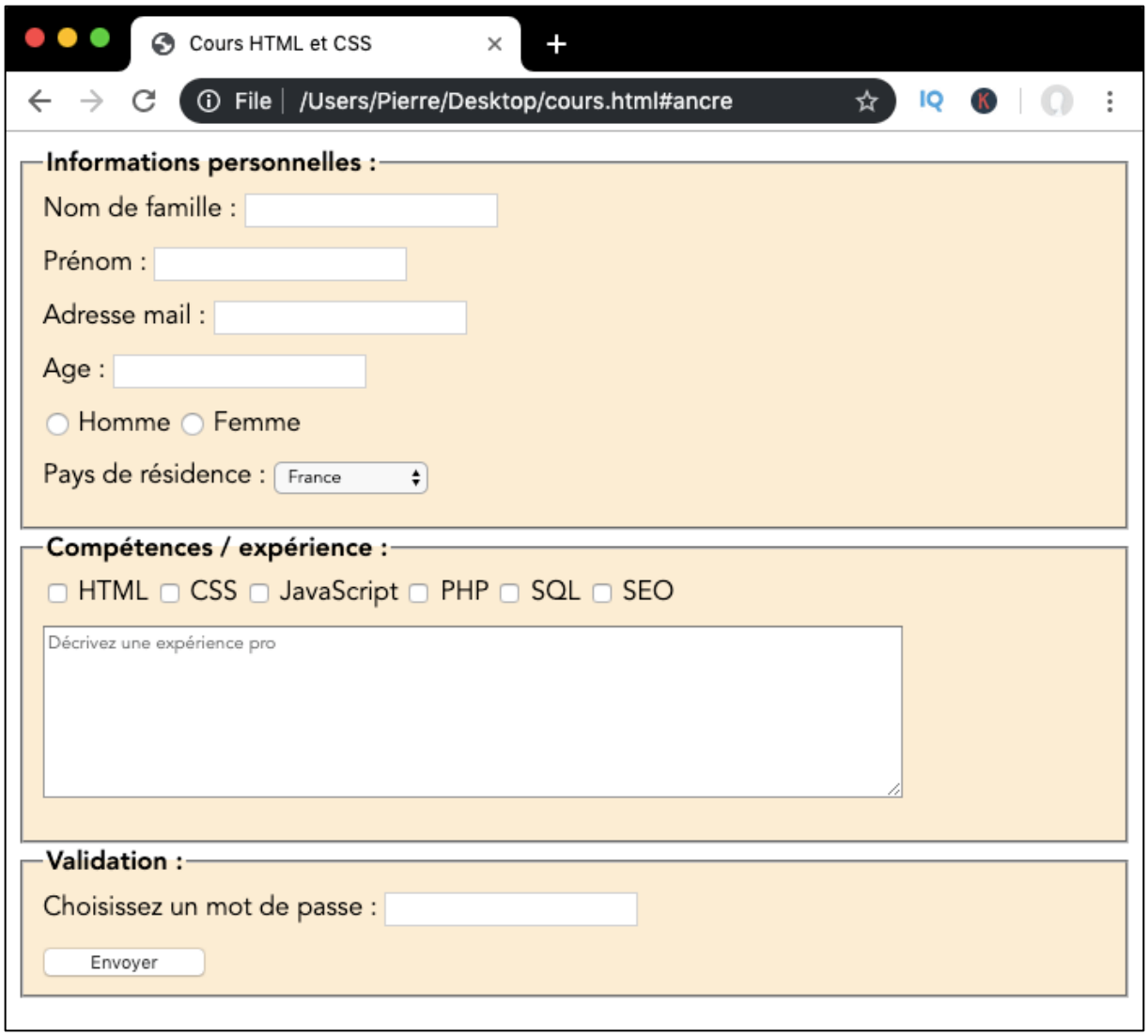
Concernant l'élément `textarea`, on utilise ici un attribut `placeholder` qui nous sert à indiquer aux utilisateurs ce qu'ils doivent remplir dans le champ en soi. Le `placeholder` prend la forme d'un texte semi transparent qui va s'effacer dès qu'un utilisateur place le curseur de sa souris dans le champ de formulaire lié.

Finalement, notre dernière section de formulaire va tout simplement demander un mot de passe à l'utilisateur et contenir le bouton d'envoi du formulaire.

```
<fieldset>
  <legend>Validation :</legend>
  <div class="champ">
    <label for="pass">Choisissez un mot de passe :</label>
    <input type="password">
  </div>
  <input type="submit" value="Envoyer">
</fieldset>
```

On peut ensuite ajouter quelques styles à notre formulaire en CSS pour le rendre un peu plus présentable :

```
*{
  font-family: Avenir, sans-serif;
}
fieldset{
  background-color: RGBa(240, 160, 0, 0.2);
}
legend{
  font-weight: bold;
}
.champ{
  margin-bottom: 10px;
}
textarea{
  width: 80%;
  height: 100px;
}
input{
  width: 150px;
}
input[type="radio"], input[type="checkbox"]{
  width: auto;
}
input[type="submit"]{
  width: 100px;
}
```


Cours HTML et CSS
File /Users/Pierre/Desktop/cours.html#ancre
Informations personnelles :
Nom de famille :
Prénom :
Adresse mail :
Age :
Homme Femme
Pays de résidence : France
Compétences / expérience :
HTML CSS JavaScript PHP SQL SEO
Décrivez une expérience pro
Validation :
Choisissez un mot de passe :
Envoyer

Attributs des formulaires et sécurité

Dans cette dernière leçon liée aux formulaires HTML, nous allons discuter de l'importance de sécuriser ses formulaires et de toujours traiter les données envoyées par les utilisateurs et je vais vous présenter quelques attributs HTML qu'il peut s'avérer intéressant d'utiliser.

Le problème des données utilisateurs

Il n'y a qu'une règle que vous devez absolument retenir concernant les formulaires HTML : il ne faut jamais faire confiance aux données envoyées par les utilisateurs.

En effet, vous ne devez jamais vous attendre à ce que vos formulaires soient correctement remplis : certains utilisateurs vont faire des fautes d'inattention ou vont avoir la flemme de remplir tous les champs de votre formulaire tandis que d'autres, malveillants, vont essayer d'exploiter les failles de votre formulaire pour récupérer des données dans votre base de données ou pour tromper vos utilisateurs.

Les mécanismes d'intrusion et de protection des formulaires sont relativement complexes pour quelqu'un qui n'a pas une vue d'ensemble sur les langages utilisés mais je vais essayer d'illustrer le problème et les solutions possibles le plus simplement possible.

Reprenons le formulaire créé à la partie précédente par exemple. Rien n'empêche un utilisateur, dans ce formulaire, d'envoyer des valeurs aberrantes comme 2000 ans pour son âge ou des données de type invalide comme des éléments HTML à la place de son nom ou encore de renvoyer le formulaire à moitié ou complètement vide.

Dans ces cas-là, les données reçues vont être inexploitables et ce n'est pas ce que nous voulons.

De plus, ici, rien n'empêcherait un utilisateur malveillant d'injecter du code JavaScript dans notre formulaire pour ensuite essayer de tromper des utilisateurs ou encore pour tenter d'injecter le code malveillant dans notre base de données.

Nous n'allons pas entrer dans les détails ici car c'est assez complexe et car vous ne pourrez pas comprendre sans connaitre le JavaScript mais grosso modo retenez qu'ici un utilisateur peut tout à fait envoyer du code dans vos champs de formulaire et que le code envoyé risque d'être exécuté à un moment ou à un autre de notre côté.

Quelles solutions pour sécuriser nos formulaires ?

Ici, il va falloir faire la différence entre deux types d'utilisateurs qui vont être gérés de façons différentes : les utilisateurs maladroits qui vont envoyer des données invalides par mégarde et les utilisateurs malveillants qui vont tenter d'exploiter des failles de sécurité dans nos formulaires pour par exemple récupérer les données personnelles d'autres utilisateurs.

Pour ce premier groupe d'utilisateurs qui ne sont pas mal intentionnés, la première action

que nous allons pouvoir prendre va être d'ajouter des contraintes directement dans notre formulaire pour limiter les données qui vont pouvoir être envoyées.

Pour cela, nous allons pouvoir utiliser des attributs HTML comme `required`, `min` et `max` que nous allons étudier dans la suite de la leçon et allons devoir faire le maximum pour préciser les bonnes valeurs dans l'attribut `type` de l'élément `input` à chaque fois.

Nous allons ensuite également pouvoir tester que les données nous conviennent dès le remplissage d'un champ ou au moment de l'envoi du formulaire grâce au HTML ou au JavaScript (principalement) et bloquer l'envoi du formulaire si des données ne correspondent pas à ce qu'on attend.

Tout cela ne va malheureusement pas être suffisant contre les utilisateurs malintentionnés pour la simple et bonne raison que n'importe qui peut neutraliser toutes les formes de vérification effectuées dans le navigateur.

Pour cela, il suffit par exemple de désactiver l'usage du JavaScript dans le navigateur et d'inspecter le formulaire pour supprimer les attributs limitatifs avant l'envoi.

Contre les utilisateurs malveillants, nous allons donc également devoir vérifier les données après l'envoi du formulaire et neutraliser les données potentiellement dangereuses. Nous allons effectuer ces vérifications en PHP, côté serveur.

Ces deux niveaux de vérifications (dans le navigateur / côté serveur) doivent être implémentés lors de la création de formulaires. En effet, n'utiliser qu'une validation dans le navigateur laisse de sérieuses failles de sécurité dans notre formulaire puisque les utilisateurs malveillants peuvent désactiver ces vérifications.

N'effectuer qu'une série de vérifications côté serveur, d'autre part, serait également une très mauvaise idée d'un point de vue expérience utilisateur puisque ces vérifications sont effectuées une fois le formulaire envoyé.

Ainsi, que faire si des données aberrantes mais pas dangereuses ont été envoyées par un utilisateur maladroit ? Supprimer les données ? Le recontacter pour qu'il soumette à nouveau le formulaire ? Il est bien plus facile dans ce cas de vérifier directement les données lorsqu'elles sont saisies dans le navigateur et de lui faire savoir si une donnée ne nous convient pas.

Vérification et validation des données de formulaire dans le navigateur

Les processus de validation des données que nous allons pouvoir mettre en place dans le navigateur vont s'effectuer avant ou au moment de la tentative d'envoi du formulaire.
L'objectif va être ici de bloquer l'envoi du formulaire si certains champs ne sont pas correctement remplis et de demander aux utilisateurs de remplir correctement les champs invalides.

Une nouvelle fois, cette série de validations va s'adresser aux utilisateurs étourdis / flemmards mais ne va pas bloquer les utilisateurs malveillants car n'importe quel utilisateur

peut supprimer à la main des parties de code HTML en inspectant le code de la page et désactiver le JavaScript dans son navigateur.

La première chose à faire ici va déjà être de s'assurer qu'on a bien précisé les bons types d'`input` pour chaque champ de formulaire car la plupart des navigateurs récents proposent des systèmes de contrôle.

Par exemple, les navigateurs bloquent aujourd'hui l'envoi de code JavaScript et empêchent l'envoi du formulaire selon que certaines données ne soient pas de la forme attendue.

Essayez par exemple de rentrer un prénom dans le champ de formulaire demandant un email : votre navigateur devrait renvoyer une erreur et vous demander d'ajouter un signe `@` dans vos données.

En plus de ces vérifications faites par le navigateur, le HTML propose aujourd'hui des options de validation relativement puissantes. La validation des données en HTML va principalement passer par l'ajout d'attributs dans les éléments de formulaire. Nous allons ainsi pouvoir utiliser les attributs suivants :

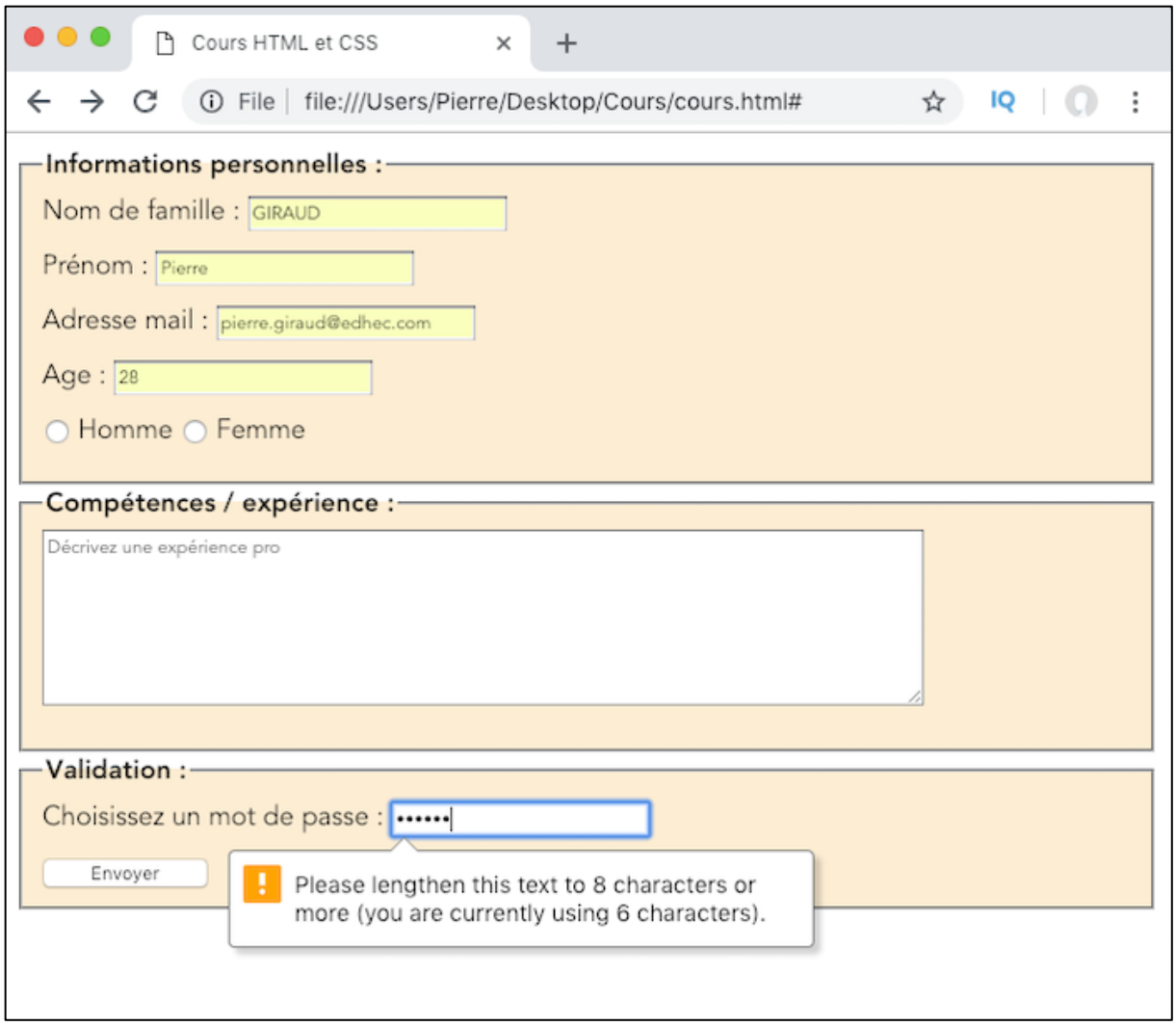
Attribut	**Définition**
size	Permet de spécifier le nombre de caractères dans un champ
minlength	Permet de spécifier le nombre minimum de caractères dans un champ
maxlength	Permet de spécifier le nombre maximum de caractères dans un champ
min	Permet de spécifier une valeur minimale pour un champ de type number ou date
max	Permet de spécifier une valeur maximale pour un champ de type number ou date
step	Permet de définir un multiple de validité pour un champ acceptant des donnés de type nombre ou date. En indiquant `step="4"`, les nombres valides seront -8, -4, 0, 4, 8, etc.
autocomplete	Permet d'activer l'auto complétion pour un champ : si un utilisateur a déjà rempli un formulaire, des valeurs lui seront proposées automatiquement lorsqu'il va commencer à remplir le champ
required	Permet de forcer le remplissage d'un champ. Le formulaire ne pourra pas être envoyé si le champ est vide
pattern	Permet de préciser une expression régulière. La valeur du champ devra respecter la contrainte de la regex pour être valide

Note : les expressions régulières constituent un langage à part. Elles vont consister en la création de « motifs » ou de chaine de caractères constituées de caractères à la signification spéciale et qui vont servir à décrire un ensemble de chaines de caractères possibles. Les expressions régulières sont utilisées conjointement avec de nombreux langages informatiques pour comparer des données envoyées avec le motif attendu et les refuser si elles ne correspondent pas. Bien évidemment, l'objet de cette leçon n'est pas de faire un cours sur les expressions régulières.

Reprenons notre formulaire précédent en se concentrant sur certains champs en particulier et ajoutons quelques contraintes sur les données que l'on souhaite recevoir :

- Les nom, prénom, adresse mail et mot de passe sont désormais obligatoires
- Les nom et prénom doivent faire au moins 2 caractères et maximum 25 caractères ;
- L'âge doit être compris entre 16 et 99 ans ;
- La description de l'expérience professionnelle ne doit pas excéder 500 caractères ;
- Le mot de passe doit contenir au moins 8 caractères.

```
<form method="post" action="form.php">
    <fieldset>
      <legend>Informations personnelles :</legend>
      <div class="champ">
        <label for="nom">Nom de famille :</label>
        <input type="text" id="nom" name="nom" minlength="2" maxlength="25" required>
      </div>
      <div class="champ">
        <label for="prenom">Prénom :</label>
        <input type="text" id="prenom" name="prenom" minlength="2" maxlength="25" required>
      </div>
      <div class="champ">
        <label for="mail">Adresse mail :</label>
        <input type="email" id="mail" name="mail" required>
       </div>
       <div class="champ">
         <label for="age">Age :</label>
         <input type="number" id="age" name="age" min="16" max="99">
       </div>
      <div class="champ">
        <input type="radio" id="h" name="sexe" value="homme">
        <label for="h">Homme</label>
        <input type="radio" id="f" name="sexe" value="femme">
        <label for="f">Femme</label>
       </div>
    </fieldset>
    <fieldset>
      <legend>Compétences / expérience :</legend>
      <div class="champ">
        <textarea name="exp" placeholder="Vos expériences pro" maxlength="500"></textarea>
       </div>
    </fieldset>
    <fieldset>
      <legend>Validation :</legend>
      <div class="champ">
        <label for="pass">Choisissez un mot de passe :</label>
        <input type="password" minlength="8" required>
      </div>
      <input type="submit" value="Envoyer">
    </fieldset>
  </form>
```



Essayez d'envoyer le formulaire ci-dessus en remplissant mal les différents champs : le navigateur va vous indiquer les différentes erreurs et va bloquer l'envoi du formulaire tant que vous ne les aurez pas corrigées.

Vous pouvez noter que l'attribut `required` possède une valeur muette : il n'est pas nécessaire de la fournir (car elle est unique) et il suffit donc de préciser le nom de l'attribut pour rendre le remplissage du champ obligatoire.

PARTIE XII

Transitions, animations et transformations

Créer des transitions en CSS

Jusqu'à présent, à chaque fois qu'on définissait le comportement d'une propriété CSS pour un élément, la valeur définie s'appliquait directement.

Les transitions CSS vont nous permettre de modifier la valeur d'une propriété CSS de manière fluide et selon une durée que l'on va pouvoir définir. On va donc pouvoir définir deux valeurs pour une propriété (une première valeur de départ ou valeur par défaut et une seconde valeur d'arrivée) et faire en sorte que la valeur change progressivement de la valeur de départ à la valeur d'arrivée.

On va ainsi par exemple pouvoir changer progressivement la couleur des textes de nos éléments ou modifier la taille d'un élément, etc.

Pour créer des transitions en CSS, nous allons pouvoir utiliser les différentes propriétés de type `transition-*` ou la propriété raccourcie `transition`.

Dans cette nouvelle leçon, nous allons commencer par étudier les propriétés complètes qui sont les suivantes :

- `transition-property` ;
- `transition-duration` ;
- `transition-timing-function` ;
- `transition-delay`.

Nous terminerons avec la création d'un transition complète en utilisant la propriété raccourcie `transition`.

Le fonctionnement d'une transition

Le principe de base d'une transition en CSS est de modifier progressivement la valeur d'une propriété. Pour cela, nous allons devoir indiquer deux valeurs pour la propriété pour laquelle on souhaite créer une transition.

Le premier souci ici est qu'on ne va pas pouvoir passer deux valeurs pour une même propriété avec un même sélecteur CSS pour créer une transition, car en faisant cela seule la dernière valeur déclarée serait lue.

Nous avons ici deux solutions pour contourner cette contrainte. La première qui va nous permettre d'exploiter tout le potentiel des animations va être d'utiliser un langage dynamique comme le JavaScript qui peut mettre à jour le nom d'un attribut `class` par exemple en fonction de certains critères.

Comme nous n'avons pas étudié le JavaScript, nous allons mettre de côté cette première solution. La deuxième méthode va être d'utiliser les transitions avec les pseudo-classes, c'est-à-dire lors d'un changement d'état d'un élément HTML.

Nous allons par exemple pouvoir changer la couleur de fond d'un élément `div` lorsqu'un utilisateur passe sa souris dessus comme ci-dessous :

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div id="tr1"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 200px;
    margin: 0 auto;
    border: 2px solid black;
    box-sizing: border-box;
}

/*Etat normal*/
#tr1{
    background-color: orange;
    transition-property: background-color;
    transition-duration: 5s;
}

/*Lors du survol*/
#tr1:hover{
    background-color: blue;
}
```

Couleur du `div` au début de la transition :

Couleur du `div` en fin de transition :

Couleur du `div` lorsqu'on passe sa souris dessus :

Ici, on a utilisé deux propriétés pour créer une transition simple : la propriété `transition-property` qui sert à définir la ou les propriété(s) dont la valeur doit être modifiée progressivement et la propriété `transition-duration` qui indique le temps que va mettre la propriété à passer de sa valeur de départ à la valeur d'arrivée.

Comme vous pouvez le constater en observant le code CSS, on appliquer ces propriétés de transition à notre `div` dans son état normal et on définit deux couleurs de fond : orange dans l'état normal et bleu lorsque le `div` est survolé.

Lorsqu'on survole le `div`, sa couleur de fond va donc changer progressivement de l'orange vers le bleu sur une durée de 5 secondes. Notez que par défaut la transition se fait dans les deux sens : lorsqu'on sort du `div`, sa couleur de fond retourne progressivement vers l'orange.

Notez finalement qu'on ne va pas pouvoir créer des transitions avec toutes les propriétés car le concept de transitions est assez récent en CSS et donc le support n'est pas encore parfait. La plupart des propriétés vont tout de même pouvoir être animées.

La propriété CSS transition-property

La propriété `transition-property` va nous permettre de définir quelles propriétés vont être les cibles de nos transitions, c'est-à-dire quelles sont les propriétés dont la valeur va devoir changer progressivement.

On va pouvoir indiquer en valeur de `transition-property` soit le nom d'une propriété CSS pour laquelle on veut créer une transition, soit le nom de plusieurs propriétés CSS qui devront alors être séparées par des virgules, soit le mot clef `all` qui signifie que toutes les propriétés vont pouvoir être sujettes aux transitions (sous réserve qu'on indique ensuite une autre valeur d'arrivée pour créer la transition bien évidemment).

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="d1">Du texte dans le div 1</div>
        <div class="d2">Du texte dans le div 2</div>
        <div class="d3">Du texte dans le div 3</div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 100px;
    margin: 20px auto;
    padding: 40px;
    border: 2px solid black;
    box-sizing: border-box;
    background-color: orange;
}

.d1{
    transition-property: background-color;
    transition-duration: 5s;
}
.d2{
    transition-property: background-color, border;
    transition-duration: 5s;
}
.d3{
    transition-property: all;
    transition-duration: 5s;
}
.d1:hover, .d2:hover, .d3:hover{
    background-color: blue;
    border: 5px solid red;
    color: white;
}
```

Dans cet exemple, on crée trois transitions pour chacun de nos trois `div`. Ici, on définit de nouvelles valeurs pour les propriétés `color` ; `background-color` et `border` de nos `div` lors de l'état `hover`.

Les transitions vont donc se faire lorsque l'utilisateur va passer sa souris sur un des `div`. Chacune des 3 transitions va durer 5 secondes.

Pour notre premier `div`, on applique la transition à la propriété `background-color` uniquement.

Pour notre deuxième `div`, on applique la transition aux propriétés `background-color` et `border`.

Pour notre troisième `div`, on applique une transition à toutes les propriétés qui ont plusieurs valeurs définies pour différents états de l'élément.

Notez que je n'ai pas défini de couleur de départ avec `color`. En effet, il n'est pas toujours strictement indispensable de définir des valeurs de départ pour les propriétés pour lesquelles on définit des transitions puisque la plupart des propriétés ont des valeurs par défaut. Pour `color`, la transition va donc se faire entre la valeur par défaut (noire) et la valeur d'arrivée spécifiée.

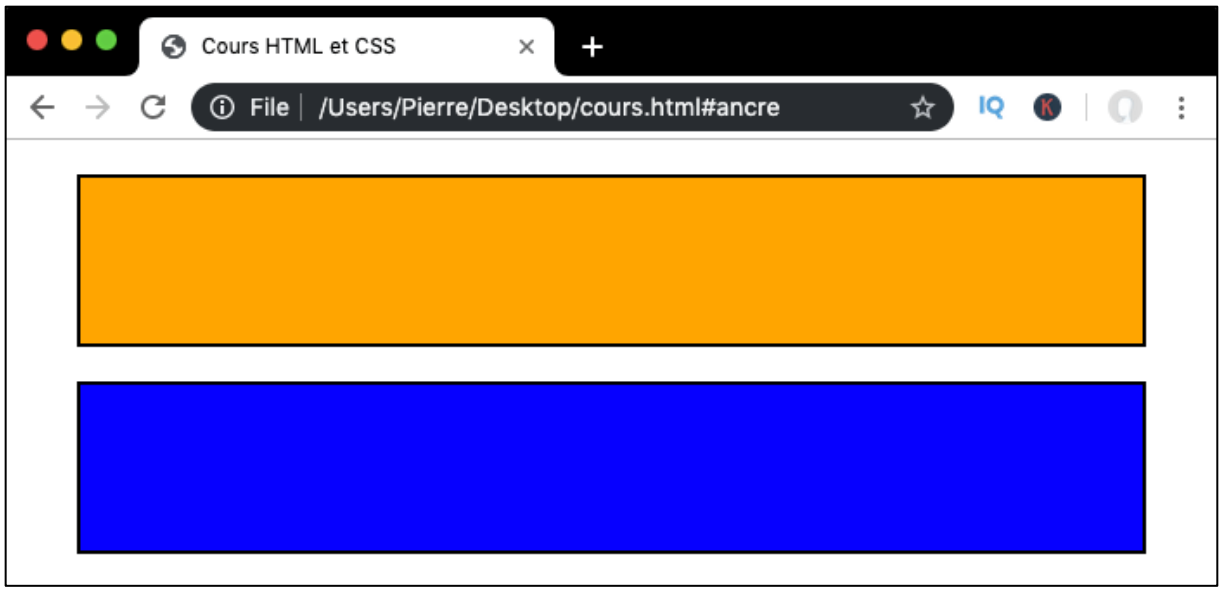
Ici, vous devez également bien comprendre qu'une transition ne va se déclencher que lors d'un changement d'état d'un élément. Ainsi, il est tout à fait possible de définir plus de

deux valeurs pour une propriété à laquelle on souhaite appliquer une transition puisqu'on va pouvoir définir une valeur pour chaque état de l'élément.

Dans le cas où plusieurs changements d'état sont provoqués en même temps, la transition se fera vers la valeur de l'état déclaré en dernier en CSS. Par exemple, on peut définir une transition sur une couleur de fond en précisant une valeur d'arrivée pour l'état hover (survol) et l'état active (cliqué) :

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="d1"></div>
        <div class="d2"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 100px;
    margin: 20px auto;
    padding: 40px;
    border: 2px solid black;
    box-sizing: border-box;
    background-color: orange;
    transition-property: background-color;
    transition-duration: 5s;
}
.d1:hover{
    background-color: blue;
}
.d1:active{
    background-color: green;
}
.d2:active{
    background-color: green;
}
.d2:hover{
    background-color: blue;
}
```



Ici, on commence par définir une couleur de fond pour l'état `active` pour notre `div .d2` puis on déclare une couleur de fond pour l'état `hover` pour ce même élément. Pour que l'élément soit cliqué (active), il faut que notre souris soit dessus (hover). Dans ce cas, la couleur de fond pour l'état `active` ne s'appliquera jamais puisque la propriété `background-color` a été définie en dernier pour l'état `hover` dans le code.

La propriété CSS transition-duration

La propriété `transition-duration` va nous permettre de définir le temps que vont mettre les propriétés passées à `transition-property` pour passer d'une valeur de départ à une valeur d'arrivée. Nous allons pouvoir lui passer un nombre de secondes en valeur.

La valeur par défaut de `transition-duration` est `0s` ce qui signifie que le changement de valeur des propriétés concernées par la transition sera immédiat.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="d1"></div>
        <div class="d2"></div>
        <div class="d3"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 100px;
    margin: 20px auto;
    padding: 40px;
    border: 2px solid black;
    box-sizing: border-box;
    background-color: orange;
}
.d1,.d2,.d3{
    transition-property: background-color;
}
.d1{
    transition-duration: 2s;
}
.d2{
    transition-duration: 4s;
}
.d3{
    transition-duration: 8s;
}
.d1:hover,.d2:hover, .d3:hover{
    background-color: blue;
}
```

Ici, on applique une transition sur la couleur de fond de nos éléments HTML `div` lors du passage de la souris sur l'un deux. Chaque transition a une durée différente définie avec `transition-duration` ce qui signifie que la couleur de fond mettra plus ou moins de temps à passer de sa couleur de départ à sa couleur d'arrivée selon le `div`.

La propriété CSS transition-timing-function

La propriété `transition-timing-function` va nous permettre de choisir la vitesse de la transition au sein de celle-ci. Nous allons ainsi pouvoir créer des transitions totalement linéaires ou, au contraire, créer des transitions qui vont s'accélérer ou ralentir au milieu. Nous allons pouvoir passer les valeurs suivantes à cette propriété :

- `ease` : valeur par défaut. Permet de créer une transition relativement lente au début puis qui s'accélère au milieu et qui se termine lentement ;
- `linear` : permet de créer une transition totalement linéaire c'est-à-dire qui va aller à la même vitesse du début à la fin ;
- `ease-in` : permet de créer une transition avec un départ lent puis qui s'accélère ensuite ;
- `ease-out` : permet de créer une transition qui va ralentir à la fin ;
- `ease-in-out` : permet de créer une transition lente au début puis qui s'accélère au milieu et qui se termine lentement. Ressemble fortement à `transition-timing-function : ease` mais démarre plus lentement ;
- `cubic-bezier(x1,y1,x2,y2)` : sert à créer une transition à la vitesse totalement personnalisé en renseignant une courbe de Bézier.

Notre but n'est ici bien évidemment pas de faire un cours sur les courbes de Bézier. Vous pouvez simplement retenir les équivalents mot-clef/Bézier suivants qui peuvent se révéler utiles :

- `transition-timing-function : ease` est équivalent à `transition-timing-function : cubic-bezier(0.25, 0.1, 0.25, 1)` ;
- `transition-timing-function : ease-in` est équivalent à `transition-timing-function : cubic-bezier(0.42, 0, 1, 1)` ;
- `transition-timing-function : ease-out` est équivalent à `transition-timing-function : cubic-bezier(0, 0, 0.58, 1)` ;
- `transition-timing-function : ease-in-out` est équivalent à `transition-timing-function : cubic-bezier(0.42, 0, 0.58, 1)`.

La propriété CSS transition-delay

La propriété `transition-delay` va nous permettre de définir quand la transition doit commencer à partir du moment où la nouvelle valeur est passée aux propriétés concernées par la transition. On va pouvoir lui passer une valeur en secondes.

La valeur par défaut est `0s` (la transition se lance dès qu'une nouvelle valeur est définie pour une propriété à laquelle on a appliqué la transition).

La propriété CSS transition

La propriété CSS `transition` est la notation raccourcie des quatre propriétés étudiées précédemment. On va pouvoir lui passer les différentes valeurs des propriétés précédentes à la suite pour créer une transition complète.

La première durée renseignée dans `transition` définira la durée de la transition. Je vous conseille de suivre l'ordre de déclaration des valeurs suivant pour être sûr que `transition` fonctionne bien :

1. La valeur relative à `transition-property` ;
2. La valeur relative à `transition-duration` ;
3. La valeur relative à `transition-timing-function` ;
4. La valeur relative à `transition-delay`.

A noter que seules les valeurs relatives aux propriétés `transition-property` et `transition-duration` doivent être obligatoirement renseignées pour créer une transition visible. Les valeurs relatives aux propriétés `transition-timing-function` et `transition-delay` sont facultatives et si rien n'est renseigné les valeurs par défaut seront utilisées.

Notez également que pour créer plusieurs transitions pour plusieurs propriétés différentes avec `transition` il va suffire de séparer les différentes déclarations par une virgule.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="d1">transition: width 4s linear 0s</div>
        <div class="d2">transition: width 4s</div>
        <div class="d3">transition: width 4s linear 0s, background-color 2s</div>
    </body>
</html>
```

```css
div{
    width: 50%;
    height: 100px;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
    background-color: orange;
}

.d1{
    transition: width 4s linear 0s;
}
.d2{
    transition: width 4s;
}
.d3{
    transition: width 4s linear 0s, background-color 2s;
}

.d1:hover, .d2:hover, .d3:hover{
    width: 100%;
}
.d3:hover{
    background-color: blue;
}
```

Ici, on renseigne toutes les valeurs pour notre première transition dans la propriété `transition`. Pour notre deuxième transition, en revanche, on omet les deux dernières valeurs. Ce seront donc les valeurs par défaut `ease` et `0s` qui seront utilisées.

Finalement, nous créons deux transitions dans notre dernier `div`. Pour faire cela, on se contente de séparer les différentes déclarations pour chacune de nos transitions par une virgule dans notre propriété `transition`.

Transition et reverse transition

Par défaut, lorsqu'on crée une transition entre les valeurs d'une ou de plusieurs propriétés, la transition va se faire dans les deux sens :

1. A partir de la valeur de départ vers la valeur d'arrivée lors du changement d'état d'un élément ;
2. A partir de la valeur actuelle vers la valeur de départ lorsque l'élément retourne dans son état initial.

Par exemple, si on crée une transition qui doit se déclencher lors du passage de la souris sur un élément, dès que l'on va déplacer notre souris en dehors de l'élément les propriétés concernées par la transition vont reprendre petit à petit leurs valeurs initiales en effectuant une transition inversée vers leurs valeurs initiales plutôt que d'être redéfinie brutalement sur leurs valeurs de départ.

Nous allons pouvoir également pouvoir gérer la transition de retour et notamment la durée de cette transition et son délai. Pour faire cela, nous allons devoir préciser en plus de nos valeurs de transition classiques des valeurs de transition dans l'état qui va déclencher la transition (à l'intérieur d'un `:hover` par exemple pour une transition qui doit démarre lorsque l'utilisateur passe sa souris sur un élément).

Attention ici : les valeurs précisées dans cet état seront utilisées pour la transition de départ et ce sont les valeurs précisées au départ qui seront utilisées pour la transition de retour. Regardez l'exemple suivant pour bien comprendre :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="d1"></div>
    </body>
</html>
```

```css
div{
    width: 50%;
    height: 200px;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
    background-color: orange;
}

.d1{
    transition: width 2s linear 5s, background-color 2s linear 5s;
}

/*Comme on n'indique qu'une valeur à transition-duration et à
 *transition-delay, la valeur indiquée pour chaque propriété
 *servira pour les deux transitions définies ci-dessus*/
.d1:hover{
    background-color: blue;
    width: 100%;
    transition-duration: 10s;
    transition-delay: 0s;
}
```

Ici, on crée une transition sur la couleur de fond et la taille de notre `div class="d1"` qui doit démarrer lorsqu'un utilisateur passe sa souris sur notre `div`.

Cette première transition ici va durer 10 secondes et avoir un délai de 0s. Dès que l'utilisateur sort sa souris de l'élément, les valeurs de nos propriétés vont revenir à leur état normal en effectuant une transition de retour. La transition de retour va durer 2 secondes et va démarrer après un délai de 5 secondes.

La relation entre les valeurs et les transitions peut paraitre contre intuitif à priori, donc faites bien attention à bien retenir cela !

Créer des animations en CSS

Les animations vont, comme les transitions, nous nous permettre de modifier la valeur d'une propriété progressivement mais en utilisant cette fois-ci des keyframes.

Nous allons pouvoir gérer le comportement des animations en CSS en définissant la durée d'animation, le nombre de répétition et le comportement de répétition.

De manière similaire aux animations, toutes les propriétés ne vont pas pouvoir être animées. Cependant, la grande majorité des propriétés courantes vont pouvoir l'être.

Pour définir une animation en CSS nous allons pouvoir utiliser les propriétés de type animation-* ou la notation raccourcie animation avec une règle @keyframes qui va contenir les propriétés à animer et leurs valeurs.

Différence entre animations et transitions en CSS

Les transitions et les animations permettent toutes les deux de modifier la valeur de propriétés de manière progressive, au cours du temps. Cependant, la façon dont vont procéder les transitions et les animations pour arriver à cela ne va pas être la même.

La grande différence entre les transitions et les animations en CSS est que les animations nous laissent à la fois une plus grande liberté et un plus grand contrôle sur le déclenchement et la progression du changement de valeur des propriétés animées.

En effet, dans le cas d'une transition, nous ne pouvons que préciser une valeur de départ et une valeur d'arrivée pour les propriétés pour lesquelles nous souhaitons créer notre transition et n'avons pas véritablement de contrôle précis sur la transition en soi tandis que dans le cas d'une animation nous allons pouvoir indiquer de manière explicite comment la « transition » entre les différentes valeurs doit se passer et pouvoir préciser différentes valeurs intermédiaires.

En cela, les animations offrent davantage de contrôle sur le changement de valeurs des propriétés concernées par l'animation puisqu'on va pouvoir contrôler ce changement de valeur dans son ensemble. Elles vont donc notre choix de prédilection lorsqu'on voudra créer des effets plus complexes ou précis.

De plus, nous n'allons plus devoir attendre un changement d'état d'un élément pour modifier la valeur d'une de ses propriétés avec les animations. En effet, nous allons pouvoir lancer une animation dès le chargement de la page ou selon un autre évènement.

La règle CSS @keyframes

La règle @keyframes va nous permettre d'indiquer quelles propriétés doivent être animées et comment elles doivent l'être. Nous allons pouvoir dans notre règle @keyframes préciser différentes valeurs auxquelles les propriétés animées doivent parvenir à certains stades de l'animation.

La règle @keyframes va donc nous permettre de définir différents stades ou étapes pour notre animation et c'est ce qui va nous permettre d'avoir un contrôle total sur la progression de l'animation.

Nous allons toujours devoir donner un nom à une règle @keyframes qu'on va ensuite réutiliser dans notre animation pour lui indiquer les propriétés à animer et comment les animer. Dans notre règle @keyframes, nous allons tout simplement renseigner les propriétés à animer et les différentes valeurs qu'elles doivent atteindre à certains moments de l'animation.

Je vous rappelle ici que pour créer une animation fonctionnelle en CSS, nous allons d'une part devoir définir notre règle @keyframes et d'autre part utiliser les propriétés de type animation-* ou la notation raccourcie animation pour définir les caractéristiques propres à notre animation (nom, durée, etc.).

Regardez plutôt l'exemple suivant pour bien comprendre :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="d1"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 200px;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}
.d1{
    background-color: orange;
    animation-name: couleur;
    animation-duration: 4s;
}
@keyframes couleur{
    from{
        background-color: orange;
    }
    to{
        background-color: blue;
    }
}
```

Ici, vous pouvez déjà noter une différence intéressante entre les animations et les transitions : une fois l'animation terminée, les propriétés animées retrouvent immédiatement leur valeur « normale ».

Regardons maintenant en détail notre règle `@keyframes`. Ici, on lui donne le nom `couleur` et on précise dedans les propriétés qui vont être animées (en l'occurrence la propriété `background-color`).

Plus précisément, on indique ici la valeur de départ pour notre ou nos propriété(s) que l'on souhaite animer à l'intérieur d'un bloc `from{}` qui signifie « à partir de » et la valeur d'arrivée pour notre ou nos propriété(s) que l'on souhaite animer à l'intérieur d'un bloc `to{}` qui signifie « vers ».

A ce niveau, il y a une chose que vous devez bien comprendre : la valeur que l'on précise dans le bloc `from{}` est bien la valeur de départ de l'animation, c'est-à-dire la valeur avec laquelle la propriété doit démarrer son animation et non pas sa valeur « normale » c'est-à-dire sa valeur en dehors du temps de l'animation (avant le lancement de l'animation et après sa fin).

C'est pour cela que j'ai également précisé une couleur de fond en dehors de ma règle `@keyframes` afin de préciser la valeur que doit avoir ma propriété lorsqu'elle n'est pas animée.

Une fois notre règle `@keyframes` définie, nous allons devoir la lier ou l'attribuer à une animation. Pour cela, nous allons également devoir déclarer une animation et définir son comportement.

C'est ce que j'ai fait ici en déclarant deux propriétés deux propriétés `animation-name` à laquelle j'ai passé notre règle `@keyframes` et `animation-duration` qui me permet de définir la durée de l'animation. Par défaut, l'animation ne va s'exécuter qu'une fois.

Dans l'exemple précédent, utiliser une animation n'a aucun intérêt par rapport à une simple transition car cette animation est très simple. Cependant, rappelez-vous qu'un des grands avantages des animations par rapport aux transitions est qu'on va pouvoir dans notre règle `@keyframes` préciser différentes valeurs que nos propriétés devront atteindre à un moment choisi de l'animation.

On va ainsi entre autres très facilement pouvoir créer une boucle sur les valeurs d'une propriété pendant une animation comme dans l'exemple suivant :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="d1"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 200px;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}
.d1{
    background-color: orange;
    animation-name: couleur;
    animation-duration: 10s;
}
@keyframes couleur{
    from{
        background-color: orange;
    }
    25%{
        background-color: blue;
    }
    50%{
        background-color: purple;
    }
    75%{
        background-color: red;
    }
    to{
        background-color: orange;
    }
}
```

Ici, en plus de mes mots clefs `from` et `to`, j'ai indiqué des pourcentages dans ma règle `@keyframes`.

Ces pourcentages correspondent à l'état d'avancement de l'animation. 50% marque donc le milieu de l'animation, c'est-à-dire au bout de 5 secondes pour une animation qui doit durer 10 secondes ; 25% correspond à 25% du temps total de l'animation, soit 2,5 secondes pour une animation de 10 secondes et etc.

Ensuite, on indique la valeur que doit avoir notre propriété animée pour chaque niveau d'avancement de l'animation défini.

Le grand avantage des animations encore une fois est que nous allons pouvoir définir autant d'étapes que l'on souhaite pour contrôler plus ou moins le l'avancement de notre animation plus. J'attribue ici la même valeur à ma propriété `background-color` au début et en fin d'animation pour donner l'impression que la couleur boucle sur elle-même.

Nous allons de plus pouvoir préciser plusieurs propriétés à animer d'un coup et leur attribuer différentes valeurs à différents moments de l'animation au sein d'une même règle `@keyframes` :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="d1"></div>
    </body>
</html>
```

```css
div{
    width: 90%;
    height: 200px;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}
.d1{
    background-color: orange;
    animation-name: couleur;
    animation-duration: 10s;
}
@keyframes couleur{
    from{
        background-color: orange;
    }
    25%{
        background-color: blue;
        width: 75%
    }
    50%{
        background-color: purple;
        width: 50%
    }
    75%{
        background-color: red;
        width: 75%;
    }
    to{
        background-color: orange;
    }
}
```

Maintenant que nous avons vu et compris le rôle de la règle `@keyframes` dans une animation, il est temps d'étudier les différentes propriétés `animation-*` qui vont nous permettre de gérer notre animation en soi avant de terminer avec la propriété raccourcie `animation`.

La propriété animation-name

La propriété `animation-name` va nous permettre de définir une liste d'animations qui doivent s'exécuter. On va donc lui passer un ou plusieurs noms qui devront correspondre aux noms des règles `@keyframes` qui définissent les propriétés à animer et les différentes valeurs qu'elle doit avoir pendant l'animation.

Si on fournit plusieurs noms à la propriété `animation-name`, alors il faudra définir à minima la durée de l'animation pour chaque animation avec la propriété `animation-duration` si on veut des animations fonctionnelles. Dans le cas où l'on fournit moins de valeurs à `animation-duration` qu'à `animation-name`, alors les animations supplémentaires vont réutiliser les valeurs de `animation-duration` dans l'ordre.

Par exemple, si on définit 5 animations avec `animation-name` et qu'on ne donne que 3 valeurs à `animation-duration`, alors la quatrième animation reprendra la première valeur de `animation-duration` et la cinquième animation reprendra sa deuxième valeur.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="d1"></div>
    </body>
</html>
```

```css
div{
    width: 100%;
    height: 100px;
    background-color: orange;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}

.d1{
    animation-name: couleur, essuieglace, taille;
    animation-duration: 10s, 2s;
}

@keyframes couleur{
    from{background-color: orange;}
    50%{background-color: purple;}
    to{background-color: orange;}
}

@keyframes taille{
    from{width: 100%}
    50%{width: 50%;}
    to{width: 100%;}
}
@keyframes essuieglace{
    from{margin-top: 0px;}
    50%{margin-top: 100px;}
    to{margin-top : 0px;}
}
```

Dans l'exemple précédent, on définit 3 règles `@keyframes` qui définissent chacune des changements de valeurs pour une propriété. Dans le cas présent, on aurait aussi bien pu tout mettre dans une seule règle `@keyframes`. Cependant, il est généralement considéré comme une bonne pratique d'avoir une règle `@keyframes` pour une propriété puisque cela va nous permettre de nous resservir indépendamment d'une règle ou d'une autre pour l'appliquer à un élément ou à un autre.

Je passe ensuite les trois noms des animations que je souhaite exécuter à ma propriété `animation-name`. Ici, vous pouvez remarquer que je n'indique que deux valeurs de durée d'animation dans ma propriété `animation-duration`.

La première animation déclarée dans `animation-name` se servira de la première valeur, la deuxième de la deuxième valeur et la troisième à nouveau de la première valeur. L'animation « couleur » va donc durer 10 secondes, l'animation « essuieglace » va durer 2 secondes et l'animation « taille » va durer 10 secondes.

Notez également ici que lorsqu'on définit plusieurs animations comme cela, les animations vont par défaut toutes se lancer en même temps et non pas à la suite les unes des autres.

La propriété animation-duration

La propriété animation-duration nous permet de définir le temps que doit durer une animation. On doit préciser une durée en secondes.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="d1"></div>
        <div class="d2"></div>
        <div class="d3"></div>
    </body>
</html>
```

```css
div{
    width: 100%;
    height: 100px;
    background-color: orange;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}
.d1{
    animation-name: couleur;
    animation-duration: 2s;
}
.d2{
    animation-name: couleur;
    animation-duration: 5s;
}
.d3{
    animation-name: couleur;
    animation-duration: 10s;
}
@keyframes couleur{
    from{background-color: orange;}
    50%{background-color: purple;}
    to{background-color: orange;}
}
```

Ici, on fournit la même règle @keyframes à chacun de nos trois div et on définit ensuite des paramètres d'animations différents à partir de cette règle en donnant des durées d'animation différentes. Une nouvelle fois, je vous rappelle qu'une règle @keyframes n'est qu'un cadre qui sert qu'à définir différentes valeurs pour une propriété à différents stades d'une animation quelconque.

Ensuite, n'importe quelle animation (qui va être construite et déclarée à proprement parler par les différentes propriétés animation-*) va pouvoir se resservir de cette

règle @keyframes. C'est la raison pour laquelle je peux ici utiliser ma règle @keyframes dans trois animations pour trois éléments différents.

La propriété animation-timing-function

La propriété animation-timing-function va nous permettre comment doit progresser l'animation entre les différentes valeurs de keyframes : la progression de l'animation peut être linéaire, s'accélérer de plus en plus, etc.

Nous allons pouvoir passer les valeurs suivantes à animation-timing-function :

- ease : valeur par défaut. Entre deux valeurs de keyframes, l'animation va commencer relativement lentement, puis accélérer au milieu et se terminer lentement ;
- linear : Entre deux valeurs de keyframes, l'animation aura une vitesse constante du début à la fin ;
- ease-in : Entre deux valeurs de keyframes, l'animation va commencer lentement puis accélérer jusqu'à atteindre la prochaine valeur de keyframe ;
- ease-out : Entre deux valeurs de keyframes, l'animation va commencer rapidement et décélérer progressivement jusqu'à atteindre la prochaine valeur de keyframe ;
- ease-in-out : Entre deux valeurs de keyframes, l'animation commence lentement, accélère au milieu et finit lentement ;
- cubic-bezier(x1, y1, x2, y2) : permet de définir une courbe de Bézier spécifique pour créer une animation à la vitesse totalement contrôlée.

Concernant la propriété animation-timing-function, vous devez bien comprendre que les valeurs passées vont dicter le comportement des animations entre chaque valeur de keyframes, c'est-à-dire que le pattern donné par animation-timing-function va être appliqué et répété entre chaque valeur de keyframes et ne va pas définir l'animation au complet.

Par exemple, si on définit une règle @keyframes avec une valeur intermédiaire à 50% et que l'animation se fait selon animation-timing-function: ease, l'animation va commencer relativement lentement à partir du from, puis accélérer pour arriver lentement au 50% puis repartir relativement lentement à partir du 50% pour accélérer à nouveau et finir à nouveau lentement au niveau du to.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="d1"></div>
        <div class="d2"></div>
        <div class="d3"></div>
    </body>
</html>
```

```css
div{
    width: 100%;
    height: 100px;
    background-color: orange;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}
.d1, .d2, .d3{
    animation-name: taille;
    animation-duration: 5s;
}
.d1{
    animation-timing-function: linear;
}
.d2{
    animation-timing-function: ease-in;
}
.d3{
    animation-timing-function: ease-out;
}
@keyframes taille{
    from{width: 100%;}
    50%{width: 50%;}
    to{width: 100%;}
}
```

La propriété animation-iteration-count

La propriété `animation-iteration-count` va nous permettre de définir combien de fois une animation va être jouée. Par défaut, une animation ne sera jouée qu'une fois.

Pour modifier ce comportement par défaut, on va pouvoir passer soit un nombre à `animation-iteration-count` qui va correspondre au nombre de fois que l'on souhaite jouer l'animation, soit le mot clef `infinite` qui signifie que l'animation va se répéter à l'infini.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="d1"></div>
        <div class="d2"></div>
        <div class="d3"></div>
    </body>
</html>
```

```css
div{
    width: 100%;
    height: 100px;
    background-color: orange;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}
.d1, .d2, .d3{
    animation-name: taille;
    animation-duration: 3s;
    animation-timing-function: linear;
}
.d2{
    animation-iteration-count: 2;
}
.d3{
    animation-iteration-count: infinite;
}
@keyframes taille{
    from{width: 100%;}
    50%{width: 50%;}
    to{width: 100%;}
}
```

Ici, ma première animation (l'animation correspondante à mon `div class="d1"`) ne va être jouée qu'une fois par défaut.

Ensuite, on demande à notre deuxième animation de se répéter, c'est-à-dire d'être jouée deux fois avec `animation-iteration-count: 2`.

Finalement, notre troisième animation va se répéter à l'infini grâce à `animation-iteration-count: infinite`.

La propriété animation-direction

La propriété `animation-direction` va nous permettre de spécifier le sens dans lequel une animation doit être jouée, c'est-à-dire si elle doit être jouée en partant du début ou de la fin pour une ou plusieurs de ses itérations ou répétitions.

Nous allons pouvoir passer les valeurs suivantes à `animation-direction` :

- `normal` : valeur par défaut. L'animation est jouée dans le sens dans lequel elle a été déclarée (du `from` vers le `to`) ;
- `reverse` : l'animation est jouée dans le sens inverse pour toutes ses itérations ;
- `alternate` : l'animation va être jouée une première fois dans le sens normal, puis dans le sens contraire, puis à nouveau dans le sens normal et etc. ;
- `alternate-reverse` : l'animation va être jouée une première fois dans le sens inverse, puis dans le sens normal, puis à nouveau dans le sens inverse et etc..

Notez qu'en inversant le sens d'une animation pour une ou plusieurs de ses itérations, les propriétés liées au timing seront également inversées. Par exemple, une animation possédant une `animation-direction : reverse` et une `animation-timing-function : ease-in` sera jouée comme si la valeur était `animation-timing-function : ease-out`.

Notez également que les valeurs `animation-direction : alternate` et `animation-direction : alternate-reverse` ne vont évidemment n'avoir un effet que pour les animations qui vont être jouées plus d'une fois.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="d1"></div>
        <div class="d2"></div>
        <div class="d3"></div>
    </body>
</html>
```

```css
div{
    width: 100%;
    height: 100px;
    background-color: orange;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}
.d1, .d2, .d3{
    animation-name: taille;
    animation-duration: 3s;
    animation-timing-function: linear;
    animation-iteration-count: infinite;
}
.d1{
    animation-direction: reverse;
}
.d2{
    animation-direction: alternate;
}
.d3{
    animation-direction: alternate-reverse;
}
@keyframes taille{
    from{width: 100%;}
    50%{width: 50%;}
    to{width: 25%;}
}
```

Ici, nous créons une règle `@keyframes` qui va modifier la valeur de la propriété `width` de 100% vers 25%. Dans son sens normal, l'animation fait donc passer nos `div` d'une taille de 100% à 25%.

Ensuite, nous demandons à chacune de nos animations de se répéter à l'infini pour bien voir le comportement lié aux valeurs `alternate` et `alternate-reverse`.

Pour notre première animation, cependant, on indique une propriété `animation-direction : reverse` qui signifie qu'on souhaite que l'animation joue dans le sens contraire. Le départ de notre animation va donc utiliser la taille `width: 25%` et la taille du `div` va grandir jusqu'à 100%.

Pour notre deuxième animation, on passe la valeur `alternate` à la propriété `animation-direction`. Notre animation va donc jouer une première fois dans le sens normal, puis dans le sens contraire, puis à nouveau dans le sens normal et etc.

La dernière animation va avoir le comportement opposé de la deuxième : elle va commencer à jouer dans le sens inverse puis alterner à chaque nouvelle itération (ou répétition) puisqu'on lui a donné une `animation-direction : alternate-reverse`

La propriété animation-play-state

La propriété `animation-play-state` va nous permettre de définir si une animation doit être jouée ou être en pause. On va pouvoir lui passer la valeur `running` (l'animation s'exécute normalement) ou `paused` (l'animation est mise en pause).

Cette propriété va pouvoir être utile pour mettre une animation en pause à un certain point de l'animation ou selon une certaine action de l'utilisateur. Par exemple, on va pouvoir proposer aux utilisateurs de mettre en pause une animation lorsqu'ils passant le curseur de leur souris sur l'élément pour lequel une animation est jouée ou lorsqu'ils cliquent (en gardant le clic enfoncé) sur l'élément en utilisant les pseudo classes `:hover` et `:active`.

La propriété animation-delay

La propriété `animation-delay` va nous permettre de définir quand une animation doit commencer c'est-à-dire s'il doit y avoir un délai avant le lancement de l'animation.

On va devoir passer une valeur en secondes à `animation-delay` qui va correspondre au délai qu'il doit s'écouler avant le lancement de l'animation.

La propriété animation-fill-mode

Par défaut, une règle `@keyframes` ou plus généralement une animation ne va pas affecter ni donc définir la valeur de la propriété animée en dehors du temps de l'animation.

Nous avons vu cela en début de tutoriel et c'est la raison pour laquelle je définis depuis le début de celui-ci un comportement « normal » ou par défaut pour les propriétés animées en CSS.

Cependant, il existe un moyen plus élégant de faire cela en utilisant la propriété `animation-fill-mode`. Cette propriété va en effet nous permettre conserver le comportement de nos propriétés défini dans l'animation en dehors de l'animation (avant ou après).

On va ainsi pouvoir appliquer la valeur de départ de notre animation à notre propriété avant que l'animation ne commence ou la valeur d'arrivée de notre animation à la propriété après que celle-ci soit finie.

Pour faire cela, nous allons pouvoir passer l'une des valeurs suivantes à `animation-fill-mode` :

- `backwards` : notre propriété animée utilisera la valeur de départ de l'animation comme valeur avant que l'animation ne commence ;
- `forwards` : notre propriété animée utilisera la valeur de fin de l'animation comme valeur après que l'animation soit terminée ;
- `both` : notre propriété animée utilisera la valeur de départ de l'animation comme valeur avant que l'animation ne commence et la valeur de fin de l'animation comme valeur après que l'animation soit terminée.

La propriété animation

La propriété `animation` correspond à la version raccourcie ou notation short-hand des propriétés `animation-*` vues ci-dessus.

Nous allons pouvoir lui passer les différentes valeurs relatives aux propriétés `animation-` pour créer simplement des animations complètes. Il est généralement considéré comme une bonne pratique de passer les valeurs dans l'ordre suivant pour être certain que l'animation fonctionne correctement :

1. La valeur relative à la propriété `animation-name` ;
2. La valeur relative à la propriété `animation-duration` ;
3. La valeur relative à la propriété `animation-timing-function` ;
4. La valeur relative à la propriété `animation-delay` ;
5. La valeur relative à la propriété `animation-iteration-count` ;
6. La valeur relative à la propriété `animation-direction` ;
7. La valeur relative à la propriété `animation-fill-mode` ;
8. La valeur relative à la propriété `animation-play-state` .

Bien évidemment, nous n'allons pas être obligé de préciser toutes les valeurs pour chaque animation : si une valeur est omise, la valeur par défaut de la propriété correspondante sera utilisée. Cependant, notez bien que la première valeur de type « secondes » fournie à `animation` sera toujours considérée comme étant la durée de l'animation tandis que la deuxième sera toujours considérée comme le délai.

On va également pouvoir créer plusieurs animations avec la propriété `animation`. Pour cela, il suffira de séparer les déclarations des différentes animations par une virgule.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="d1"></div>
        <div class="d2"></div>
        <div class="d3"></div>
    </body>
</html>
```

```css
div{
    height: 100px;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}
.d1{
    animation: couleur 5s linear 2s infinite alternate;
}
.d2{
    animation: taille 5s linear 2s 5 alternate both;
}
.d3{
    animation: couleur 5s linear 2s infinite alternate,
               taille 5s linear 2s 5 alternate both;
}
@keyframes couleur{
    from{background-color: orange;}
    50%{background-color: purple;}
    to{background-color: blue;}
}
@keyframes taille{
    from{width: 100%;}
    50%{width: 50%;}
    to{width: 100%;}
}
```

Ici, dans notre premier exemple, on anime la couleur de fond de notre élément `div` sur une durée de 5 secondes. L'animation doit progresser de façon linéaire et commencer après un délai de 2 secondes. De plus, on demande à l'animation de se répéter à l'infini et d'alterner le sens dans lequel elle est jouée à chaque fois.

Dans notre deuxième exemple, on anime la taille du `div` sur 5 secondes. L'animation doit à nouveau progresser de façon linéaire et commencer après un délai de 2 secondes. L'animation doit se répéter 5 fois et nous demandons à ce que la propriété utilise la valeur de départ de l'animation avant que celle-ci ne commence et la valeur d'arrivée après la fin de l'animation.

Notre troisième exemple fait jouer les deux animations précédentes en même temps. Pour cela, on sépare les différentes déclarations par une virgule dans animation, tout simplement.

Créer des transformations en CSS

Dans les leçons précédentes, nous avons pu étudier les transitions et les animations en CSS qui permettent de modifier la valeur de certaines propriétés progressivement en fonction de certains critères (chargement de la page, passage de souris sur l'élément, etc.) et donc d'ajouter un côté interactif à nos pages web.

Le CSS va également nous permettre d'appliquer des transformations à nos éléments : on va pouvoir incliner nos éléments, les déformer, les translater, etc.

Dans cette leçon, nous allons expliquer en détail comment fonctionnent les transformations et apprendre à créer des transformations plus ou moins complexes.

Définir une transformation en CSS

La possibilité d'effectuer des transformations en CSS est récente et les possibilités et fonctionnalités des transformations sont donc aujourd'hui relativement limitées.

Cependant, on peut s'attendre à ce que de nouvelles fonctionnalités viennent s'ajouter aux transformations dans un futur proche.

A terme, les transformations en CSS devraient être définies par 3 critères qui vont pouvoir être renseignés via 3 propriétés CSS différentes :

- La propriété `transform-box` va nous permettre de définir une boite de référence qui va être utilisée pour calculer le point d'origine et pour réaliser la transformation en soi ;
- La propriété `transform-origin` va nous permettre de définir un point d'origine à partir duquel réaliser la transformation ;
- La propriété `transform` va nous permettre de définir un effet de transformation.

La propriété `transform-box` ne fait pas encore partie des recommandations du W3C et est toujours en développement. Il est donc déconseillé de l'utiliser pour l'instant puisque sa définition n'est pas encore stable et que le support par les navigateurs n'est pas assuré. Nous ne l'étudierons ainsi pas ici.

La propriété `transform-origin` permet de définir un point d'origine pour une transformation. Par défaut, le point d'origine est le centre de l'élément.

Cette propriété va pouvoir prendre une ou deux valeurs. En n'indiquant qu'une valeur, la valeur sera utilisée pour définir les coordonnées dans l'axe horizontal et dans l'axe vertical du point d'origine. En indiquant deux valeurs, la première valeur va permettre de définir la coordonnée horizontale du point d'origine tandis que la seconde valeur va permettre de définir sa coordonnée verticale.

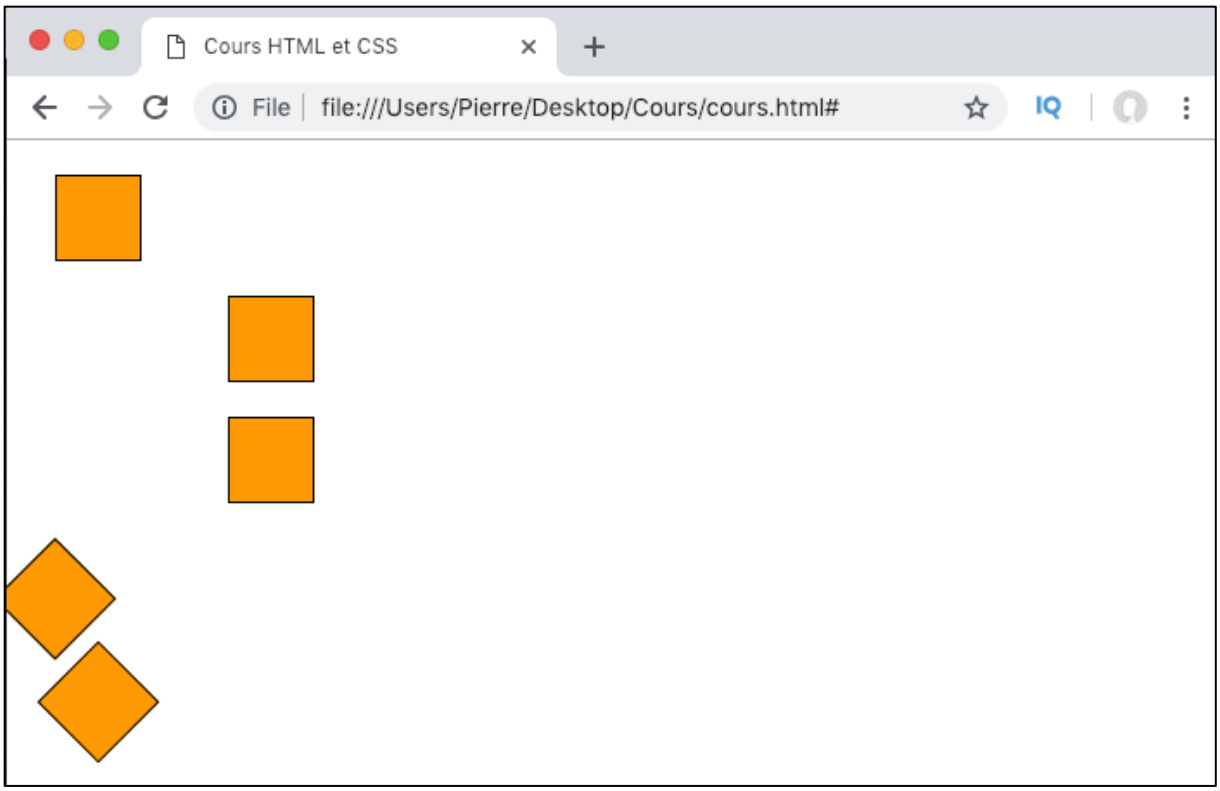
Nous allons pouvoir passer les mots clefs `top`, `right`, `bottom`, `left` et `center` ainsi que des longueurs ou des pourcentages à cette propriété. Les valeurs de type longueur ou pourcentage vont définir l'éloignement du point d'origine à partir du bord supérieur gauche de la boite de référence.

La propriété `transform` va nous permettre de définir un ou plusieurs effets de transformation à appliquer à un élément : inclinaison, rotation, déformation, etc.

Cette propriété va accepter différents mots clefs ou plus exactement différentes fonctions qui vont définir le type de transformation qui va être appliqué à un élément. Nous allons étudier ces valeurs dans la suite de la leçon.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div></div>
        <div class="d1"></div>
        <div class="d2"></div>
        <div class="d3"></div>
        <div class="d4"></div>
    </body>
</html>
```

```css
div{
  width: 50px;
  height: 50px;
  background-color: orange;
  border: 1px solid black;
  box-sizing: border-box;
  margin: 20px;
}
.d1{
  transform-origin: 0 0;
  transform: translate(100px);
}
.d2{
  transform-origin: 50% 50%;
  transform: translate(100px);
}
.d3{
  transform-origin: 0 0;
  transform: rotate(45deg);
}
.d4{
  transform-origin: 50% 50%;
  transform: rotate(45deg);
}
```



Dans le code ci-dessus, nous avons défini 4 transformations pour nos 4 `div` portant des attributs `class`.

Nous appliquons un premier effet de transformation à nos deux premiers `div` qui va être une translation, c'est-à-dire un déplacement selon une direction. Ici, on va donc déplacer nos deux `div` de 100px à partir de leur point d'origine vers la droite.

Ensuite, nous appliquons un effet de rotation à nos deux derniers `div`. La rotation va s'effectuer dans le sens des aiguilles d'une montre.

Le point d'origine de la transformation pour le premier `div` est son coin supérieur gauche tandis que cela va être le centre pour le deuxième. Comme la transformation n'est ici qu'un déplacement horizontal, modifier le point d'origine ne change pas le résultat de la transformation.

En revanche, cela va être différent pour une rotation : notre troisième `div` va pivoter autour d'un point central qui va être son coin supérieur gauche tandis que notre quatrième `div` va pivoter autour de son point central. Dans ce cas-là, modifier le point d'origine de la transformation change le résultat obtenu.

Finalement, vous pouvez noter ici qu'à la différence des transitions et des animations pour lesquelles on précise une durée, les transformations sont permanentes et les éléments vont conserver la transformation.

Exemples de transformations 2D

Nous allons pour le moment nous concentrer sur les effets de transformations en deux dimensions. En effet, vous devez savoir qu'on peut également aujourd'hui créer des

transformations en 3D en rajoutant une perspective à nos éléments. Nous discuterons de cette possibilité plus tard.

Modifier la taille ou l'échelle d'un élément avec scale()

La fonction `scale()` permet de modifier la taille ou plus exactement l'échelle de l'élément. Nous allons pouvoir lui passer deux nombres qui vont correspondre au pourcentage d'agrandissement en largeur et en hauteur de l'élément.

Par exemple, en écrivant `transform : scale(2, 0.5)`, l'élément va doubler en largeur et être diminué de moitié en hauteur.

Notez que cette fonction ne va pas affecter que les propriétés `width` et `height` de l'élément mais également s'appliquer au `font-size` et au `padding`.

Finalement, vous devez savoir que la fonction `scale()` est la notation raccourcie des fonctions `scaleX()` et `scaleY()` qui permettent de ne modifier que la largeur et la hauteur respectivement d'un élément.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div></div>
        <div class="d1"></div>
        <div class="d2"></div>
        <div class="d3"></div>
    </body>
</html>
```

```css
div{
    width: 200px;
    height: 50px;
    background-color: orange;
    border: 1px solid black;
    box-sizing: border-box;
    margin: 25px auto;
}
.d1{
    transform: scale(2);
}
.d2{
    transform: scale(2, 0.5);
}
.d3{
    transform: scaleX(3);
}
```

Ici, on ne passe qu'une valeur à notre fonction `scale()` pour notre première transformation : cette valeur va donc être utilisée pour calculer la mise à l'échelle horizontale et verticale et notre `div` fera deux fois sa taille d'origine.

Pour notre deuxième transformation, on demande à ce que le `div` devienne deux fois plus large et qu'il soit deux fois moins haut.

Finalement, on utilise la fonction `scaleX()` pour définir notre dernière transformation qui ne va donc impacter que la largeur de l'élément.

Notez bien ici qu'appliquer une transformation à des éléments ne va pas modifier la taille de l'espace qui leur était attribué à la base ni donc faire bouger les autres éléments autour. Il faudra donc faire bien attention à anticiper les chevauchements et dépassements possibles entre éléments transformés et leurs voisins.

Déformer un élément avec skewX() et skewY()

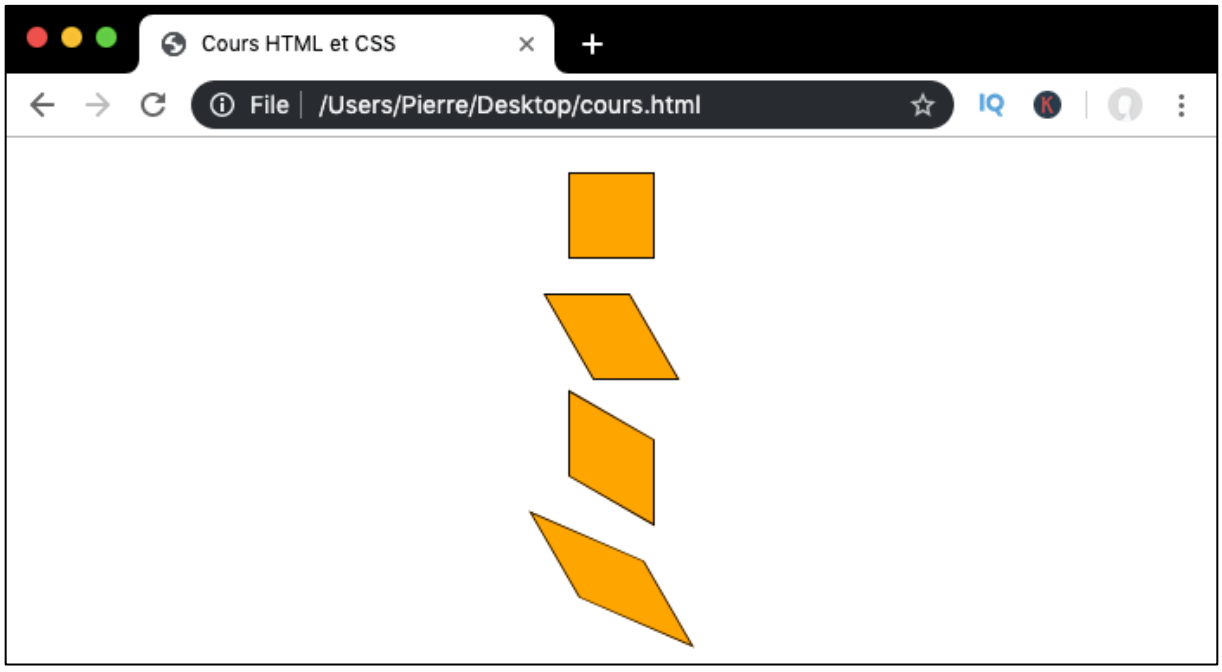
Avant toute chose, il faut savoir que la fonction `skew()`, si elle existe, n'a été créée que pour des questions de compatibilité et qu'on ne devrait jamais l'utiliser.

A la place, il faudra plutôt utiliser les fonctions `skewX()` et `skewY()` qui vont nous permettre de déformer un élément selon son axe horizontal ou vertical.

Nous allons devoir passer un angle (généralement en `deg`) à ces deux fonctions qui va représenter l'angle selon lequel l'élément doit être déformé le long de l'axe correspondant.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div></div>
        <div class="d1"></div>
        <div class="d2"></div>
        <div class="d3"></div>
    </body>
</html>
```

```css
div{
    width: 50px;
    height: 50px;
    background-color: orange;
    border: 1px solid black;
    box-sizing: border-box;
    margin: 20px auto;
}
.d1{
    transform: skewX(30deg);
}
.d2{
    transform: skewY(30deg);
}
.d3{
    transform: skewX(30deg) skewY(30deg);
}
```



Effectuer une translation avec translate(X,Y)

La fonction `translate()` va nous permettre de créer une translation, c'est-à-dire de déplacer un élément selon un certain vecteur (ou selon une certaine distance et direction, pour faire simple).

Là encore, on va pouvoir passer deux valeurs de type longueur à `translate()` pour spécifier les caractéristiques de la translation dans l'axe horizontal et dans l'axe vertical ou utiliser les versions complètes `translateX()` et `translateY()`.

Les valeurs passées vont pouvoir être positives ou négatives. Une valeur positive pour l'axe horizontal va déplacer l'élément vers la droite tandis qu'une valeur positive pour l'axe vertical va déplacer l'élément vers le bas et inversement pour des valeurs négatives.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div></div>
        <div class="d1"></div>
        <div class="d2"></div>
        <div class="d3"></div>
    </body>
</html>
```

```css
div{
    width: 50px;
    height: 50px;
    background-color: orange;
    border: 1px solid black;
    box-sizing: border-box;
    margin: 20px;
    padding: 15px 0px;
    text-align: center;
}
.d1{
    transform: translate(100px, -50px);
    background-color: yellow;
}
.d2{
    transform: translateX(100px);
    background-color: green;
}
.d3{
    transform: translateY(50px);
    background-color: blue;
}
```

Effectuer une rotation avec rotate()

La fonction `rotate()` va nous permettre de faire pivoter un élément ou de lui faire effectuer une rotation selon un certain angle. Nous allons pouvoir lui fournir un angle (généralement en `deg`) en valeur.

La rotation va s'effectuer dans le sens horaire (sens des aiguilles d'une montre). Ainsi, indiquer `rotate(90deg)` va faire pivoter l'élément d'un quart de tour.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div></div>
        <div class="d1">d1</div>
        <div class="d2">d2</div>
        <div class="d3">d3</div>
    </body>
</html>
```

```css
div{
    width: 50px;
    height: 50px;
    background-color: orange;
    border: 1px solid black;
    box-sizing: border-box;
    margin: 20px;
    padding: 15px 0px;
    text-align: center;
}
.d1{
    transform: rotate(45deg);
    background-color: yellow;
}
.d2{
    transform: rotate(90deg);
    background-color: green;
}
.d3{
    transform: rotate(225deg);
    background-color: blue;
}
```

Définir une matrice de transformation avec matrix()

La fonction `matrix()` va nous permettre de définir notre propre matrice de transformation. Son utilisation est réservée aux personnes qui possèdent un bon degré de connaissance des transformations ET en mathématiques. Comme son usage est très marginal, je ne pense pas qu'il soit pertinent d'expliquer son fonctionnement en détail dans ce cours.

Appliquer plusieurs transformations d'un coup

Nous allons pouvoir définir plusieurs transformations à appliquer à un élément avec transform. Pour cela, il va suffire d'indiquer les différents effets de transformation à la suite.

Notez bien ici que les transformations ne vont pas toutes s'effectuer en même temps mais plutôt les unes à la suite des autres selon leur ordre de déclaration.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div></div>
        <div class="d1">d1</div>
        <div class="d2">d2</div>
    </body>
</html>
```

```css
div{
    width: 50px;
    height: 50px;
    background-color: orange;
    border: 1px solid black;
    box-sizing: border-box;
    margin: 20px;
    padding: 15px 0px;
    text-align: center;
}
.d1{
    transform: translateX(100px) rotate(45deg) ;
    background-color: yellow;
}
.d2{
    transform: rotate(45deg) translateX(100px);
    background-color: green;
}
```

Expliquons le code ci-dessus. Ici, on applique les mêmes transformations à nos deux `div` mais pas dans le même ordre.

Le premier `div` auquel on applique une transformation va d'abord effectuer une translation de 100px vers la droite puis une rotation de 45 degrés dans le sens horaire.

Le deuxième `div`, au contraire, va lui d'abord effectuer une rotation de 45 degrés puis une translation de 100px. La différence est ici qu'après sa rotation l'axe horizontal du `div` (qu'il serait plus correct d'appeler l'axe des abscisses ou axe des X) est également incliné de 45 degrés et la translation va se faire selon cet axe.

Notre second `div` va donc être décalé de 100px selon cet axe incliné à 45 degrés.

Animer des transformations

Notez enfin qu'on va tout à fait pouvoir utiliser les transformations au sein d'animations en CSS. Cela va nous permettre d'ajouter un certain dynamisme à nos pages web.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="d1">d1</div>
        <div class="d2">d2</div>
        <div class="d3">d3</div>
    </body>
</html>
```

```css
div {
  display: inline-block;
  background-color: RGBa(240,160,0,0.5);
  height: 100px;
  width: 100px;
  margin: 50px 5px;
  animation: rotation 5s linear infinite;

}
.d2{
  animation: rotation 5s linear infinite reverse;
}
@keyframes rotation {
  0% {
    transform: rotate(0) translateX(0px);
  }
  25%{
    transform: rotate(90deg) translateX(50px);
  }
  50%{
    transform: rotate(180deg) translateX(0px);
  }
  75%{
    transform: rotate(270deg) translateX(50px);
  }
  100% {
    transform: rotate(360deg) translateX(0px);
  }
}
```

Les transformations 3D

Finalement, vous devez savoir qu'on va également pouvoir créer des transformations en 3D, c'est-à-dire en rajoutant un axe Z qui va créer une perspective. Cet axe va nous permettre de simuler une profondeur et on va pouvoir ainsi faire comme si des éléments se rapprochaient ou s'éloignaient de l'utilisateur.

Avec les transformations 3D, nous commençons à toucher à des choses vraiment complexes en CSS et qui ne sont pas forcément pertinentes dans le cadre de ce cours complet car elles sont très peu utilisées et très spécifiques.

J'aborde ce sujet par souci d'exhaustivité mais je vais me contenter de vous expliquer rapidement le principe de fonctionnement des transformations 3D et vous donner quelques exemples qui me semblent les plus pertinents car encore une fois je ne pense pas qu'il soit pertinent de s'arrêter trop longtemps sur ces notions et d'essayer à tout prix de comprendre comment elles fonctionnent en détail.

Pour qu'une transformation 3D ait l'effet visuel escompté, il va déjà falloir créer un effet de perspective c'est-à-dire créer une impression d'espace 3D. Pour cela, nous allons utiliser la fonction `perspective()`avec nos effets de transformation.

La valeur passée à la fonction `perspective()` va définir l'intensité de l'effet 3D. On peut considérer qu'elle va représenter la distance entre l'utilisateur et l'élément. Plus la valeur passée à `perspective()` va être grande, plus l'élément sera éloigné de l'utilisateur au départ et moins l'effet 3D sera visible.

Ensuite, nous allons pouvoir définir nos effets de transformation 3D : mises à l'échelle en 3D, translations 3D ou rotations 3D. Nous allons également pouvoir créer nos propres transformations 3D grâce à la fonction `matrix3d()` mais il faudra pour cela comprendre comment fonctionne le calcul matriciel en mathématiques.

Pour chaque effet de transformation cité ci-dessus, nous allons soit pouvoir utiliser les fonctions `scale3d()`, `translate3d()` et `rotate3d()`, soit utiliser les notations longues `scaleX()`, `scaleY()`, etc. et en les complétant avec `scaleZ()`, `translateZ()` et `rotateZ()`.

Voici deux exemples de translation et de rotation 3D qu'on va effectuer durant une animation :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="d1">d1</div>
        <div class="d2">d2</div>
        <div class="d3">d3</div>
    </body>
</html>
```

```css
div {
  display: inline-block;
  background-color: RGBa(240,160,0,0.5);
  height: 100px;
  width: 100px;
  margin: 50px 5px;
}
.d2{
  animation: translation 5s linear infinite;
}
@keyframes translation {
  0% {
    transform: perspective(100px) translate3d(0px, 0px, 0px);
  }
  25%{
    transform: perspective(100px) translate3d(0px, 0px, -100px)
  }
  50% {
    transform: perspective(100px) translate3d(0px, 0px, 0px);
  }
  75% {
    transform: perspective(100px) translate3d(0px, 0px, 50px);
  }
  100% {
    transform: perspective(100px) translate3d(0px, 0px, 0px);
  }
}
```

```css
div {
  display: inline-block;
  background-color: RGBa(240,160,0,0.5);
  height: 100px;
  width: 100px;
  margin: 50px 5px;
  animation: rotation 5s linear infinite;

}
@keyframes rotation {
  0% {
    transform: perspective(100px) rotate3d(1,1,1,0deg);
  }
  50%{
    transform: perspective(100px) rotate3d(1,1,1,180deg);
  }
  100% {
    transform: perspective(100px) rotate3d(1,1,1,360deg);
  }
}
```

EXERCICE #3 : Créer un diaporama en HTML et CSS

L'objectif de ce nouvel exercice est d'essayer de créer un diaporama, c'est-à-dire un enchainement fluide d'images, en HTML et en CSS.

L'idée ici va être d'utiliser judicieusement la propriété `background-image` et les animations et transformations pour modifier l'image de fond d'un élément et ainsi créer un effet de diaporama.

Nous n'allons évidemment pas pouvoir coder certaines fonctionnalités que possèdent les diaporamas « complets » créés avec du JavaScript mais allons essayer de réaliser quelque chose qui va s'en rapprocher.

Je vous propose ici de créer deux animations différentes pour créer deux diaporamas : une première animation de fondu et une seconde qui va faire défiler nos images comme dans le cas d'un diaporama classique.

De plus, pour chaque animation de diaporama, nous allons créer deux versions : une version avec un diaporama de taille fixe et une version avec un diaporama qui va s'adapter en fonction de la taille de la fenêtre.

Création d'un diaporama avec effet de fondu

Pour notre premier diaporama, nous allons vouloir créer un effet de fondu, c'est-à-dire de disparition progressive d'une image et d'apparition progressive d'une autre.

Pour cela, je vous propose de travailler en 3 étapes :

1. Création des cadres du diaporama en HTML ;
2. Mise en forme des cadres en CSS ;
3. Création de l'animation en CSS.

Le code HTML du diaporama

A manière la plus simple de créer un diaporama en HTML et en CSS va être de faire défiler des images de fond.

Pour pouvoir faire cela, il va nous falloir un élément conteneur ou cadre auquel on va ensuite pouvoir passer des images de fond.

Nous allons ainsi nous contenter en HTML d'utiliser des éléments `div` comme cadres.

On va créer un premier `div class="d1"` qui va représenter notre cadre de diaporama à dimension fixe et un deuxième `div class="d2"` qu'on va lui-même placer dans un `div class="conteneur"` et qui va représenter notre cadre de diaporama aux dimensions adaptables.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="d1"></div>
        <div class="conteneur">
            <div class="d2"></div>
        </div>
    </body>
</html>
```

Mise en forme des cadres en CSS

Pour créer nos diaporamas en CSS nous allons nous servir de la propriété `background-image` que nous allons ensuite animer grâce à la propriété `animation`.

En pratique, la propriété `background-image` s'utilise souvent pour apporter un fond à un élément HTML qui possède du contenu et donc une hauteur définie.

Ici, nos `div` qui servent de cadres à notre diaporama ne vont contenir que des images de fond et aucun « vrai » contenu. Par défaut, leur hauteur va donc être nulle puisqu'une image de fond n'est qu'un fond par définition et qu'un fond ne peut pas impacter la taille d'un élément.

Nous allons donc devoir préciser explicitement une hauteur pour nos conteneurs. Nous allons pouvoir le faire de deux façons différentes selon qu'on souhaite créer un diaporama avec un cadre possédant des dimensions fixes ou en pourcentage.

Cadre de diaporama avec dimensions fixes

Pour utiliser un cadre de diaporama avec dimensions fixes, nous allons tout simplement préciser une largeur et une hauteur explicites en `px` pour notre `div`.

Il faudra ensuite recadrer nos images de fond à la même taille que le `div` ou à minima faire en sorte qu'elles possèdent le même ratio largeur / hauteur que celui-ci afin d'avoir un bon affichage.

L'avantage de cette première méthode est qu'on va avoir un comportement stable et prévisible pour les dimensions de notre diaporama. En contrepartie, le diaporama fera toujours la même taille quelle que soit la taille de l'écran de vos visiteurs.

Notez qu'on va pouvoir définir plusieurs tailles fixes selon certains paliers de tailles grâce aux media queries que nous étudierons dans la suite de ce cours.

```css
.d1{
    width: 576px;
    height: 432px;
    margin: 50px auto;
    box-shadow: 0px 15px 10px -5px #777;
    background-color: #EDEDED;
    background-size: contain;
}
```

Ici, on indique explicitement des dimensions pour notre `div` qui va contenir notre diaporama par défaut ainsi qu'une couleur de fond. Pour le moment, nous créons simplement le cadre et n'utilisons donc pas d'image.

Il est toutefois toujours essentiel de préciser une couleur de fond au cas où les images du diaporama ne pourraient pas pour une raison ou une autre s'afficher.

Ensuite, on définit un `background-size: contain` pour que nos images de fond soient à la fois contraintes dans le conteneur mais occupe le plus d'espace dans celui-ci tout en conservant leur proportions d'origine.

On crée également un effet d'ombre sous notre `div` avec `box-shadow` pour donner l'impression qu'il est au-dessus de la page.

Cadre de diaporama avec des dimensions en %

Créer un cadre de diaporama qui va se redimensionner en même temps qu'on va changer la taille de la fenêtre va être un peu plus complexe.

En effet, je vous rappelle qu'une image de fond n'est pas considérée comme un contenu en soi et donc que la hauteur de notre cadre est nulle par défaut.

Ici, quand on agrandit ou rétrécit la fenêtre, il va falloir que la taille de notre cadre s'adapte et que nos images de fond s'affichent entièrement dans tous les cas.

Pour faire cela, nous allons utiliser un petit hack CSS. Nous allons définir explicitement une hauteur nulle pour notre `div` et une largeur égale à 100% et utiliser la propriété `padding-top`.

Le `padding-top` va ici servir à donner une hauteur au cadre. On va lui passer une valeur en pourcentage qui va correspondre au ratio hauteur / largeur de l'image qu'on souhaite voir s'afficher.

Par exemple, si mon image de fond fait 1200px de large par 900px de haut, le ratio hauteur / largeur est de 900/1200 = 3/4 = 75%. On indiquera donc un `padding-top: 75%` dans ce cas.

Cela va faire que notre cadre va se redimensionner en gardant toujours ce ratio de ¾, c'est-à-dire qu'il aura toujours une hauteur égale à 75% de sa largeur.

```css
.d2{
    width: 100%;
    height: 0px;
    padding-top: 75%;
    margin: 50px auto;
    box-shadow: 0px 0px 10px #777;
    background-color: #EDEDED;
    background-size: contain;
}
```

Ici, d'un point de vue purement esthétique, on fait cette fois-ci le choix d'utiliser un `box-shadow` centré autour du `div`.

L'avantage de cette deuxième méthode est que notre cadre de diaporama va pouvoir se redimensionner en même temps que la fenêtre. L'inconvénient est qu'on ne va pas pouvoir maitriser la hauteur du diaporama qui va pouvoir atteindre une très grande taille sur de grands écrans, ce qui peut être un comportement indésirable.

C'est la raison pour laquelle nous avons placé notre `div class="d2"` dans un autre `div class="conteneur"`qui va nous servir de conteneur et pour lequel on va préciser une taille maximale avec `max-width` afin de limiter la taille de notre diaporama à partir d'une certaine taille de fenêtre.

```css
.d2{
    width: 100%;
    height: 0px;
    padding-top: 75%;
    box-shadow: 0px 0px 10px #777;
    background-color: #EDEDED;
    background-size: contain;
}
```

```css
.conteneur{
    max-width: 576px;
    margin: 50px auto;
}
```

Création de l'animation de fondu du diaporama

Il ne nous reste plus qu'à créer notre animation de fondu pour rendre notre diaporama fonctionnel. Pour ce diaporama, je vais utiliser trois images que vous pouvez télécharger en cliquant ici (la quatrième image dans le dossier va être utilisée pour notre prochain diaporama).

L'animation de fondu va être relativement simple à réaliser : nous allons simplement passer nos différentes images à la propriété `background-image` à différents moments de l'animation.

Comme le principe d'une animation est de passer progressivement d'une valeur de départ à une valeur d'arrivée, nos images vont s'enchainer avec un effet de fondu de façon naturelle.

```
@keyframes fondu{
    0%{background-image: url("diapo1.png");}
    33.33%{background-image: url("diapo2.png");}
    66.67%{background-image: url("diapo3.png");}
    100%{background-image: url("diapo1.png");}
}
```

Nous allons répéter l'animation à l'infini pour que nos images s'enchainent constamment et allons préciser le comportement du `background-image` avant et après l'animation au cas où celle-ci ne puisse pas se lancer.

```
.d1{
    width: 576px;
    height: 432px;
    margin: 50px auto;
    box-shadow: 0px 15px 10px -5px #777;
    background-color: #EDEDED;
    background-size: contain;
    animation: fondu 15s ease-in-out infinite both;
}
.conteneur{
    max-width: 576px;
    margin: 50px auto;
}
.d2{
    width: 100%;
    height: 0px;
    padding-top: 75%;
    box-shadow: 0px 0px 10px #777;
    background-color: #EDEDED;
    background-size: contain;
    animation: fondu 15s ease-in-out infinite both;
}
```

Finalement, nous allons vouloir mettre en pause l'animation lorsqu'un utilisateur passe sa souris sur le cadre du diaporama, pour lui laisser le temps d'apprécier l'image.

```
.d1:hover, .d2:hover{
    animation-play-state: paused;
}
```

Voilà tout pour notre premier effet de diaporama.

Création d'un diaporama avec défilement des images

Essayons maintenant de créer un diaporama avec un effet de défilement des images en fond.

L'idée ici va être de créer une maxi-image qui va contenir toutes les images de notre diaporama. Nous allons ensuite placer cette image en fond et allons animer sa position pour la faire dérouler.

Pour animer la position de l'image de fond, nous allons pouvoir soit utiliser la propriété `background-position` soit la propriété `transform` avec sa fonction `translate()`. Cette deuxième méthode est à privilégier car elle est plus performante.

Code HTML du diaporama avec effet de défilement

Nous allons à nouveau essayer de créer deux diaporamas : un avec des dimensions fixes et un qui va s'adapter en fonction de la taille de la fenêtre.

Nous allons donc déjà avoir besoin de deux cadres pour nos diaporamas qui vont être représentés par deux `div`. Ici, nous allons placer chacun de ces cadres dans un autre `div` conteneur.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur1">
            <div class="d1"></div>
        </div>
        <div class="conteneur2">
            <div class="d2"></div>
        </div>
    </body>
</html>
```

Diaporama de dimensions fixes avec effet de défilement

Commençons déjà par essayer de créer un diaporama avec effet de défilement et des dimensions fixes.

L'idée derrière l'effet de défilement va être ici de n'utiliser qu'une grande image qui va être en fait composée de plusieurs images et de la faire bouger.

On peut donc déjà définir notre image de fond avec `background-image`. Je vais pour ma part utiliser une image qui a été créée à partir des 3 images du diaporama précédent et que j'ai appelée `diapo123.png`.

Mon image fait ici 2880px de largeur par 720px de hauteur et est composée de 3 images de tailles identiques (960px*720px).

On va vouloir que l'effet de défilement soit infini et nous allons créer cet effet avec une transformation de type translation.

On va déjà commencer par donner une taille égale à notre grande image + la largeur d'une sous image à notre `div` afin de pouvoir créer une boucle infini fluide. Dans mon cas, il faut

donc que je donne une taille de 2880px + 960px = 3840px à mon div. Nous allons fixer une hauteur du div égale à celle de l'image de fond à savoir 720px.

```
.d1{
    width: 3840px;
    height: 720px;
    background-color: #EDEDED;
    background: url("diapo123.png");
    background-size: contain;
}
```

Nous avons donc un cadre de 3840px de largeur. Cependant, nous voulons que la partie visible du diaporama soit égale à la largeur de chacune de nos sous images (on pourrait tout à fait décider d'une largeur différente mais c'est la largeur qui fait le plus de sens selon moi).

Pour cela, nous allons passer une largeur maximale à notre div conteneur égale à la largeur de nos sous images et également lui passer un overflow : hidden pour cacher ce qui dépasse du conteneur.

```
.conteneur1{
    overflow: hidden;
    max-width: 960px;
    margin: 50px auto;
    box-shadow: 0px 15px 10px -5px #777;
}
```

Finalement, nous allons créer notre effet de défilement en animant une transformation de type translation. Au début de notre animation, on va vouloir afficher la première sous image dans le cadre puis on va vouloir faire défiler la grande image jusqu'à arriver à nouveau sur une vue montrant la première sous image.

```
@keyframes defilement1{
    0%{transform: translate(0,0);}
    100%{transform: translate(-2880px,0);}
}
```

A la fin de notre animation, l'image s'est déplacée de sa taille exactement de sa largeur. On utilise ici la fait qu'une image de fond est par défaut répétée pour remplir le fond d'un élément, ce qui fait que notre image de fin d'animation est la même que celle du début (la première sous image est bien répétée du fait que la largeur du cadre soit égale à celle de l'image de fond + la largeur d'une sous-image).

Ensuite, il ne nous reste plus qu'à répéter cette boucle à l'infini en définissant un nombre d'animations infini.

```css
.d1{
    width: 3840px;
    height: 720px;
    background-color: #EDEDED;
    background: url("diapo123.png");
    background-size: contain;
    animation: defilement1 12s linear infinite;
}
```

Diaporama avec effet de défilement aux dimensions adaptables

Nous allons finalement pouvoir créer un diaporama avec effet de défilement et aux dimensions adaptables sur le même modèle que ce qu'on a pu faire précédemment mais en convertissant les différentes valeurs en pourcentage.

La principale difficulté/ astuce ici va être de jouer avec le ratio de notre image et la largeur de notre cadre pour faire en sorte que les sous images s'affichent complètement à chaque fois et que le défilement soit fluide.

Pour rappel, notre image de fond fait 2880px*720px ce qui signifie qu'elle est quatre fois plus large que haute. Chacune des sous images qu'elle contient à un ratio largeur/ hauteur de 4 : 3.

Ici, on va commencer par passer une largeur `width : 400%` à notre cadre. Ensuite, on va lui attribuer un `padding-top : 75%` afin que la hauteur du `div` soit toujours égale à 75% de la largeur visible du `div`.

La partie visible du `div` cadre aura donc toujours un ratio de 4 : 3. En utilisant `background-size : contain`, la première répétions de notre maxi image de fond va donc prendre une largeur égale à 300% de la partie visible du `div`.

Comme notre `div` cadre possède une largeur de 400%, notre image de fond va donc s'afficher une fois complètement dedans puis un tiers de l'image va se répéter (ce qui va correspondre à notre première sous image se répétant).

```css
.d2{
    width: 400%;
    height: 0;
    padding-top: 75%;
    background-color: #EDEDED;
    background-image: url("diapo123.png");
    background-size: contain;
}
```

On va ensuite passer une largeur maximale à notre `div` conteneur qu'on va définir ici comme égale à une de nos sous images pour éviter que notre diaporama ne dépasse une certaine taille sur les grands écrans. On passe également un `overflow : hidden` pour cacher la partie du diaporama qui dépasse de l'écran.

```css
.conteneur2{
    max-width: 960px;
    overflow: hidden;
    margin: 50px auto;
    box-shadow: 0px 0px 10px #777;
}
```

Il ne nous reste plus qu'à définir notre `@keyframes` avec notre translation et les propriétés de notre animation en soi.

De manière similaire à ce qu'on a pu faire précédemment, on va faire en sorte que la première boucle de l'animation des termine exactement lorsque notre première sous image occupe à nouveau l'espace visible dans le cadre.

```css
@keyframes defilement2{
    0%{transform: translate(0,0);}
    100%{transform: translate(-75%,0);}
}
```

Ici, on indique un `transform : translate(-75%,0)` en fin d'animation. En effet, notre cadre fait 400% de largeur avec un ratio visible de 4 : 3. Notre image de fond va occuper 75% de cette largeur et va se répéter pour les derniers 25%.

Ainsi, `transform : translate(-75%,0)` nous ramène exactement à la position de l'image de fond en début d'animation et nous n'avons plus qu'à répéter l'animation à l'infini.

```css
.d2{
    width: 400%;
    height: 0;
    padding-top: 75%;
    background-color: #EDEDED;
    background-image: url("diapo123.png");
    background-size: contain;
    animation: defilement2 12s linear infinite;
}
```

PARTIE XIII

Modèle des boites flexibles

Introduction au modèle des boites flexibles ou flexbox

Dans ce cours, nous avons vu que les éléments HTML pouvaient être affichés de différentes manières et que le type d'affichage conditionnait la disposition par défaut d'un élément :

- Affichage en bloc avec `display : block` ;
- Affichage en ligne avec `display : inline` ;
- Affichage sous forme de tableau avec `display : table` ;
- Etc.

Le but de cette partie est de vous présenter un nouveau mode d'affichage et de disposition très puissant : la disposition selon un modèle de boites flexibles ou flexbox avec `display : flex` (ou `display : inline-flex`) et les différentes propriétés CSS liées à ce modèle.

Définition du modèle des boites flexibles ou flexbox

Le flexbox est un modèle de disposition très puissant qui va nous permettre de contrôler facilement et avec précision l'alignement, la direction, l'ordre et la taille de nos éléments (ou plus précisément de nos boites).

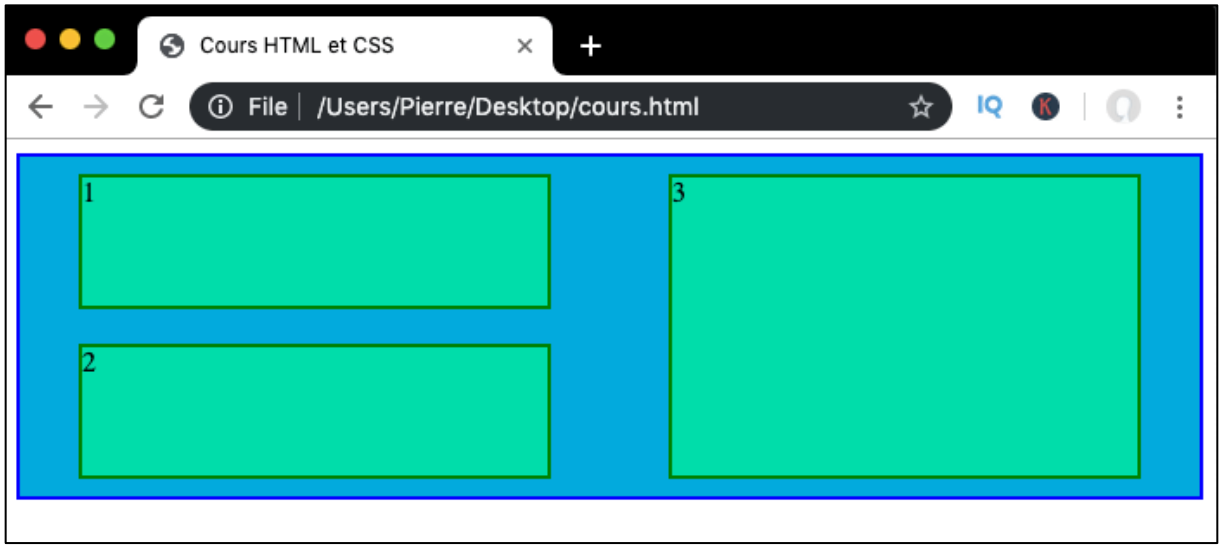
Avec le modèle des boites flexibles, nous allons pouvoir définir des conteneurs flexibles. Ces conteneurs sont dits « flexibles » car leurs enfants directs vont être des éléments flexibles qui vont pouvoir se réarranger (se redimensionner, se réaligner, etc.) automatiquement dans leur conteneur lorsque celui-ci change de dimension.

Le flexbox représente ainsi en quelque sorte une alternative beaucoup plus puissante à la propriété `float` qui nous permettait de positionner nos boites horizontalement puisqu'on va avoir cette fois-ci un contrôle total sur la disposition des éléments.

Voici un premier exemple d'organisation des éléments qu'on peut obtenir avec quelques lignes de code simplement en utilisant le modèle des boites flexibles et ses propriétés :

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur-flexible">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    flex-flow: column wrap;
    align-items: center;
    justify-content: space-around;
    align-content: space-around;
    background-color: #0AD; /*Bleu*/
    width: 100%;
    height: 200px;
    border: 2px solid blue;
    box-sizing: border-box;
}

.element-flexible{
    flex: 1 1 50px;
    background-color: #0DA; /*Vert*/
    width: 40%;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 10px;
}
```



N’essayez pas pour le moment de comprendre le code ci-dessus : nous allons étudier toutes les spécificités du flexbox dans les leçons à venir !

Conteneur et éléments flexibles

Le modèle des boites flexibles fait la distinction entre deux types de boites auxquelles on va pouvoir appliquer différentes propriétés : des conteneurs flexibles d’un côté et des éléments flexibles ou « flex items » de l’autre.

Concrètement, nous allons définir un conteneur flexible en attribuant un `display : flex` à un élément. Tous les éléments directement contenus dans ce conteneur (c’est-à-dire tous les enfants directs) vont alors automatiquement devenir des éléments flexibles.

Nous allons ensuite pouvoir manipuler chaque flex item de manière indépendante grâce à certaines propriétés du modèle des boites flexibles et / ou leur appliquer des propriétés de disposition en tant que groupe en manipulant le conteneur flexible.

Notez également qu’on va pouvoir créer autant de niveau d’imbrication de conteneurs et d’éléments flexibles que souhaité : il va en effet suffire d’ajouter un `display : flex` à un flex item pour qu’il serve également de conteneur flexible pour ses propres enfants.

Finalement, notez qu’on va pouvoir choisir de définir un conteneur flexible qui va avoir les mêmes propriétés qu’un élément de niveau `block` avec `display : flex` ou un conteneur qui va posséder les mêmes propriétés qu’un élément de niveau `inline` avec `display : inline-flex`.

Définir un conteneur flexible avec `display : flex` ou `display : inline-flex` n’impactera que le comportement du conteneur et pas celui de ses enfants directs qui seront toujours des éléments flexibles.

Notez bien ici que les conteneurs flexibles définissent un nouveau contexte de formatage, ce qui signifie que certaines propriétés qu’on va pouvoir utiliser avec des éléments de type `block` par exemple ne vont pas se comporter de la même façon avec les éléments flexibles ou ne vont simplement pas pouvoir être utilisées.

En particulier, on notera que les propriétés `float`, `clear` et `vertical-align` ne vont pas pouvoir s’appliquer à des éléments flexibles et que les marges haute et basse d’un conteneur flexible ne vont pas fusionner avec celles de ses enfants.

Axe principal et axe secondaire des boites flexibles

En plus de la distinction conteneur flexible / éléments flexibles, vous devez absolument comprendre la notion d’axe principal et d’axe secondaire des boites flexibles.

En effet, le comportement (et le résultat) de la plupart des propriétés liées au modèle des boites flexibles va être intimement lié à cette notion d’axe.

Pour le dire très simplement, deux axes vont intervenir dans le modèle des boites flexibles : l'axe horizontal et l'axe vertical.

La propriété `flex-direction` va nous permettre de définir quel va être l'axe principal pour un conteneur flexible et ses éléments flexibles ainsi que la direction des éléments le long de cet axe. Le deuxième axe sera ainsi appelé axe secondaire.

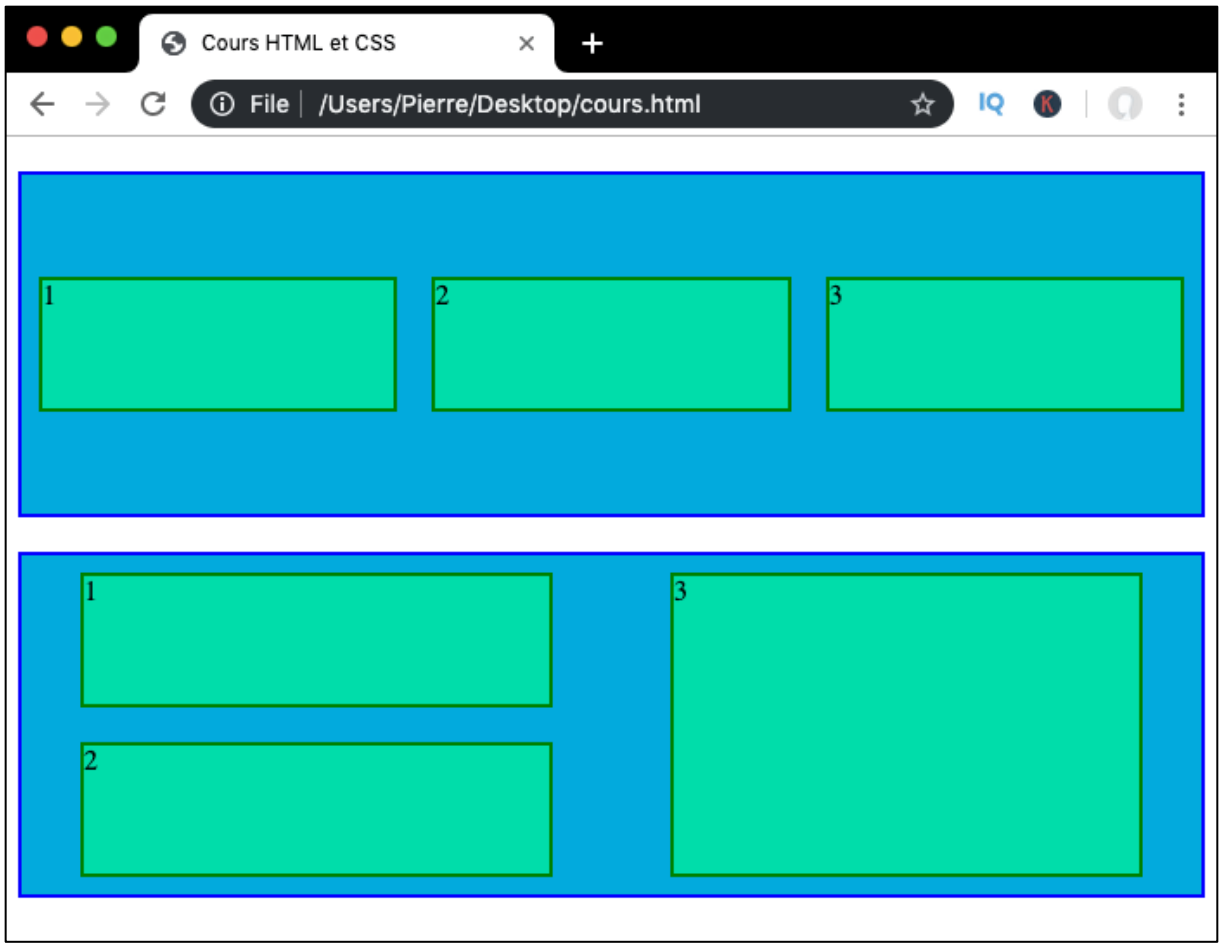
Par défaut, l'axe principal est l'axe horizontal (et la direction va être celle de l'écriture) et l'axe secondaire est l'axe vertical (et la direction va être de haut en bas).

Une nouvelle fois, la plupart des propriétés du flexbox vont permettre d'organiser les éléments selon un axe (certaines selon l'axe principal, d'autres selon l'axe secondaire) et selon la direction. Il va donc être essentiel de bien savoir quel est notre axe principal et quel est notre axe secondaire !

Regardez plutôt l'exemple ci-dessous pour vous en convaincre. La seule différence entre les deux conteneurs flexibles est que l'axe principal du premier défini avec `flex-direction` est l'axe horizontal tandis que l'axe principal du second est l'axe vertical.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur-flexible ligne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    flex-wrap: wrap;
    align-items: center;
    justify-content: space-around;
    align-content: space-around;
    background-color: #0AD; /*Bleu*/
    width: 100%;
    height: 200px;
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 20px 0px;
}
.ligne{
    flex-direction: row; /*Axe principal = axe horizontal*/
}
.colonne{
    flex-direction: column; /*Axe principal = axe vertical*/
}

.element-flexible{
    flex: 1 1 50px;
    width: 40%;
    height: 40%;
    background-color: #0DA; /*Vert*/
    border: 2px solid green;
    box-sizing: border-box;
    margin: 10px;
}
```


Cours HTML et CSS
File
/Users/Pierre/Desktop/cours.html
1
2
3
1
2
3

Gérer la direction des éléments flexibles (« flex-items »)

Le flexbox est un modèle qui va nous permettre de contrôler avec précision la disposition de nos éléments.

Pour disposer nos éléments flexibles, il va déjà falloir définir un axe de direction des éléments principal et un axe secondaire.

Pour définir l'axe principal, nous allons pouvoir utiliser la propriété `flex-direction` ou la version raccourcie `flex-flow` en les appliquant au conteneur flexible.

L'objectif de cette leçon est de bien comprendre cette notion d'axe et d'apprendre à modifier la direction des éléments flexibles.

L'axe principal et la propriété flex-direction

La propriété `flex-direction` nous permet de définir l'axe principal des éléments flexibles. Nous allons utiliser cette propriété sur notre conteneur flexible.

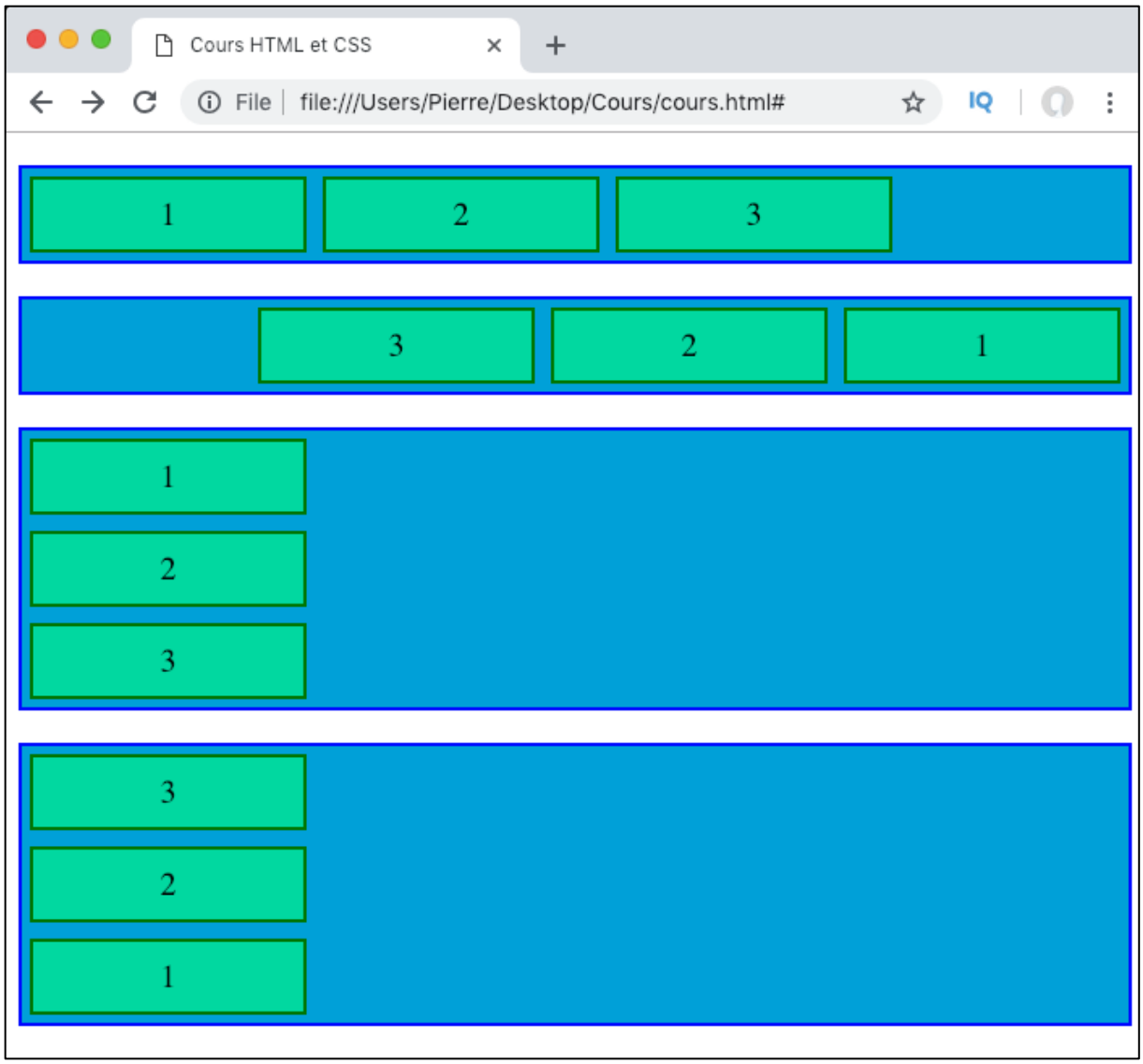
Nous allons pouvoir choisir parmi les valeurs de direction suivantes :

- `row` : Valeur par défaut. L'axe principal est l'axe horizontal et les éléments flexibles vont se placer en ligne, dans le sens de la lecture (de gauche à droite pour un français) ;
- `row-reverse` : L'axe principal est l'axe horizontal et les éléments vont se placer en ligne. Cette fois-ci, les éléments se placent dans le sens inverse de la lecture ;
- `column` : L'axe principal est l'axe vertical et les éléments vont se placer en colonne, en partant du début du conteneur vers la fin (du haut vers le bas par défaut) ;
- `column-reverse` : L'axe principal est l'axe vertical et les éléments vont se placer en colonne, en partant de la fin du conteneur vers le début (du bas vers le haut par défaut).

Prenons immédiatement un premier exemple pour illustrer cela :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible ligne-inverse">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne-inverse">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 20px 0px;
}
.ligne{
    flex-direction: row; /*Axe principal = axe horizontal*/
}
.ligne-inverse{
    flex-direction: row-reverse; /*Axe principal = axe horizontal*/
}
.colonne{
    flex-direction: column; /*Axe principal = axe vertical*/
}
.colonne-inverse{
    flex-direction: column-reverse; /*Axe principal = axe vertical*/
}
.element-flexible{
    flex-basis: 25%;
    width: 25%;
    height: 25%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
```



Nous définissons ici 4 conteneurs flexibles. Les conteneurs flexibles sont à priori identiques et contiennent chacun 3 éléments désormais flexibles par définition.

Nous passons ensuite une valeur de `flex-direction` différente à chaque conteneur flexible pour définir la direction des éléments flexibles et également l'axe principal.

Pour nos deux premiers conteneurs, l'axe principal sera l'axe horizontal. Les éléments flexibles vont ainsi se positionner en ligne, les uns à côté des autres.

Avec la valeur `row`, les éléments suivent le sens de la lecture : les éléments se positionnent de gauche à droite avec le premier élément flexible tout à gauche, puis le deuxième à sa droite et etc.

Avec la valeur `row-reverse`, les éléments se positionnent selon le sens inverse à la lecture : le premier élément sera positionné à droite, le deuxième à sa gauche et etc.

Pour nos deux derniers conteneurs flexibles, l'axe principal sera l'axe vertical. Les éléments flexibles vont ainsi se positionner en colonne, les uns au-dessous des autres.

La valeur `column` va faire se positionner les éléments à partir du début (haut) du conteneur et vers la fin (bas) de celui-ci. Ainsi, le premier élément flexible sera en haut du conteneur, le deuxième sera en dessous du premier et etc.

Avec la valeur `column-reverse`, les éléments flexibles vont continuer de se positionner en colonne mais à partir de la fin du conteneur et vers le début.

Gérer les dépassements des éléments flexibles du conteneur

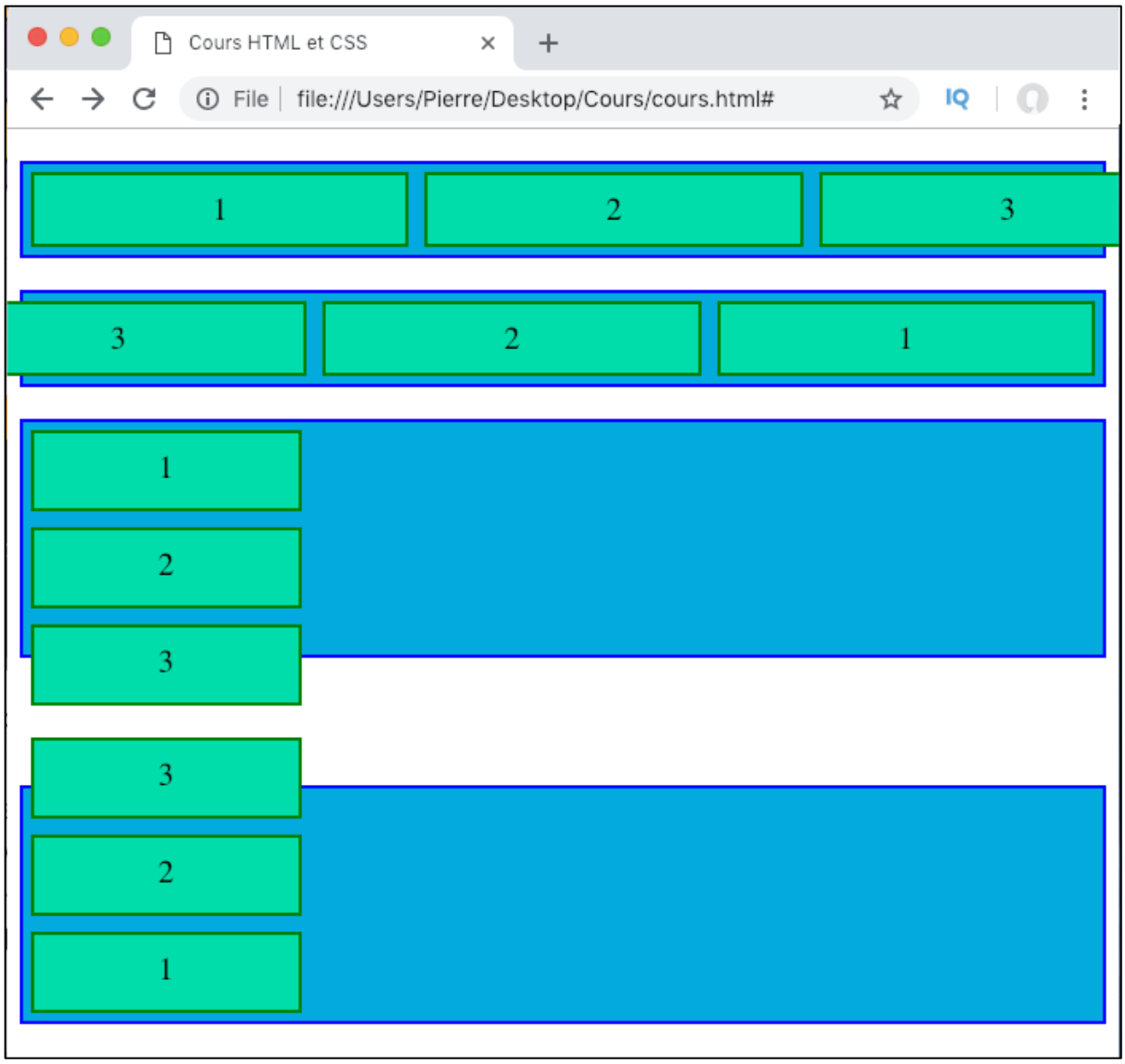
Dans l'exemple précédent, j'ai choisi les dimensions de mes différents éléments de façon à ce que les éléments flexibles ne dépassent pas de leur conteneur.

Cependant, vous devez savoir que si les éléments flexibles occupent plus de place selon leur axe principal que leur conteneur, alors ils vont par défaut dépasser de celui-ci.

Cela est dû au fait que les éléments flexibles n'ont pas le droit par défaut d'aller à la ligne (si l'axe principal est l'axe horizontal) ou de se placer sur une autre colonne (si l'axe principal est l'axe vertical).

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible ligne-inverse">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne-inverse">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 20px 0px;
}
.ligne{
    flex-direction: row; /*Axe principal = axe horizontal*/
}
.ligne-inverse{
    flex-direction: row-reverse; /*Axe principal = axe horizontal*/
}
.colonne{
    flex-direction: column; /*Axe principal = axe vertical*/
    height: 150px;
    margin-bottom: 80px;
}
.colonne-inverse{
    flex-direction: column-reverse; /*Axe principal = axe vertical*/
    height: 150px;
}
.element-flexible{
    flex: 0 0 35%;
    width: 25%;
    height: 25%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
```



Dans l'exemple précédent, j'ai défini une dimension par défaut pour chacun de mes éléments flexibles égale à 35% de la taille du conteneur avec la propriété `flex` que nous étudierons plus tard.

Pour comprendre l'exemple ci-dessus, vous devez savoir que le `flex : 0 0 35%` va définir la largeur des éléments flexibles si l'axe principal est l'axe horizontal ou la hauteur de ces éléments si l'axe principal est l'axe vertical.

J'ai également ici défini une hauteur fixe pour nos deux conteneurs dont l'axe principal est l'axe vertical car sinon les éléments ne pourraient jamais dépasser. En effet, je vous rappelle que par défaut la hauteur d'un élément va dépendre de celle de ses enfants sauf si une hauteur explicite est précisée.

Nous avons donc finalement ici des éléments flexibles qui sont plus grands que leur conteneur (selon leur axe principal) et qui vont donc par défaut dépasser de celui-ci.

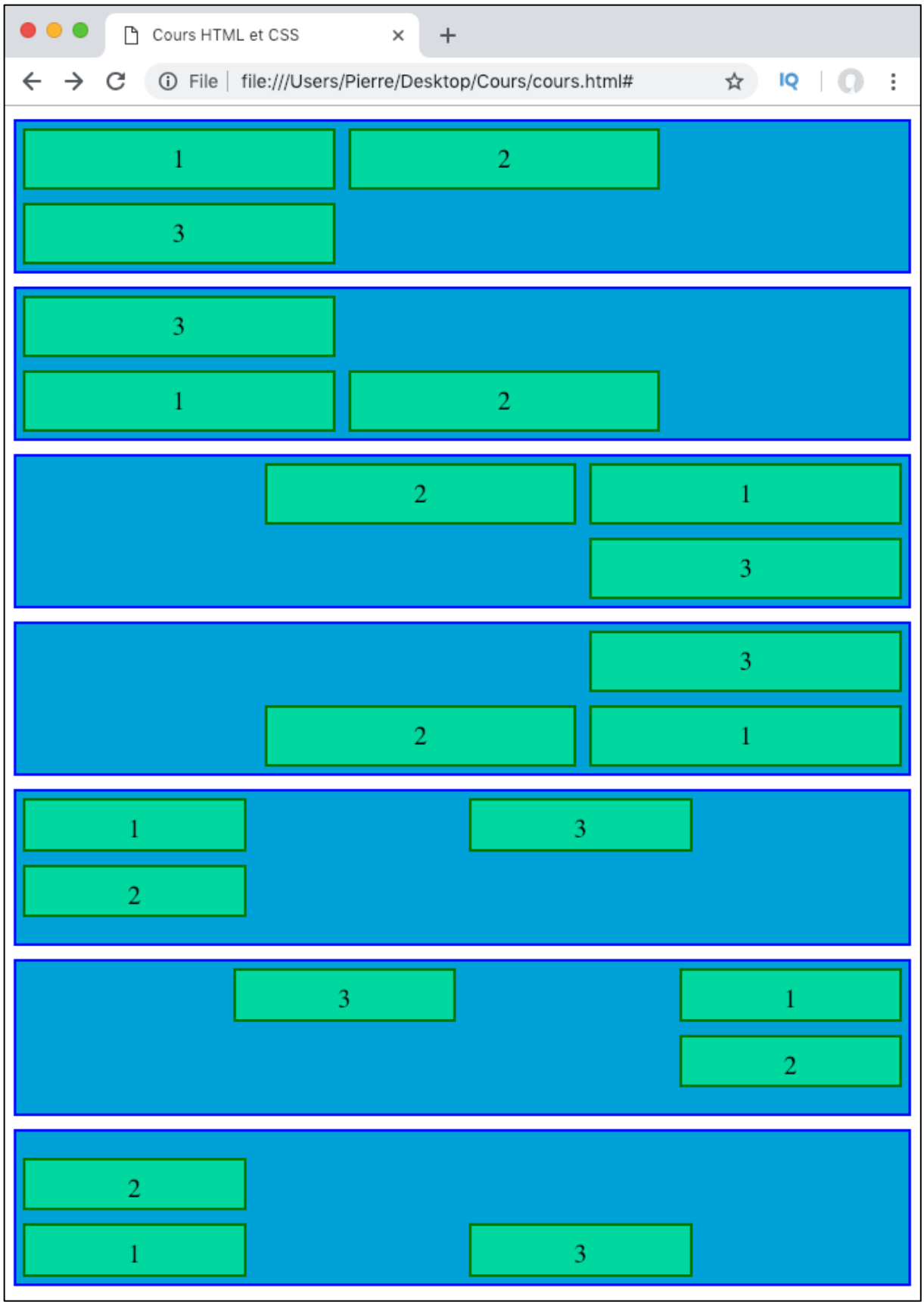
Pour éviter ce comportement souvent non souhaité, nous allons pouvoir utiliser la propriété `flex-wrap` qui va pouvoir donner la possibilité aux éléments flexibles d'aller à la ligne ou en colonne s'ils dépassent de leur conteneur.

Cette propriété va à nouveau s'appliquer au conteneur et accepte les valeurs suivantes :

- `nowrap` : Valeur par défaut. Les éléments vont rester sur la même ligne ou sur la même colonne (selon leur axe principal) ;
- `wrap` : Les éléments qui dépassent du conteneur vont avoir la possibilité d'aller à la ligne ou de se positionner sur une nouvelle colonne à partir du début du conteneur et en allant vers la fin de celui-ci (de haut en bas ou de gauche à droite par défaut);
- `wrap-reverse` : Les éléments qui dépassent du conteneur vont avoir la possibilité d'aller à la ligne ou de se positionner sur une nouvelle colonne à partir cette fois-ci de la fin du conteneur et en allant vers le début (de bas en haut ou de droite à gauche par défaut).

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne wrap">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible ligne wrap-reverse">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible ligne-inverse wrap">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible ligne-inverse wrap-reverse">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne wrap">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne wrap-reverse">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne-inverse wrap">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
}
.ligne{
    flex-direction: row; /*Axe principal = axe horizontal*/
}
.ligne-inverse{
    flex-direction: row-reverse; /*Axe principal = axe horizontal*/
}
.colonne{
    flex-direction: column; /*Axe principal = axe vertical*/
    height: 120px;
}
.colonne-inverse{
    flex-direction: column-reverse; /*Axe principal = axe vertical*/
    height: 120px;
}
.wrap{
    flex-wrap: wrap;
}
.wrap-reverse{
    flex-wrap: wrap-reverse;
}
.element-flexible{
    flex: 0 0 35%;
    width: 25%;
    height: 25%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
```



La direction des éléments de mon premier conteneur est définie avec `flex-direction : row` et on laisse la possibilité aux éléments d'aller à la ligne en partant du début du conteneur

vers la fin avec `flex-wrap : wrap`. L'élément qui dépasse va donc être placé sous les autres, sur la gauche.

La direction des éléments du deuxième conteneur est également définie avec `flex-direction : row` mais cette fois-ci on laisse la possibilité aux éléments d'aller à la ligne en partant de la fin du conteneur vers le début avec `flex-wrap : wrap-reverse`.

Selon l'axe horizontal, le « début » du conteneur est sa première ligne tandis que la « fin » du conteneur est sa dernière ligne. Les éléments vont donc ici se positionner à partir du bas du conteneur et les éléments qui dépassent vont se positionner au-dessus.

La direction des éléments des troisième et quatrième conteneurs est définie avec `flex-direction : row-reverse` : les éléments vont se positionner en ligne à partir de la droite et vers la gauche.

On applique un `flex-wrap : wrap` à notre troisième conteneur, ce qui fait que les éléments vont pouvoir se positionner sur plusieurs lignes en partant du haut du conteneur et en allant vers le bas. Les éléments qui dépassent vont donc être renvoyés sur la ligne du dessous.

On applique un `flex-wrap : wrap-reverse` à notre troisième conteneur, ce qui fait que les éléments vont pouvoir se positionner sur plusieurs lignes en partant du bas du conteneur. Les éléments qui dépassent vont être renvoyés vers le haut du conteneur, sur une ligne supérieure.

Le même principe va s'appliquer pour les éléments des conteneurs flexibles dont la direction est définie avec `column` ou `column-reverse`.

Ici, la valeur `wrap` de la propriété `flex-wrap` va permettre aux éléments qui dépassent de se positionner sur une nouvelle colonne, en partant de la gauche et en allant vers la droite. Les éléments qui dépassent vont donc être renvoyés sur d'autres colonnes à droite des premiers éléments.

La valeur `wrap-reverse` va elle nous permettre de renvoyer les éléments qui dépassent sur une nouvelle colonne à gauche des premiers éléments.

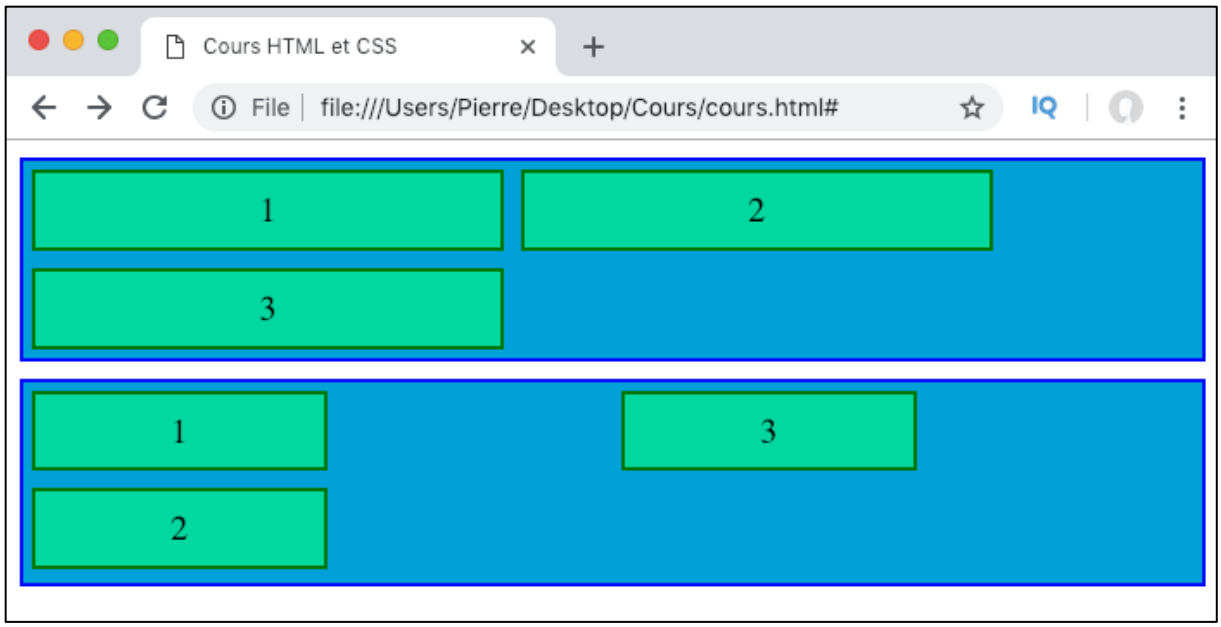
La notation raccourcie flex-flow

Les deux propriétés `flex-direction` et `flex-wrap` peuvent être abrégées grâce à la propriété raccourcie `flex-flow`.

Si une valeur est omise, alors la valeur par défaut de la propriété correspondante sera utilisée à savoir `row` pour `flex-direction` et `nowrap` pour `flex-wrap`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
}
.ligne{
    flex-flow: row wrap;
}
.colonne{
    flex-flow: column wrap;
    height: 120px;
}
.element-flexible{
    flex: 0 0 40%;
    width: 25%;
    height: 25%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
```



L'ordre d'affichage des éléments flexibles dans le conteneur

En plus de définir une direction pour les éléments de notre conteneur flexible, nous allons également pouvoir choisir leur ordre d'affichage élément par élément grâce à la propriété `order`.

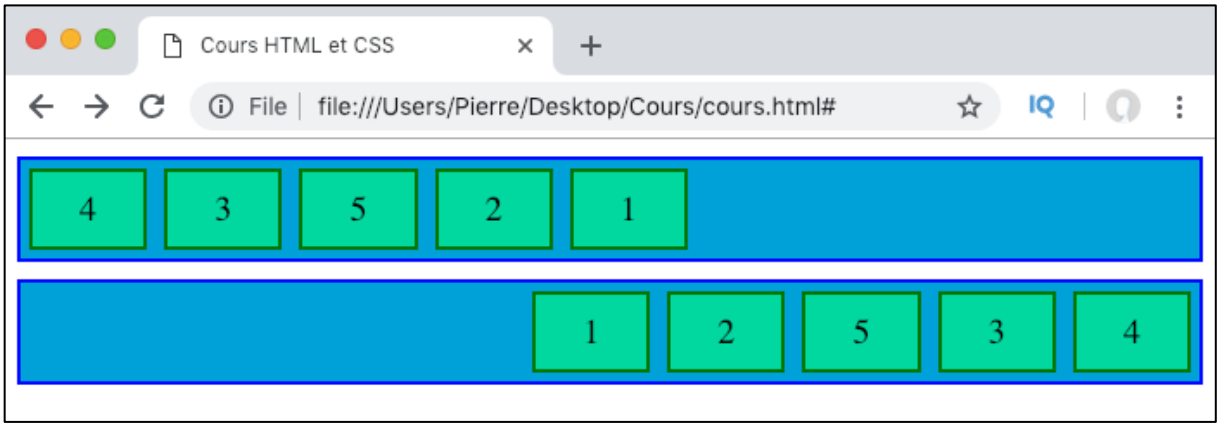
Nous allons appliquer `order` aux éléments flexibles et non au conteneur et allons pouvoir lui passer un nombre.

La valeur de `order` d'un élément va être comparée à celle des autres et l'élément possédant la plus petite valeur sera affiché en premier, celui possédant la deuxième plus petite valeur en deuxième et etc.

La valeur par défaut de `order` est 0 et si plusieurs éléments possèdent la même valeur pour leur propriété `order` ils seront affichés selon l'ordre d'écriture du code.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne">
            <div class="element-flexible o4">1</div>
            <div class="element-flexible o2">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible omoins1">4</div>
            <div class="element-flexible">5</div>
          </div>
          <div class="conteneur-flexible ligne-reverse">
            <div class="element-flexible o4">1</div>
            <div class="element-flexible o2">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible omoins1">4</div>
            <div class="element-flexible">5</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
}
.ligne{
    flex-flow: row wrap;
}
.ligne-reverse{
    flex-flow: row-reverse wrap;
}
.element-flexible{
    flex: 0 0 10%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
.omoins1{
    order: -1;
}
.o2{
    order: 2;
}
.o4{
    order: 4;
}
```



Ici, nous avons deux conteneurs flexibles identiques possédant chacun 5 éléments. La seule différence entre nos deux conteneurs va être la direction de leur axe principal : l'axe principal de notre premier conteneur est horizontal selon une direction qui suit le sens de l'écriture (gauche à droite pour moi) tandis que l'axe principal de notre deuxième conteneur est horizontal selon une direction contraire au sens de l'écriture.

A chaque fois, on attribue un `order : 4` au premier élément des conteneurs, un `order : 2` au deuxième élément et un `order : -1` au quatrième élément.

On ne définit pas de valeur pour la propriété `order` pour nos troisième et cinquième éléments. La valeur par défaut, `0`, sera donc utilisée.

L'élément qui possède la valeur la plus petite pour sa propriété `order` est donc le quatrième élément qui sera affiché en premier puis vont suivre dans l'ordre d'écriture dans le code les troisième et cinquième élément puis le deuxième et enfin le premier.

Notez bien ici que l'ordre des éléments va dépendre de la direction de l'axe du conteneur flexible : pour notre deuxième conteneur, le début du conteneur se situe à droite et la fin à gauche. Le premier élément va donc s'afficher à droite, le deuxième à sa gauche et ainsi de suite vers la gauche.

Gérer l'alignement des éléments flexibles

Dans la leçon précédente, nous avons vu comment définir l'axe principal d'un conteneur flexible et avons appris à gérer la direction des éléments dans ce conteneur.

Le modèle des boites flexibles va nous permettre d'aller plus loin et de définir l'alignement des éléments à l'intérieur du conteneur selon l'axe principal ou l'axe secondaire.

Dans cette nouvelle leçon, nous allons apprendre à aligner les éléments d'un conteneur flexible par rapport à leur axe principal ou secondaire en nous servant des propriétés suivantes :

- `justify-content` ;
- `align-items` ;
- `align-content` ;
- `align-self`.

Attention : cette partie n'est pas simple. Il faudra bien comprendre les concepts d'axe principal et d'axe secondaire et bien se représenter mentalement l'espace que prend une ligne ou une colonne et l'espace que prennent les éléments dans chaque ligne ou colonne. J'ai essayé d'illustrer au maximum pour que vous puissiez bien vous représenter tout cela. N'hésitez pas à faire des pauses, à lire la leçon plusieurs fois et à expérimenter !

Gérer l'alignement des éléments selon l'axe principal avec justify-content

La propriété `justify-content` nous permet d'aligner les éléments d'un conteneur flexibles selon l'axe principal.

Plus précisément, cette propriété va nous permettre de définir de quelle manière doit être distribué l'espace restant dans le conteneur. On va l'appliquer au conteneur et pouvoir lui passer l'une des valeurs suivantes :

- `flex-start` : Valeur par défaut. Les éléments vont être concentrés au début du conteneur (selon leur axe principal) ;
- `flex-end` : Les éléments vont être concentrés à la fin du conteneur (selon leur axe principal) ;
- `center` : Les éléments vont être centrés dans le conteneur (selon leur axe principal) ;
- `space-between` : Les éléments vont être régulièrement distribués dans le conteneur. Les éléments se trouvant contre un bord du conteneur vont être collés au bord ;
- `space-around` : Les éléments vont être régulièrement distribués dans le conteneur. Chaque élément va posséder le même espace et les espaces vont être cumulatifs entre deux éléments, ce qui fait que la taille de l'espace entre le conteneur et un élément contre le bord du conteneur sera deux fois plus petite qu'entre deux éléments ;

- space-evenly Les éléments vont être régulièrement distribués dans le conteneur. L'espace entre le bord du conteneur et un élément sera le même que celui entre deux éléments.

Prenons immédiatement un exemple pour illustrer le fonctionnement de cette propriété :

```
<body>
    <div class="conteneur-flexible ligne debut">
        <div class="element-flexible">1</div>
        <div class="element-flexible">2</div>
        <div class="element-flexible">3</div>
        <div class="element-flexible">4</div>
        <div class="element-flexible">5</div>
    </div>
    <div class="conteneur-flexible ligne fin">
        <div class="element-flexible">1</div>
        <div class="element-flexible">2</div>
        <div class="element-flexible">3</div>
        <div class="element-flexible">4</div>
        <div class="element-flexible">5</div>
    </div>
    <div class="conteneur-flexible ligne milieu">
        <div class="element-flexible">1</div>
        <div class="element-flexible">2</div>
        <div class="element-flexible">3</div>
        <div class="element-flexible">4</div>
        <div class="element-flexible">5</div>
    </div>
    <div class="conteneur-flexible ligne space-between">
        <div class="element-flexible">1</div>
        <div class="element-flexible">2</div>
        <div class="element-flexible">3</div>
        <div class="element-flexible">4</div>
        <div class="element-flexible">5</div>
    </div>
    <div class="conteneur-flexible ligne space-around">
        <div class="element-flexible">1</div>
        <div class="element-flexible">2</div>
        <div class="element-flexible">3</div>
        <div class="element-flexible">4</div>
        <div class="element-flexible">5</div>
    </div>
    <div class="conteneur-flexible ligne space-evenly">
        <div class="element-flexible">1</div>
        <div class="element-flexible">2</div>
        <div class="element-flexible">3</div>
        <div class="element-flexible">4</div>
        <div class="element-flexible">5</div>
    </div>
</body>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
}
.ligne{
    flex-flow: row wrap;
}
.debut{
    justify-content: flex-start;
}
.fin{
    justify-content: flex-end;
}
.milieu{
    justify-content: center;
}
.space-between{
    justify-content: space-between;
}
.space-around{
    justify-content: space-around;
}
.space-evenly{
    justify-content: space-evenly;
}
.element-flexible{
    flex: 0 0 10%;
    width: 25%;
    height: 25%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
```

Dans cette première série d'exemples, notre axe principal est l'axe horizontal et nos éléments flexibles n'occupent qu'une seule ligne dans leur conteneur. La direction d'affichage des éléments est celle du sens de la lecture (de gauche à droite).

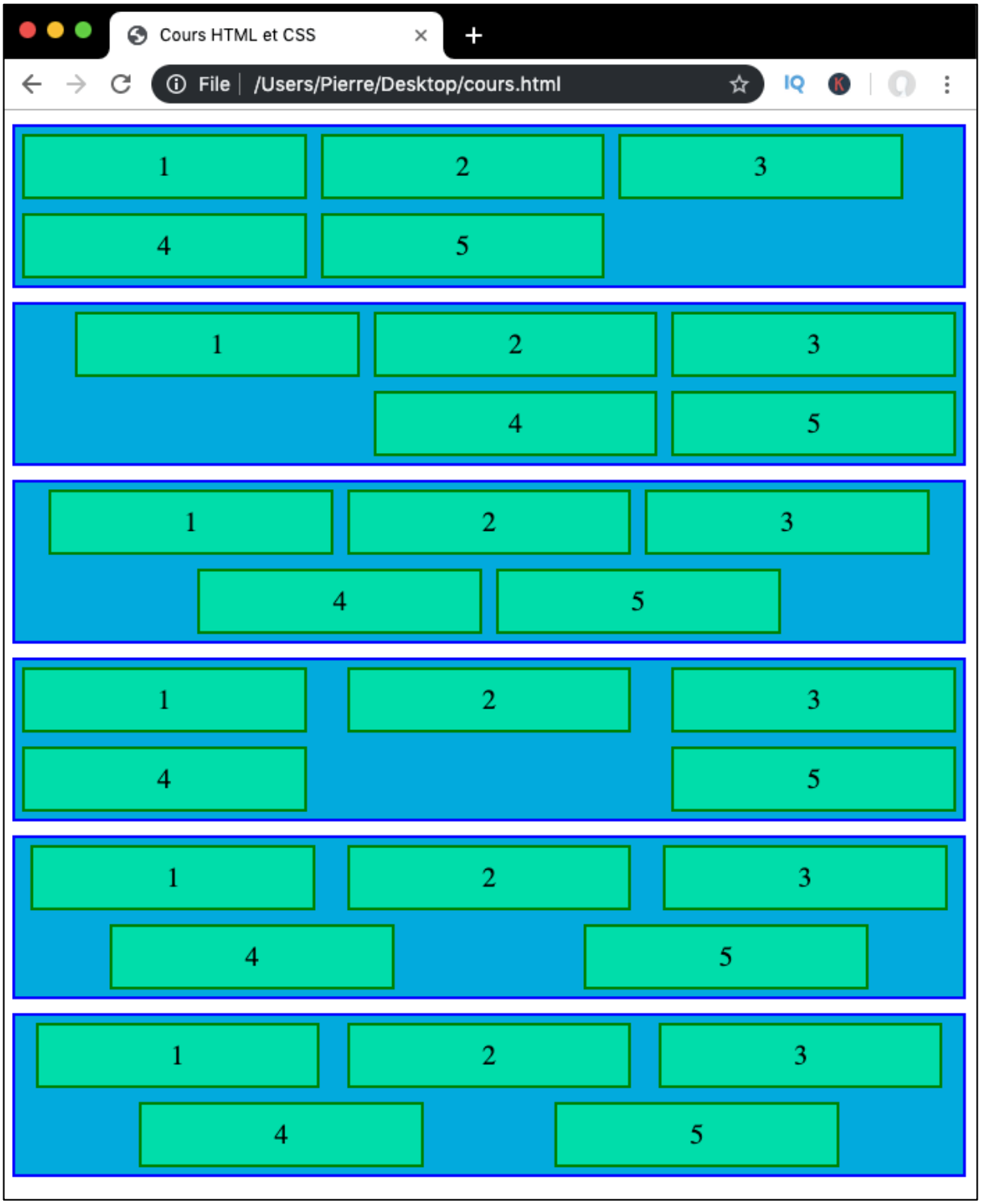
Ici, la valeur `flex-start` va aligner les éléments au début du conteneur c'est-à-dire dans le cas présent à gauche de celui-ci tandis que `flex-end` va les aligner sur la droite et que `flex-center` va nous permettre de centrer horizontalement les éléments dans le conteneur.

Comme vous pouvez le voir, l'espace vide est régulièrement distribué dans le conteneur avec les valeurs `space-between`, `space-around` et `space-evenly`. La seule différence entre ces valeurs va être l'espace qui va être attribué entre les bords du conteneur et les éléments collés contre les bords.

La propriété `justify-content` va se comporter exactement de la même façon dans le cas où les éléments occupent plusieurs lignes dans le conteneur :

```
<body>
    <div class="conteneur-flexible ligne debut">
      <div class="element-flexible">1</div>
      <div class="element-flexible">2</div>
      <div class="element-flexible">3</div>
      <div class="element-flexible">4</div>
      <div class="element-flexible">5</div>
    </div>
    <div class="conteneur-flexible ligne fin">
      <div class="element-flexible">1</div>
      <div class="element-flexible">2</div>
      <div class="element-flexible">3</div>
      <div class="element-flexible">4</div>
      <div class="element-flexible">5</div>
    </div>
    <div class="conteneur-flexible ligne centre">
      <div class="element-flexible">1</div>
      <div class="element-flexible">2</div>
      <div class="element-flexible">3</div>
      <div class="element-flexible">4</div>
      <div class="element-flexible">5</div>
    </div>
    <div class="conteneur-flexible ligne space-between">
      <div class="element-flexible">1</div>
      <div class="element-flexible">2</div>
      <div class="element-flexible">3</div>
      <div class="element-flexible">4</div>
      <div class="element-flexible">5</div>
    </div>
    <div class="conteneur-flexible ligne space-around">
      <div class="element-flexible">1</div>
      <div class="element-flexible">2</div>
      <div class="element-flexible">3</div>
      <div class="element-flexible">4</div>
      <div class="element-flexible">5</div>
    </div>
    <div class="conteneur-flexible ligne space-evenly">
      <div class="element-flexible">1</div>
      <div class="element-flexible">2</div>
      <div class="element-flexible">3</div>
      <div class="element-flexible">4</div>
      <div class="element-flexible">5</div>
    </div>
</body>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
}
.ligne{flex-flow: row wrap;}

.debut{justify-content: flex-start;}
.fin{justify-content: flex-end;}
.centre{justify-content: center;}
.space-between{justify-content: space-between;}
.space-around{justify-content: space-around;}
.space-evenly{justify-content: space-evenly;}

.element-flexible{
    flex: 0 0 30%;
    width: 25%;
    height: 25%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
```

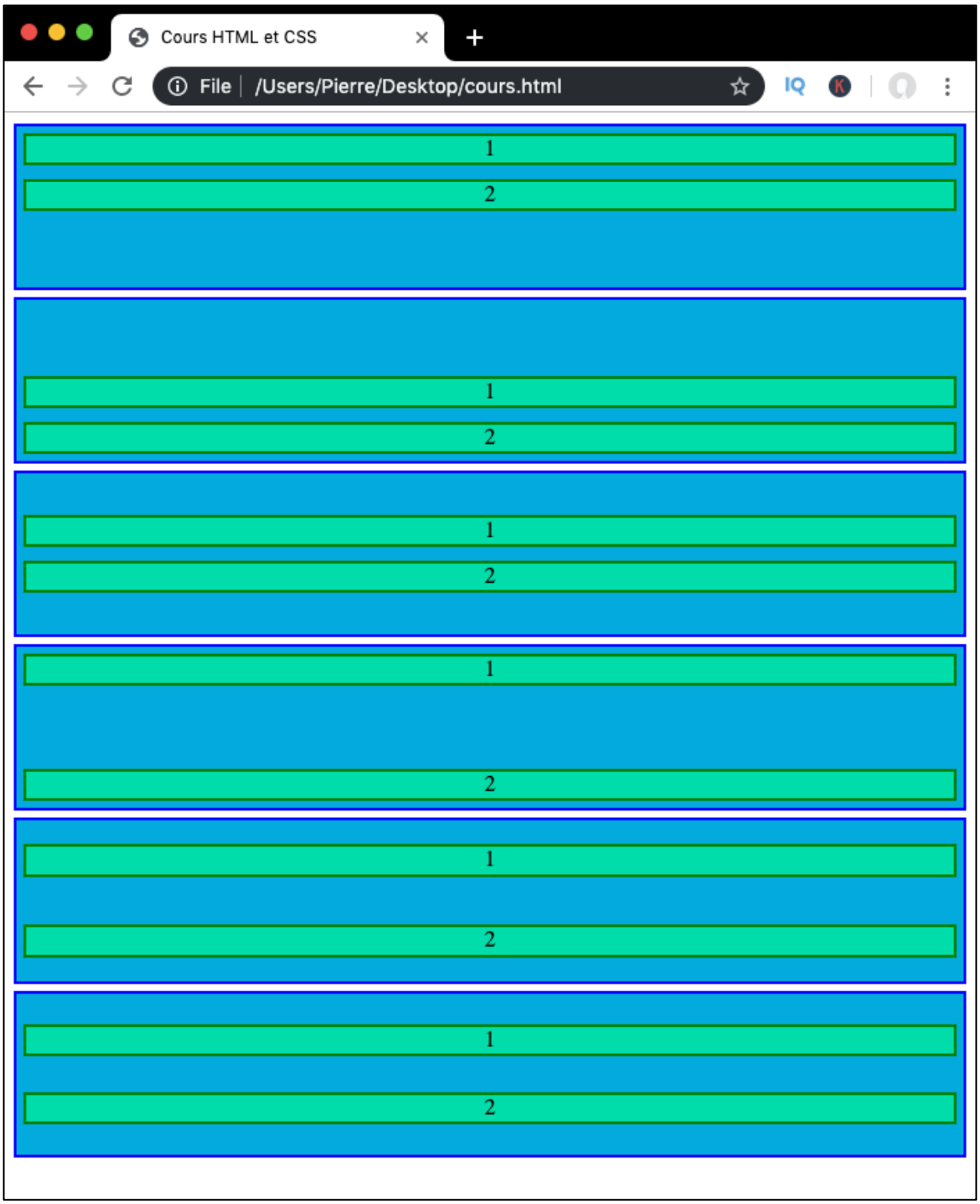


Regardons de plus près les valeurs `space-between`, `space-around` et `space-evenly` : vous pouvez remarquer que l'espace entre le premier élément et le dernier élément de chaque ligne et le bord du conteneur va être le même pour chaque ligne d'éléments. Ce comportement est tout à fait normal et voulu.

Enfin, la propriété `justify-content` va à nouveau avoir le même effet si l'axe principal est l'axe vertical.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur-flexible colonne debut">
          <div class="element-flexible">1</div>
          <div class="element-flexible">2</div>
        </div>
        <div class="conteneur-flexible colonne fin">
          <div class="element-flexible">1</div>
          <div class="element-flexible">2</div>
        </div>
        <div class="conteneur-flexible colonne centre">
          <div class="element-flexible">1</div>
          <div class="element-flexible">2</div>
        </div>
        <div class="conteneur-flexible colonne space-between">
          <div class="element-flexible">1</div>
          <div class="element-flexible">2</div>
        </div>
        <div class="conteneur-flexible colonne space-around">
          <div class="element-flexible">1</div>
          <div class="element-flexible">2</div>
        </div>
        <div class="conteneur-flexible colonne space-evenly">
          <div class="element-flexible">1</div>
          <div class="element-flexible">2</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 16px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 5px 0px;
}
.colonne{
    flex-flow: column wrap;
    height: 120px;}

.debut{justify-content: flex-start;}
.fin{justify-content: flex-end;}
.centre{justify-content: center;}
.space-between{justify-content: space-between;}
.space-around{justify-content: space-around;}
.space-evenly{justify-content: space-evenly;}

.element-flexible{
    flex: 0 0 20%;
    height: 20%;
    background-color: #0DA; /*Vert*/
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
```



Cette fois-ci, la valeur `flex-start` va nous permettre d'aligner les éléments en haut du conteneur tandis que `flex-end` va nous permettre de les aligner en bas de celui-ci et que `center` va les centrer verticalement dans le conteneur.

De la même manière c'est l'espace vide dans le sens de la hauteur qui sera distribué régulièrement avec les valeurs `space-between`, `space-around` et `space-evenly`.

Ici, faites bien attention à ne pas confondre les effets des propriétés `justify-content` et `flex-direction` : la propriété `justify-content` va gérer la distribution de l'espace vide dans le

conteneur tandis que flex-direction va permettre d'indiquer un axe et ou une direction pour nos éléments flexibles et va donc justement permettre de définir où est le « début » et la « fin » du conteneur.

Ce que vous devez absolument comprendre ici est que dans le cas où la valeur de flex-direction est row, l'axe principal sera horizontal et le début du conteneur sera son bord gauche.

En revanche, dans le cas où flex-direction vaut row-reverse, alors le début du conteneur sera son bord droit.

La propriété justify-content va ensuite aligner les éléments du début du conteneur. Ainsi, lorsqu'on applique un justify-content : flex-start sur un conteneur dont la direction est flex-direction : row-reverse, l'effet va être de concentrer les éléments sur la droite du conteneur puisque le début du conteneur correspond à son bord droit.

Note : Tous ces concepts de direction sont dépendants de la direction de l'écriture de votre document. Si votre document est défini dans une langue dont l'écriture se fait de droite à gauche alors le début d'un conteneur flexible dont la direction a été définie avec flex-direction : row sera son bord droit.

Gérer l'alignement des éléments selon l'axe secondaire

La gestion de l'alignement des éléments flexibles selon l'axe secondaire va pouvoir se faire de différentes façons et avec plusieurs propriétés :

- La propriété align-items va nous permettre de gérer l'alignement des éléments au sein d'une ligne (ou d'une colonne selon les axes principal et secondaires choisis) dans l'axe secondaire ;
- La propriété align-content va nous permettre de gérer l'alignement des lignes (ou des colonnes selon l'axe) les unes par rapport aux autres dans l'axe secondaire et ne va donc avoir un effet que si nous avons plusieurs lignes (ou colonnes) d'éléments flexibles ;
- La propriété align-self va nous permettre de gérer l'alignement d'un élément flexible en particulier dans l'axe secondaire.

A partir d'ici, nous allons avoir besoin d'avoir des éléments flexibles de différentes tailles pour bien voir l'effet des différentes propriétés. Pour le moment, je vais me contenter de modifier la hauteur des éléments en attribuant à certains éléments un padding haut et bas en sachant que dans les exemples suivants l'axe principal sera l'axe horizontal sauf mention contraire.

Gérer l'alignement des éléments dans une ligne ou dans une colonne selon l'axe secondaire avec align-items

La propriété align-items va nous permettre de définir l'alignement des éléments flexibles au sein d'une ligne (ou d'une colonne) selon l'axe secondaire.

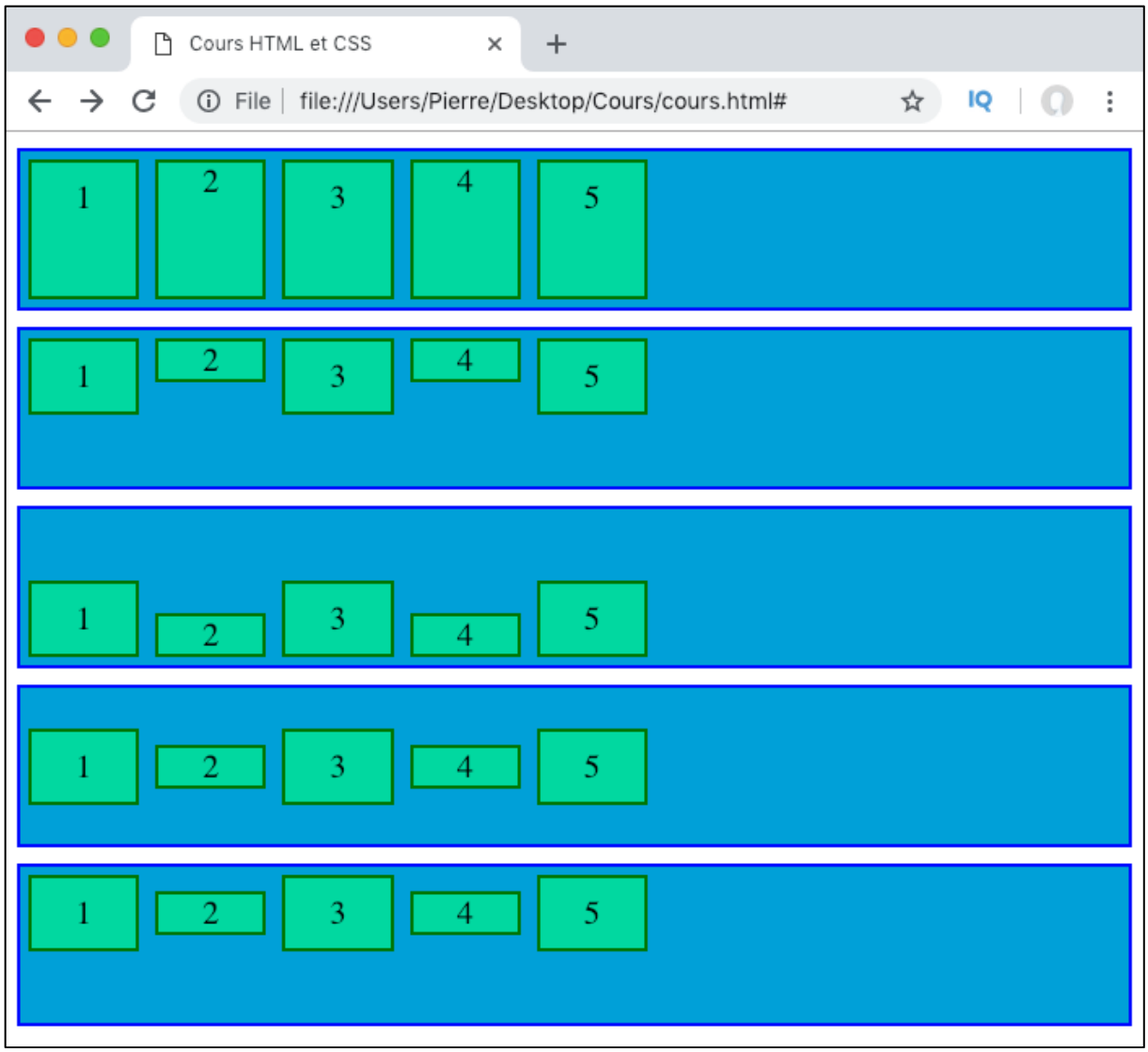
Cette propriété est l'équivalent de la propriété justify-content mais va permettre l'alignement des éléments selon leur axe secondaire et non pas principal cette fois-ci.

Nous allons une nouvelle fois devoir définir `align-items` pour le conteneur flexible et allons pouvoir lui passer les valeurs suivantes :

- `stretch` : Valeur par défaut. Les éléments vont s'étirer dans leur axe secondaire jusqu'à remplir tout l'espace disponible ;
- `flex-start` : Les éléments vont être placés au début de leur conteneur en fonction de l'axe secondaire ;
- `flex-end` : Les éléments vont être placés à la fin de leur conteneur en fonction de l'axe secondaire ;
- `center` : Les éléments vont être placés au milieu de leur conteneur en fonction de l'axe secondaire ;
- `baseline` : Les éléments vont être alignés dans leur axe secondaire de telle sorte à ce que leurs lignes de base (ligne imaginaire sur laquelle est écrit le texte) soient alignées.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne stretch-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
          </div>
          <div class="conteneur-flexible ligne debut-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
          </div>
          <div class="conteneur-flexible ligne fin-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
          </div>
          <div class="conteneur-flexible ligne centre-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
          </div>
          <div class="conteneur-flexible ligne baseline-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
    height: 100px;
}
/*L'axe principal est horizontal, le début du conteneur est à gauche
 *L'axe secondaire est vertical, le début du conteneur pour cet axe est en haut*/
.ligne{
    flex-flow: row wrap;
}
.stretch-sec{
    align-items: stretch;
}
.debut-sec{
    align-items: flex-start;
}
.fin-sec{
    align-items: flex-end;
}
.centre-sec{
    align-items: center;
}
.baseline-sec{
    align-items: baseline;
}
.element-flexible{
    flex: 0 0 10%;
    width: 25%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
.petit{
  padding: 0px;
}
```



Dans les exemples ci-dessus, l'axe principal de mon conteneur flexible est l'axe horizontal et l'axe secondaire est donc l'axe vertical.

La propriété `align-items` va donc impacter l'alignement des éléments selon l'axe vertical qui est ici l'axe secondaire.

Par défaut, un conteneur va posséder la même hauteur que le plus grand de ses éléments. Ici, la hauteur de mes éléments dépend de leur contenu et des propriétés `padding` et `border`.

Afin que l'effet des propriétés liées à l'alignement que l'on va étudier soit bien visible, je donne une hauteur fixe plus grande que celle de mes éléments à mon conteneur pour créer de l'espace vide.

Comme vous pouvez le constater, la valeur `stretch` va ici étirer les éléments de façon à ce qu'ils occupent tout l'espace disponible dans leur conteneur selon leur axe secondaire qui est ici l'axe vertical.

La valeur flex-start va ici aligner les éléments en haut du conteneur ce qui correspond au début du conteneur dans l'axe vertical tandis que la valeur flex-end va avoir l'effet contraire.

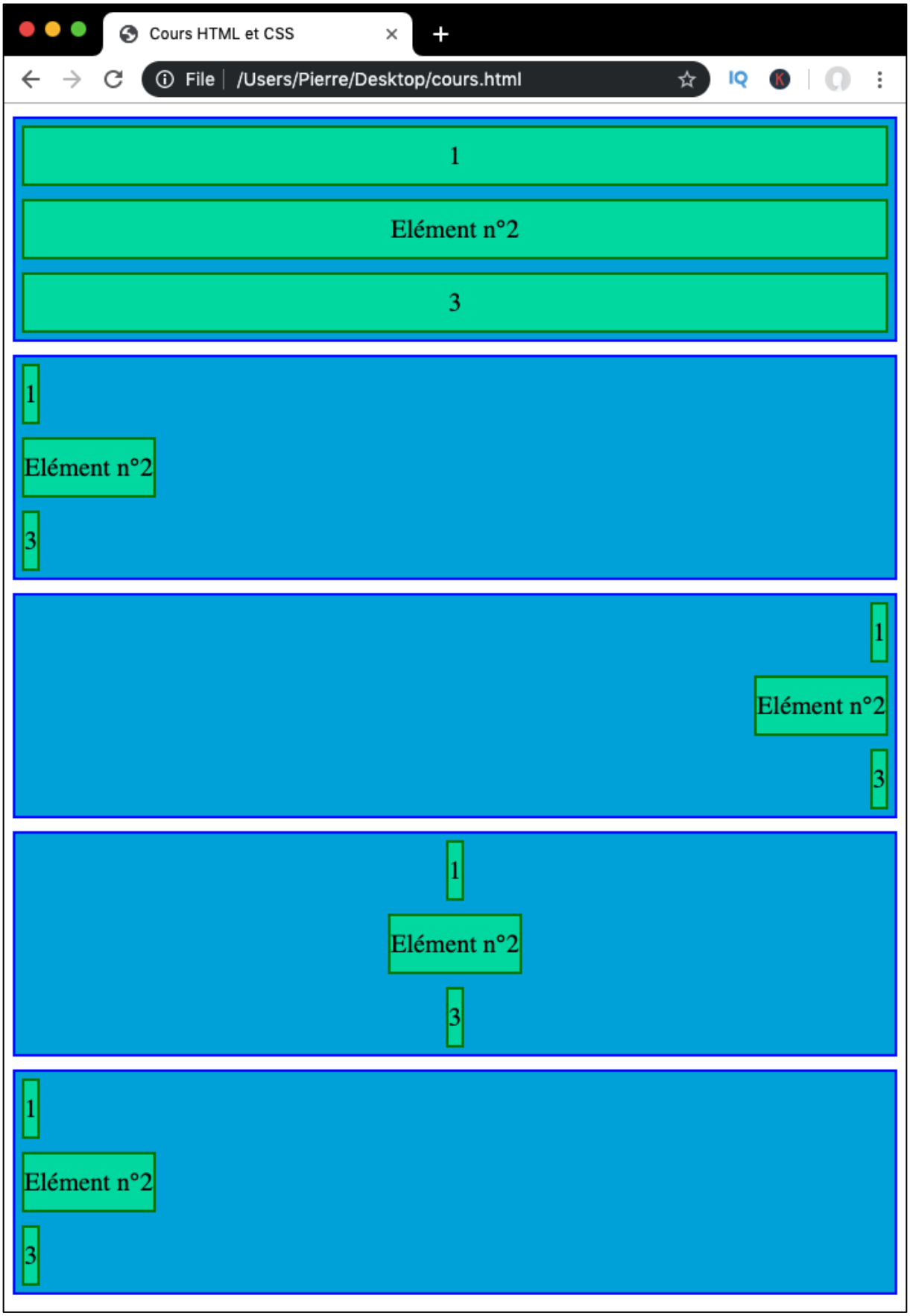
La valeur center nous permet ici très simplement de centrer verticalement les éléments dans leur conteneur.

Finalement, la valeur baseline permet d'aligner les éléments selon leur ligne de base qui est la ligne sur laquelle sont écrits les textes.

De manière similaire, si l'axe vertical est défini comme axe principal pour un conteneur flexible, alors la propriété align-items alignera les éléments selon leur axe horizontal.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur-flexible colonne stretch-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible">Elément n°2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne debut-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible">Elément n°2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne fin-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible">Elément n°2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne centre-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible">Elément n°2</div>
            <div class="element-flexible">3</div>
        </div>
        <div class="conteneur-flexible colonne baseline-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible">Elément n°2</div>
            <div class="element-flexible">3</div>
        </div>
    </body>
</html>
```

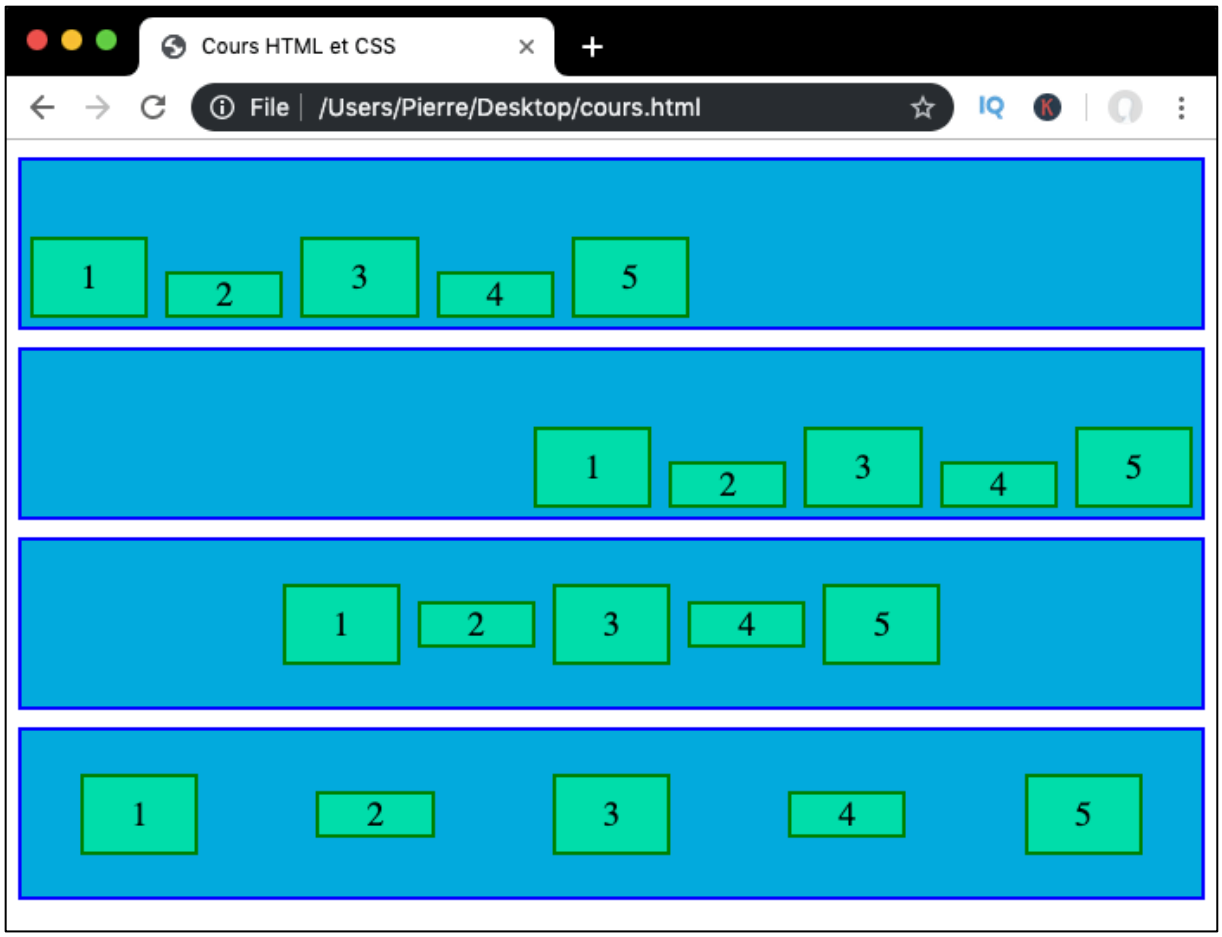
```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
}
/*Axe principal = vertical; Axe secondaire = horizontal*/
.colonne{flex-flow: column wrap;}

.stretch-sec{align-items: stretch;}
.debut-sec{align-items: flex-start;}
.fin-sec{align-items: flex-end;}
.centre-sec{align-items: center;}
.baseline-sec{align-items: baseline;}

.element-flexible{
    flex: 0 0 10%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
```


Cours HTML et CSS
File /Users/Pierre/Desktop/cours.html
1
Elément n°2
3
1
Elément n°2
3
1
Elément n°2
3
1
Elément n°2
3
1
Elément n°2
3

Bien évidemment, nous allons pouvoir utiliser les propriétés justify-content et align-items ensemble pour aligner les éléments à la fois selon leur axe principal et selon leur axe secondaire.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur-flexible ligne début-pri fin-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
        </div>
        <div class="conteneur-flexible ligne fin-pri fin-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
        </div>
        <div class="conteneur-flexible ligne centre-pri centre-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
        </div>
        <div class="conteneur-flexible ligne sparound-pri centre-sec">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
    height: 100px;
}
/*L'axe principal est horizontal, le début du conteneur est à gauche
 *L'axe secondaire est vertical, le début du conteneur pour cet axe est en haut*/
.ligne{flex-flow: row wrap;}

.debut-pri{justify-content: flex-start;}
.fin-pri{justify-content: flex-end;}
.centre-pri{justify-content: center;}
.sparound-pri{justify-content: space-around;}
.debut-sec{align-items: flex-start;}
.fin-sec{align-items: flex-end;}
.centre-sec{align-items: center;}

.element-flexible{
    flex: 0 0 10%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}

.petit{padding: 0px;}
```



Ici, les éléments de notre premier conteneur se placent au début de celui-ci selon l'axe principal qui est l'axe horizontal c'est-à-dire à gauche et à la fin selon l'axe secondaire (axe vertical) c'est-à-dire en bas.

Les éléments du deuxième conteneur s'alignent à la fin de celui-ci à la fois selon l'axe principal et selon l'axe secondaire c'est-à-dire à droite et en bas.

Les éléments de notre troisième conteneur sont centrés à la fois dans l'axe principal et dans l'axe secondaire, ils vont ainsi être centrés à la fois horizontalement et verticalement dans le conteneur. On va ainsi très simplement pouvoir obtenir un centrage parfait en utilisant le flexbox.

Les éléments de notre quatrième conteneur ont un alignement de type `space-around` (distribution régulière) selon leur axe principal (horizontal) et sont centrés selon leur axe secondaire c'est-à-dire centrés verticalement.

Les propriétés flex-wrap et align-items

Il convient ici de ne pas confondre les effets des propriétés `flew-wrap` et `align-items`. La propriété `flex-wrap` va définir la possibilité pour les éléments qui dépassent du conteneur de se placer sur une nouvelle ligne ou une nouvelle colonne.

Lorsqu'on autorise les éléments à se placer sur une nouvelle ligne ou une nouvelle colonne, c'est-à-dire lorsqu'on passe les valeurs `wrap` ou `wrap-reverse` à la propriété `flex-wrap`, alors cette propriété va également définir la direction de l'axe secondaire.

Par défaut, la direction liée à la valeur wrap va être de gauche à droite si l'axe secondaire est l'axe horizontal et de haut en bas si l'axe secondaire est l'axe vertical.

Au contraire, si la valeur de flex-wrap est wrap-reverse, alors la direction de l'axe secondaire sera de droite à gauche si c'est l'axe horizontal et de bas en haut si l'axe secondaire est l'axe vertical.

La propriété align-items va elle définir l'alignement des éléments dans chaque ligne (ou colonne) selon leur axe secondaire. Le résultat lié aux valeurs de align-items va être intimement lié à la direction définie pour l'axe secondaire.

Un flex-flow : row wrap-reverse par exemple va définir que l'axe principal est l'axe horizontal et que sa direction pour nous va être de gauche à droite et que la direction de l'axe vertical sera inversée c'est-à-dire partir du bas (début) vers le haut (fin) pour nous.

Ainsi, si on applique également un align-items : flex-end au conteneur flexible, les éléments se placeront à la fin de leur ligne dans le conteneur selon l'axe secondaire c'est-à-dire… en haut de leur ligne !

Gérer l'alignement des lignes ou des colonnes les unes par rapport aux autres selon l'axe secondaire avec align-content

La propriété align-content ne va pas nous permettre de gérer l'alignement des éléments au sein d'une ligne ou d'une colonne mais plutôt de gérer l'alignement des différentes lignes (ou colonnes) les unes par rapport aux autres.

Cette propriété n'aura d'effet que si le conteneur flexible possède plusieurs lignes (si l'axe principal est horizontal) ou plusieurs colonnes (si l'axe principal est vertical).

On va à nouveau définir cette propriété au niveau de notre conteneur et allons pouvoir lui passer une des valeurs suivantes :

- stretch : Valeur par défaut. Les lignes (ou les colonnes) vont essayer de s'étendre pour prendre tout l'espace disponible dans le conteneur ;
- flex-start : Les lignes (ou les colonnes) vont être concentrées au début du conteneur ;
- flex-end : Les lignes (ou les colonnes) vont être concentrées à la fin du conteneur;
- center : Les lignes (ou les colonnes) vont être concentrées au milieu du conteneur;
- space-between : Les lignes (ou les colonnes) vont être régulièrement distribuées dans le conteneur. La première ligne (ou colonne) va être collée contre le début du conteneur et la dernière sera collée contre la fin du conteneur ;
- space-around : Les lignes (ou les colonnes) vont être régulièrement distribuées dans le conteneur. L'espace autour de chaque ligne (colonne) va être le même et cumulatif entre deux lignes (colonnes).

Il convient de ne pas confondre align-items et align-content. En effet, ces deux propriétés agissent sur des choses différentes.

Imaginez un conteneur flexible défini avec flex-flow : row wrap et contenant deux lignes d'éléments. En appliquant un align-items : flex-start, les éléments du conteneur vont être

placé au début de celui-ci selon leur axe secondaire c'est-à-dire en haut de celui-ci ou plus exactement les différents éléments vont être placés au début de leur ligne.

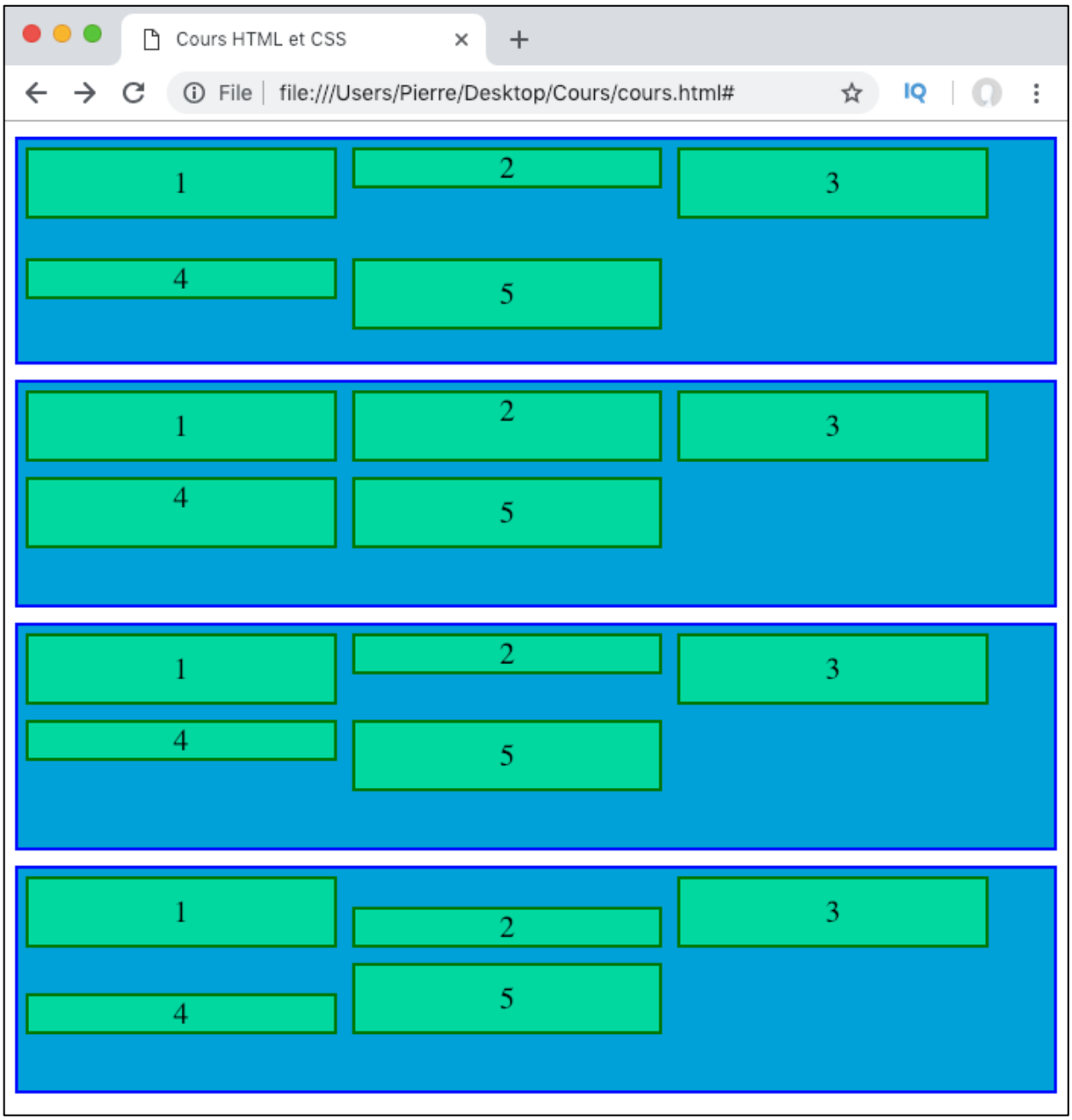
En effet, chaque ligne (ou chaque colonne) d'un conteneur flexible agit indépendamment des autres et peut être considérée comme un conteneur à part. Les conteneurs flexibles ont été créés comme cela justement pour nous donner un contrôle sur les éléments dans les lignes (colonnes) et sur les lignes (colonnes) en soi.

La propriété `align-content` va-elle donc nous permettre ensuite d'aligner les lignes ou colonnes du conteneur.

Regardez l'exemple ci-dessous pour bien comprendre :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne items-start">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
          </div>
          <div class="conteneur-flexible ligne content-start">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
          </div>
          <div class="conteneur-flexible ligne items-start content-start">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
          </div>
          <div class="conteneur-flexible ligne items-end content-start">
            <div class="element-flexible">1</div>
            <div class="element-flexible petit">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible petit">4</div>
            <div class="element-flexible">5</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
    height: 150px;
}
.ligne{
    flex-flow: row wrap;
}
.items-start{
    align-items: flex-start;
}
.items-end{
    align-items: flex-end;
}
.content-start{
    align-content: flex-start;
}
.element-flexible{
    flex: 0 0 30%;
    width: 30%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
.petit{
    padding: 0px;
}
```



L'axe secondaire pour chacun de nos conteneurs flexibles est l'axe vertical. On définit un `align-items : flex-start` pour notre premier conteneur mais pas de `align-content`. La valeur par défaut `stretch` sera donc utilisée. Les lignes vont donc prendre chacune le maximum d'espace dans le conteneur c'est-à-dire 50% chacune et les éléments vont se placer au début de chaque ligne grâce à `align-items : flex-start`.

Ici, vous devez bien comprendre que le fait qu'une ligne s'étende au maximum dans le conteneur ne signifie pas que les éléments qu'elle contient vont également s'étendre.

Pour notre deuxième conteneur, on demande cette fois-ci aux lignes de se positionner au début du conteneur. Les lignes vont donc se concentrer en haut du conteneur et avoir par défaut une hauteur égale à celle du plus grand des éléments de la ligne.

On ne définit cette fois-ci pas de valeur pour `align-items`. La valeur par défaut, `stretch` va donc être utilisée et les éléments vont s'étendre un maximum dans la ligne.

On définit pour notre troisième conteneur un `align-items : flex-start` et un `align-content : flex-start`. Les lignes vont donc se concentrer au début du conteneur et les éléments vont se concentrer au début de la ligne.

Finalement, on définit un `align-items : flex-end` et un `align-content : flex-start` pour notre quatrième conteneur. Les lignes vont don à nouveau se concentrer en début de conteneur mais les éléments vont cette fois-ci se concentrer à la fin de leur ligne.

Gérer l'alignement d'un élément en particulier dans son axe secondaire avec align-self

La propriété `align-self` va finalement nous permettre de gérer l'alignement d'un élément en particulier dans une ligne (ou une colonne) selon l'axe secondaire.

Nous allons cette fois-ci non pas appliquer la propriété au conteneur mais bien aux éléments flexibles et allons pouvoir lui passer les mêmes valeurs qu'à `align-items` à savoir :

- `stretch` ;
- `flex-start` ;
- `flex-end` ;
- `center` ;
- `baseline`.

Cette propriété va surcharger la valeur de `align-items` pour les éléments pour lesquels elle a été définie. Cela signifie en d'autres termes que ce sera la valeur de `align-self` qui sera retenue pour l'alignement par-dessus la valeur de `align-items`.

De manière similaire à `align-items`, cette propriété n'aura un effet visible que si la ligne (ou la colonne) est plus grande que les éléments qu'elle contient, c'est-à-dire s'il existe des espaces vides.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne items-start">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible self-end">3</div>
        </div>
        <div class="conteneur-flexible ligne items-start">
            <div class="element-flexible">1</div>
            <div class="element-flexible self-end">2</div>
            <div class="element-flexible">3</div>
            <div class="element-flexible self-end">4</div>
            <div class="element-flexible">5</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
    height: 150px;
}
.ligne{
    flex-flow: row wrap;
}
.items-start{
    align-items: flex-start;
}
.self-end{
    align-self: flex-end;
}
.element-flexible{
    flex: 0 0 30%;
    width: 30%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}
```

Dans cet exemple, l'axe secondaire de nos conteneurs est à nouveau l'axe vertical. Notre premier conteneur ne possède qu'une ligne. La hauteur de la ligne va ainsi être égale à celle du conteneur. On définit ici un `align-items : flex-start` ce qui place les éléments en haut du conteneur et on surcharge ce comportement pour un élément en particulier qu'on place en bas avec `align-self : flex-end`.

Notre deuxième conteneur possède deux lignes. Aucune valeur n'est définie pour `align-content` et c'est donc sa valeur par défaut `stretch` qui va s'appliquer. On place ensuite les éléments en haut de chaque ligne avec `align-items` et on surcharge ce comportement pour deux éléments flexibles avec `align-self : flex-end` ce qui va les envoyer en bas de leur ligne.

Gérer la taille et la flexibilité des éléments flexibles

Dans les leçons précédentes, nous avons vu comment définir la direction des éléments flexibles, leur ordre et avons appris à les aligner sur leurs axes principal et secondaire.

Le modèle des boites flexibles nous offre une dernière série de propriété qui vont nous permettre de gérer la taille des éléments et de définir leur capacité à grossir et à se rétracter en cas de besoin dans le conteneur, c'est-à-dire définir leur capacité à être « flexible ».

Nous allons dans cette nouvelle leçon nous intéresser aux propriétés `flex-grow`, `flex-shrink` et `flex-basis` ainsi qu'à la propriété raccourcie `flex`.

Définir une taille de départ pour nos éléments flexibles avec flex-basis

La propriété `flex-basis` va nous permettre de définir une taille de départ pour les différents éléments d'un conteneur flexible avant distribution éventuelle de l'espace vide du conteneur ou au contraire rétrécissement des éléments.

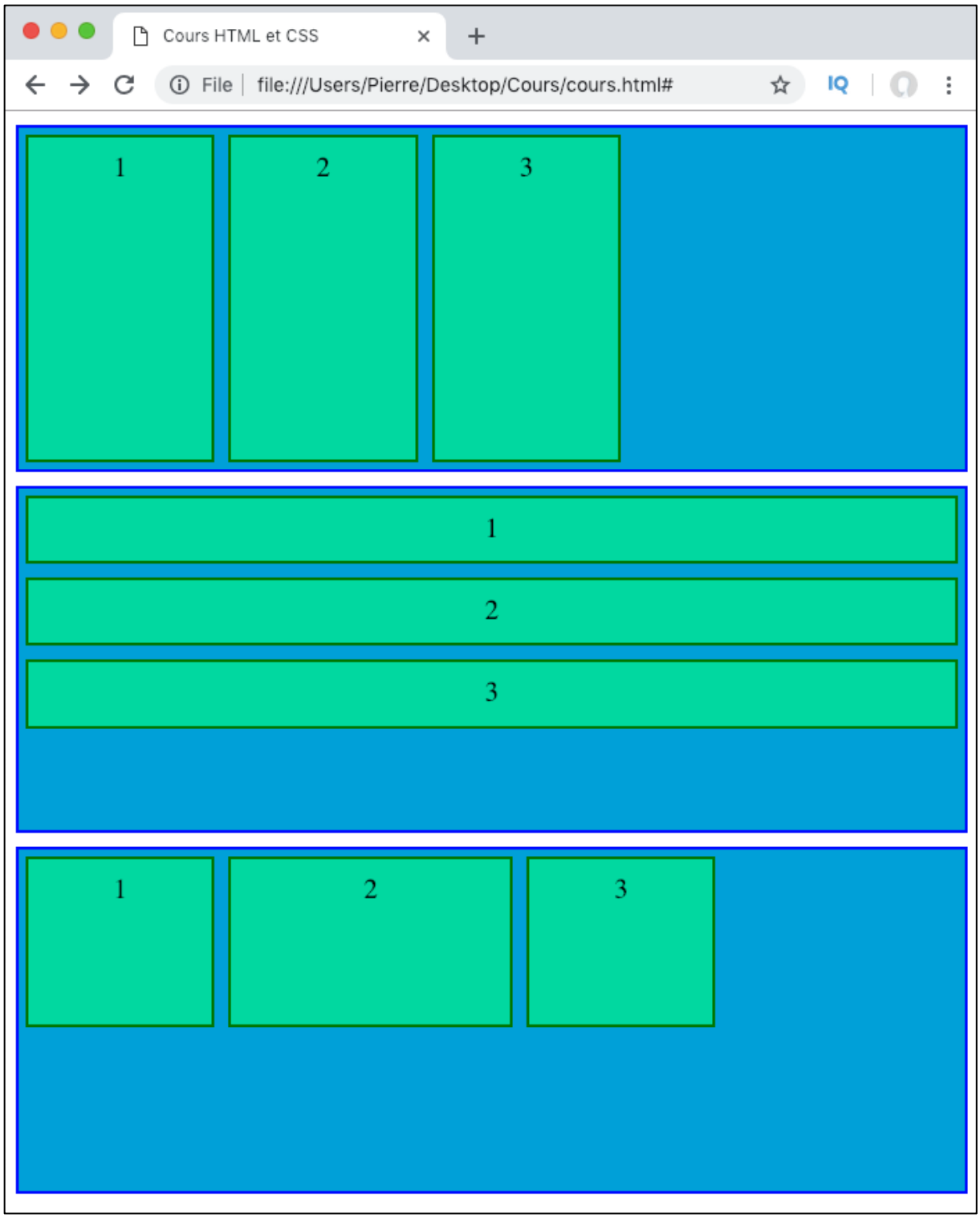
Nous allons appliquer cette propriété directement aux éléments flexibles et allons pouvoir lui passer une valeur, que nous exprimerons généralement en pourcentage de la taille du conteneur.

Notez ici que la propriété `flex-basis` ne va définir que la dimension de base d'un élément liée à son axe principal, c'est-à-dire sa largeur si l'axe principal est horizontal et sa hauteur si l'axe principal est vertical.

Cette propriété va par ailleurs surcharger la valeur éventuellement donnée à `width` ou à `height` selon que l'axe principal soit horizontal ou vertical.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
          </div>
          <div class="conteneur-flexible colonne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
          <div class="conteneur-flexible ligne dim">
            <div class="element-flexible">1</div>
            <div class="element-flexible ef30">2</div>
            <div class="element-flexible">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
    height: 250px;
}
.ligne{
    flex-flow: row wrap;
}
.colonne{
    flex-flow: column wrap;
}
.element-flexible{
    flex-basis: 20%;
    flex-grow: 0;
    flex-shrink: 0;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}

.dim .element-flexible{
    width: 50%;
    height: 50%;
}
.ef30{
    flex-basis: 30%;
}
```



Ici, nous définissons trois conteneurs flexibles qui contiennent trois éléments flexibles chacun. L'axe principal du premier et du troisième conteneurs est horizontal et l'axe principal du deuxième conteneur est vertical. J'ai précisé une hauteur explicite pour nos conteneurs afin de bien voir l'effet des propriétés que nous allons étudier.

Je définis ensuite un `flex-basis : 20%` pour chacun des éléments flexibles ainsi qu'un `flex-grow : 0` et un `flex-shrink : 0` que nous allons étudier ensuite pour interdire à nos éléments de grossir ou rétrécir.

Les éléments de notre premier conteneur vont ainsi avoir une largeur de départ égale à 20% de la largeur du conteneur tandis que les éléments de notre deuxième conteneur auront une hauteur également à 20% de la hauteur du conteneur.

Dans notre troisième conteneur, nous définissons un `flex-basis` différent pour un élément en particulier. L'élément en question aura donc ici une largeur de départ égale à 30% de celle de son conteneur.

Notez que nous passons également une largeur et une hauteur explicites à chacun des éléments flexibles de notre troisième conteneur. L'axe principal de ce conteneur est l'axe horizontal : la propriété `flex-basis` va donc nous permettre de définir la largeur de départ des éléments flexibles et la propriété `width` va être surchargée. La propriété `height` va en revanche ici bien s'appliquer.

Gérer la capacité des éléments flexibles à grossir avec flex-grow

La propriété `flex-grow` va nous permettre de définir la capacité des éléments à s'étirer dans leur conteneur pour remplir l'espace vide. Nous allons à nouveau l'appliquer aux éléments flexibles.

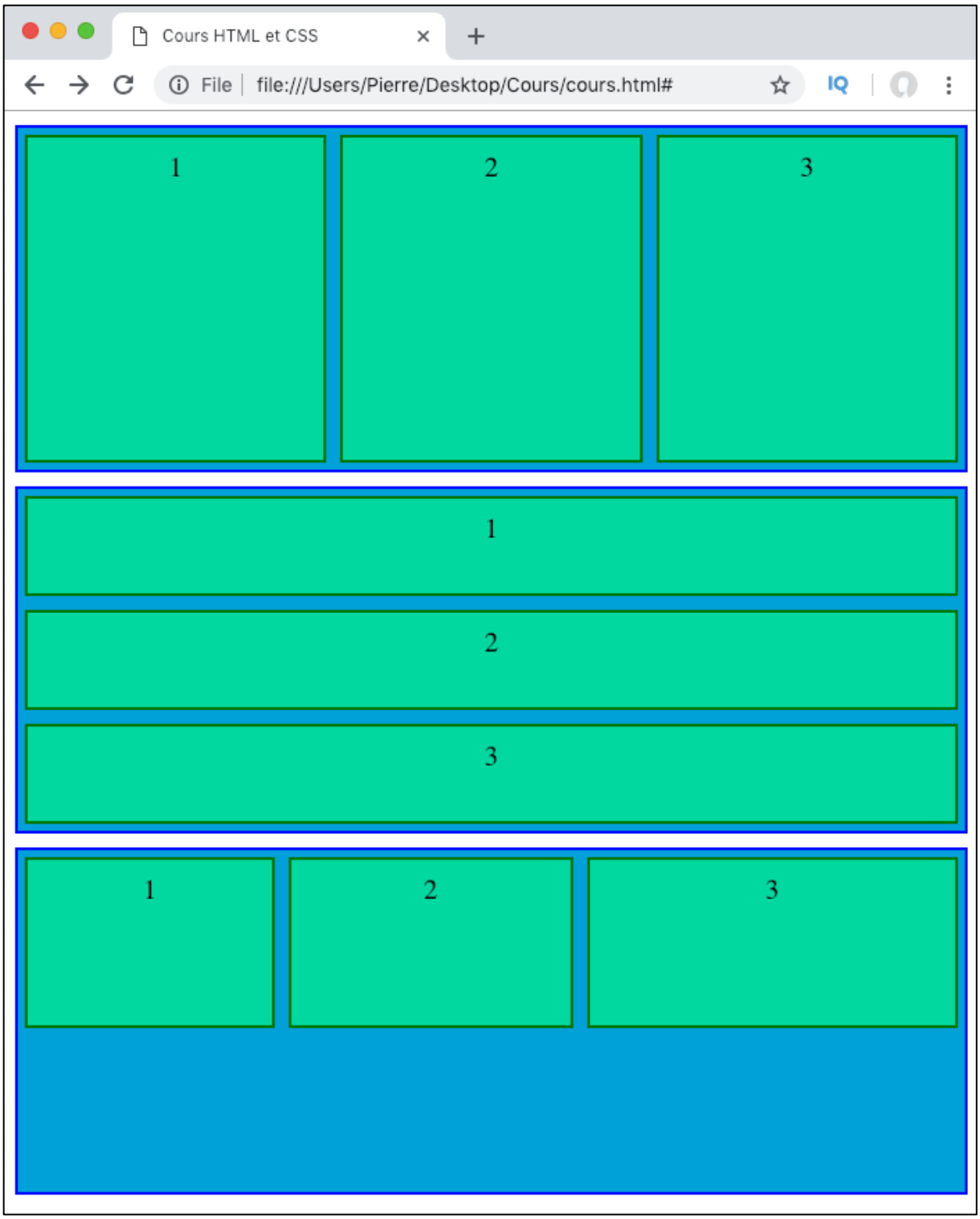
On va passer un nombre positif à `flex-grow` qui va définir la quantité d'espace disponible qu'un élément doit récupérer par rapport aux autres.

La valeur par défaut de `flew-grow` est `0` ce qui signifie que les éléments n'ont pas le droit de grossir dans leur conteneur.

Reprenons l'exemple précédent et appliquons cette fois-ci un `flex-grow : 1` aux éléments flexibles.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
          </div>
          <div class="conteneur-flexible colonne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
          <div class="conteneur-flexible ligne dim">
            <div class="element-flexible fg1">1</div>
            <div class="element-flexible ef30 fg0">2</div>
            <div class="element-flexible fg3">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
    height: 250px;
}
.ligne{
    flex-flow: row wrap;
}
.colonne{
    flex-flow: column wrap;
}
.element-flexible{
    flex-basis: 20%;
    flex-grow: 1;
    flex-shrink: 0;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}

.dim .element-flexible{
    width: 50%;
    height: 50%;
}
.ef30{
    flex-basis: 30%;
}
.fg0{
    flex-grow: 0;
}
.fg1{
    flex-grow: 1;
}
.fg3{
    flex-grow: 3;
}
```



Ici, on définit les dimensions de départ de nos éléments ave la propriété `flex-basis`. Nos éléments auront donc une largeur ou hauteur selon l'axe principal égale à 20% de leur conteneur sauf pour l'élément possédant la classe `.ef30` dont la largeur de départ va être égale à 30% du conteneur.

Vous devez bien comprendre ici que `flex-basis` sert à définir les dimensions de base des éléments flexibles avant un éventuel étirement ou rétrécissement.

On va également laisser la possibilité à nos éléments flexibles de s'étirer pour occuper l'espace vide dans le conteneur avec `flew-grow : 1`.

L'espace disponible dans le conteneur va donc être équitablement réparti entre les différents éléments de nos deux premiers conteneurs.

Pour notre troisième conteneur, on définit trois comportements de `flex-grow` différents pour nos trois éléments : notre premier élément va avoir le droit de s'étirer, le deuxième n'aura pas le droit, et le troisième aura le droit et va essayer de récupérer une quantité de l'espace restant dans le conteneur 3 fois plus importante que le premier élément.

Si vous êtes familiers des mathématiques, vous pouvez considérer que les nombres passés à `flex-grow` représentent une pondération. Sinon, retenez simplement que les nombres passés à `flex-grow` vont définir combien d'espace libre un élément peut avaler par rapport aux autres éléments.

Gérer la capacité des éléments flexibles à rétrécir avec flex-shrink

En plus de pouvoir gérer la capacité des éléments à grossir pour absorber l'espace vide dans un conteneur, nous allons également pouvoir gérer leur capacité à rétrécir pour éviter que des éléments ne dépassent du conteneur avec la propriété `flex-shrink`.

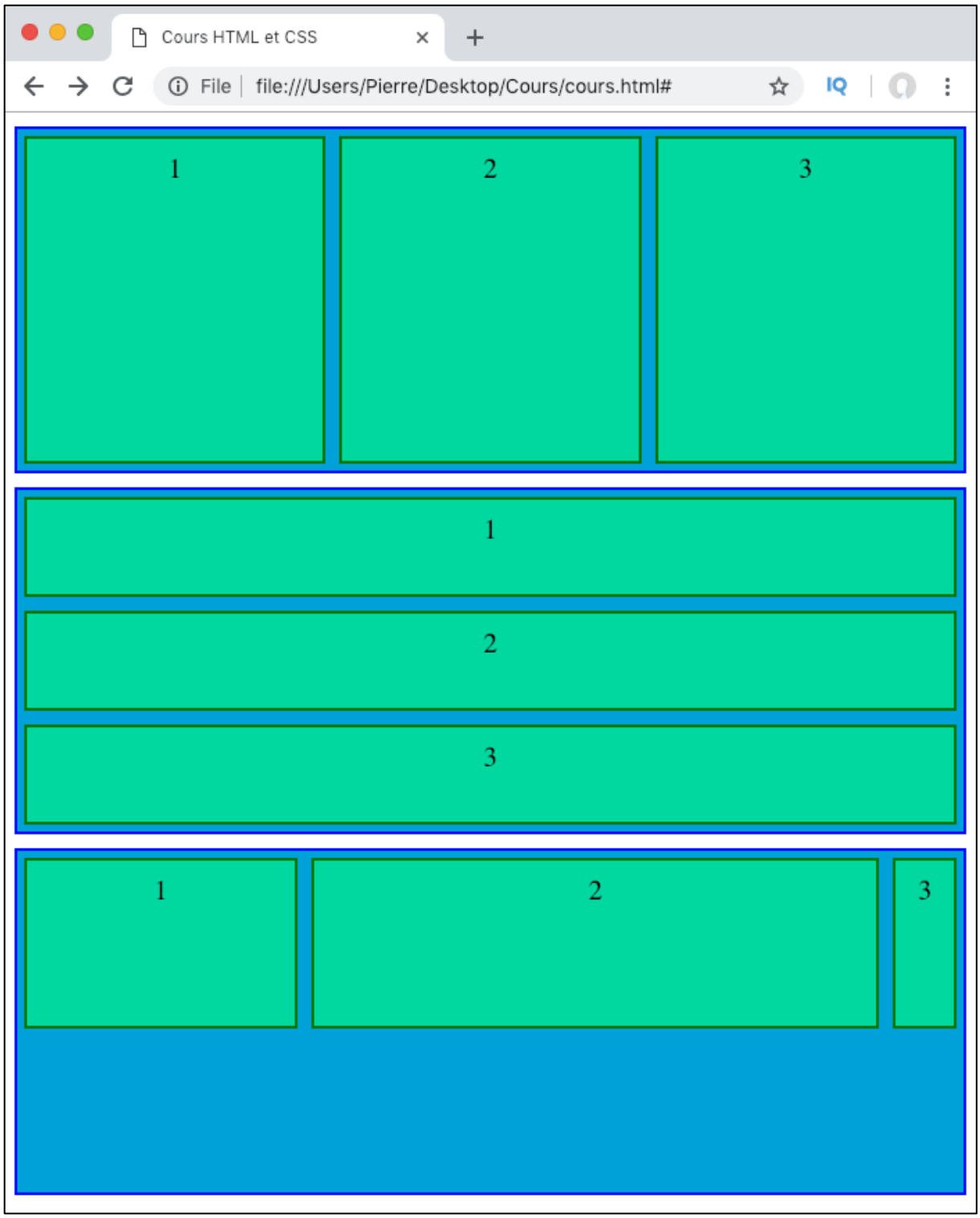
Cette propriété va s'utiliser de manière similaire à la précédente : nous allons à nouveau pouvoir passer un nombre positif à `flex-shrink` qui va indiquer la capacité des éléments à rétrécir ainsi que l'importance de leur rétrécissement les uns par rapport aux autres.

Cependant, à la différence de `flex-grow`, la valeur par défaut de `flex-shrink` est `1` ce qui correspond à laisser la capacité aux éléments de rétrécir.

Notez par ailleurs qu'un élément ne va pouvoir rétrécir que jusqu'au point où son contenu est prêt à dépasser de l'élément.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
          </div>
          <div class="conteneur-flexible colonne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
          <div class="conteneur-flexible ligne dim">
            <div class="element-flexible fs1">1</div>
            <div class="element-flexible ef60 fs0">2</div>
            <div class="element-flexible fs3">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
    height: 250px;
}
.ligne{
    flex-flow: row nowrap;
}
.colonne{
    flex-flow: column nowrap;
}
.element-flexible{
    flex-basis: 40%;
    flex-grow: 1;
    flex-shrink: 1;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}

.dim .element-flexible{
    width: 50%;
    height: 50%;
}
.ef60{flex-basis: 60%;}
.fs0{flex-shrink: 0;}
.fs1{flex-shrink: 1;}
.fs3{flex-shrink: 3;}
```



Cette fois-ci, on définit une taille de base de 40% pour chacun de nos éléments avec `flex-basis : 40%` et de 60% pour un élément en particulier dans le troisième conteneur.

On interdit également aux éléments de se placer sur plusieurs lignes ou sur plusieurs colonnes en passant la valeur `nowrap` pour la propriété `flex-wrap` aux différents conteneurs afin que les éléments dépassent bien par défaut de leur conteneur.

Finalement, on laisse la possibilité aux éléments de rétrécir afin qu'ils essaient de ne pas dépasser du conteneur avec `flex-shrink : 1`.

Dans notre troisième conteneur, nous définissons des comportements de rétrécissement différents pour chacun de nos éléments flexibles : le premier élément va pouvoir rétrécir selon un `flex-shrink : 1`, le deuxième ne pourra pas rétrécir et le troisième va également pouvoir rétrécir selon un `flex-shrink : 3`, ce qui signifie qu'il va essayer de rétrécir 3 fois plus que le premier élément flexible du conteneur.

La notation raccourcie flex

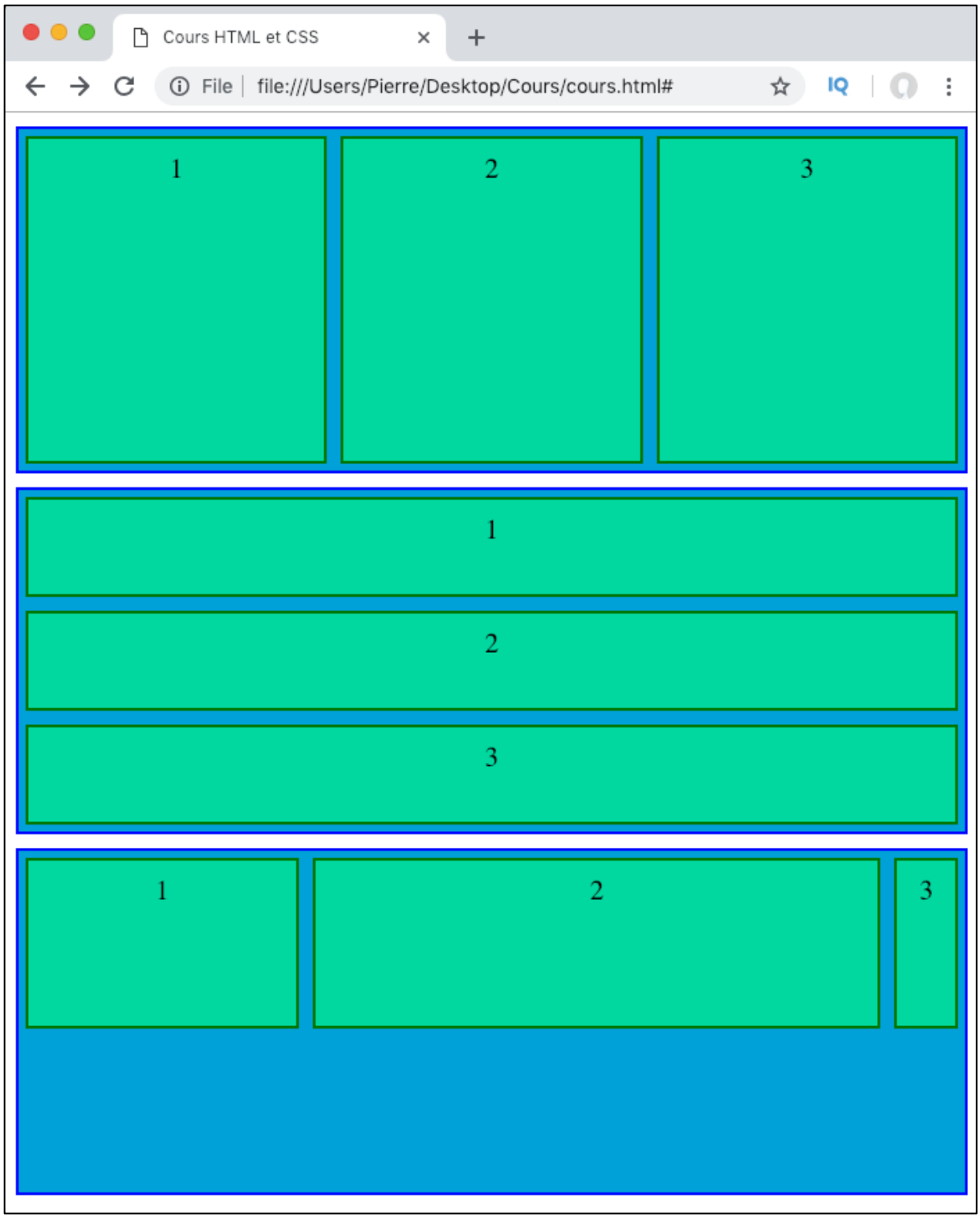
Nous allons pouvoir définir les trois propriétés `flex-grow`, `flex-shrink` et `flex-basis` d'un coup en utilisant la propriété raccourcie `flex`.

Pour que la propriété fonctionne correctement, il faudra lui passer d'abord la valeur liée à `flex-grow`, puis celle de `flex-shrink` et finalement celle de `flex-basis`.

Faites bien attention une nouvelle fois avec l'utilisation des propriétés raccourcies : si l'une des valeurs de la propriété raccourcie a été définie avant alors elle sera écrasée par la valeur de la propriété raccourcie.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flexible ligne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
          </div>
          <div class="conteneur-flexible colonne">
            <div class="element-flexible">1</div>
            <div class="element-flexible">2</div>
            <div class="element-flexible">3</div>
        </div>
          <div class="conteneur-flexible ligne dim">
            <div class="element-flexible fs1">1</div>
            <div class="element-flexible ef60 fs0">2</div>
            <div class="element-flexible fs3">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flexible{
    display: flex;
    font-size: 20px;
    background-color: #0AD; /*Bleu*/
    border: 2px solid blue;
    box-sizing: border-box;
    margin: 10px 0px;
    height: 250px;
}
.ligne{
    flex-flow: row nowrap;
}
.colonne{
    flex-flow: column nowrap;
}
.element-flexible{
    flex: 1 1 40%;
    background-color: #0DA; /*Vert*/
    padding: 10px 0px;
    text-align: center;
    border: 2px solid green;
    box-sizing: border-box;
    margin: 5px;
}

.dim .element-flexible{
    width: 50%;
    height: 50%;
}
.ef60{flex-basis: 60%;}
.fs0{flex-shrink: 0;}
.fs1{flex-shrink: 1;}
.fs3{flex-shrink: 3;}
```


Cours HTML et CSS
File | file:///Users/Pierre/Desktop/Cours/cours.html#
1
2
3
1
2
3
1
2
3

Cas d'utilisation et limites du flexbox

Le modèle des boites flexibles ou flexbox est un modèle de disposition très puissant qui permet de créer un conteneur et des éléments flexibles, c'est-à-dire des éléments qui vont pouvoir se réorganiser lorsque le conteneur change de dimension mais également les uns par rapport aux autres.

Nous allons pouvoir utiliser les propriétés suivantes pour déclarer un conteneur flexible et disposer nos différents éléments :

- `display : flex` nous permet de créer un conteneur flexible ;
- `flex-direction` nous permet de choisir l'axe principal et la direction des éléments flexibles selon cet axe ;
- `flex-wrap` nous permet de laisser la possibilité aux éléments flexibles qui dépassent du conteneur d'aller à la ligne ou sur une nouvelle colonne et de définir la direction des éléments selon l'axe secondaire ;
- `order` permet de choisir l'ordre d'affichage des éléments ;
- `justify-content` permet de définir l'alignement des éléments dans leur axe principal ;
- `align-items` permet de définir l'alignement des éléments dans leur axe secondaire ;
- `align-self` permet de définir l'alignement d'un élément en particulier dans son axe secondaire ;
- `align-content` permet de définir l'alignement des lignes ou des colonnes selon l'axe secondaire ;
- `flex-grow` permet de laisser la possibilité aux éléments flexibles de grossir pour absorber l'espace vide dans le conteneur ;
- `flex-shrink` permet de laisser la possibilité aux éléments flexibles de rétrécir dans le conteneur ;
- `flex-basis` permet de définir une taille de base pour les éléments flexibles.

En pratique, nous allons pouvoir recourir au modèle des boites flexibles pour résoudre simplement de nombreuses questions d'affichage et de positionnement qui pouvaient poser problème par le passé.

Exemples concrets d'utilisation du flexbox

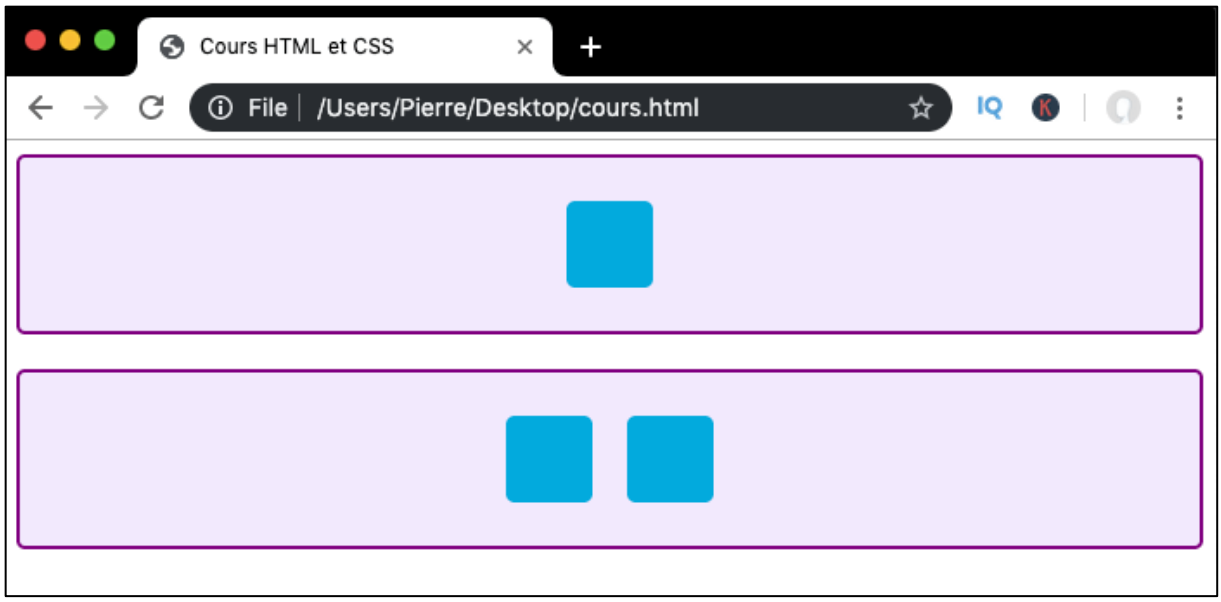
Effectuer un centrage parfait

Pendant très longtemps, il a été très difficile d'obtenir un centrage parfait en toute circonstance pour un élément ou un ensemble d'éléments et notamment un centrage vertical.

Cela devient très facile avec le flexbox puisqu'on va pouvoir centrer les éléments dans leur axe principal et dans leur axe secondaire.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur-flexible">
            <div class="element-flexible"></div>
        </div>

        <div class="conteneur-flexible">
            <div class="element-flexible"></div>
            <div class="element-flexible"></div>
        </div>
    </body>
</html>
```

```css
div{
    border-radius: 5px;
}
.conteneur-flexible{
    display: flex;
    align-items: center;
    justify-content: center;
    height: 100px;
    border: 2px solid purple;
    margin-bottom: 20px;
    background-color: RGBa(120,0,240,0.1)
}
.element-flexible{
    width: 50px;
    height: 50px;
    background-color: #0AD;
    margin: 10px;
}
```



Ici, on centre le ou les éléments par rapport à leur axe principal dans leur conteneur avec `justify-content : center` et on les centre également dans leur axe secondaire avec `align-items : center`.

Créer un menu avec le flexbox

Dans les parties précédentes, je vous ai proposé de créer un menu horizontal simple puis un menu déroulant.

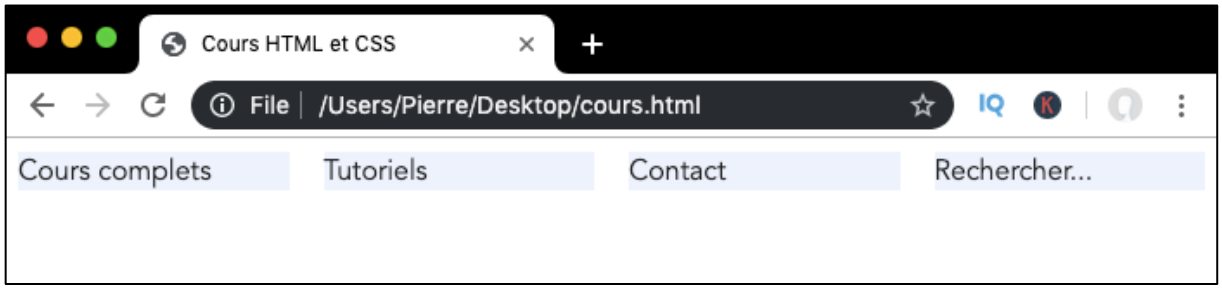
Pour que notre menu principal s'affiche de manière horizontale, nous avons utilisé la propriété `float`.

Ensuite, pour que les éléments du menu s'affichent correctement, nous leur avons attribué une largeur égale à 100% divisée par le nombre d'éléments.

Cette méthode n'était pas parfaite, loin de là. En effet, imaginons qu'on modifie le nombre d'éléments dans le menu : il faudra alors modifier la largeur donnée à chaque élément de menu. De même, selon la taille de certains écrans, on aimerait pouvoir distribuer l'espace dans le conteneur d'une façon ou d'une autre. Nous allons pouvoir gérer tout cela facilement avec le flexbox.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <nav>
            <ul class="conteneur-flex">
              <li><a href="#">Cours complets</a></li>
              <li><a href="#">Tutoriels</a></li>
              <li><a href="#">Contact</a></li>
              <li><a href="#">Rechercher...</a></li>
            </ul>
        </nav>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  color: #222;
  padding: 0px;
}
nav ul{
  display: flex;
  list-style-type: none;
  margin: 0px -10px; /*J'explique cela plus loin dans cette leçon !*/
}

nav ul li{
  flex: 1 1 20%;
  margin: 0px 10px;
  background-color: RGBa(60,120,240,0.1);
}

nav ul li a{
  display: block;
  text-decoration: none;
  color: inherit;
}
```



Ici, on attribue un `flex : 1 1 0` à nos éléments de menu. Cela signifie que nos différents éléments de menu vont tous avoir la même taille de base avec `flex-basis : 0` et vont pouvoir

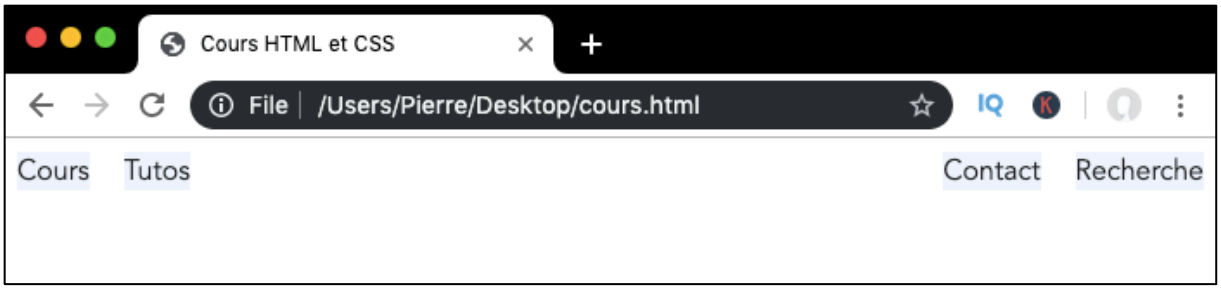
grandir ou rétrécir pour absorber l'espace vide du menu ou pour faire de la place à de nouveaux éléments.

Grouper des éléments

Le flexbox va également nous permettre simplement de créer des groupes d'éléments et de les aligner d'un côté ou de l'autre du conteneur. Pour cela, nous allons utiliser la propriété `margin` et sa valeur `auto` qui va faire que la marge va consommer tout l'espace vide dans le conteneur.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <nav>
            <ul class="conteneur-flex">
              <li><a href="#">Cours </a></li>
              <li><a href="#">Tutos</a></li>
              <li class="droite"><a href="#">Contact</a></li>
              <li><a href="#">Recherche</a></li>
            </ul>
        </nav>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  color: #222;
  padding: 0px;
}
nav ul{
  display: flex;
  list-style-type: none;
  margin: 0px -10px; /*J'explique cela plus loin dans cette leçon !*/
}
nav ul li{
  margin: 0px 10px;
  background-color: RGBa(60,120,240,0.1);
}
nav ul li a{
  display: block;
  text-decoration: none;
  color: inherit;
}
.droite{
  margin-left: auto;
}
```



Ici, on attribue un `margin-left : auto` au troisième élément de notre menu. Cela fait que sa marge gauche va consommer tout l'espace vide du conteneur et va donc repousser l'élément et les éléments suivants sur la droite. Cela va être particulièrement utile pour la création de menus.

Styliser les formulaires

Imaginons que l'on souhaite créer un formulaire d'inscription à une newsletter. La seule information qu'on va demander aux utilisateurs va être leur adresse mail.
Ici, plutôt que d'afficher notre unique `input` sur une ligne puis le bouton d'envoi du formulaire en dessous, on aimerait tout avoir sur la même ligne pour des questions d'ergonomie.

Cependant, pour que nos éléments de formulaire alignés s'affichent toujours bien, il va falloir qu'ils se redimensionnent les uns par rapport aux autres en fonction de l'espace disponible. On va pouvoir faire cela très simplement avec le flexbox.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <form method="#" action="#">
            <div class="conteneur-flex">
              <input type="email" name="newsletter">
              <input type="submit" value="Envoyer">
            </div>
        </form>

        <form method="#" action="#">
            <div class="conteneur-flex">
              <label for="mail">Votre mail :</label>
              <input type="email" name="newsletter" id="mail">
              <input type="submit" value="Envoyer">
            </div>
        </form>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
}

form{
  margin-bottom: 20px;
}

.conteneur-flex{
  display: flex;
  border: 1px solid #58A;
  align-items: center;
}

.conteneur-flex input[type="email"]{
  flex: 1 1 auto;
  background-color: RGBa(80,130,180,0.2);
 }

input{
  border: none;
  padding: 10px;
  background-color: RGBa(80,130,180,0.8);
  color: white;
}
label{
  padding: 10px;
  background-color: RGBa(80,80,80,0.8);
  color: white;
}
```

Ici, on applique `flex : 1 1 auto` à notre input de type `email` afin que celui-ci se redimensionne en fonction des éléments autour et pour qu'il occupe toujours tout l'espace vide dans le conteneur.

Les limites du flexbox

Le modèle des boites flexibles est à l'heure actuelle l'un des modèles de disposition les plus puissants que l'on puisse utiliser. Cependant, comme ce modèle est relativement récent, l'implémentation n'est pas encore parfaite et certaines propriétés manquent pour le moment.

Cela va être notamment le cas pour la gestion des gouttières, c'est-à-dire de l'espace entre les éléments flexibles. En effet, par défaut, nos différents éléments flexibles vont venir se coller les uns aux autres.

Pour créer des gouttières, c'est-à-dire pour créer un espace entre ces éléments, j'ai jusqu'à présent utilisé la propriété `margin`. Le problème ici est que cela va également créer un espace entre les éléments et le bord du conteneur, ce qui n'est pas toujours le comportement voulu.

Pour annuler cet espace, la solution actuellement admise est d'attribuer des marges négatives au conteneur qui vont venir contrebalancer les marges attribuées aux éléments flexibles.

Notez qu'on ne sera bientôt plus obligé d'utiliser cette technique puisque deux nouvelles propriétés qui vont nous permettre justement de créer des gouttières sont en train d'être ajoutées au modèle des boites : les propriétés `row-gap` et `column-gap`. Ces propriétés ne sont néanmoins pour le moment pas supportées par la plupart des navigateurs.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur">
            <div class="conteneur-flex">
              <div class="element-flex">1</div>
              <div class="element-flex">2</div>
              <div class="element-flex">3</div>
            </div>
            <div class="conteneur-flex">
              <div class="element-flex marge">1</div>
              <div class="element-flex marge">2</div>
              <div class="element-flex marge">3</div>
            </div>
            <div class="conteneur-flex margeneg">
              <div class="element-flex marge">1</div>
              <div class="element-flex marge">2</div>
              <div class="element-flex marge">3</div>
            </div>
            <div class="conteneur-flex gouttiere">
              <div class="element-flex">1</div>
              <div class="element-flex">2</div>
              <div class="element-flex">3</div>
            </div>
        </div>
    </body>
</html>
```

```css
.conteneur{
    overflow: hidden;
}
.conteneur-flex{
    display: flex;
    flex-wrap: wrap;
    background-color: RGBa(80,130,180,0.7);
    margin-bottom: 20px;
}
.element-flex{
    background-color: RGBa(80,180,130,0.7);
    width: 50px;
    height: 50px;
    border: 2px solid green;
}
.marge{
    margin: 0px 10px;
}
.margeneg{
    margin: 0px -10px 20px -10px;
}
.gouttiere{
    column-gap: 10px;
}
```

EXERCICE #4 : Création d'un menu flex

Dans un exercice précédent, nous avons réussi à créer un menu déroulant et sticky en HTML et en CSS en utilisant notamment les propriétés `float` et `position`.

Pour rappel, voici le code et le résultat auquel on était parvenu :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Menu déroulant sticky HTML et CSS</h1>
        <nav>
          <ul>
            <li class="deroulant"><a href="#">Cours Complets  </a>
              <ul class="sous">
                <li><a href="#">Cours HTML et CSS</a></li>
                <li><a href="#">Cours JavaScript</a></li>
                <li><a href="#">Cours PHP et MySQL</a></li>
              </ul>
            </li>
            <li class="deroulant"><a href="#">Articles  </a>
              <ul class="sous">
                <li><a href="#">CSS display</a></li>
                <li><a href="#">CSS position</a></li>
                <li><a href="#">CSS float</a></li>
              </ul>
            </li>
            <li><a href="#">Contact</a></li>
            <li><a href="#">A propos</a></li>
          </ul>
        </nav>

        <div class="conteneur">
          <p>Du contenu sous le menu</p>
        </div>
    </body>
</html>
```

```css
/*Reset CSS*/
*{margin: 0px; padding: 0px; font-family: Avenir, sans-serif;}
nav{width: 100%;
    margin: 0 auto;
    background-color: white;
    position: sticky;
    top: 0px;}

nav ul{list-style-type: none;}
nav ul li{
    float: left;
    width: 25%;
    text-align: center;
    position: relative;}

nav ul::after{content: ""; display: table; clear: both;}
nav a{
    display: block;
    text-decoration: none;
    color: black;
    border-bottom: 2px solid transparent;
    padding: 10px 0px;}

nav a:hover{color: orange; border-bottom: 2px solid gold;}
.sous{
    display: none;
    box-shadow: 0px 1px 2px #CCC;
    background-color: white;
    position: absolute;
    width: 100%;
    z-index: 1000;}

nav > ul li:hover .sous{display: block;}
.sous li{float: none; width: 100%; text-align: left;}
.sous a{padding: 10px; border-bottom: none;}
.sous a:hover{border-bottom: none; background-color: RGBa(200,200,200,0.1);}
.deroulant > a::after{content:" ▼";font-size: 12px;}
.conteneur{margin: 50px 20px; height: 1500px;}
```

Le résultat était relativement bon mais pas parfait. Une chose était particulièrement ennuyeuse avec ce menu : le fait qu'on avait dû définir un `width : 25%` pour chaque élément de menu.

Cela rend le menu peu flexible puisque chaque élément va prendre exactement la même place quel que soit son contenu et car il faudra changer la valeur de largeur dès qu'on enlève ou ajoute un élément pour garder un affichage correct.

Ici, le flexbox semble tout indiqué pour résoudre nos problèmes puisqu'on va pouvoir laisser les éléments se redimensionner en fonction des autres et grossir ou rétrécir selon l'espace restant dans le conteneur. Reprenons donc notre menu et transformons-le en utilisant le modèle des boites flexibles.

Création d'un menu horizontal en utilisant le flexbox

Nous allons conserver exactement le même code HTML que dans notre menu précédent et n'allons modifier que le CSS.

La première chose que nous allons faire ici va être de retirer tout ce qui a rapport aux flottants.

Ensuite, nous allons définir un conteneur flexible en ajoutant un `display : flex` à notre menu principal (représenté par un élément `ul`).

```
nav ul{
    list-style-type: none;
    display: flex;
}
```

On va alors pouvoir retirer la propriété `width` pour nos éléments flexibles `li` et ajouter plutôt un `flex : 1 1 auto` ce qui va permettre aux éléments d'avoir une taille de base relative à leur contenu et ensuite de pouvoir grandir pour occuper l'espace vide ou rétrécir pour laisser de la place à de nouveaux éléments.

```css
nav ul li{
    flex: 1 1 auto;
    text-align: center;
    position: relative;
}
```

Nous allons toujours vouloir afficher nos sous menus en colonne et, ça tombe bien, le flexbox nous permet de gérer la direction de nos éléments flexibles.

On va donc déclarer nos conteneurs `.sous` comme des conteneurs flexibles et leur ajouter un `flex-flow : column wrap`. On va ensuite à nouveau passer un `flex : 1 1 auto` à nos éléments flexibles.

```css
nav > ul li:hover .sous{
    display: flex;
    flex-flow: column wrap;
}
.sous li{
    flex: 1 1 auto;
    text-align: left;
}
```

Et voilà tout ! Notre menu est désormais flexible et les éléments de menu vont occuper une place relative à leur contenu et pouvoir rétrécir ou s'agrandir en fonction de la taille de la fenêtre et des autres éléments du menu.

Voici le résultat visuel auquel vous devriez arriver :

Note : Si vous voulez que les différents éléments de menu occupent tous le même espace de base, il suffit de changer la valeur de `flex-basis` de `auto` à `0`.

PARTIE XIV

Responsive design

Introduction au responsive design

Dans cette dernière partie, nous nous attaquons à une notion clef du développement d'un site Internet : le responsive design ou « design adaptable ».

Nous allons voir ici les principaux défis apportés par la multiplication des appareils pouvant aller sur le web et comment y répondre efficacement.

Comprendre les défis apportés par la multiplication des écrans

Il y a quelques années encore, on ne pouvait accéder à Internet et au web que grâce à des ordinateur de bureau.

Avec la miniaturisation des composants et les capacités de connexion de plus en plus puissante, cependant, de nouveaux appareils connectés ont vu le jour et les sites web peuvent maintenant être consultés à partir de nombreux appareils différents : ordinateurs de bureau, ordinateurs portables, tablettes, smartphones, montres connectées, etc.

Cela a multiplié les défis en termes d'ergonomie et de design pour les concepteurs de site : en effet, comment faire en sorte qu'un site Internet s'affiche correctement à la fois sur un écran de bureau et sur un smartphone ? Doit-on créer deux sites différents ? Deux versions d'un même site ? Dans ce cas-là, est-il préférable de retirer des informations pour rendre la version mobile plus légère ou peut-on se « contenter » de réarranger le contenu ?

Une définition rapide du responsive design

Lorsqu'on parle de « responsive design » ou de « design adaptable » en français, on fait référence à l'idée selon laquelle un site web devrait s'afficher aussi bien sur un écran de PC que sur un écran de smartphone ou sur n'importe quel type d'appareil.

Aujourd'hui, nous pouvons utiliser principalement trois méthodes pour répondre aux défis amenés par les différentes tailles d'écran. On peut :

- Créer une application dédiée pour les mobiles ;
- Créer une « copie mobile » de notre site en utilisant l'initiative AMP (« Accelerated Mobile Pages ») de Google ;
- Utiliser les Media Queries ou requêtes media.

L'idée de base du responsive design est qu'un site web devrait présenter les mêmes informations quel que soit l'appareil qui l'affiche, en les réarrangeant pour conserver la meilleure ergonomie possible. Nous allons pouvoir achever cela en utilisant les Media Queries qui seront donc la notion centrale de cette partie.

Définition rapide des Media Queries

Les Media Queries vont nous permettre d'appliquer certaines règles CSS de manière conditionnelle. Par exemple, on va pouvoir définir une largeur pour un élément pour certaines tailles d'écrans et une largeur différente pour le même élément pour d'autres tailles d'écran.

Cela va nous permettre d'afficher des pages avec des organisations différentes selon la taille de l'écran d'un visiteur. Attention ici à ne pas confondre les Media Queries et les valeurs de taille relatives et en % ou le flexbox.

Les valeurs en % et le flexbox vont pouvoir permettre aux éléments de grandir ou de rétrécir selon la taille d'un écran mais selon la même règle CSS.

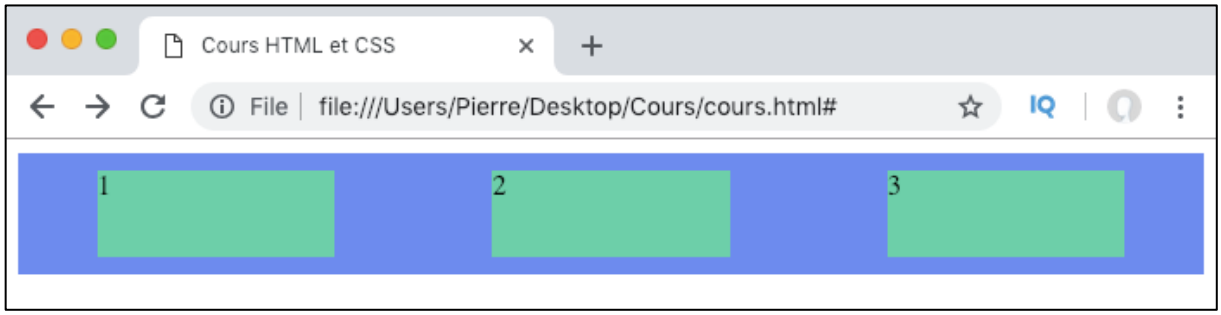
Avec les Media Queries, nous allons pouvoir appliquer des règles CSS totalement différentes selon les tailles d'écrans. Notez par ailleurs qu'on va tout à fait pouvoir utiliser le flexbox, etc. dans nos Media Queries.

Regardez l'exemple ci-dessous pour bien comprendre :

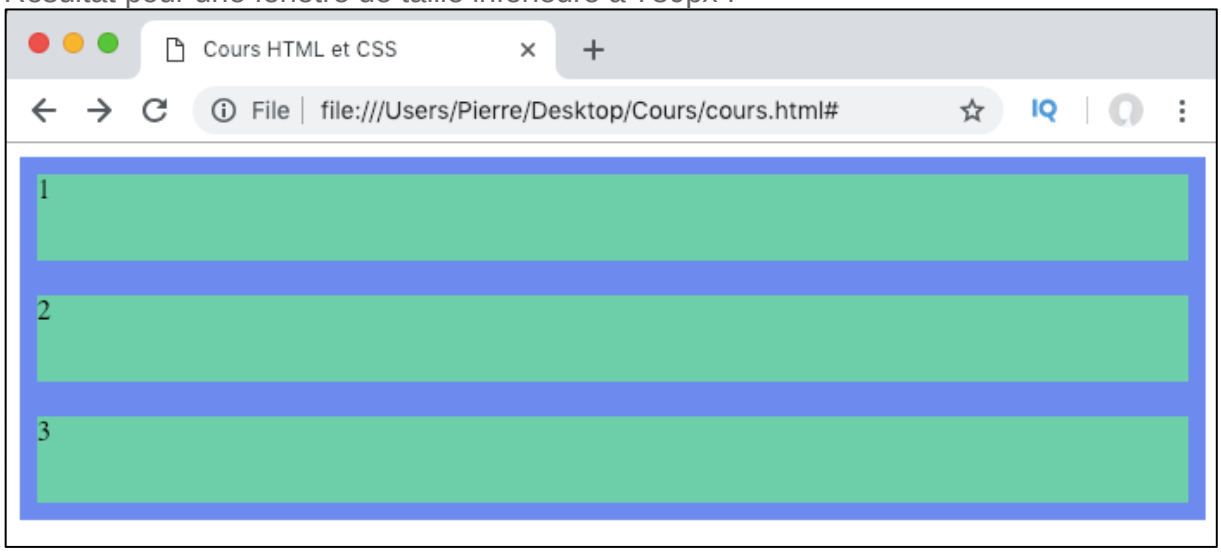
```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flex">
            <div class="element-flex">1</div>
            <div class="element-flex">2</div>
            <div class="element-flex">3</div>
        </div>
    </body>
</html>
```

```css
.conteneur-flex{
    display: flex;
    flex-flow : row wrap;
    justify-content: space-around;
    background-color: RGBa(120,150,240,1);
}

.element-flex{
    flex: 0 1 20%;
    background-color: RGBa(120,240,150,0.7);
    height: 50px;
    margin: 10px;
}

@media screen and (max-width: 780px){
    .conteneur-flex{
        flex-flow: column wrap;
    }
}
```

Résultat pour une fenêtre de taille supérieure à 780px :



Résultat pour une fenêtre de taille inférieure à 780px :



Ici, on commence par définir un conteneur flexible qui va contenir trois éléments flexibles. On définit ensuite l'axe principal et la direction de notre conteneur avec `flex-flow : row wrap`. Notre axe principal va donc être l'axe horizontal par défaut et les éléments flexibles

vont s'afficher les uns à côté des autres. Les éléments seront distribués régulièrement le long de leur axe principal avec `justify-content: space-around`.

On définit finalement une dimension de `20%` pour nos éléments flexbiles dans leur axe principal et on leur interdit de grossir automatiquement.

Ensuite, on va définir une règle media avec `@media`. Ici, notre règle va s'appliquer à tous les appareils disposant d'un écran (screen) de taille inférieure ou égale à 780px (`max-width : 780px`).

Dans cette règle media, nous allons pouvoir écrire autant de règles CSS que l'on souhaite. Ces règles CSS ne vont être appliqués que lorsqu'un utilisateur affiche la page avec un écran ou dans une fenêtre de taille inférieure à 780px.

Ici, on choisit de définir la direction de notre conteneur flexible avec `flex-flow: column wrap` pour tous les écrans de taille inférieure à 780px.

Notez qu'en cas de conflit (c'est-à-dire si une même propriété a déjà été définie en dehors d'une règle media ou dans une autre règle media), c'est la valeur de la propriété définie dans la règle la plus précise ou la plus restrictive qui va s'appliquer.

Dans le cas présent, nos éléments flexibles s'afficheront donc en ligne de manière générale et en colonne dès que la fenêtre fera moins de 780px.

Ce type de modification de structure ne serait pas possible en utilisant juste le flexbox ou des valeurs en `%` et ne va pouvoir être réalisé qu'avec les Media Queries.

La balise meta viewport

Avant d'étudier les Media Queries et la création de design responsives en profondeur, il est important de comprendre le rôle de la balise `<meta name= "viewport">`.

L'élément HTML meta

L'élément HTML `meta` est utilisé pour définir des métadonnées pour un document HTML.

Une métadonnée est une donnée qui ne va pas être affiché sur la page mais qui va pouvoir servir aux différents robots pour comprendre et afficher la page.

On va pouvoir ajouter différents attributs à l'élément `meta` qui vont nous servir à spécifier différents types de données.

Les attributs les plus courants de l'élément `meta` vont être les attributs `charset`, `name` et `content`.

Nous connaissons déjà l'attribut `charset` qui sert à indiquer le jeu de caractères ou l'encodage de notre document. Cet attribut va notamment permettre aux navigateurs d'utiliser le bon jeu de caractères et d'afficher correctement les caractères de notre page et notamment les accents et autres caractères spéciaux comme les cédilles et etc.

L'attribut `name` va nous permettre d'indiquer le type de métadonnées que l'on souhaite passer. Cet attribut va aller de pair avec l'attribut `content` qui va lui nous permettre de passer une métadonnée en soi.

L'attribut `name` va notamment pouvoir prendre les valeurs suivantes :

- `author` : la valeur passée à `content` sera considérée comme étant le nom de l'auteur du document ;
- `description` : la valeur passée à `content` pourra être utilisée par les moteurs de recherche comme extrait pour décrire le sujet de notre page ;
- `viewport` : la valeur passée à `content` va nous permettre d'indiquer comment le navigateur doit afficher la page sur différents appareils.

Le viewport et les problèmes d'affichage sur mobile

Le viewport représente de manière schématique la partie visible d'une page web par un utilisateur ou la fenêtre active.

La taille de cette fenêtre va bien évidemment varier en fonction de la taille de l'écran de l'utilisateur et de l'appareil utilisé.

Le problème qui s'est posé avec les smartphones est que la taille du viewport, c'est-à-dire la taille de la fenêtre d'affichage des pages web va souvent être différente de la taille physique des écrans.

En effet, la plupart des smartphones utilisent un viewport plus grand que la taille réelle de leur écran afin d'éviter aux utilisateurs d'avoir à dézoomer dans le cas où ils consulteraient un site non optimisé pour une navigation sur mobile.

Ainsi, imaginons par exemple qu'un smartphone utilise un viewport de 980px de large. Le site va donc s'afficher dans cette fenêtre. Seulement, notre smartphone n'a une taille réelle d'écran que de 400px.
Bien évidemment, ici, le viewport ne va pas dépasser de l'écran physique mais va être recadré pour s'afficher dans l'écran. Ainsi, notre site va apparaitre comme dézoomé, puisque le niveau de zoom sera de 400 / 980 = 0,41x.

Le vrai problème ici est tous les smartphones et les navigateurs mobiles n'utilisent pas les mêmes tailles de viewport ni les mêmes ratios. Ainsi, nos pages vont apparaitre plus ou moins dézoomées selon les appareils utilisés.

La balise `meta name= "viewport"` a été créée pour nous permettre de reprendre le contrôle du viewport et notamment de sa taille et de son échelle afin de proposer la meilleure version de notre site pour les différents appareils.

La balise meta name= »viewport »

La balise `meta name= "viewport"` va permettre de donner des instructions relatives à la taille et à l'échelle du viewport aux navigateurs mobiles afin que les différents éléments d'une page s'affichent au mieux.

Nous allons pouvoir passer plusieurs propriétés à l'attribut `content`.

Les propriétés `width` et `height` vont nous permettre de contrôler la taille du viewport dans lequel notre page doit s'afficher. On peut leur passer un nombre ou le mot clef `device-width` qui correspond à la taille de l'écran en pixels CSS à l'échelle 100%.

Ici, il me semble intéressant de définir ce qu'est « un pixel CSS ». Les pixels CSS correspondent à la surface utilisable de l'écran. Ce sont des pixels virtuels que l'appareil « pense » avoir. L'idée importante ici est qu'un pixel CSS n'est pas toujours égal à un pixel physique.
Les pixels physiques correspondent aux pixels réels qui composent un écran. C'est également ce qu'on appelle la définition d'un écran. Les écrans retina et haute définition possèdent généralement 4 fois plus de pixels réels que de pixels CSS.
La propriété `user-scalable` permet à l'utilisateur de zoomer dans la page (avec la valeur `yes`) ou, au contraire, lui interdit de la faire (avec la valeur `no`).

Cette propriété est souvent utilisée avec les propriétés `minimum-scale` et `maximum-scale` auxquelles on va pouvoir passer un nombre entre 0 et 10 et qui va représenter le niveau de dézoom ou de zoom que l'utilisateur est autorisé à faire.

Finalement, la propriété `initial-scale` permet de définir de niveau de zoom initial du viewport, c'est-à-dire son échelle. Nous allons également pouvoir lui passer un nombre entre 0 et 10.

Définir le meta viewport : quelles valeurs choisir ?

Si vous avez suivi jusque-là, vous devriez avoir compris qu'il est essentiel de définir une balise `meta name= "viewport"` ave des propriétés et des valeurs adaptées afin que les différents navigateurs mobiles ne définissent pas eux-mêmes leur propre viewport et leur niveau de zoom de notre page.

Généralement, nous définirons une largeur de viewport égale à la largeur de l'appareil dans le viewport ainsi qu'un niveau de zoom initial égal à 1 et interdirons les utilisateurs de zoomer ou de dézoomer. Nous allons donc utiliser la balise suivante : .

```
<head>
    <title>Cours HTML et CSS</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
    <link rel="stylesheet" href="cours.css">
</head>
```

Les media queries

Dans cette nouvelle leçon, nous allons découvrir dans le détail ce que sont les requêtes media ou Media Queries, comprendre comment elles fonctionnent et apprendre à les utiliser.

Définition des Media Queries

Les Media Queries correspondent à des styles CSS conditionnels. Les Media Queries se basent sur la règle CSS `@media` qui va nous permettre de définir différents styles CSS pour différents types d'appareils media et selon différents critères.

Nous allons ainsi, grâce aux Media Queries, pouvoir présenter le même contenu HTML de différentes façons en fonction de l'appareil utilisé par nos visiteurs pour accéder à nos pages.

On va ainsi par exemple déjà pouvoir changer la disposition des éléments ou la couleur de fond de nos pages en fonction de la taille de l'écran d'un utilisateur.

Le fonctionnement des Media Queries (syntaxe)

Dans notre règle `@media`, nous allons pouvoir placer deux types de conditions ou de contraintes : une condition sur le media utilisé pour afficher la page et une condition sur les caractéristiques du media.

A l'origine, on pensait que créer une condition sur le type de media serait suffisante et il n'y avait pas de conditions sur les caractéristiques des media.

Cependant, au fil du temps nous nous sommes rendu compte qu'il était beaucoup plus simple et plus logique d'ajouter des conditions sur les caractéristiques d'un media plutôt que sur son type.

Ainsi, on peut s'attendre à ce que la condition sur le type de media soit complètement dépréciée dans le futur. A l'heure actuelle, nous pouvons choisir parmi les types de media suivants :

- `all` : Valeur par défaut. Nos règles vont s'appliquer à tous les appareils ;
- `screen` : Nos règles ne vont s'appliquer qu'aux appareils dotés d'un écran ;
- `printer` : Nos règles ne s'appliqueront que pour les imprimantes ;
- `speech` : Nos règles ne s'appliqueront qu'aux liseurs d'écran qui sont capable de rendre le contenu d'une page de manière sonore.

Nous allons également pouvoir inverser la valeur logique d'un test avec le mot clef `not`. En plaçant le mot clef `not` avant le type de media, nos règles s'appliqueront à tous les appareils media sauf celui spécifié.

Ensuite, nous allons pouvoir créer des conditions sur les caractéristiques du media. Notez déjà qu'on peut créer autant de conditions sur des caractéristiques différentes que l'on souhaite.

Nous allons devoir entourer chaque condition sur une caractéristique media avec un couple de parenthèses et allons pouvoir séparer deux conditions avec les mots clefs `and` (et) ou `or` (ou).

Dans le cas où on utilise `and`, les deux conditions devront être vérifiées. Dans le cas où on utilise `or`, il suffira qu'une condition soit vérifiée.

Il existe de nombreuses caractéristiques sur lesquelles on peut effectuer nos tests. Cependant, en pratique, nous utiliserons généralement des conditions de taille pour distinguer entre différents appareils et utiliserons pour cela les propriétés `width`, `min-width` et `max-width`.

Sachez toutefois qu'on peut également conditionner l'application de nos styles CSS à la hauteur d'un media, sa résolution, son processus de scan, à la présence d'un appareil de pointage parmi les mécanismes de saisie et sa précision ou encore à la capacité de l'appareil à survoler les éléments.

Utilisation pratique des Media Queries

Depuis quelques années, la majorité des recherches web se font sur mobile. C'est la raison principale qui a amené Google à aujourd'hui indexer la version mobile des sites Internet et non plus leur version bureau. Cela fait qu'il est indispensable d'avoir une version mobile performante aujourd'hui.

Pour cette raison, il est considéré comme une bonne pratique aujourd'hui de créer son site en version mobile d'abord puis d'utiliser les Media Queries pour modifier la disposition du code pour les écrans d'ordinateurs ou de tablettes.

Fonctionner de cette manière peut vous sembler tout à fait logique si vous n'avez jamais développé auparavant mais je peux vous assurer que c'est une façon de procéder qu'il est difficile de faire admettre comme norme pour des développeurs expérimentés et habitués à créer des sites version bureau puis à les décliner pour mobile.

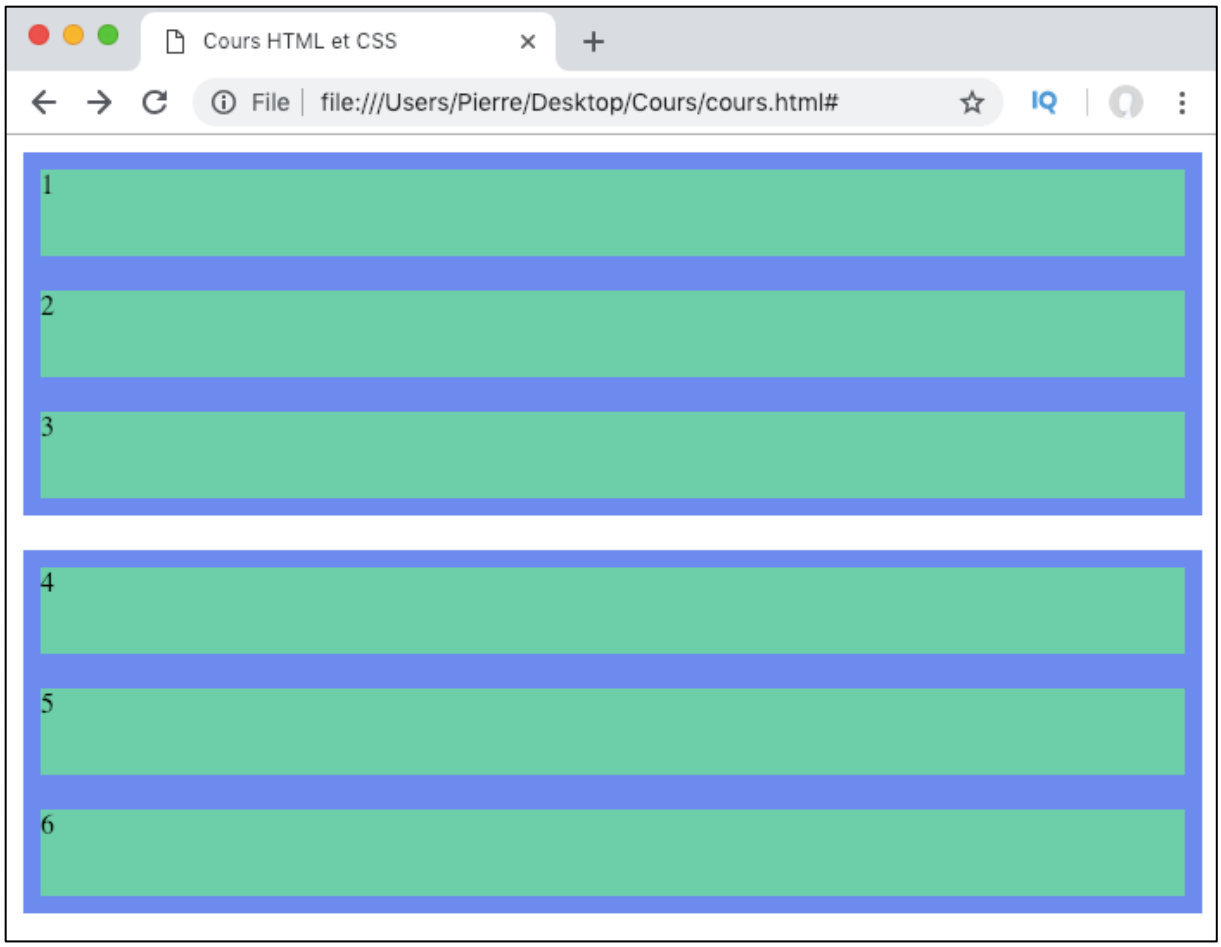
Dorénavant, nous travaillerons comme cela : on construira la version mobile de nos pages d'abord et nous utiliserons les Media Queries pour adapter nos pages pour des grands écrans.

Illustrons cela immédiatement avec un exemple :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <meta name="viewport"
              content="width=device-width, initial-scale=1.0, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flex">
            <div class="sous-conteneur-flex">
                <div class="element-flex">1</div>
                <div class="element-flex">2</div>
                <div class="element-flex">3</div>
            </div>
            <div class="sous-conteneur-flex">
                <div class="element-flex">4</div>
                <div class="element-flex">5</div>
                <div class="element-flex">6</div>
            </div>
        </div>
    </body>
</html>
```

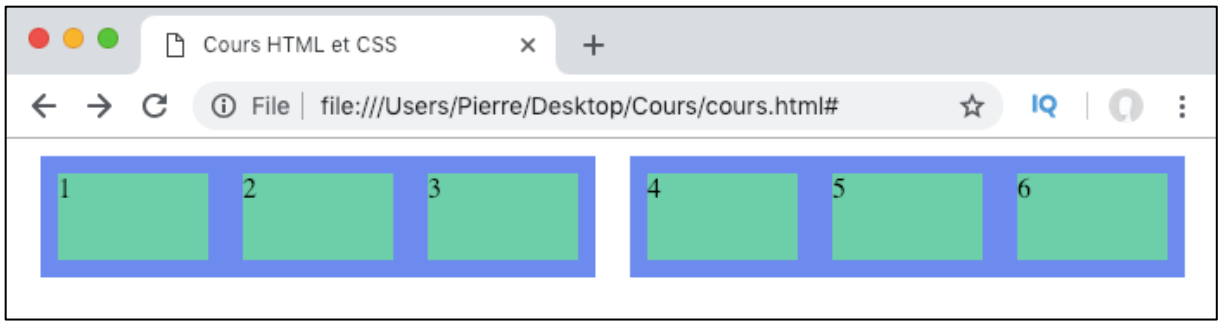
```css
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

.conteneur-flex{
    display: flex;
    flex-flow: column wrap;
    margin: 10px;
}
.sous-conteneur-flex{
    flex: 1 1 auto;
    display: flex;
    flex-flow: column wrap;
    justify-content: space-around;
    background-color: RGBa(120,150,240,1);
    margin: 0px 0px 20px 0px;
}

.element-flex{
    flex: 1 1 auto;
    background-color: RGBa(120,240,150,0.7);
    height: 50px;
    margin: 10px;
}

@media screen and (min-width: 780px) and (max-width: 979px){
    .conteneur-flex{
        flex-flow: row wrap;
    }
    .sous-conteneur-flex{
        margin: 0px 10px;
    }
}
@media screen and (min-width: 980px){
    .conteneur-flex{
        flex-flow: row wrap;
    }
    .sous-conteneur-flex{
        flex-flow: row wrap;
        margin: 0px 10px;
    }
}
```

Résultat pour des petits écrans (type mobile / smartphone) :



Résultat pour des écrans de taille moyenne (type tablette) :

Résultat pour des grands écrans (type ordinateur) :



Dans cet exemple, nous travaillons avec trois niveaux de `div` : un premier élément conteneur global, deux sous-conteneurs et des `div` qui servent d'éléments dans ces sous conteneurs.

En CSS, on commence donc par créer nos styles mobiles. On va ici appliquer un `display : flex` à notre conteneur et à nos sous conteneurs afin d'en faire des conteneurs flexibles.

Nos sous conteneurs vont donc à la fois être des conteneurs flexibles et des éléments flexibles. Comme les écrans de mobile sont les plus petits, on choisit une organisation des éléments verticale en appliquant un `flex-flow : column wrap` à la fois à notre conteneur flexible principal et à nos sous conteneurs.

On applique également des couleurs de fond et des marges externes à tous les `div` afin de bien pouvoir les différencier visuellement.

Ensuite, on définit deux règles `@media` qui vont s'adresser à différents appareils.

Notre première règle `@media` est la suivante : `@media screen and (min-width: 780px) and (max-width: 979px){}`. Cette règle va cibler les appareils media dotés d'un écran dont la largeur est comprise entre 780px et 979px.

Si un utilisateur utilise un appareil qui remplit ces conditions, alors les styles CSS indiqués dans la règle `@media` seront appliqués. Ici, notre règle s'adresse aux écrans de taille moyenne de type tablette.

On va vouloir pour ces écrans disposer nos sous conteneurs en ligne mais conserver un affichage des éléments dans les sous conteneurs en colonne. On va donc seulement modifier la direction du conteneur principal.

Ici, il faut bien comprendre que les styles de notre règle `@media` vont être traités de manière prioritaire et surcharger les styles définis globalement si les conditions relatives à l'appareil sont remplies.

Cependant, les autres propriétés définies globalement et qui ne sont pas reprécisées dans notre règle `@media` vont continuer à s'appliquer normalement (comme la couleur de fond par exemple ou la direction des sous conteneurs et etc).

Finalement, notre deuxième règle `@media` s'adresse aux grands écrans. On va ici vouloir afficher tous nos éléments en ligne. Nous allons donc attribuer un `flex-flow : row wrap` à la fois à notre conteneur flexible principal et à nos sous conteneurs flexibles.

Images responsives

Grâce aux Media Queries, nous avons pu réorganiser nos éléments HTML en créant des règles CSS spécifiques qui se s'appliquaient qu'à certaines tailles d'écran.

Cependant, on ne va pas pouvoir « réorganiser » une image en CSS. Nous allons bien évidemment pouvoir placer notre image dans un conteneur en HTML puis utiliser par exemple le flexbox pour rendre l'image « responsive » mais cela ne va pas être une solution optimale et ceci pour deux raisons :

- En n'utilisant qu'une seule image, nous allons être obligés de choisir la version la plus grande possible de celle-ci afin qu'elle ne pixellise pas lorsqu'un utilisateur l'affiche sur un grand écran. Or, plus une image est grande et plus elle est lourde : nous allons donc imposer aux mobiles le téléchargement d'une image très lourde sans raison ce qui va ralentir la vitesse d'affichage de nos pages ;
- En n'utilisant qu'une seule image, il est fort probable que l'image ne rende pas bien sur mobile car comme celle-ci sera plus petite à cause de la taille de l'écran le sujet dans l'image va apparaitre lointain ou comme dézoomé.

Ces deux problématiques font qu'on aimerait pouvoir proposer différentes versions d'une image pour différents types d'appareil. Nous allons voir comment faire dans ce chapitre.

Pixels physiques, pixels CSS et écrans retina

Nous l'avons vu dans le chapitre sur le viewport : un pixel n'est pas toujours égal à un pixel ! Comprendre cela va être très important pour afficher des images qui vont s'afficher correctement sur tous les appareils.

Ici, retenez que les écrans retina ont généralement une résolution deux fois plus importante que les écrans standards. Cela signifie que chaque « pixel retina » est l'équivalent de 4 « pixels standards » (2 en largeur, 2 en hauteur).

Ainsi, pour qu'une image s'affiche correctement sur un écran retina, il faudra qu'elle soit deux fois plus grande que l'écran sur lequel elle doit s'afficher (1960x 980px par exemple pour un affichage sur un écran retina de 980x 490px).

Notez également ici que certains écrans retina possèdent une résolution 3 fois plus importante qu'un écran standard, auquel cas il faudra une image 3 fois plus grande et etc.

Une première solution : utiliser des images SVG

La première solution, qui semble la plus évidente pour gérer les différentes tailles d'écran et les différentes résolutions est d'utiliser des images au format SVG pour Scalable Vector Graphic.

Comme le nom l'indique, ces images sont vectorielles, ce qui signifie qu'on va pouvoir les agrandir ou les rapetisser à l'infini sans perte de qualité.

Cependant, cela ne résout qu'une partie du problème puisqu'en n'utilisant qu'une seule image pour toutes les versions de notre site, cette image risque d'apparaitre comme trop imposante pour la version bureau ou trop dézoomée pour la version mobile.

De plus, souvent, vous serez obligés d'intégrer des photos ou images d'un format différent comme `jpeg` ou `png`.

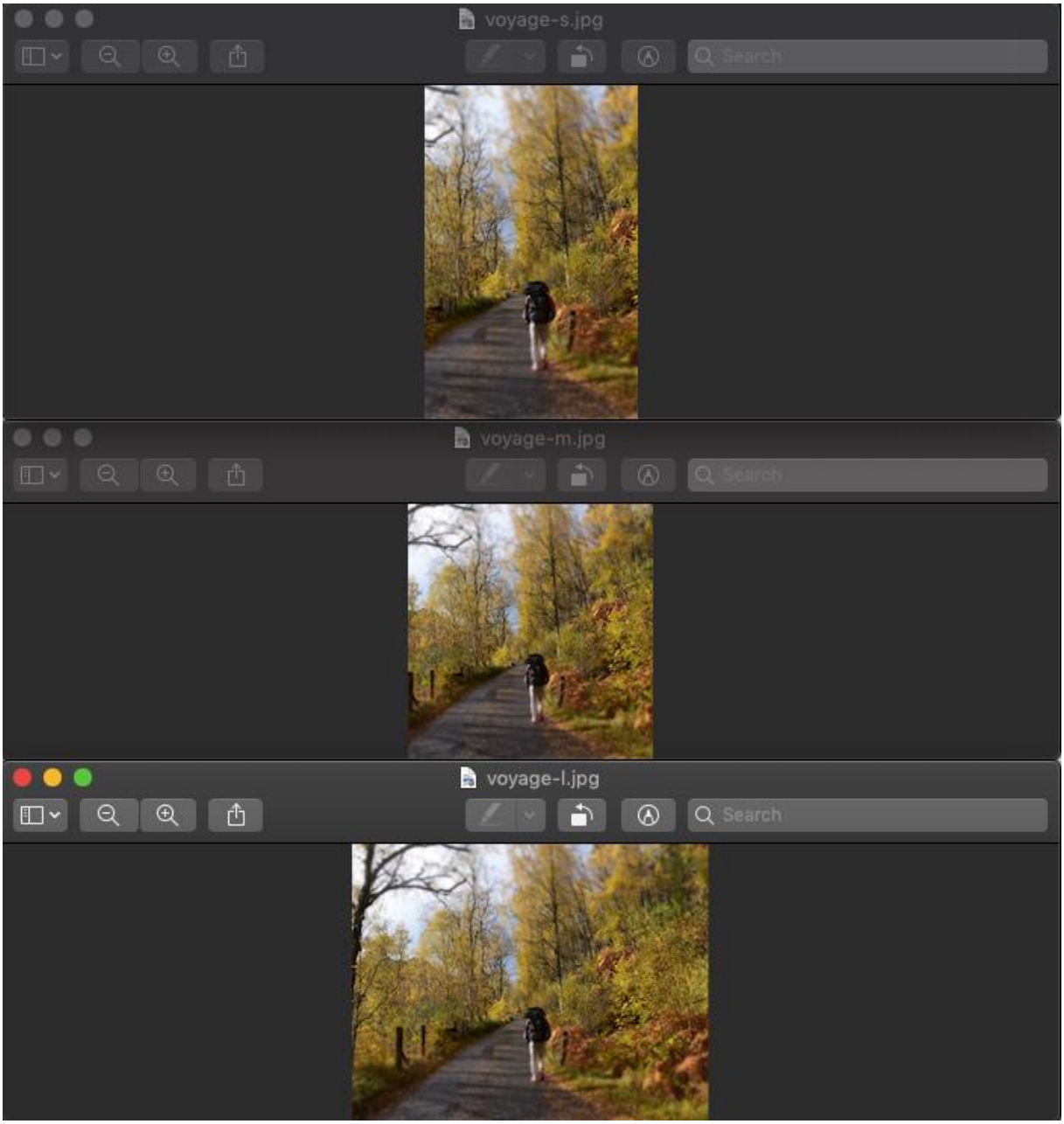
Proposer plusieurs versions d'une image en utilisant les attributs HTML srcset et sizes

Nous allons pouvoir ajouter un attribut `srcset` dans notre élément `img` qui va nous permettre de fournir plusieurs sources d'images (c'est-à-dire concrètement de fournir plusieurs images différentes) au navigateur parmi lesquelles choisir.

Ce qui va nous intéresser ici va être de proposer plusieurs versions d'une même image avec des tailles et des cadrages différents. Pour cet exercice, je vais utiliser trois versions d'une même image que j'ai prédécoupé avant :

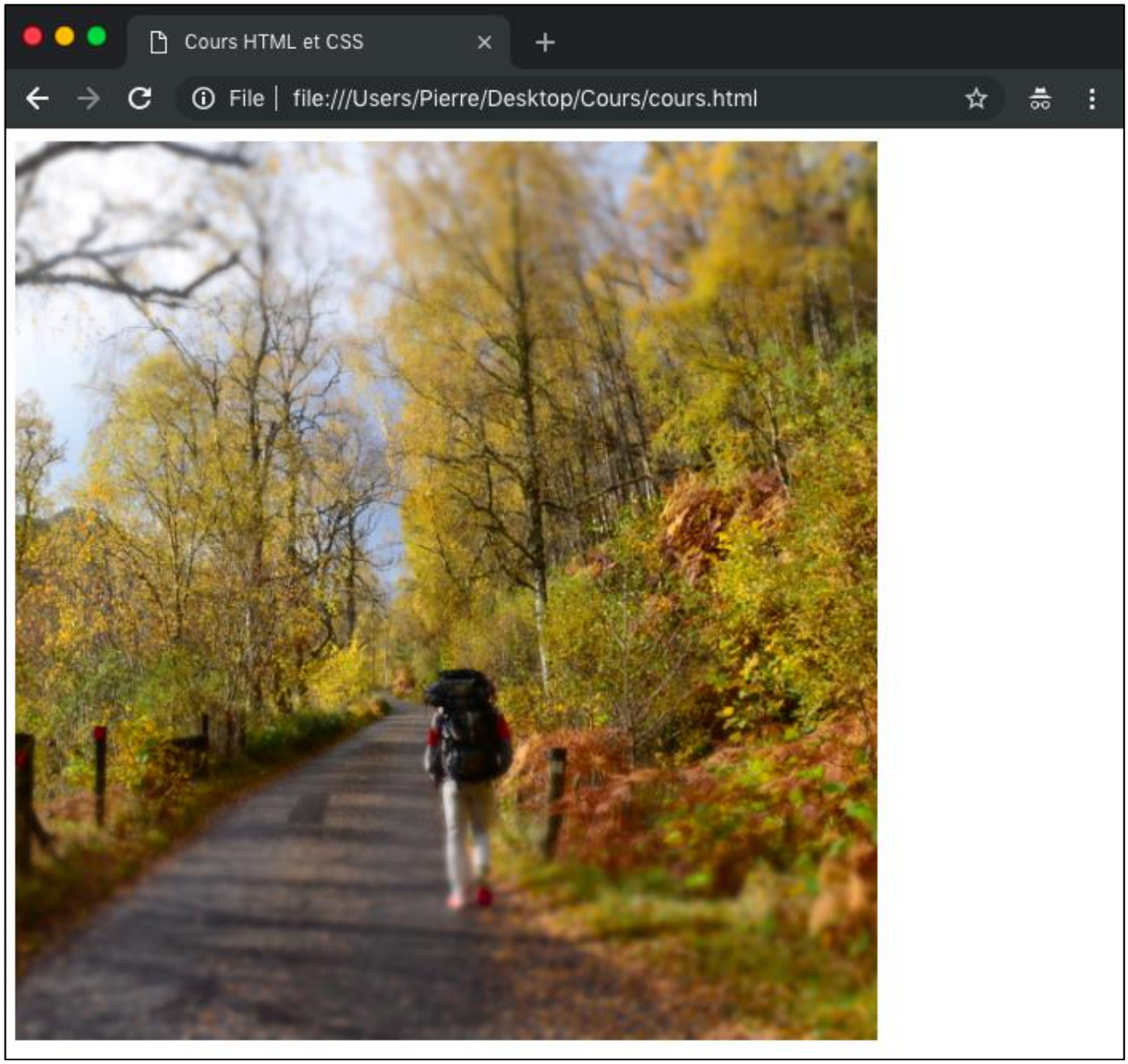
- Une version complète « L » de l'image de dimensions 800x 625px que l'on va servir aux grands écrans ;
- Une version « M » rognée avec un premier centrage sur le sujet pour les écrans de taille moyenne ;
- Une version « S » de dimensions 400x 625px complètement centrée sur le sujet pour les petits écrans.

Voici à quoi ressemblent mes trois photos :



L'attribut srcset va être accompagné d'un attribut sizes qui va nous permettre de préciser un ensemble de conditions relatives au terminal utilisé par un utilisateur (généralement des conditions de tailles de fenêtre) et d'indiquer la largeur que doit occuper l'image dans chaque situation.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset= "utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <img srcset="voyage-s.jpg 400w,
                     voyage-m.jpg 600w,
                     voyage-l.jpg 800w"
             sizes="(max-width: 576px) 380px,
                    (max-width: 768px) 540px,
                    700px"
             src="voyage-l.jpg"
             alt="Voyage en Ecosse">
    </body>
</html>
```



Ici, nous renseignons chacune des versions de notre image dans l'attribut `srcset` avec leur valeur intrinsèque en unités `w` qui sont un équivalent pixel. Pour faire très simple, nous indiquons la largeur de chacune de nos images en pixels avec l'unité `w`.

Ensuite nous allons préciser dans l'attribut `size` un ensemble de conditions et la largeur de l'espace que doit occuper l'image dans l'écran si une condition est vérifiée.

Les conditions seront généralement des conditions sur la taille de la fenêtre et vont être le strict équivalent de nos Media Queries CSS. Par exemple la condition `(max-width : 576px)` va être vérifiée pour toutes les tailles de fenêtre inférieures à 576px et l'image devra alors être affichée dans un espace de 380px de large.

Si cette condition est vérifiée, alors le navigateur utilisera l'image dont les dimensions sont le plus proche de l'espace dans lequel elle devra être affichée.

Notez que l'ordre d'écriture des conditions dans `sizes` compte puisque c'est toujours la première condition vérifiée qui va être retenue. Ainsi, nous déclarerons toujours nos conditions de la plus restrictive à la moins restrictive afin que le navigateur récupère la meilleure version à chaque fois.

Le navigateur va donc ici commencer par calculer la taille de l'écran de vos visiteurs puis passer en revue les conditions données jusqu'à ce qu'une condition soit vérifiée. Dès que c'est le cas, le navigateur va noter la largeur de la place que doit occuper l'image et va charger l'image dans `srcset` dont la taille est la plus proche de la largeur précisée dans notre condition.

Attention cependant ici : l'attribut `srcset` ne donne qu'une indication de préférence au navigateur et celui-ci est libre de l'ignorer pour charger la version de l'image qu'il souhaite. Ainsi, il est possible que le navigateur d'un visiteur ne charge pas la version souhaitée dans le cas où, par exemple, le navigateur possèderait déjà une version avec une meilleure résolution de l'image en cache.

En effet, la plupart des navigateurs comprennent bien que l'attribut `srcset` est utilisé à des fins d'optimisation. Or, si le navigateur possède déjà une version de qualité supérieure cachée d'un média, c'est-à-dire une version déjà téléchargée et prête à l'affichage, alors il est normal qu'il l'affiche puisque cela optimisera le rendu (supposément) et les performances.

Faites donc bien attention de votre côté si vous testez `srcset` à le tester en navigation privée ou à supprimer le cache de votre navigateur afin de bien voir l'image changer en fonction de la taille de l'écran.

EXERCICE #5 : Création d’un menu déroulant responsive

Après un menu horizontal simple / sticky, un menu horizontal déroulant et un menu utilisant le flexbox, nous arrivons finalement à notre dernier exercice de création de menu dans lequel nous allons cette fois-ci essayer de créer un menu responsive, c’est-à-dire un menu dont la disposition va s’adapter en fonction de l’écran de chaque visiteur.

Nous n’allons pas ici pouvoir créer quelque chose de parfait. En effet, idéalement, nous voudrons utiliser du JavaScript afin de déclencher l’ouverture et la fermeture du menu mobile. Cependant, nous allons pouvoir déjà faire de belles choses en n’utilisant que du HTML et du CSS !

Ici, nous allons à nouveau nous inspirer de la dernière version de notre menu utilisant le flexbox et modifier certains styles en se concentrant cette fois-ci sur la partie mobile qu’on va vouloir définir comme version standard du menu.

Pour rappel, le code de notre dernier menu flexible était le suivant :

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <h1>Menu déroulant sticky HTML et CSS avec flexbox</h1>
        <nav>
          <ul>
            <li class="deroulant"><a href="#">Cours Complets  </a>
              <ul class="sous">
                <li><a href="#">Cours HTML et CSS</a></li>
                <li><a href="#">Cours JavaScript</a></li>
                <li><a href="#">Cours PHP et MySQL</a></li>
              </ul>
            </li>
            <li class="deroulant"><a href="#">Articles  </a>
              <ul class="sous">
                <li><a href="#">CSS display</a></li>
                <li><a href="#">CSS position</a></li>
                <li><a href="#">CSS float</a></li>
              </ul>
            </li>
            <li><a href="#">Contact</a></li>
            <li><a href="#">A propos</a></li>
          </ul>
        </nav>

        <div class="conteneur">
          <p>Du contenu sous le menu</p>
        </div>
    </body>
</html>
```

```css
*{margin: 0px; padding: 0px; font-family: Avenir, sans-serif;}
nav{
    width: 100%;
    margin: 0 auto;
    background-color: white;
    position: sticky;
    top: 0px;}

nav ul{list-style-type: none; display: flex;}
nav ul li{flex: 1 1 auto; text-align: center; position: relative;}

nav a{
    display: block;
    text-decoration: none;
    color: black;
    border-bottom: 2px solid transparent;
    padding: 10px 0px;}

nav a:hover{color: orange; border-bottom: 2px solid gold;}
.sous{
    display: none;
    box-shadow: 0px 1px 2px #CCC;
    background-color: white;
    position: absolute;
    width: 100%;
    z-index: 1000;}

nav > ul li:hover .sous{display: flex; flex-flow: column wrap;}
.sous li{flex: 1 1 auto; text-align: left;}
.sous a{padding: 10px; border-bottom: none;}
.sous a:hover{
    border-bottom: none;
    background-color: RGBa(200,200,200,0.1);}

.deroulant > a::after{content:" ▼"; font-size: 12px;}
.conteneur{margin: 50px 20px; height: 1500px;}
```

Choix esthétiques pour la version mobile

Lorsqu'on se lance dans un projet de développement, il est important de commencer par définir les « spéc » (spécifications techniques) du projet en question avant de commencer à coder.

En effet, le fait de savoir précisément ce qu'on souhaite obtenir et de définir les différentes choses qu'on va pouvoir utiliser pour l'obtenir évite de faire des allers-retours dans son code, d'en supprimer des parties, d'en modifier d'autres et etc. et fait au final gagner beaucoup de temps.

Pour notre menu mobile, nous avons deux spécifications majeures : nous voulons que les éléments de menu s'affichent en colonne et voulons également que le menu soit replié par défaut pour éviter qu'il ne consomme trop de place sur l'écran.

Squelette HTML du menu responsive

Ici, nous allons récupérer le squelette HTML de notre menu déroulant utilisant le flexbox. Nous allons toutefois ajouter deux choses à ce code HTML : une balise `meta name="viewport"` que nous avons découvert dans cette partie ainsi qu'un `label` avec un élément `input type="checkbox"` associé qui vont nous servir de « hack » pour nous permettre de déplier et de replier le menu.

L'idée ici va être de n'afficher le menu que si la case a été cochée et de le cacher à nouveau dès que celle-ci est décochée.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <meta name="viewport"
              content="width=device-width, initial-scale=1.0, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Menu déroulant responsive HTML et CSS utilisant le flexbox</h1>
        <nav>
          <div class="conteneur-nav">
            <label for="mobile">Afficher / Cacher le menu</label>
            <input type="checkbox" id="mobile" role="button">
            <ul>
              <li class="deroulant"><a href="#">Cours Complets  </a>
                <ul class="sous">
                  <li><a href="#">Cours HTML et CSS</a></li>
                  <li><a href="#">Cours JavaScript</a></li>
                  <li><a href="#">Cours PHP et MySQL</a></li>
                </ul>
              </li>
              <li class="deroulant"><a href="#">Articles  </a>
                <ul class="sous">
                  <li><a href="#">CSS display</a></li>
                  <li><a href="#">CSS position</a></li>
                  <li><a href="#">CSS float</a></li>
                </ul>
              </li>
              <li><a href="#">Contact</a></li>
              <li><a href="#">A propos</a></li>
            </ul>
          </div>
        </nav>

        <div class="conteneur">
          <p>Du contenu sous le menu</p>
        </div>
    </body>
</html>
```

Styles CSS du menu responsive

Nous allons donc ici commencer par définir la version de mobile de notre menu qui sera sa version standard. Nous utiliserons ensuite les Media Queries pour définir une version pour grands écrans.

La première chose que l'on va faire ici va être d'appliquer un `display : none` à notre élément de liste représentant notre menu puisqu'on veut que celui-ci soit caché par défaut.

Nous allons vouloir afficher le menu seulement lorsque notre case à cocher a été effectivement cochée. Dans ce cas-là, nous lui appliquerons un `display : flex` et un `flex-flow : column wrap` pour que les éléments s'affichent bien en colonne. Nous allons également déjà en profiter pour appliquer une couleur de fond à tout notre menu.

Pour faire cela en CSS, nous allons utiliser la pseudo-classe `:checked` qui va nous permettre d'appliquer des styles seulement lorsqu'un élément a été coché. Ici, on veut afficher notre menu lorsque notre case à cocher a été cochée. On va donc appliquer notre pseudo-classe au sélecteur `nav input[type=checkbox]` et allons pouvoir utiliser le symbole `+` pour appliquer les styles à notre menu puisque l'élément `ul` suit directement notre élément `input`.

```
nav ul{
    display: none;
    list-style-type: none;
    background-color: #555;
}
nav input[type=checkbox]:checked + ul{
    display: flex;
    flex-flow : column wrap;
}
```

Nous avons lié notre `input` à un `label` en HTML grâce aux attributs `for` et `id`. Cela signifie qu'il suffit à un utilisateur de cliquer sur le `label` pour cocher et décocher la case. Cela va donc nous permettre de ne pas afficher la case mais d'afficher uniquement le label.

On va vouloir que le `label` occupe toute la largeur disponible à l'écran. Pour cela, on lui appliquer un `display : inline-block` et on lui passe une largeur égale à 100% de son élément conteneur. Par ailleurs, nous conservons les styles liés à l'élément `nav` de notre précédent menu qui nous conviennent très bien.

```
nav{
    width: 100%;
    margin: 0 auto;
    background-color: white;
    position: sticky;
    top: 0px;
}
nav input[type=checkbox]{
    display: none;
}
nav label{
    display: inline-block;
    width: 100%;
    padding: 10px 0px;
    text-align: center;
    background-color: gold;
}
```

Voilà pour la première partie du menu. Ensuite, nous allons supprimer tout ce qui dépend de la pseudo classe `:hover`. En effet, les mobiles ne disposent pas de curseur puisque les écrans sont tactiles et donc cette pseudo classe ne fait aucun sens pour la navigation sur mobile.

De gros changements vont être faits ici par rapport à notre version de menu précédente. Nous allons ici vouloir afficher l'intégralité du contenu de nos sous menus en toute circonstance.

On va donc en profiter pour supprimer ce qui était lié aux propriétés `position`. Pour ne pas que l'affichage de notre menu pousse les autres éléments vers le bas, nous allons plutôt appliquer une `position : absolute` au `div class="conteneur-nav"` créé pour cela ainsi qu'une largeur égale à 100%.

On va également en profiter pour supprimer les styles liés au pseudo-élément `::after`.

```
.conteneur-nav{
    position: absolute;
    width: 100%;
}
.sous{
    display: flex;
    flex-flow: column wrap;
    z-index: 1000;
}

.sous li{
    flex: 1 1 auto;
    text-align: left;
}
.sous a{
    padding: 10px;
    border-bottom: none;
    background-color: RGBa(200,200,200,0.8);
}
```

Voilà tout pour notre version mobile. Il n'y a plus qu'à s'occuper de la version bureau qu'on avait déjà concrètement créée lors du dernier exercice.

Nous allons utiliser ici les Media Queries et cibler en particulier les appareils qui ont un écran de taille supérieure à 980px. Pour ces écrans, on va vouloir que notre menu s'affiche de la même manière que lors de l'exercice précédent.

```
@media screen and (min-width: 980px){}
```

Pour minimiser la taille de notre code et économiser un maximum nos ressources, nous n'allons pas faire de copier-coller du menu précédent ici (ce qui dans tous les cas ne fonctionnerait pas bien car nous avons modifié la structure HTML du menu) mais ne préciser que les styles qui doivent changer.

Ici, nous allons déjà nous occuper des éléments qui ont été rajoutés pour le menu mobile. Nous allons passer une `position : static` à notre `div class="conteneur-nav"` afin qu'il n'interfère pas avec notre menu et un `display : none` à notre `label` et à notre `input`. On ne peut pas donner de `display : none` à notre `div` ici car celui-ci contient tout notre menu.

```css
@media screen and (min-width: 980px){
    .conteneur-nav{
        position: static;
    }
    nav label, nav input{
        display: none;
    }
```

Nous allons ensuite rendre à notre menu principal sa disposition en ligne. Ici, il faut également faire bien attention à gérer le cas où un utilisateur s'amuse à jouer avec la taille de la fenêtre et où la case à cocher a été cochée avant que le menu ne se transforme en sa version bureau. On va donc également appliquer un `flex-flow : row wrap` lorsque c'est le cas pour que notre menu s'affiche bien dans tous les cas.

```css
    nav input[type=checkbox]:checked + ul, nav ul{
        display: flex;
        flex-flow: row wrap;
        background-color: white;
    }
```

Aucun piège pour le reste du menu principal : on se contente de rétablir les styles utilisés dans l'exercice précédent avec notamment l'utilisation de la propriété `position` :

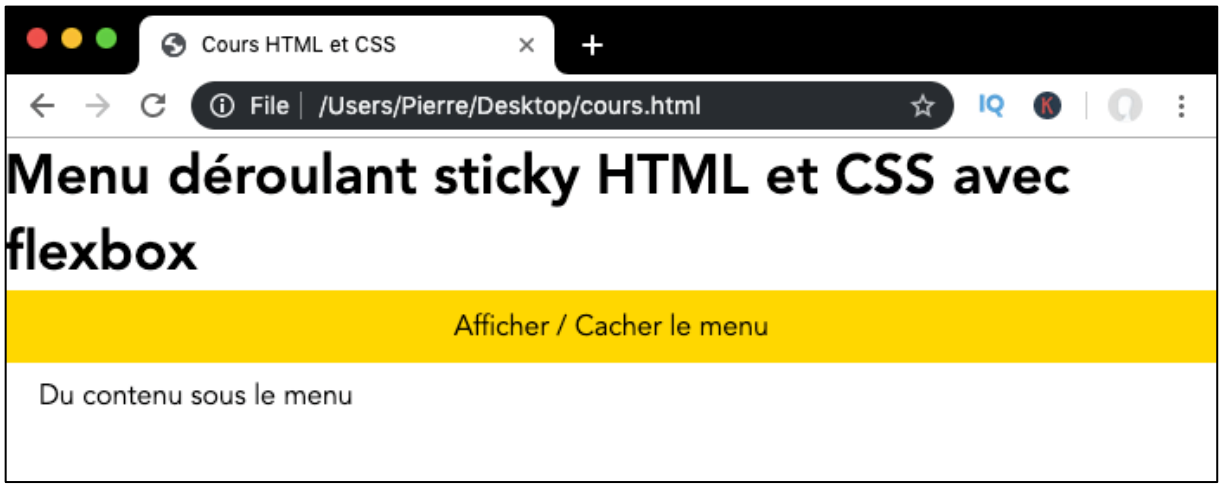
```css
    nav ul li{
        position: relative;
    }
    nav > div > ul > li > a{
        color: black;
    }
    nav a{
        border-bottom: 2px solid transparent;
    }
    nav a:hover{
        color: orange;
        border-bottom: 2px solid gold;
    }
```

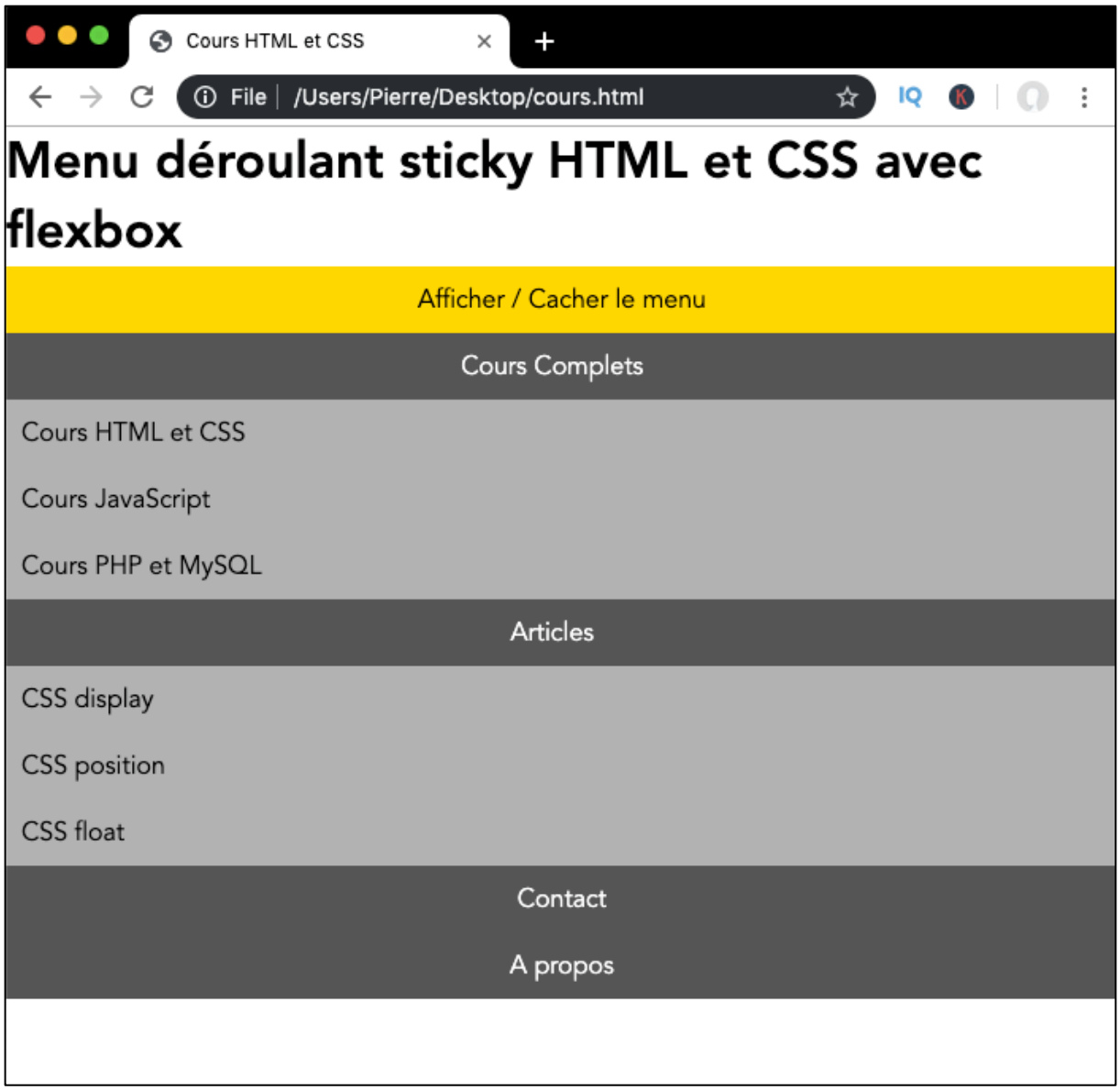
Le dernier petit « piège » va se trouver dans le sélecteur qui cible nos sous-menus pour les afficher en colonne lorsque l'utilisateur passe sur un élément du menu principal : on utilise le symbole `>` dans ce sélecteur qui va cibler les enfants directs.

Ici, il faut bien se souvenir qu'on a imbriqué un élément `div` entre notre `nav` et notre menu principal `ul`. Il faudra donc le mentionner au milieu de ce sélecteur.

```css
.sous{
    display: none;
    box-shadow: 0px 1px 2px #CCC;
    background-color: white;
    position: absolute;
    width: 100%;
}
nav > div > ul li:hover .sous{
    display: flex;
    flex-flow: column wrap;
}
.sous a{
    border-bottom: none;
    background-color: white;
}
.sous a:hover{
    border-bottom: none;
    background-color: RGBa(200,200,200,0.1);
}
.deroulant > a::after{
    content:" ▼";
    font-size: 12px;
}
```

Voilà, notre menu est cette fois-ci définitivement terminé et il va être compliqué d'aller plus loin avec le HTML et le CSS. Voici le résultat final :





Si cet exercice vous a semblé compliqué, encore une fois, ne vous inquiétez pas : c'est tout à fait normal ! Ce menu utilise plusieurs fonctionnalités parmi les plus avancées du HTML et du CSS et il est normal que vous n'ayez pas encore tout à fait assimilé comment toutes ces fonctionnalités peuvent fonctionner ensemble.

Encore une fois, l'assimilation et la compréhension ne pourra se faire complétement qu'en pratiquant régulièrement et en se confrontant aux difficultés et en cherchant toujours à comprendre les différents comportements obtenus.

PARTIE XV

Sémantique et éléments structurants

Sémantique et éléments structurants du HTML

Dans cette leçon, nous allons repréciser le rôle du HTML et insister sur l'importance d'utiliser toujours les éléments les plus cohérents pour définir nos différents contenus.

Nous allons également en profiter pour présenter différents éléments introduits avec le HTML5 et qui vont justement nous permettre de donner du sens aux différentes parties de nos pages ou de préciser la nature de certains contenus.

Le HTML et la sémantique

L'objectif principal du HTML est de structurer du contenu ou de lui donner du sens. En effet, les robots et programmes ne parlent pas la même langue que nous, ne possèdent pas nos yeux ni notre intelligence pour pouvoir distinguer des contenus et les comprendre par eux-mêmes.

C'est la raison pour laquelle nous utilisons le HTML puisque les différents robots et programmes vont très bien le comprendre.

Cela va ainsi permettre aux moteurs de recherche et aux navigateurs de « lire » nos pages et de « comprendre » ce que représente chaque type de contenu pour les afficher au mieux dans le cas d'un navigateur ou pour pouvoir les présenter à des gens effectuant une recherche sur un sujet que notre page traite dans le cas d'un moteur de recherche.

Grâce au HTML, nous allons pouvoir marquer nos différents contenus avec de nombreux éléments qui vont permettre de leur donner du sens. Utiliser les « bons » éléments à chaque fois et ceux qui font le plus de sens va ainsi être essentiel pour que notre page soit la plus lisible et la plus compréhensible possible pour les différents robots et programmes.

Le HTML5 nous offre justement une série de nouveaux éléments dit « structurants » et qui vont nous permettre de préciser le sens de certains contenus. Ces éléments sont dits « structurants » puisqu'ils ont une visée purement sémantique.

Certains de ces éléments vont notamment pouvoir être utilisés à la place des classiques `div` et `span` qui sont deux conteneurs génériques c'est-à-dire des éléments qui n'apportent aucun sens au contenu.

Liste et description des éléments structurants

Le HTML5 apporte les éléments structurants suivants :

Nom de l'élément	Description
header	Représente l'en-tête ou le haut d'une page

nav	Représente un menu de navigation dans une page
main	Représente le contenu principal de l'élément body
section	Représente une section dans une page, c'est-à-dire un groupement thématique de contenu
article	Représente une partie de page qui se suffit à elle-même, comme un post sur un blog par exemple
aside	Représente une partie de contenu non lié au reste
footer	Représente le pied d'une page

L'élément header

L'élément header représente l'en-tête de nos pages. Cet élément va généralement contenir le logo du site et le menu principal de navigation. Dans le cas d'un blog, si nous avons un article de blog qui est placé dans un élément article, on pourra également utiliser un élément header dans l'élément article et y placer les informations relatives à l'auteur, au temps de lecture, etc.

L'élément nav

L'élément nav représente un menu de navigation. Nous allons par exemple pouvoir utiliser cet élément pour entourer le menu de nos pages. Cela va permettre d'indiquer aux navigateurs et moteurs de recherche que notre élément ul (utilisé pour créer un menu) n'est pas qu'une liste mais bien un menu de navigation.

L'élément main

L'élément main représente le contenu principal ou « majoritaire » du body. La zone de contenu principale d'un document comprend un contenu unique à ce document et exclut le contenu répété sur un ensemble de documents, tels que des liens de navigation de site, des informations de copyright, des logos de site et des bannières.

La spécification officielle (HTML5.1) indique que l'élément main ne doit pas être imbriqué dans un autre élément structurant et qu'on ne doit utiliser qu'un élément main par page. Cependant, ces limitations sont levées dans le Living Standard (qui possède un peu d'avance sur la spéc officielle).

L'élément article

L'élément article représente une partie indépendante d'une page. Nous utiliserons cet élément pour entourer du contenu qui pourrait être extrait de notre page et distribué ailleurs tout en continuant à faire du sens.

Imaginez par exemple une page catégorie d'un blog qui affiche les différents articles de la catégorie. Chacun des différents articles n'a pas besoin des autres ou du reste de la page

pour faire du sens et pourrait tout à fait être extrait de la page et intégré sur un autre site et rester tout à fait lisible. Dans ce cas-là, il serait donc intéressant de placer chaque article dans un élément `article`.

L'élément section

L'élément `section` représente à nouveau une partie d'une page cohérente mais qui ne va pas forcément faire du sens si on l'extrait seule de la page pour l'intégrer ailleurs. L'élément `section` est à mi-chemin entre les éléments `article` et `div` et nous pourrons nous en servir pour organiser nos pages.

Un cas classique d'utilisation de `section` est le cas d'une page de présentation d'une entreprise. Sur ce type de pages, il y a généralement une partie dédiée à la présentation de l'activité de l'entreprise, une autre partie pour l'équipe, etc. Nous pourrons ici encadrer chaque partie avec un élément `section` pour les séparer les unes des autres.

Il va être tout à fait possible d'inclure un ou plusieurs éléments `section` dans un élément `article` si on souhaite diviser le contenu de notre article en plusieurs parties.

Notez que nous allons également pouvoir faire le contraire, c'est-à-dire placer un ou plusieurs éléments `article` dans un élément `section`. Pour des raisons de respect de la sémantique nous éviterons cependant d'imbriquer les éléments dans cet ordre.

L'élément aside

L'élément `aside` permet d'indiquer un contenu en marge du reste de la page, c'est-à-dire différent du reste de la page. Imaginons par exemple que nous ayons un blog et que chaque article de blog se trouve dans un élément `article`.

Dans nos articles, nous affichons des publicités. Les publicités n'ont aucune cohérence avec le contenu de l'article (ou tout au moins ne sont pas utiles à l'article). Nous pourrons donc les placer dans un élément `aside` pour le signaler aux moteurs de recherche et autres types de robots.

L'élément footer

L'élément `footer` va avoir un rôle et une signification sémantique identiques à l'élément `header` mais pour le pied de page. Comme son nom l'indique, nous utiliserons cet élément pour entourer les informations de pied de page (notice de copyright, menu de navigation bas, etc.).

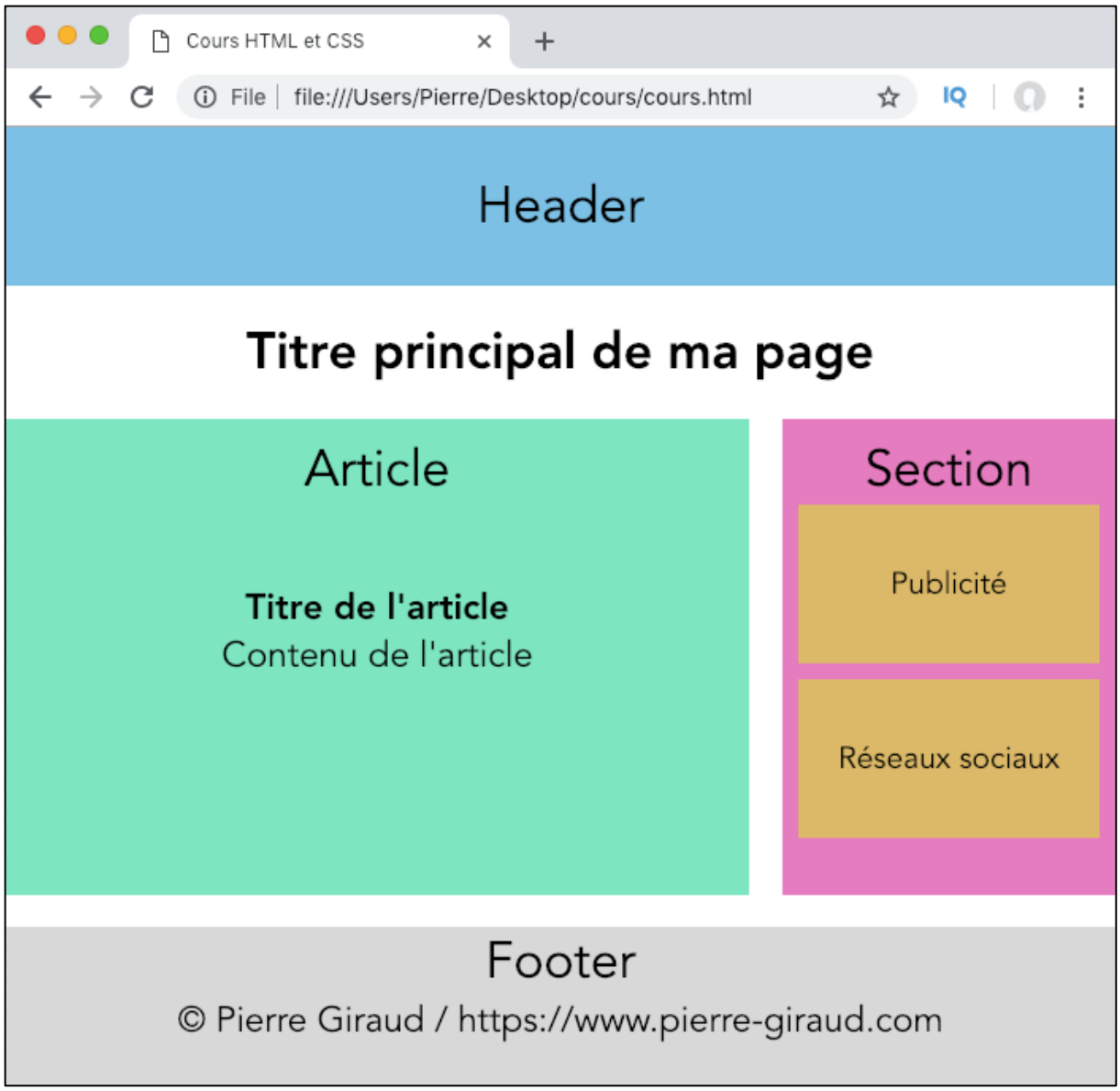
Utilisation des éléments HTML structurants

Nous allons donc en pratique utiliser nos éléments structurants pour mieux structurer nos pages et apporter du contexte et plus de sens aux différents objets dans nos pages.

Voici par exemple une utilisation des éléments structurants HTML. Le schéma que je vous propose ici est un schéma d'une page simple et relativement classique :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <meta name="viewport"
              content="width=device-width,initial-scale=1.0,user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <!--En tête de la page-->
        <header>
            <img src="#" alt="Mon logo">
            <nav><!--Menu de navigation principal--></nav>
        </header>

        <h1>Titre principal de ma page</h1>

        <!--Contenu qui se suffit à lui même-->
        <article>
            <header>
                <nav><!--Menu de navigation interne--></nav>
            </header>
            <article>
                <h1>Titre de l'article</h1>
                <p>Contenu de l'article</p>
            </article>
            <footer><!--Infos sur l'auteur, etc.--></footer>
        </article>

        <!--Autre contenu pas forcément pertinent hors contexte-->
        <section>
            <aside><!--Une publicité--></aside>
            <aside><!--Liens vers les réseaux sociaux--></aside>
        </section>

        <!--Pied de la page-->
        <footer>
            <p>© Pierre Giraud / https://www.pierre-giraud.com</p>
        </footer>
    </body>
</html>
```

Et voici une représentation imagée de la structure HTML ci-dessus (j'ai bien évidemment positionné mes éléments en CSS et leur ai donné des couleurs pour arriver à cela :



Autres éléments utilisés pour renforcer la sémantique d'une page

Les éléments suivants ne vont pas servir à « structurer » une page en soi mais vont nous permettre de donner du sens à certains types de contenus ou de données. Nous allons ici étudier les éléments suivants :

- `time` ;
- `address` ;
- `figure` ;
- `figcaption`.

L'élément time

L'élément time va pouvoir représenter soit une heure de la journée soit une date complète. Il faut savoir que les dates sont très difficiles à analyser pour les robots car la plupart des auteurs utilisent des syntaxes différentes dans l'écriture d'un moment de la journée ou d'une date.

Pour cette raison, nous allons ajouter un attribut datetime à l'élément time. Nous allons ensuite simplement passer une date en valeur de cet attribut en respectant le format yyyy-mm-dd hh:mmTIMEZONE.

La date devra être au format 24h et le timezone va être exprimé par rapport à GMT (on écrira donc +0100 pour Paris par exemple).

Cette date pourra être récupérée et affichée à côté de l'extrait dans la liste des résultats de Google par exemple.

L'élément address

L'élément address représente des informations de contact. On pourra par exemple l'utiliser pour encadrer l'adresse mail d'un auteur dans le cas d'un blog.

Les éléments figure et figcaption

Nous connaissons déjà les deux derniers éléments structurants qui sont les éléments figure qui représente une figure (photo, illustration, etc.) et figcaption qui va nous servir à ajouter une légende à un élément figure.

EXERCICE #6 : Création d'un site personnel monopage

Dans ce nouvel exercice, je vous propose de créer une page complète en HTML et en CSS qui sera une page de CV. Pour cela, nous allons utiliser les dernières notions que nous avons étudié ensemble comme le flexbox, le responsive design et les éléments HTML5 structurants.

Réalisation d'un projet en HTML et CSS : par où commencer ?

Lorsqu'on est face à un projet qui commence à être complexe, la difficulté principale est bien souvent de savoir par où commencer. Cela est d'autant plus vrai lorsqu'on n'a pas beaucoup d'expérience en développement.

La meilleure façon de faire, selon moi, est de commencer par réfléchir aux différents langages qu'on va devoir utiliser et d'identifier les points les plus complexes du projet. Ensuite, l'idée va être de découper le projet dans son ensemble en sous-projets ou en modules qu'on attaquera les uns après les autres.

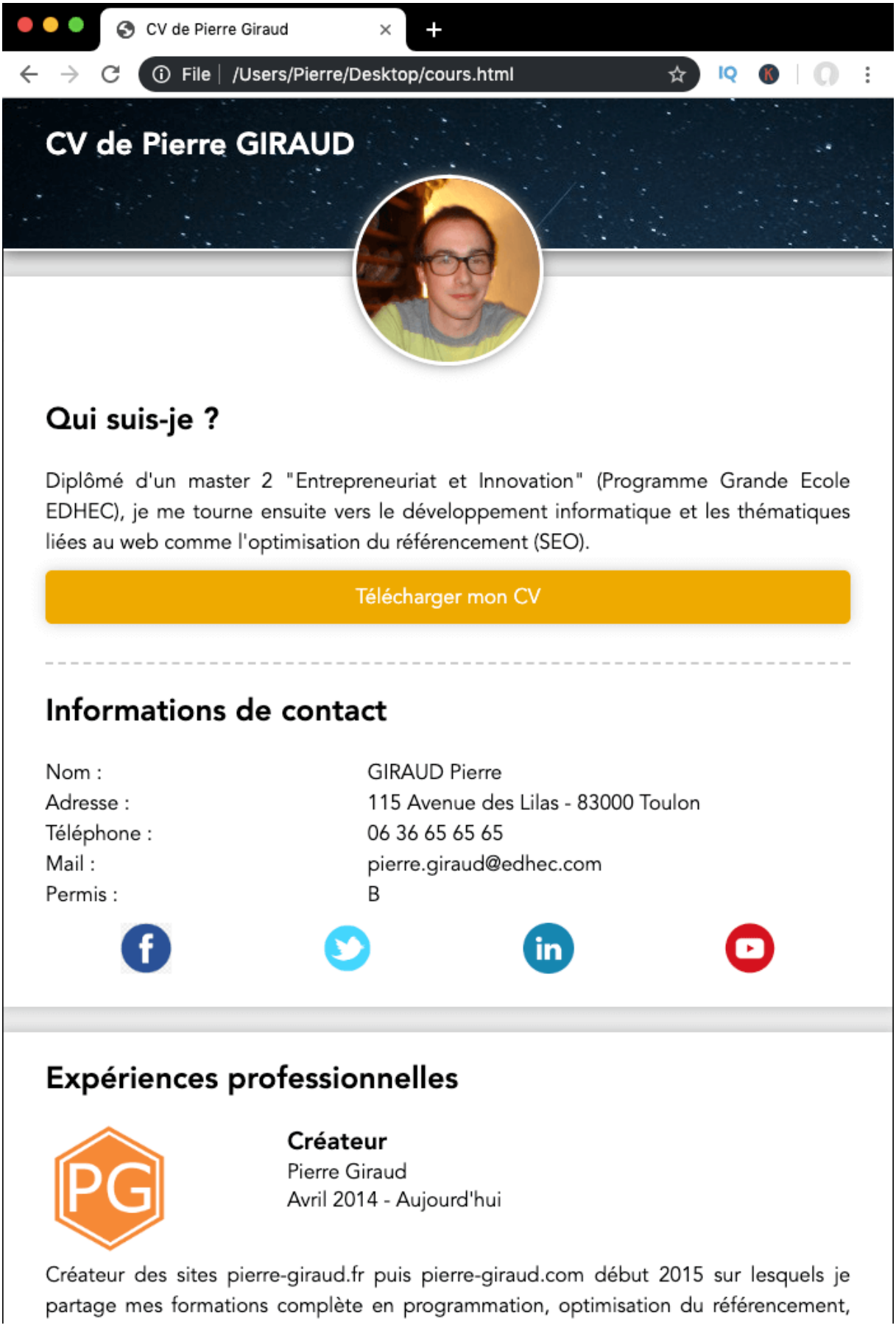
Dans le cas présent, cela reste relativement simple car nous n'avons qu'une page à créer et que nous nous contentons de n'utiliser que du HTML et du CSS. La réflexion principale va ici se faire sur le design général de la page.

Si on crée une page de CV, nous allons déjà vouloir insérer une photographie avec un paragraphe de présentation et des informations de contact.

Ensuite, nous voudrons présenter nos expériences professionnelles, puis notre formation académique, nos compétences et enfin finir avec nos centres d'intérêt.

Nous allons commencer avec la version mobile de notre CV qui sera la version standard. Pour créer cette version, nous allons utiliser le modèle des boites flexibles en choisissant une orientation en colonne principalement.

Voici le résultat auquel on souhaite parvenir :


CV de Pierre Giraud
File | /Users/Pierre/Desktop/cours.html
CV de Pierre GIRAUD
Qui suis-je ?
Diplômé d'un master 2 "Entrepreneuriat et Innovation" (Programme Grande Ecole EDHEC), je me tourne ensuite vers le développement informatique et les thématiques liées au web comme l'optimisation du référencement (SEO).
Télécharger mon CV
Informations de contact
Nom : GIRAUD Pierre
Adresse : 115 Avenue des Lilas - 83000 Toulon
Téléphone : 06 36 65 65 65
Mail : pierre.giraud@edhec.com
Permis : B
Expériences professionnelles
PG
Créateur
Pierre Giraud
Avril 2014 - Aujourd'hui
Créateur des sites pierre-giraud.fr puis pierre-giraud.com début 2015 sur lesquels je partage mes formations complète en programmation, optimisation du référencement,

etc.

Responsable SEO et contenu
LegalPlace
Novembre 2017 - Décembre 2018

Responsable SEO du site et plus globalement de la stratégie de production de contenu : recherche de mots clefs, réécriture, contrôle qualité du contenu publié, etc.

Stage marketing
PrestaShop
Mai 2013 - Novembre 2013

Contrôle de la qualité et de l'intégrité des modules proposés, modernisation de la marketplace & passerelle avec la communauté.

Formation

EDHEC Programme Grande Ecole
2011 - 2015

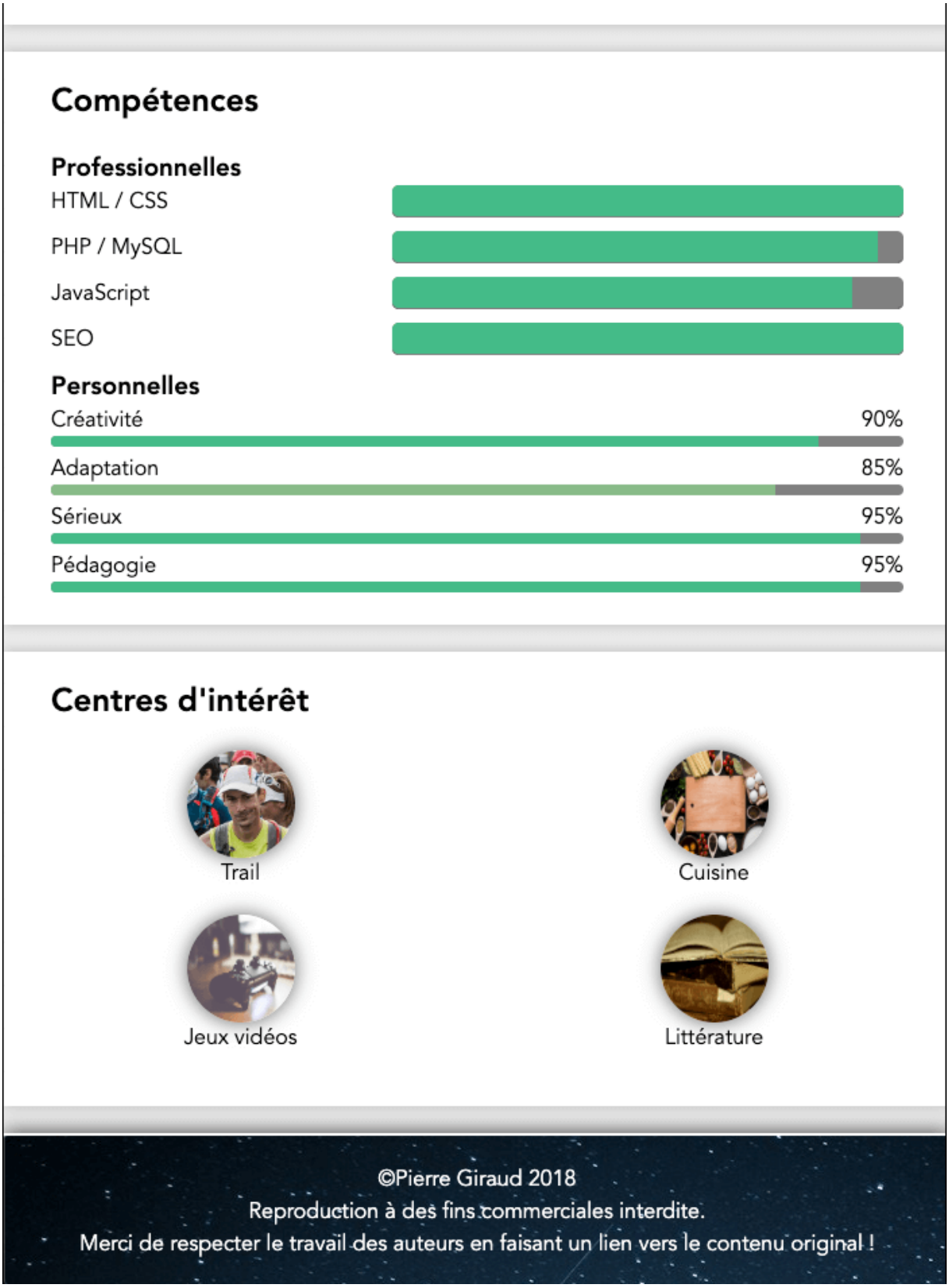
Programme grande école EDHEC, master 2 entrepreunariat et innovation

Prépa ECS option Maths
2009 - 2011

Classes préparatoires aux grandes écoles

Bac S option Maths
2005 - 2009

Bac scientifique avec options sport et mathématique



Structure HTML de notre page de CV

Comme d'habitude, nous commençons par créer la structure HTML avant de définir les styles particuliers en CSS.

Nous allons déjà ici commencer avec la structure minimale d'une page valide et allons également renseigner le `meta name="viewport"` et lier notre fichier HTML à un fichier CSS.

```
<!DOCTYPE html>
<html>
    <head>
        <title>CV de Pierre Giraud</title>
        <meta charset="utf-8">
        <meta name="viewport"
              content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cv.css">
    </head>
    <body>

    </body>
</html>
```

Ensuite, nous allons créer une zone d'en-tête pour notre page avec un élément `header` et allons simplement placer le titre principal de notre page de CV ici.

```
<!DOCTYPE html>
<html>
    <head>
        <title>CV de Pierre Giraud</title>
        <meta charset="utf-8">
        <meta name="viewport"
              content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cv.css">
    </head>
    <body>
        <header>
            <h1>CV de Pierre GIRAUD</h1>
        </header>

    </body>
</html>
```

Sous notre `header`, nous voulons créer une première `section` de présentation pour notre CV.

Cette section va contenir :

- Un bloc contenant une photo de profil ;
- Un bloc contenant un paragraphe de présentation et un lien de téléchargement du CV au format PDF ;
- Un bloc d'informations de contact et des liens vers nos réseaux sociaux.

Ici, je vous propose la structure suivante :

```
<section>
    <div class="photo">
        <img src="cv-profil.jpg" alt="Ma photo de profil">
    </div>
    <div class="prez">
        <h2>Qui suis-je ?</h2>
        <p>Diplômé d'un master 2 "Entrepreneuriat et Innovation" (Programme Grande
        Ecole EDHEC), je me tourne ensuite vers le développement informatique et les
        thématiques liées au web comme l'optimisation du référencement (SEO).</p>
        <a href="#" download>Télécharger mon CV</a>
    </div>
    <div class="contact">
        <h2>Informations de contact</h2>
        <div class="contact-flex">
            <p>Nom : </p>
            <p>GIRAUD Pierre</p>
        </div>
        <div class="contact-flex">
            <p>Adresse : </p>
            <p>115 Avenue des Lilas - 83000 Toulon</p>
        </div>
        <div class="contact-flex">
            <p>Téléphone :</p>
            <p>06 36 65 65 65</p>
        </div>
        <div class="contact-flex">
            <p>Mail : </p>
            <p><a href="mailto:pierre.giraud@edhec.com">pierre.giraud@edhec.com</a></p>
        </div>
        <div class="contact-flex">
            <p>Permis :</p>
            <p>B</p>
        </div>
        <div class="social">
            <a href="#"><img src="fb.png" alt="Logo Fb"></a>
            <a href="#"><img src="tw.png" alt="Logo Tw"></a>
            <a href="#"><img src="lk.png" alt="Logo Lk"></a>
            <a href="#"><img src="yt.png" alt="Logo Yt"></a>
        </div>
    </div>
</section>
```

On commence par créer un premier bloc avec un `div` dans lequel on place notre photo de profil. Ensuite, on déclare deux nouveaux `div` pour chacun de nos deux blocs de section pour pouvoir ensuite les mettre en forme.

Notre premier bloc de contenu possède une structure très simple : un titre, un paragraphe et un lien.

La structure de notre deuxième bloc est un peu plus complexe : on place les différentes informations de contact dans différents `div class="contact-flex"`. L'idée va être de créer des conteneurs flexibles pour aligner les éléments. On crée également un bloc avec nos icônes sociales qui seront cliquables (je laisse ici les liens vides).

A ce stade, je pense qu'il est important que vous notiez une chose : ici, je vous propose une structure en sachant que j'ai déjà réalisé l'exercice. En pratique, bien évidemment, on ne trouvera quasiment jamais la structure optimale du premier coup et serons souvent obligés de revenir sur le code HTML pour l'ajuster.

Pour éviter de trop nombreux allers-retours dans le code et de vous retrouver au final avec un code brouillon, je ne saurais que trop vous recommander de bien réfléchir à ce que vous souhaitez obtenir avant de commencer à coder.

Sous cette première section de présentation, nous allons avoir deux sections qui vont avoir une structure similaire : première une section décrivant nos expériences professionnelles et une seconde avec notre parcours scolaire.

Chacune de ces deux sections va être divisée en blocs, chaque bloc représentant une expérience. Chaque bloc expérience va suivre le même schéma : un logo, des informations « méta » par rapport à l'expérience et une description de celle-ci.

Je vous propose ici la structure suivante :

```html
<section>
    <h2>Expériences professionnelles</h2>
    <div class="exp">
        <div class="exp-logo">
            <a href="#"><img src="pierre.png" alt="Logo de Pierre Giraud"></a>
        </div>
        <div class="exp-info">
            <h3>Créateur</h3>
            <h4>Pierre Giraud</h4>
            <h4>Avril 2014 - Aujourd'hui</h4>
        </div>
        <div class="exp-desc">
            <p>Créateur des sites pierre-giraud.fr puis pierre-giraud.com
            début 2015 sur lesquels je partage mes formations complète en
            programmation, optimisation du référencement, etc.</p>
        </div>
    </div>
    <div class="exp">
        <div class="exp-logo">
            <a href="#"><img src="legalplace.png" alt="Logo de LegalPlace"></a>
        </div>
        <div class="exp-info">
            <h3>Responsable SEO et contenu</h3>
            <h4>LegalPlace</h4>
            <h4>Novembre 2017 - Décembre 2018</h4>
        </div>
        <div class="exp-desc">
            <p>Responsable SEO du site et plus globalement de la stratégie
            de production de contenu : recherche de mots clefs, réécriture,
            contrôle qualité du contenu publié, etc.</p>
        </div>
    </div>
    <div class="exp">
        <div class="exp-logo">
            <a href="#"><img src="prestashop.png" alt="Logo de PrestaShop"></a>
        </div>
        <div class="exp-info">
            <h3>Stage marketing</h3>
            <h4>PrestaShop</h4>
            <h4>Mai 2013 - Novembre 2013</h4>
        </div>
        <div class="exp-desc">
            <p>Contrôle de la qualité et de l'intégrité des modules proposés,
            modernisation de la marketplace & passerelle avec la communauté.</p>
        </div>
    </div>
</section>
```

```
<section>
    <h2>Formation</h2>
    <div class="exp">
        <div class="exp-logo">
            <a href="#"><img src="edhec.png" alt="Logo EDHEC"></a>
        </div>
        <div class="exp-info">
            <h3>EDHEC Programme Grande Ecole</h3>
            <h4>2011 - 2015</h4>
        </div>
        <div class="exp-desc">
            <p>blablabla</p>
        </div>
    </div>
    <div class="exp">
        <div class="exp-logo">
            <a href="#"><img src="dd.png" alt="Logo Dumont"></a>
        </div>
        <div class="exp-info">
            <h3>Prépa ECS option Maths</h3>
            <h4>2009 - 2011</h4>
        </div>
        <div class="exp-desc">
            <p>blablabla</p>
        </div>
    </div>
    <div class="exp">
        <div class="exp-logo">
            <a href=""><img src="dd.png" alt=""></a>
        </div>
        <div class="exp-info">
            <h3>Bac S option Maths</h3>
            <h4>2005 - 2009</h4>
        </div>
        <div class="exp-desc">
            <p>blablabla</p>
        </div>
    </div>
</section>
```

Pour notre quatrième section présentant nos compétences, nous allons utiliser un système de barres plus ou moins remplies qui vont indiquer notre pourcentage de maitrise de la compétence en question. Ici, je vais vous proposer deux designs différents en CSS.

Dans tous les cas, la structure de cette section va ressembler à notre bloc d'informations de contact de notre première section.

```
<section>
    <h2>Compétences</h2>
    <h3 class="h3gauche">Professionnelles</h3>
    <div class="comp">
        <p>HTML / CSS</p>
        <div class="conteneur-barre"><span class="barre c100"></span></div>
    </div>
    <div class="comp">
        <p>PHP / MySQL</p>
        <div class="conteneur-barre"><span class="barre c95"></span></div>
    </div>
    <div class="comp">
        <p>JavaScript</p>
        <div class="conteneur-barre"><span class="barre c90"></span></div>
    </div>
    <div class="comp">
        <p>SEO</p>
        <div class="conteneur-barre"><span class="barre c100"></span></div>
    </div>
    <h3 class="h3gauche">Personnelles</h3>
    <div class="comp2">
        <p>Créativité</p>
        <p>90%</p>
        <div class="conteneur-barre2"><span class="barre c90"></span></div>
    </div>
    <div class="comp2">
        <p>Adaptation</p>
        <p>85%</p>
        <div class="conteneur-barre2"><span class="barre c85"></span></div>
    </div>
    <div class="comp2">
        <p>Sérieux</p>
        <p>95%</p>
        <div class="conteneur-barre2"><span class="barre c95"></span></div>
    </div>
    <div class="comp2">
        <p>Pédagogie</p>
        <p>95%</p>
        <div class="conteneur-barre2"><span class="barre c95"></span></div>
    </div>
</section>
```

Enfin, notre dernière section va nous permettre de présenter nos centres d'intérêt à travers des vignettes.

```
<section>
    <h2>Centres d'intérêt</h2>
    <figure class="interet">
        <img src="trail.png" alt="Trail">
        <figcaption>Trail</figcaption>
    </figure>
    <figure class="interet">
        <img src="cuisine.png" alt="Cuisine">
        <figcaption>Cuisine</figcaption>
    </figure>
    <figure class="interet">
        <img src="gaming.png" alt="Gaming">
        <figcaption>Jeux vidéos</figcaption>
    </figure>
    <figure class="interet">
        <img src="livre.png" alt="Littérature">
        <figcaption>Littérature</figcaption>
    </figure>
</section>
```

Je vais également en profiter pour rajouter un pied de page à mon CV pour avoir un effet de symétrie avec l'en-tête.

```
    <footer>
        <p><a href="https://www.pierre-giraud.com">©Pierre Giraud</a> 2018</p>
        <p>Reproduction à des fins commerciales interdite.</p>
        <p>Merci de respecter le travail des auteurs en faisant un lien vers
        le contenu original !</p>
    </footer>
</body>
```

Styles CSS de la version mobile de notre CV

Comme on l'a décidé précédemment, nous allons commencer par créer les styles de la version mobile de notre CV qui seront donc les styles standards puis allons ensuite utiliser les Media Queries pour les styles de la version bureau.

Nous allons déjà ici nous occuper d'effectuer un reset CSS des marges internes et externes sur tous nos éléments pour avoir le même comportement dans tous les navigateurs et pouvoir les positionner précisément ensuite.

Nous allons également en profiter pour définir quelques styles généraux comme la police d'écriture, l'alignement par défaut des textes, la façon dont va être calculée la taille des éléments ou encore l'affichage des éléments de lien.

```css
/*STYLES GENERAUX*/
*{
    font-family: Avenir, sans-serif;
    padding: 0px;
    margin: 0px;
    text-align: justify;
    box-sizing: border-box;
}
body{
    background: #ededed;
}
p{
    font-size: 1em;
    line-height: 1.5em;
}
a{
    text-decoration: none;
    color: #000;
}
```

Ensuite, nous allons nous occuper de régler les styles du `header` et du `footer` qui sont les parties les plus simples de notre CV.

En version mobile, on va vouloir que le CV s'affiche en pleine page, c'est-à-dire que les différents blocs occupent tout l'espace en largeur et allons ensuite centrer les éléments à l'intérieur de façon à ce qu'on ait un espace vide égal à 5% de la largeur totale de la page de chaque côté.

Pour faire cela, on va demander aux éléments dans nos blocs de n'occuper que 90% de la largeur des blocs qui eux occuperont 100% de la largeur disponible et allons centrer les éléments dans les blocs.

```css
/*HEADER & FOOTER*/
header, footer{
    width: 100%;
    padding: 20px 0px;
    background: url("cv-cover.jpg") #4f4f4f;
    box-shadow: 0px 0px 15px #333;
}
header{
    border-bottom: 2px solid #fff;
    height: 120px;
}
header h1{
    color: white;
    font-size: 1.5em;
    width: 90%;
    margin: 0 auto;
}

footer{
    border-top: 2px solid #fff;
    background-image: url("cv-cover.jpg");
}
footer p{
    width: 90%;
    margin: 0 auto;
    color: #fff;
    text-align: center;
}
footer a{
    color: #fff;
}
```

Nous allons ensuite définir les styles généraux communs à toutes nos sections. Nos éléments `section` vont tous être des conteneurs flexibles et nous allons choisir l'axe horizontal comme axe principal.

Nous allons également aligner les éléments au centre selon l'axe principal (l'axe horizontal donc) avec `justify-content : center`. Ensuite, on va définir des styles purement esthétiques pour nos sections : couleur de fond, ombres, etc.

```css
/*SECTIONS*/
section{
    display: flex;
    flex-flow: row wrap;
    justify-content: center;
    width: 100%;
    padding-bottom: 20px;
    margin: 20px auto;
    box-shadow: 0px 0px 10px #bbb;
    background-color: #fff;
}

section h2{
    margin: 20px 0;
    width: 90%;
}
section > div{
    width: 90%;
}
```

Nos éléments section sont des conteneurs flexibles. Cela signifie que tous leurs enfants directs sont des éléments flexibles. Pour notre première section, nos 3 div class="photo-profil", div class="prez" et div class="contact" sont donc des éléments flexibles.

Ces trois blocs occupent par défaut 90% de l'espace disponible en largeur dans le conteneur avec width: 90% tant qu'aucun flex-basis n'est défini. De plus, ils sont alignés au centre selon leur axe principal qui est l'axe horizontal grâce à justify-content : center.

Il va suffire d'ajouter un text-align : center à notre premier div pour centrer notre photo dans la section.

D'un point de vue esthétique, on va régler les dimensions de la photo sur 150*150px et allons lui attribuer une bordure blanche ainsi et allons créer un effet d'ombre autour. Ensuite, on applique un border-radius égal à la moitié de ses dimensions, ce qui aura pour effet « d'arrondir » notre image.

Ici, nous allons également vouloir que notre image se trouve à cheval entre notre header et notre première section. Pour réaliser cela de manière simple, nous allons devoir utiliser un petit trick CSS qui consiste à attribuer une marge haute négative à l'image pour la faire sortir de son conteneur.

```css
.photo{
    text-align: center;
}
.photo img{
    width: 150px;
    height: 150px;
    border: 3px solid #fff;
    box-shadow: 0px 0px 10px #777;
    border-radius: 50%;
    margin-top: -80px;
}
```

Pour notre premier bloc de présentation, nous allons nous contenter d'ajouter une bordure basse et de mettre en forme notre lien de téléchargement.

```
.prez{
    border-bottom: 2px dashed #ccc;
}
.prez a{
    display: block;
    text-align: center;
    padding: 10px;
    margin: 10px auto 30px auto;
    color: #fff;
    background-color: #EA0;
    box-shadow: 0px 0px 10px #ccc;
    border-radius: 5px;
}
```

Notre deuxième bloc contenant nos informations de contact va subir une mise en forme un peu plus complexe. L'idée ici va être d'aligner les informations de contact les unes sous les autres.

Pour cela, on va appliquer un `display : flex` à nos éléments portant la classe `contact-flex` en définissant à nouveau l'axe horizontal comme axe principal. On utilise `align-items: center` pour centrer les éléments en hauteur au cas où certains prendraient plusieurs lignes.

On cible ensuite le premier paragraphe de chaque conteneur flexible avec `.contact-flex p:first-child` pour lui attribuer une largeur de base égale à 40% du conteneur flexible et le dernier avec `.contact-flex p:last-child` pour lui attribuer une largeur de base égale à 60% du conteneur flexible. Cela va permettre aux différents couples d'informations d'être bien alignés les uns sous les autres.

```
.contact-flex{
    display: flex;
    flex-flow: row wrap;
    align-items: center;
}
.contact-flex p:first-child{
    flex: 0 0 40%;
}
.contact-flex p:last-child{
    flex: 0 0 60%;
}
```

On transforme également notre conteneur `div class="social"` en conteneur flexible. L'axe principal est par défaut l'axe horizontal. On choisit de repartir l'espace libre selon l'axe principal dans le conteneur équitablement entre les différents éléments avec `justify-content: space-around`.

Ici, on règle manuellement la taille des différents éléments flexibles en définissant explicitement une hauteur et une largeur mais pas de `flex-basis`.

```css
.social{
    display: flex;
    justify-content: space-around;
    margin: 10px auto 0px auto;
}
.social img{
    width: 40px;
    height: 40px;
}
```

Nos deuxième et troisième sections de CV vont recevoir la même mise en forme. Ici, chaque expérience est composée de trois blocs : une image, des informations importantes et une description.
Nous allons transformer nos conteneurs `div class="exp"` qui contiennent une expérience chacun en conteneurs flexibles et allons définir l'axe horizontal comme axe principal.

Nous allons ensuite demander à notre premier élément flexible qui est notre conteneur d'image de n'occuper que 25% de la largeur du conteneur jusqu'à 100px. On règle également la largeur de notre image en soi à 100% de la taille du conteneur.
Notre deuxième élément flexible (`exp-info`) occupera lui 70% de la largeur et on définit une marge à gauche `auto` ce qui signifie que la marge va absorber l'espace restant dans le conteneur.

Notre dernier élément flexible va occuper par défaut tout l'espace disponible.
On va ensuite également mettre en forme l'intérieur de nos `div class="exp-info"`. Pour cela, on les transforme également en conteneur flexible et on choisit l'axe vertical comme axe principal. Cela va avoir pour effet d'afficher les différentes informations en colonne.

```css
.exp{
    display: flex;
    flex-flow: row wrap;
    border-bottom: 1px solid #bbb;
    padding-bottom: 10px;
    margin-bottom: 10px;
}
.exp-info{
    display: flex;
    flex-flow: column wrap;
}
.exp-logo{
    flex: 0 0 25%;
    max-width: 100px;
}
.exp-info{
    flex: 0 0 70%;
    margin-left: auto;
}
.exp img{
    width: 100%;
}
.exp h3{
    font-size:  1.2em;
}
.exp h4{
    font-size:  1em;
    font-weight: normal;
}
```

Pour la section compétence de notre CV HTML et CSS, je vais vous proposer deux designs différents. Nous allons appliquer le premier design au bloc de compétences professionnelles et le deuxième design au bloc de compétences personnelles.

Le niveau de maitrise de nos compétences va être représenté par une barre plus ou moins remplie et plus ou moins verte. Pour nos compétences professionnelles, nous allons vouloir placer les barres à côté des compétences tandis que pour les compétences personnelles nous allons placer les barres en dessous.

On va donc commencer par transformer nos `div class="comp"` en conteneurs flexibles et définir l'axe horizontal comme axe principal.

Ensuite, on va vouloir que les textes décrivant les compétences occupent 40% de l'espace pour les compétences professionnelles et que le conteneur de barre occupe 60% de l'espace du conteneur.

```css
.comp{
    display:flex;
    flex-flow: row wrap;
    margin-bottom: 10px;
}
.comp p{
    flex: 0 0 40%;
}
.conteneur-barre{
    flex: 0 0 60%;
    border-radius: 5px;
    background-color: grey;
    border-bottom: 1px ridge grey;
}
```

Ensuite, nous allons devoir créer notre barre en soi. Pour cela, on utilise un `display : block` pour nos `span` et une hauteur de 100% afin que les barres aient bien la même hauteur que les paragraphes.

On applique finalement la couleur sur une certaine largeur du conteneur grâce aux classes `c85`, `c90`, etc. Pour les barres, on pense bien à répliquer le `border-radius` du conteneur lorsqu'elles touchent un angle du conteneur afin d'avoir le meilleur rendu possible.

```css
.barre{
    display: block;
    height: 100%;
    border-top-left-radius: 5px;
    border-bottom-left-radius: 5px;
}
.c85{
    width: 85%;
    background-color: #8B8;
}
.c90{
    width: 90%;
    background-color: #4B8;
}
.c95{
    width: 95%;
    background-color: #4B8;
}
.c100{
    width: 100%;
    background-color: #4B8;
    border-radius: 5px;
}
```

On va également appliquer un `display : flex` aux conteneurs de nos compétences personnelles, toujours en choisissant l'axe horizontal comme axe principal et en laissant la possibilité aux éléments flexibles de passer à la ligne.

Ici, on utilise également `justify-content: space-between` pour coller le texte de la compétence contre un bord du conteneur et le pourcentage d'acquisition contre le bord opposé.

On passe un `flex: 0 0 100%` à nos conteneurs de barre afin qu'ils occupent tout l'espace disponible et passe de fait à la ligne. Ici, on veut créer une barre moins épaisse que précédemment. On définit donc la hauteur du conteneur comme étant égale à la moitié de la hauteur de nos paragraphes. On réutilise ensuite les barres créées précédemment.

```
.comp2{
    display: flex;
    flex-flow: row wrap;
    justify-content: space-between;
}
.conteneur-barre2{
    flex: 0 0 100%;
    height: 0.5em;
    margin-bottom: 0.25em;
    border-radius: 5px;
    background-color: grey;
}
```

Pour la dernière section de notre CV présentant nos centres d'intérêt, nous allons vouloir afficher des vignettes avec une légende en dessous et afficher deux vignettes par ligne en centrant le tout.

On va un `flex: 0 1 50%` à nos différents éléments flexibles `figure` et également les transformer en conteneur flex en choisissant cette fois-ci l'axe vertical comme axe principal et en centrant les éléments selon leur axe secondaire, c'est-à-dire horizontalement.

Finalement, on définit une taille fixe pour nos différentes vignettes ainsi qu'un `border-radius : 50%` afin que nos images apparaissent comme rondes.

```
.interet{
    flex: 0 1 50%;
    display: flex;
    flex-flow: column wrap;
    align-items: center;
    margin-bottom: 20px;
}
.interet img{
    width: 80px;
    height: 80px;
    border-radius: 50%;
    box-shadow: 0px 0px 15px #555;
}
```

Styles CSS de la version bureau de notre CV

Pour la version bureau de notre CV HTML et CSS, nous allons pouvoir nous permettre de placer certains éléments de section côte-côte plutôt que les uns en dessous des autres.

Pour créer cette version bureau, nous allons utiliser la règle CSS `@media screen and (min-width: 980px){}`. Les styles définis dans cette règle vont donc s'appliquer pour tous les appareils disposant d'un écran et lorsque la fenêtre est plus large que 980px.

Ici, on va déjà commencer par modifier la largeur de nos sections afin qu'elles n'occupent que 80% de la largeur de la page. On en profite également pour aligner notre titre principal sur la nouvelle largeur des sections et pour définir une couleur pour les liens au survol.

```
@media screen and (min-width: 980px){
    section{
        width: 80%;
        box-shadow: 0px 0px 10px #bbb;
    }
    a:hover{
        color: #EA0;
    }
    header h1{
        width: 80%;
    }
```

L'axe principal de nos sections est encore l'axe horizontal. Notre photo est toujours centrée car le `div` qui la contient occupe 90% de la largeur de la `section`, qu'il est lui-même centré et qu'on lui a appliqué un `text-align : center`.

Nous allons cette fois-ci vouloir que les deux blocs de contenu de la première section de notre CV s'affichent côte-à-côte. On définit donc un `flex: 0 0 45%` pour chacun et des marges gauche pour le premier et droite pour le second automatique afin qu'elles absorbent l'espace restant. Ici, le `flex-basis` défini va être prioritaire que la largeur des `div`.

On remplace également la bordure horizontale de séparation de nos deux blocs par une bordure verticale entre les deux et on en profite pour appliquer des styles à notre lien de téléchargement et notamment au survol de la souris.

```css
.prez, .contact{
    flex: 0 0 45%;
}
.prez{
    border-bottom: none;
    border-right: 2px solid #ccc;
    padding-right: 20px;
    margin-left: auto;
}
.contact{
    padding-left: 20px;
    margin-right: auto;
}
.prez a{
    border: 2px solid transparent;
}
.prez a:hover{
    color: #f28835;
    background-color: #fff;
    border: 2px solid #f28835;
    box-shadow: 0px 0px 20px #666;
}
```

Pour nos deux sections suivantes, on se contente de réduire la taille de base des logos dans la section et d'augmenter la taille prise pas les informations. On en profite également pour cette fois-ci aligner la description sous les informations de chaque expérience.

```css
.exp-logo{
    flex: 0 0 10%;
}
.exp-info, .exp-desc{
    flex: 0 0 85%;
    margin-left: auto;
}
```

On ne touche pas au design de la section compétence qui nous convient très bien en l'état.

Finalement, on place chacun des blocs représentant nos centres d'intérêt côte-à-côte en définissant un `flex: 0 1 25%` pour chacun de nos éléments `figure`.

```css
.interet{
    flex: 0 1 25%;
}
```

PARTIE XVI

Modèle des grilles

Introduction aux grilles CSS

Dans cette nouvelle partie, nous allons étudier un dernier modèle de disposition : le modèle des grilles.

Implémenté en 2017, le modèle des grilles est le modèle de disposition le plus récent et également le plus puissant que nous allons pouvoir utiliser en CSS.

La plupart des propriétés qu'on va pouvoir utiliser avec ce modèle vont ressembler aux propriétés des boites flexibles car ces deux modèles possèdent des principes de base communs.

Première définition du modèle des grilles et différences avec le flexbox

Le modèle des grilles est un modèle bidimensionnel, ce qui signifie que c'est un modèle qui va nous permettre de placer nos éléments en fonction de deux axes.

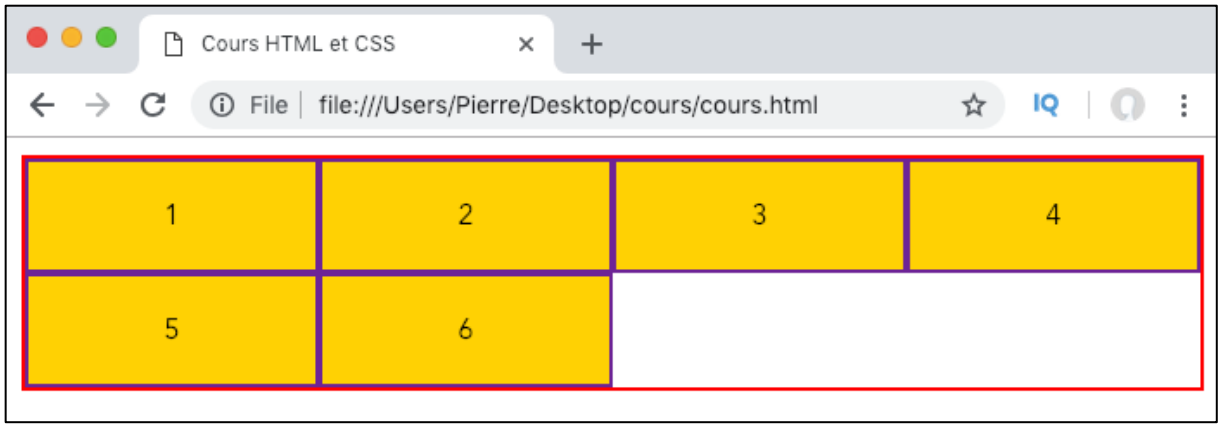
A la différence du modèle des boites flexibles, il n'est plus question ici d'axe principal et d'axe secondaire. Avec le modèle des grilles, les deux axes peuvent être manipulés de la même façon et vont être définis de cette manière :

- Un axe de bloc qu'on pourra également appeler (pour simplifier) axe vertical ou axe des colonnes ;
- Un axe en ligne ou axe horizontal ou encore axe des rangées.

Le modèle des grilles va donc s'avérer encore plus puissant que le modèle des boites flexibles qui était principalement unidimensionnel. Pour illustrer cela, imaginons la situation suivante :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-flex">
            <div>1</div>
            <div>2</div>
            <div>3</div>
            <div>4</div>
            <div>5</div>
            <div>6</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}

.conteneur-flex{
  display: flex;
  flex-flow: row wrap;
  margin: 10px;
  border: 2px solid red;
}
.conteneur-flex > div{
  flex: 0 0 25%;
  padding: 20px 0px;
  background-color: gold;
  border: 2px solid RGB(120,40,160);
}
```



Ici, j'ai défini un conteneur flexible dont l'axe principal est l'axe horizontal et qui possède 6 éléments flexibles. On définit une taille de base de 25% pour les éléments flexibles et on ne leur laisse pas la possibilité de s'agrandir ou dé rétrécir. En revanche, on donne le droit aux éléments d'aller à la ligne si nécessaire.

Nos éléments vont donc se placer les uns à côté des autres en partant du début du conteneur puis passer à la ligne. Avec les boites flexibles, il faut bien comprendre que chaque ligne agit comme un conteneur flexible indépendant.

Maintenant, imaginons que je souhaite aligner mon sixième élément flexible dans la ligne. Cela va être impossible avec le flexbox puisqu'il n'existe pas de propriété comme `justify-self` dans ce modèle.

Les grilles ne vont pas posséder cette limitation et nous permettre au contraire de définir l'alignement de chaque élément de grille selon l'axe de bloc et l'axe en ligne.

Autre limitation du flexbox par rapport aux grilles : le modèle des boites flexibles ne possède pas au jour d'aujourd'hui de propriété nous permettant de définir la taille des gouttières d'un élément.

Une gouttière est l'équivalent d'une marge mais uniquement entre deux éléments flexibles ou de grille et non pas entre un élément et son conteneur.

Avec le flexbox, on était donc obligé d'utiliser la propriété `margin` pour espacer les éléments flexibles les uns des autres puis d'utiliser des marges négatives sur le conteneur pour supprimer les marges créées entre le conteneur et les éléments.

Non seulement cette solution n'est pas optimale d'un point de vue propreté du code mais en plus le fait de rajouter des marges aux éléments flexibles risque de poser des problèmes dans la disposition des éléments puisque les marges externes vont venir s'ajouter à la taille des éléments.

Les grilles possèdent elles un ensemble de propriétés qui vont nous permettre de définir les gouttières des éléments d'une bien meilleure façon.

Vous pouvez donc retenir l'idée suivante pour définir s'il est préférable d'utiliser le flexbox ou les grilles : si vous avez besoin de contrôler la disposition des éléments selon les deux axes ou si vous avez besoin d'espacer précisément les différents éléments, alors utilisez les grilles. Sinon, utilisez le flexbox.

Notez par ailleurs que nous allons tout à fait pouvoir utiliser ces deux modèles conjointement et créer des dispositions de page complexes, un créant des éléments de grille qui vont contenir des éléments flexibles par exemple.

Élément grille conteneur et éléments de grille

Pour définir une grille, nous allons devoir appliquer un `display : grid` (la grille sera de type `block`) ou un `display : inline-grid` (la grille sera de niveau `inline`) à un élément.

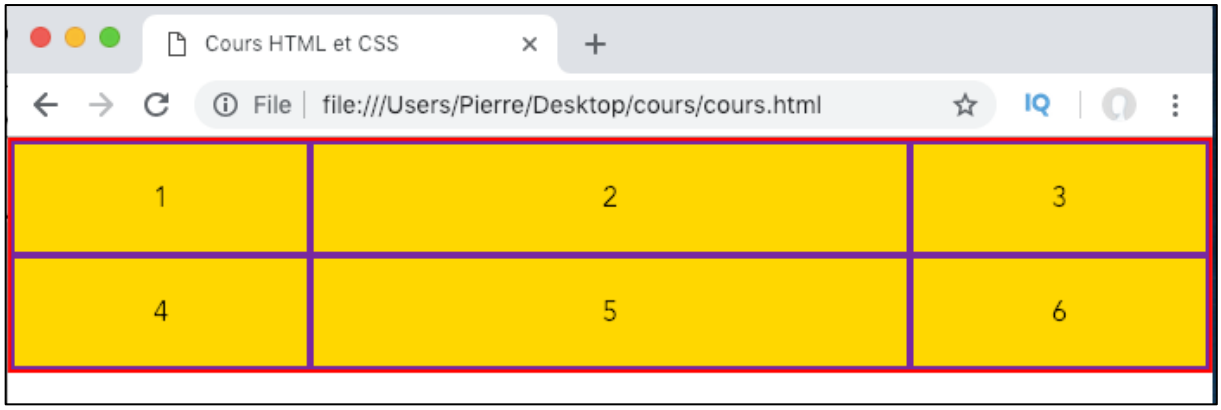
L'élément auquel on applique un `display : grid` ou `display : inline- grid` va automatiquement devenir un élément grille conteneur.

De manière similaire au modèle des boites flexibles, tous les enfants directs de notre élément grille conteneur (et seulement les enfants directs) vont automatiquement être des éléments de grille.

Voici ci-dessous une première grille :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid">
            <div>1</div>
            <div>2</div>
            <div>3</div>
            <div>4</div>
            <div>5</div>
            <div>6</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  border: 2px solid red;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: gold;
  border: 2px solid RGB(120,40,160);
}
```



Ici, nous définissons un conteneur de grille avec `display : grid`. Les enfants directs du conteneur vont ainsi automatiquement devenir des éléments de grille. Ensuite, je définis

les colonnes de ma grille avec `grid-template-columns`. N'essayez pas de tout comprendre tout de suite, nous aurons l'occasion d'étudier cette propriété par la suite.

Le vocabulaire des grilles

Les grilles sont des structures relativement complexes et il est donc essentiel de définir précisément les différentes parties d'une grille que nous allons être amenés à manipuler ou qui vont pouvoir nous être utiles.

Le premier concept à comprendre est qu'une grille est constituée de lignes ou de « droites » horizontales et verticales. Ces lignes ne sont pas visibles à l'écran et il faut donc se les imaginer.

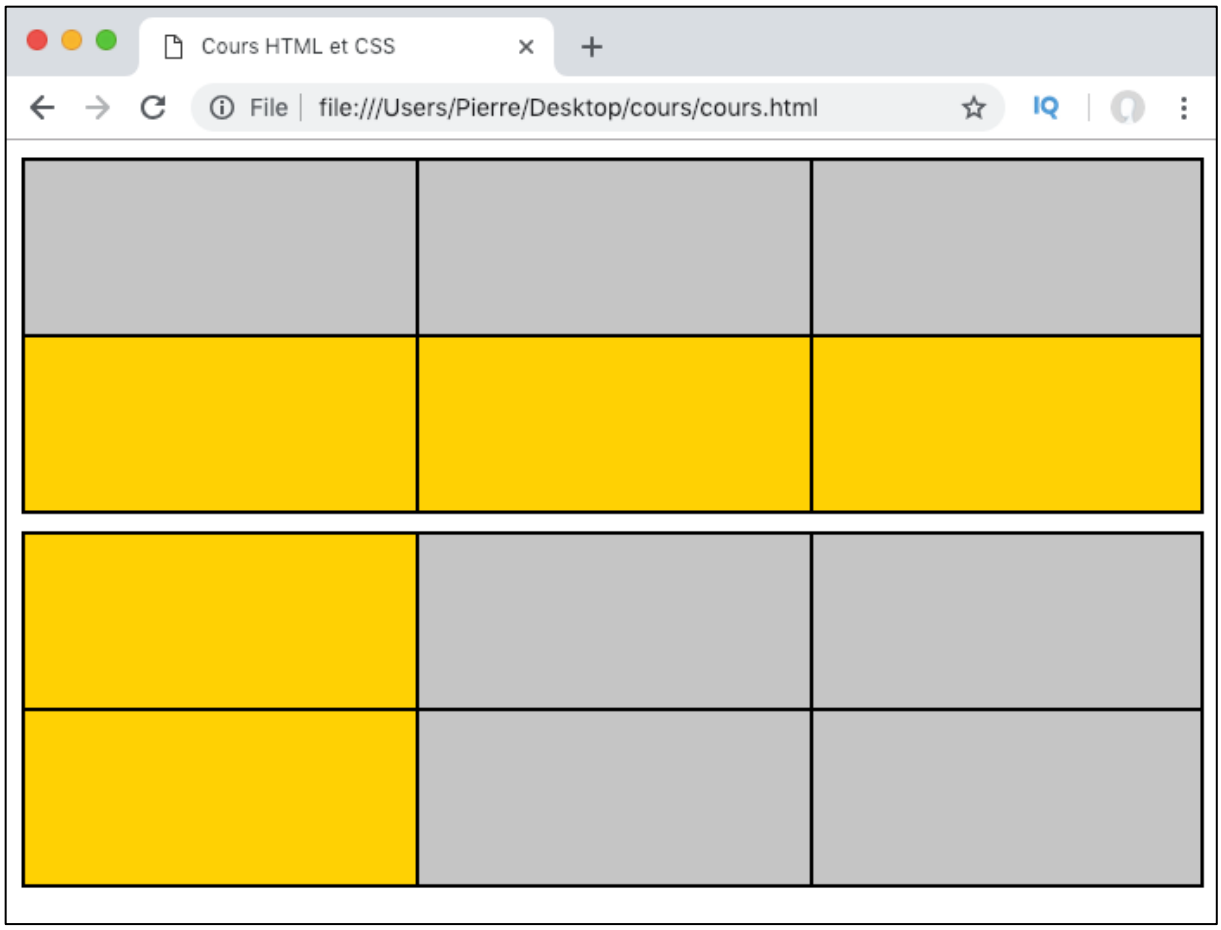
Ces lignes vont être disposées de chaque côté des colonnes et des rangées d'une grille.

Note : Pour les grilles, nous parlerons de « rangées » pour définir les « rows » en anglais et non pas de lignes afin de ne pas les confondre avec les lignes que nous venons de définir ci-dessus.

Une grille possédant 3 colonnes et 2 rangées va donc posséder 4 lignes verticales et 3 lignes horizontales, une grille possédant 4 colonnes et 4 rangées va posséder 5 lignes verticales et 5 lignes horizontales et etc.

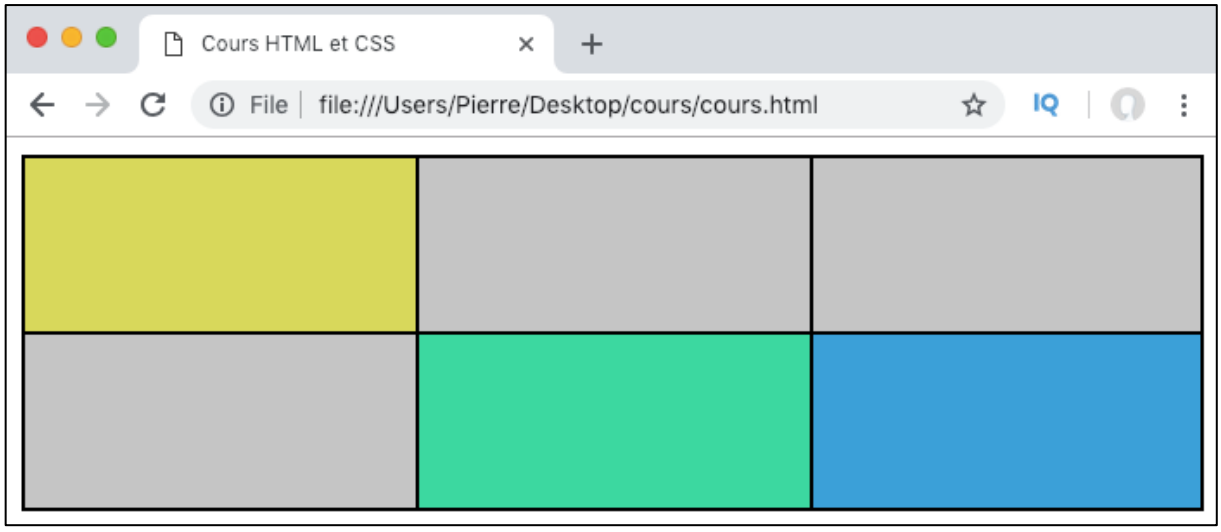
Ci-dessous, vous pouvez retrouver une représentation visuelle des lignes d'une grille. Ma grille contient ici 3 colonnes et 2 rangées. Les lignes verticales ont été dessinées en bleu et les lignes horizontales ont été dessinées en rouge.

L'espace entre deux lignes adjacentes est ce qu'on appelle une piste. Le terme piste sert donc tout simplement à désigner indifféremment une colonne ou une rangée dans une grille.



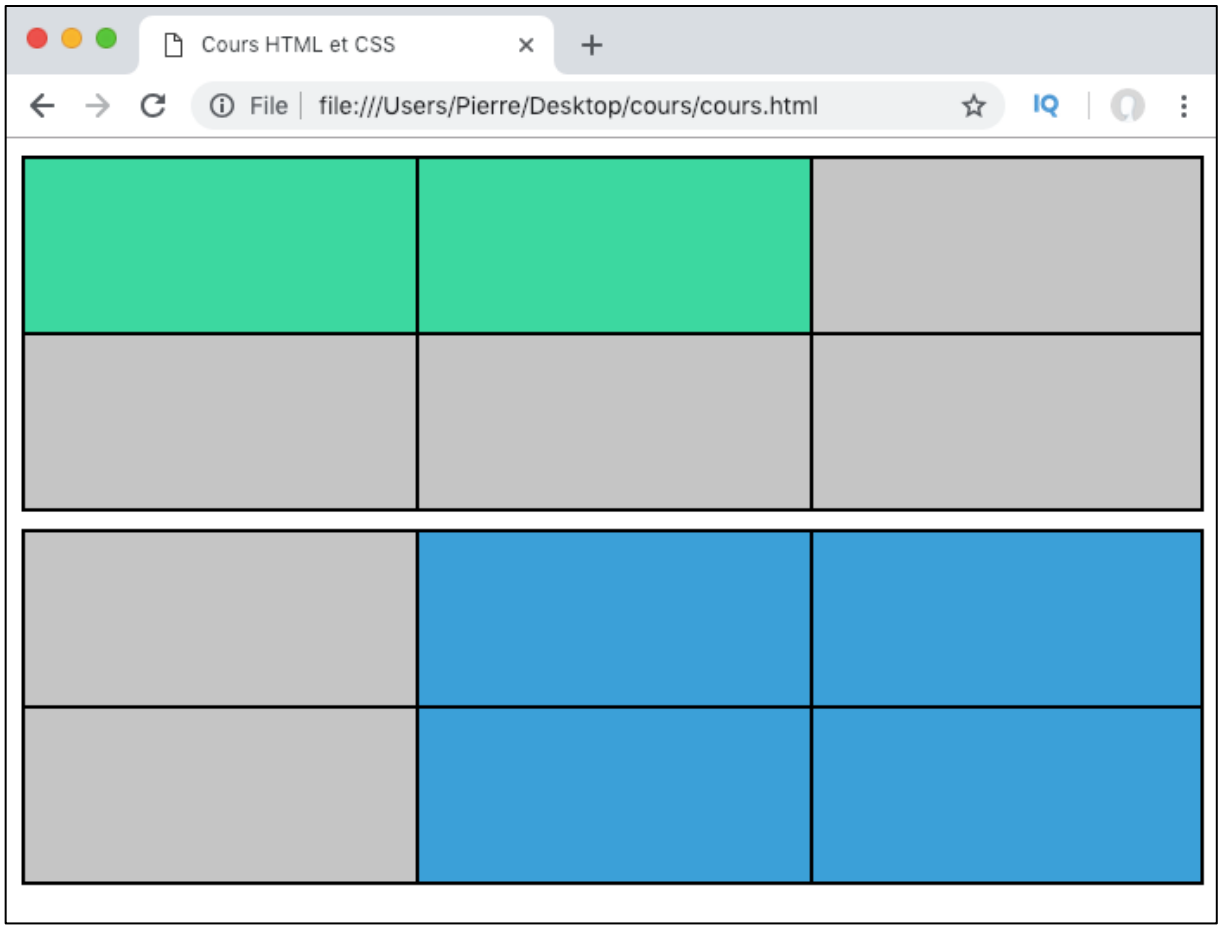
Ci-dessus, j'ai créé deux grilles composées de 3 colonnes et de 2 rangées. En jaune, j'ai colorié une piste pour chacune des deux grilles.

Une piste est composée de cellules. Tout comme pour les tableaux, une cellule correspond visuellement à l'espace délimité par deux lignes de colonnes et deux lignes de rangées adjacentes.



Ci-dessus, j'ai colorié 3 cellules de ma grille en jaune, vert et bleu.

Finalement, une zone de grille correspond à l'espace délimité par deux lignes de colonnes et deux lignes de rangées qui ne sont pas nécessairement adjacentes. Ci-dessous, j'ai dessiné une zone dans chacune de mes deux grilles (la première couvre 2 cellules et la deuxième couvre 4 cellules).



Créer une grille et définir des pistes

Dans la leçon précédente, nous avons vu qu'il suffisait d'appliquer un `display : grid` à un élément pour le transformer en un élément grille conteneur et ainsi définir notre grille.

Notre grille ainsi créée ne va par défaut être composée que d'une seule colonne et d'autant de rangées qu'il y a d'éléments de grille.

Dans ce cas, on dit que les pistes sont définies de manière implicite (car elles sont créées par la grille elle-même). Nous allons cependant également pouvoir définir les pistes de nos grilles nous-mêmes, c'est-à-dire explicitement.

Nous allons apprendre dans cette leçon à définir explicitement le nombre et la taille des pistes de nos grilles ainsi qu'à maitriser le comportement des pistes définies implicitement.

Définir explicitement les colonnes et les rangées d'une grille

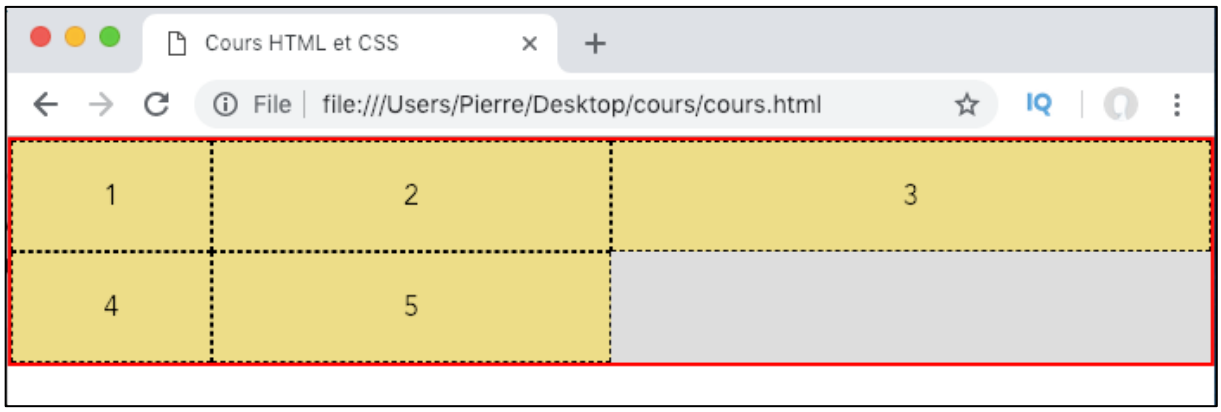
Pour définir explicitement le nombre et les dimensions de colonnes et de rangées de notre grille, nous allons pouvoir utiliser les propriétés `grid-template-columns` et `grid-template-rows`.

Ces propriétés vont pouvoir accepter n'importe quelle valeur de type dimension ainsi qu'un type de valeurs spécifique aux grilles : le `fr` qui représente une fraction de l'espace disponible dans le conteneur et qui va donc nous permettre de définir la taille d'une piste en fonction des autres.

Le nombre de valeurs passées à `grid-template-columns` et à `grid-template-rows` va déterminer le nombre de colonnes et de rangées de notre grille.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid">
            <div>1</div>
            <div>2</div>
            <div>3</div>
            <div>4</div>
            <div>5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: 1fr 2fr 3fr;
  border: 2px solid red;
  background-color: #DDD;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
```

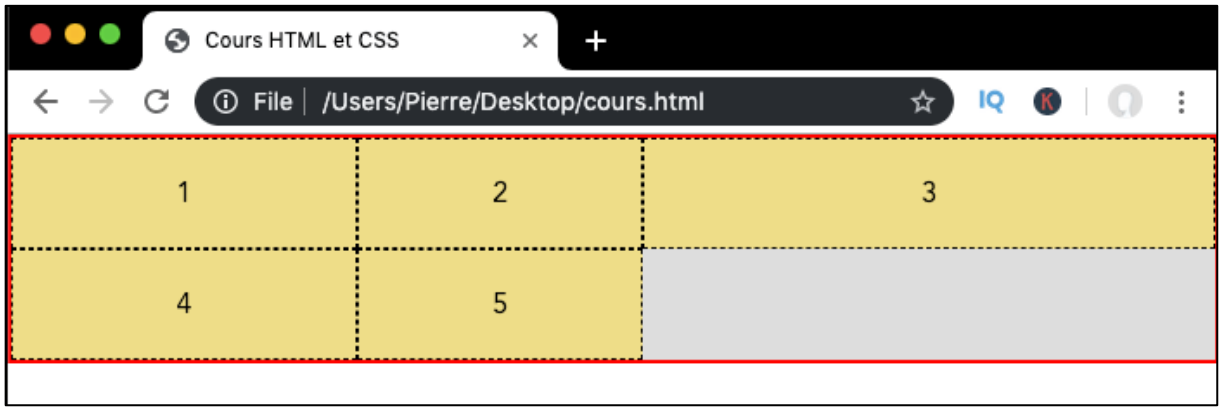


Ci-dessus, nous avons une grille qui contient 5 éléments. Nous créons trois colonnes dans notre grille avec `grid-template-columns : 1fr 2fr 3fr`. Dans ce cas, la deuxième colonne occupera deux fois plus d'espace que la première tandis que la troisième occupera trois fois plus d'espace que la première.

Notez qu'on va également tout à fait pouvoir définir nos pistes en utilisant un mélange de différents types d'unités.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur-grid">
            <div>1</div>
            <div>2</div>
            <div>3</div>
            <div>4</div>
            <div>5</div>
        </div>
    </body>
</html>
```

```css
*{
    font-family: Avenir, sans-serif;
    font-size: 1em;
    text-align: center;
    margin: 0px;
    padding: 0px;
    box-sizing: border-box;
}
.conteneur-grid{
    display: grid;
    grid-template-columns: 200px 1fr 2fr;
    border: 2px solid red;
    background-color: #DDD;
}
.conteneur-grid > div{
    padding: 20px 0px;
    background-color: #ED8;
    border: 1px dashed black;
}
```



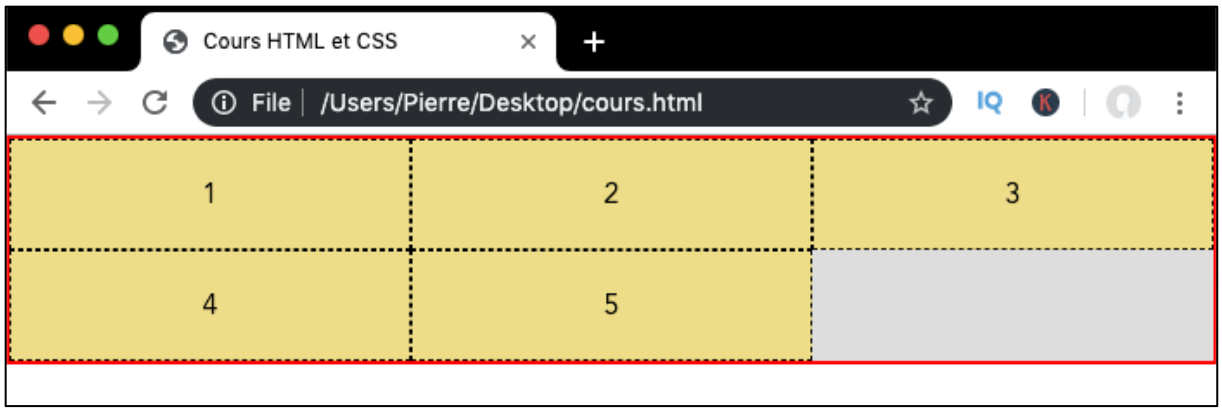
Dans cet exemple, on crée à nouveau une grille à trois colonnes. Cette fois-ci, on demande à ce que notre première colonne occupe un espace de 200px en largeur. L'espace restant

dans le conteneur sera ensuite partagé entre les deux autres colonnes : 1/3 de l'espace restant pour la deuxième colonne et les deux autres tiers pour la dernière colonne.

Nous allons encore pouvoir passer une fonction `repeat()` à `grid-template-columns` et à `grid-template-rows` pour créer rapidement plusieurs pistes aux caractéristiques similaires. Cette fonction va accepter deux valeurs : le nombre de pistes à créer ainsi que la taille de chaque piste.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur-grid">
            <div>1</div>
            <div>2</div>
            <div>3</div>
            <div>4</div>
            <div>5</div>
        </div>
    </body>
</html>
```

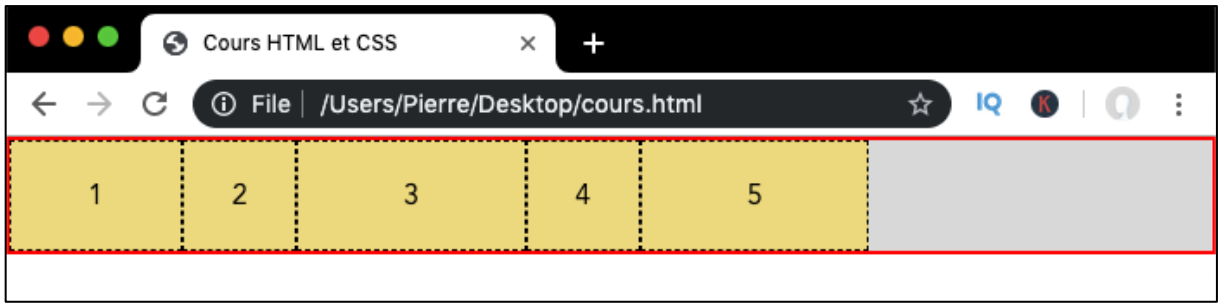
```css
*{
    font-family: Avenir, sans-serif;
    font-size: 1em;
    text-align: center;
    margin: 0px;
    padding: 0px;
    box-sizing: border-box;
}
.conteneur-grid{
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    border: 2px solid red;
    background-color: #DDD;
}
.conteneur-grid > div{
    padding: 20px 0px;
    background-color: #ED8;
    border: 1px dashed black;
}
```



Ici, on définit à nouveau trois colonnes dans notre grille. Chaque colonne occupera la même place. Notez qu'on va pouvoir utiliser la fonction `repeat()` pour définir des motifs. Notez également qu'on va pouvoir définir certaines pistes avec `repeat()` et certaines pistes individuellement.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <div class="conteneur-grid">
            <div>1</div>
            <div>2</div>
            <div>3</div>
            <div>4</div>
            <div>5</div>
        </div>
    </body>
</html>
```

```css
*{
    font-family: Avenir, sans-serif;
    font-size: 1em;
    text-align: center;
    margin: 0px;
    padding: 0px;
    box-sizing: border-box;
}
.conteneur-grid{
    display: grid;
    grid-template-columns: 100px repeat(3, 1fr 2fr);
    border: 2px solid red;
    background-color: #DDD;
}
.conteneur-grid > div{
    padding: 20px 0px;
    background-color: #ED8;
    border: 1px dashed black;
}
```



Dans l'exemple ci-dessus, on crée une première colonne qui va occuper une place de 100px en largeur puis on définit un motif de 2 colonnes à répéter 3 fois avec `repeat()`. A chaque fois, la première colonne aura une taille de `1fr` et la seconde une taille de `2fr`.

Règles de création et taille des pistes implicites

Dans les exemples ci-dessus, nous n'avons à chaque fois défini que les colonnes de notre grille et avons laissé la grille définir implicitement son nombre de rangées.

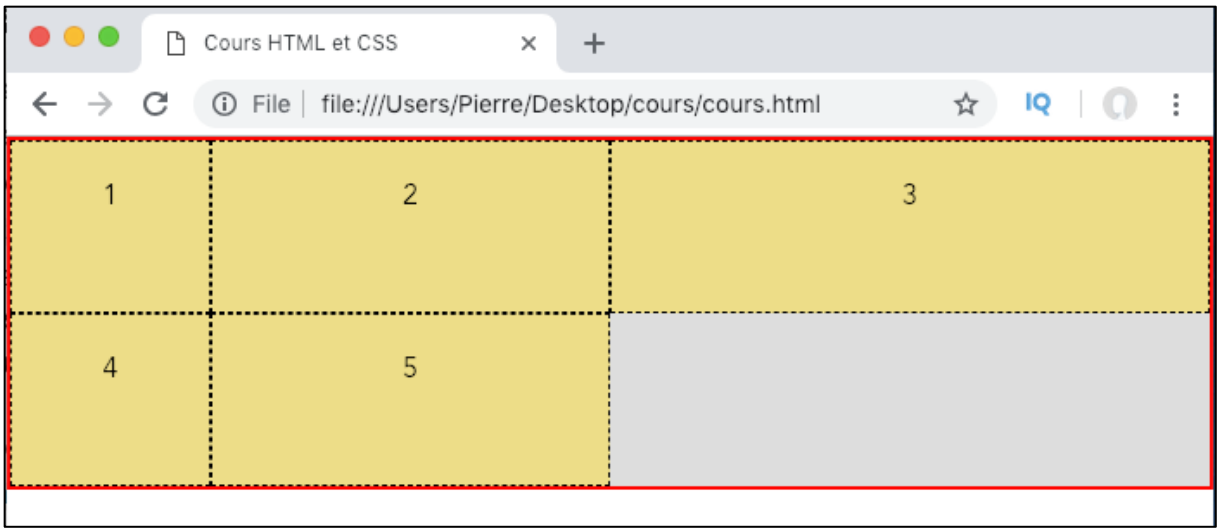
On dit qu'une piste est définie implicitement dès qu'elle n'a pas été créée avec `grid-template-columns` ou `grid-template-rows`.

Une grille va ainsi créer implicitement de nouvelles rangées ou de nouvelles colonnes dès qu'elle va devenir trop petite pour contenir un élément. Cela peut arriver dans deux situations différentes : si on a tenté de placer explicitement un élément en dehors de la grille ou si nos éléments possèdent trop de contenu pour que celui-ci rentre dans la grille.

Par défaut, les dimensions des pistes créées implicitement auront la valeur `auto` ce qui signifie que les pistes ajusteront leur taille selon leur contenu. On va pouvoir définir un autre comportement et une taille pour les pistes créées implicitement grâce aux propriétés `grid-auto-rows` et `grid-auto-columns`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid">
            <div>1</div>
            <div>2</div>
            <div>3</div>
            <div>4</div>
            <div>5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: 1fr 2fr 3fr;
  grid-auto-rows: 100px;
  border: 2px solid red;
  background-color: #DDD;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
```



Dans l'exemple ci-dessus, on crée une grille et on définit explicitement 3 colonnes avec `grid-template-columns : 1fr 2fr 3fr`. On ne précise pas de propriété `grid-template-rows` : les rangées vont donc être créées implicitement.

Nous allons donc renseigner une propriété `grid-rows-auto` pour maitriser la hauteur des rangées créées implicitement par la grille.

Définir un intervalle de tailles valides pour les pistes d'une grille

Les propriétés `grid-template-columns`, `grid-template-rows`, `grid-auto-columns` et `grid-auto-rows` vont également pouvoir accepter une fonction `minmax()` en valeur pour une ou plusieurs pistes.

La fonction `minmax()` va s'avérer très intéressante puisqu'elle va nous permettre de définir des bornes, c'est-à-dire un intervalle de dimensions valides pour nos pistes, que celles-ci aient été définies explicitement ou implicitement.

Nous allons passer deux valeurs à cette fonction qui vont correspondre à la borne basse et à la borne haute. Les types de valeurs acceptés par `minmax()` sont les suivantes :

- Une longueur en `px` par exemple ;
- Un pourcentage ;
- Une unité de fraction `fr` ;
- Le mot clef `max-content` qui va représenter la taille idéale de la piste c'est-à-dire la plus petite taille permettant d'afficher tout le contenu sur une ligne ;
- Le mot clef `min-content` qui va représenter la plus petite taille que le piste peut avoir sans que son contenu ne déborde (avec un contenu éventuellement sur plusieurs lignes) ;
- Le mot clef `auto` qui va laisser la piste s'adapter en fonction de son contenu et par rapport à son autre borne.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid">
            <div>1</div>
            <div>2</div>
            <div>3</div>
            <div>4</div>
            <div>5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: minmax(100px, 1fr) 2fr 3fr;
  grid-auto-rows: minmax(80px, auto);
  border: 2px solid red;
  background-color: #DDD;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
```

Dans l'exemple précédent, on utilise `grid-auto-rows : minmax(80px, auto)` pour indiquer que les rangées créées implicitement ne peuvent pas faire moins de 80px de hauteur et doivent s'adapter à leur contenu si celui-ci est plus grand que la taille minimale.

On va également utiliser `minmax()` en valeur de notre propriété `gris-template-columns` pour indiquer que notre première colonne doit occuper une largeur minimum de `100px` et maximum de `1fr`.

Ici, dans le cas où la borne haute (`1fr`) s'avère plus petite que la borne basse (`100px`), alors elle sera ignorée et `minmax()` ne servira qu'à définir une taille minimale.

Positionner des éléments dans une grille

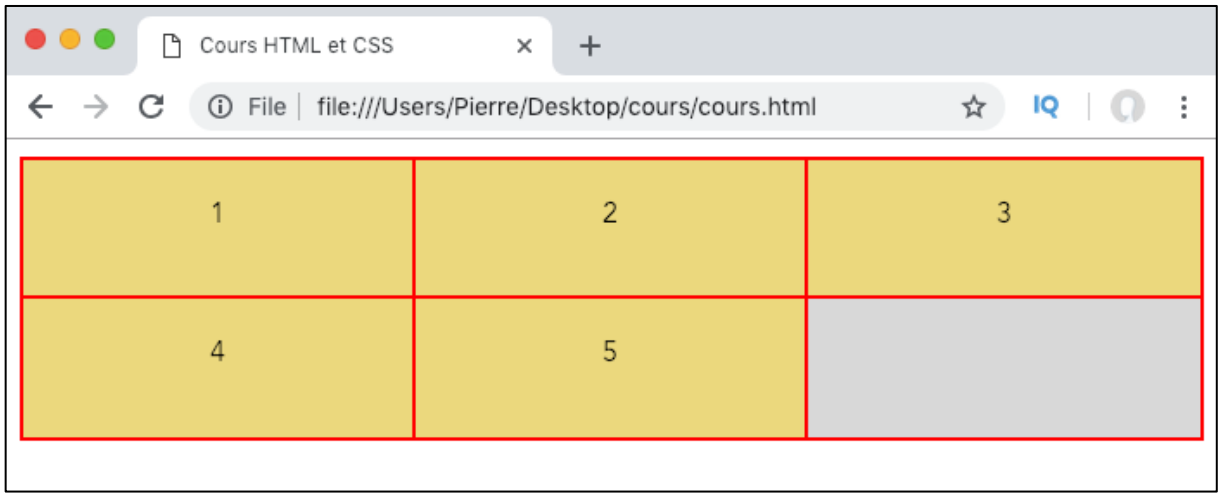
Nous savons désormais comment créer des grilles et comment définir précisément les colonnes et les rangées dont elles se composent.

Dans cette leçon, nous allons apprendre à positionner les éléments de grille dans une grille.

Les fondamentaux du positionnement des éléments de grille

Les éléments de grille vont être positionnés à partir des lignes invisibles créées automatiquement par la grille. Pour rappel, ces lignes vont se trouver de chaque côté des différentes pistes d'une grille. Une grille possédant 3 colonnes et 2 rangées va ainsi disposer de 4 lignes verticales et de 3 lignes horizontales.

Dans l'illustration ci-dessous, j'ai tracé en rouge les différentes lignes d'une telle grille qui possède ici 5 éléments :



Ces lignes vont par défaut être numérotées dans chacun des deux axes. Elles vont donc posséder un ordre sur lequel on va pouvoir se baser pour positionner nos éléments de grille.

La numérotation va dépendre du sens de l'écriture du document. Pour un langage qui se lit de gauche à droite, par exemple, la ligne tout à gauche sera la ligne n°1, celle à sa droite sera la ligne n°2 et etc. dans l'axe de bloc. De même, la ligne tout en haut sera la n°1, celle en dessous la n°2 et etc. dans l'axe en ligne.

Pour positionner les éléments de grille dans la grille, nous allons pouvoir indiquer 4 lignes : de lignes de départ (horizontal / vertical) à partir desquelles l'élément doit être placé et deux lignes d'arrivée (horizontal / vertical) où l'élément doit finir.

Nous allons donc indiquer une surface qu'un élément doit couvrir et, en même temps qu'on définit son positionnement, définir de facto la taille de l'élément. Cette façon de faire est différente de ce qu'on a pu voir jusqu'à présent et va imposer aux éléments de grille de couvrir toujours un nombre entier de cellules.

Ce que vous devez bien comprendre ici est que la définition des tailles / dimensions va se faire au moment où on définit les pistes de notre grille. Pour cette raison, il faudra bien définir au départ si on veut créer une grille à 3, 6, 12, etc. colonnes et également bien définir le comportement des rangées.

Ensuite, nous n'allons faire que placer des éléments dans la grille. Fonctionner comme cela peut sembler contraignant mais c'est au final une excellente méthode pour organiser ses éléments et créer des designs complexes puisque toutes les contraintes se font sur la grille et cela va nous éviter d'avoir à envisager toutes les situations de comportement non voulu des différents éléments.

Positionner des éléments de grille en pratique

Par défaut, les éléments de grille vont se positionner à la suite les uns des autres selon le sens de l'écriture du document. Chaque élément va occuper une cellule, c'est-à-dire ne va s'étendre que sur une rangée et sur une colonne.

Pour positionner manuellement (ou explicitement) les éléments dans la grille, nous allons pouvoir utiliser les propriétés suivantes :

- `grid-column-start` : Indique la ligne de départ de l'élément selon l'axe de bloc (ligne verticale) ;
- `grid-column-end` : Indique la ligne de fin de l'élément selon l'axe de bloc (ligne verticale) ;
- `grid-row-start` : Indique la ligne de départ de l'élément selon l'axe en ligne (ligne horizontale) ;
- `grid-row-end` : Indique la ligne de fin de l'élément selon l'axe en ligne (ligne horizontale) ;
- `grid-area` : Indique une zone de grille dans laquelle l'élément doit de placer.

Nous allons également pouvoir utiliser les versions raccourcies `grid-column`, `grid-row`.

Positionner les éléments de grille en utilisant les lignes des grilles

Utiliser la numérotation des lignes pour placer les éléments

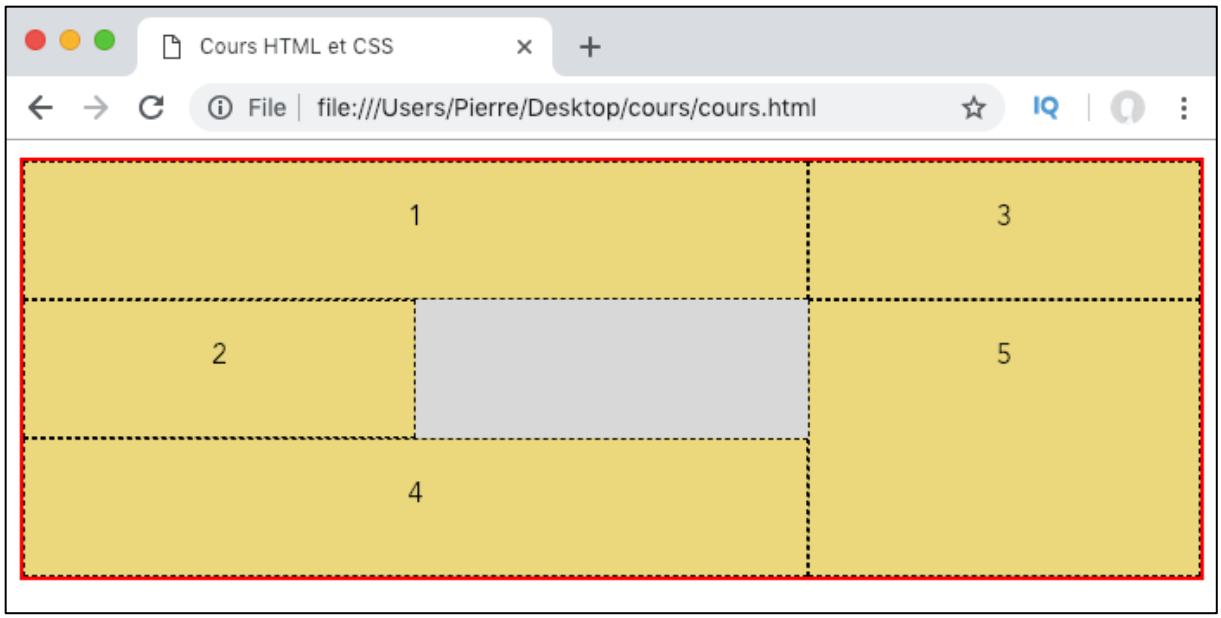
Nous allons déjà pouvoir passer un chiffre aux propriétés `grid-column-start`, `grid-column-end`, `grid-row-start` et `grid-row-end`.

Le chiffre passé va indiquer le numéro de la ligne où l'élément de grille doit commencer (pour les propriétés `-start`) ou la ligne où il doit s'arrêter (pour les propriétés `-end`.

Commençons avec un exemple simple d'une grille possédant 5 éléments tous positionnés explicitement :

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(3,minmax(80px,1fr));
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
.g1{grid-column-start: 1;
    grid-column-end: 3;
    grid-row-start: 1;
    grid-row-end: 2;}

.g2{grid-column-start: 1;
    grid-column-end: 2;
    grid-row-start: 2;
    grid-row-end: 3;}

.g3{grid-column-start: 3;
    grid-column-end: 4;
    grid-row-start: 1;
    grid-row-end: 2;}

.g4{grid-column-start: 1;
    grid-column-end: 3;
    grid-row-start: 3;
    grid-row-end: 4;}

.g5{grid-column-start: 3;
    grid-column-end: 4;
    grid-row-start: 2;
    grid-row-end: 4;}
```



Ici, on indique que notre premier élément doit de positionner à partir de la première ligne verticale et couvrir l'espace jusqu'à la troisième ligne verticale. L'élément sera également compris entre la première et la deuxième ligne horizontale.

Notre deuxième élément devra se positionner entre la première et la deuxième ligne verticale et entre la deuxième et la troisième ligne horizontale.

On demande à l'élément 3 de se positionner entre la troisième et la quatrième ligne verticale et entre la première et la deuxième ligne horizontale.

Notre quatrième élément de grille va se placer entre la première et la troisième ligne verticale et entre la troisième et la quatrième ligne horizontale.

Finalement, notre cinquième et dernier élément de grille va recouvrir l'espace entre la troisième et la quatrième ligne verticale et entre la deuxième et la quatrième ligne horizontale de la grille.

Notez qu'on ne va pas être obligé de préciser les 4 propriétés pour positionner un élément dans une grille. Pour être tout à fait précis, seule une des deux propriétés `grid-column-start` et `grid-row-start` est strictement nécessaire.

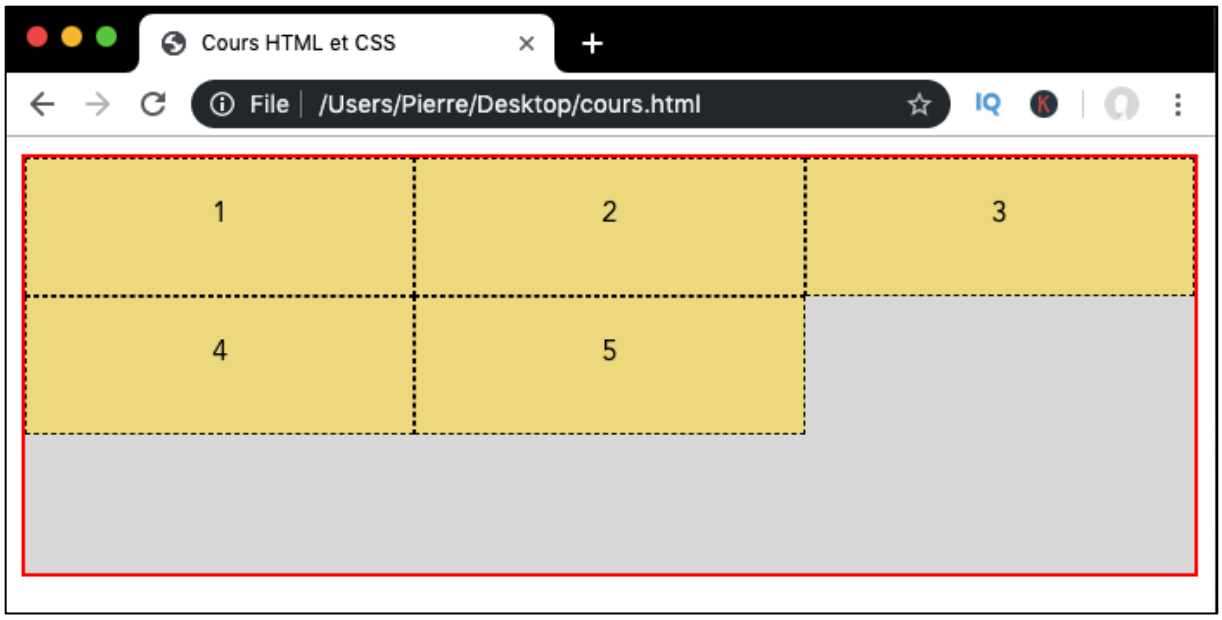
Attention cependant : si certaines propriétés sont omises pour certains éléments mais précisées pour d'autres, alors les éléments dont les propriétés ont été précisées viendront se placer avant les autres dans la grille et cela peut décaler les autres éléments.

En omettant l'une des deux propriétés `grid-column-start` ou `grid-row-start`, l'élément sera positionné selon sa position par défaut par rapport à l'axe non défini.

En omettant les propriétés `grid-column-end` et / ou `grid-row-end`, l'élément suivra son comportement par défaut qui est de n'occuper qu'une piste dans l'axe non défini.

Pour le moment, contentons-nous simplement d'omettre les propriétés `-end` lorsque l'élément n'est positionné que sur une piste. Nous verrons les cas de positionnement plus complexes plus tard.

```css
*{
    font-family: Avenir, sans-serif;
    font-size: 1em;
    text-align: center;
    margin: 0px;
    padding: 0px;
    box-sizing: border-box;
}
.conteneur-grid{
    display: grid;
    grid-template-columns: repeat(3,1fr);
    grid-template-rows: repeat(3,minmax(80px,1fr));
    border: 2px solid red;
    background-color: #DDD;
    margin: 10px;
}
.conteneur-grid > div{
    padding: 20px 0px;
    background-color: #ED8;
    border: 1px dashed black;
}
/*Produit le même résultat que précédemment*/
.g1{
    grid-column-start: 1;
    grid-column-end: 3;
    grid-row-start: 1;
}
.g2{
    grid-column-start: 1;
    grid-row-start: 2;
}
.g3{
    grid-column-start: 3;
    grid-row-start: 1;
}
.g4{
    grid-column-start: 1;
    grid-column-end: 3;
    grid-row-start: 3;
}
.g5{
    grid-column-start: 3;
    grid-row-start: 2;
    grid-row-end: 4;
}
```



Finalement, notez également qu'on va pouvoir placer un élément à partir de la fin de la grille en indiquant des numéros de lignes négatifs. Dans ce cas, -1 correspond à la dernière ligne définie explicitement, -2 à l'avant dernière etc. En pratique, cette notation est très peu utilisée et va répondre à des besoins très spécifiques.

Nommer les lignes pour positionner les éléments

Jusqu'à présent, nous nous sommes contentés d'utiliser le système de numérotation des lignes créé automatiquement par les grilles.

Nous allons également pouvoir nommer nos lignes pour pouvoir ensuite les manipuler plus facilement.

Pour cela, nous allons devoir indiquer le nom des lignes entre crochets dans la définition des pistes de la grille avec les propriétés `grid-template-columns` et `grid-template-rows` que nous avons étudié précédemment.

Reprenons notre grille précédente et nommons des lignes :

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: [ext-gauche] 1fr [centre-gauche] 2fr [centre-droite] 1fr [ext-droite];
  grid-template-rows: minmax(100px, 1fr) [body-sup] minmax(200px, 2fr) [body-inf] minmax(100px, 1fr);
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
.g1{
    grid-column-start: ext-gauche;
    grid-column-end: centre-droite;
    grid-row-start: 1;
}
.g2{
    grid-column-start: ext-gauche;
    grid-row-start: body-sup;
}
.g3{
    grid-column-start: centre-droite;
    grid-row-start: 1;
}
.g4{
    grid-column-start: ext-gauche;
    grid-column-end: centre-droite;
    grid-row-start: body-inf;
}
.g5{
    grid-column-start: centre-droite;
    grid-row-start: body-sup;
    grid-row-end: 4;
}
```

Ici, ma grille possède à nouveau 3 colonnes et 3 rangées, donc 4 lignes verticales et 4 lignes horizontales. Ici, vous devez bien comprendre que l'on va à la fois nommer les lignes et définir nos pistes avec les propriétés grid-template-columns et grid-template-rows.

Regardons de plus près la déclaration relative à grid-template-columns : je commence par donner un nom à ma première ligne, puis je définis la taille de ma première colonne, puis je passe le nom de ma deuxième ligne, puis la taille de ma deuxième colonne et etc. jusqu'au nom de ma dernière ligne.

Bien sûr, nous ne sommes pas obligés de nommer toutes les lignes. Par exemple, je n'ai donné de nom qu'au deux lignes horizontales les plus à l'intérieur de ma grille dans grid-template-rows.

Ensuite, nous allons pouvoir utiliser ces noms plutôt que les numéros pour positionner nos éléments de grille.

Utiliser les propriétés raccourcies grid-column et grid-row pour positionner les éléments dans la grille en fonction des lignes

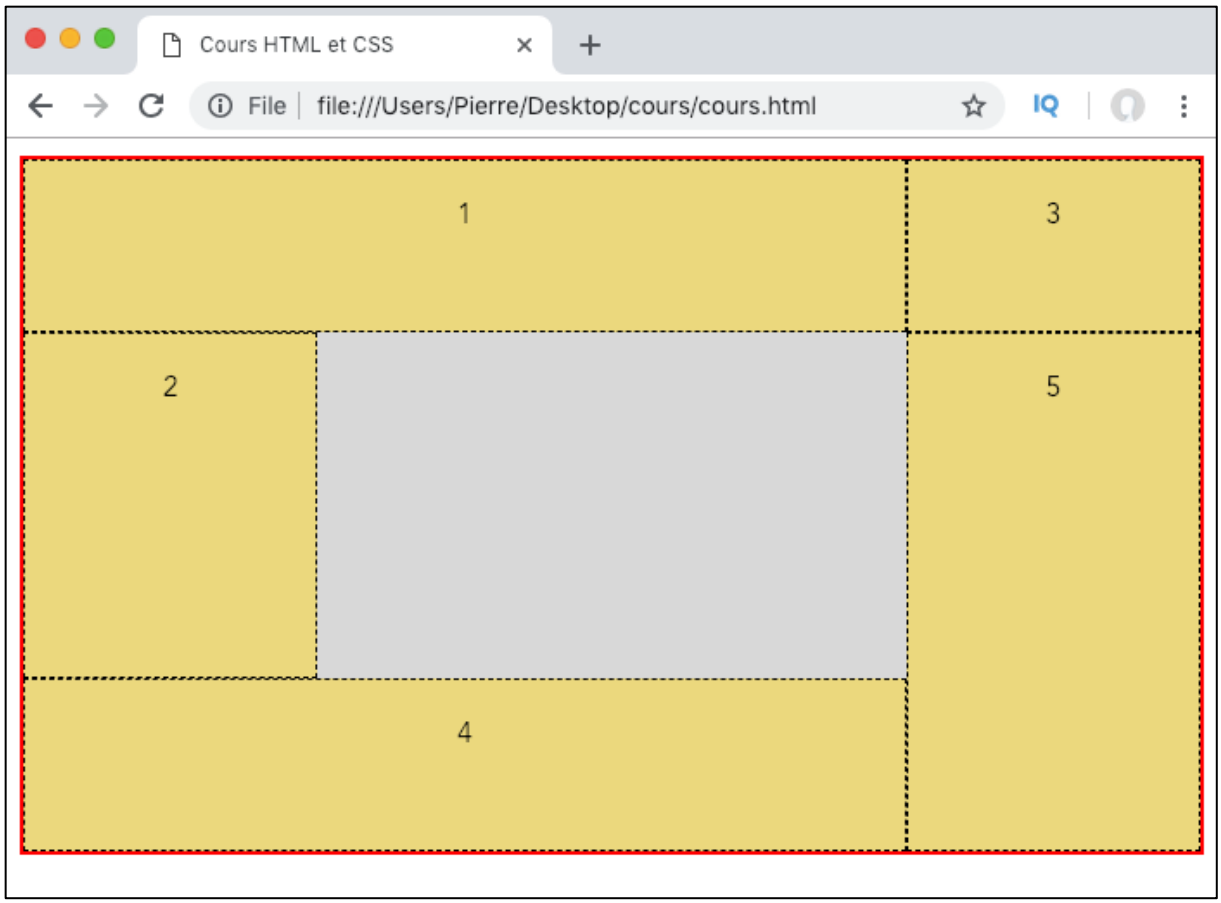
La propriété grid-column est la version raccourcie des propriétés grid-column-start et grid-column-end.

La propriété grid-row est la version raccourcie des propriétés grid-row-start et grid-row-end.

Nous allons donc pouvoir passer deux valeurs à chacune de ces propriétés. Les valeurs devront être séparées par un slash `/`. Les valeurs vont pouvoir être numérotées ou être des noms de lignes. Dans le cas où une seule valeur est passée, elle sera considérée comme étant la valeur de départ.

En reprenant notre grille précédente, on va ainsi pouvoir écrire :

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns:
    [ext-gauche] 1fr [centre-gauche] 2fr [centre-droite] 1fr [ext-droite];
  grid-template-rows:
    minmax(100px,1fr) [body-sup] minmax(200px,2fr) [body-inf] minmax(100px,1fr);
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
.g1{
    grid-column: ext-gauche / centre-droite;
    grid-row: 1;
}
.g2{
    grid-column: ext-gauche;
    grid-row: body-sup;
}
.g3{
    grid-column: centre-droite;
    grid-row: 1;
}
.g4{
    grid-column: ext-gauche / centre-droite;
    grid-row: body-inf;
}
.g5{
    grid-column: centre-droite;
    grid-row: body-sup / 4;
}
```



Positionner les éléments en utilisant les zones des grilles

Utiliser la numérotation des lignes pour définir des zones et placer les éléments

La propriété grid-area va nous permettre de définir des zones dans une grille. Pour cela, nous allons déjà pouvoir lui passer 4 numéros en valeur séparés par des slash.

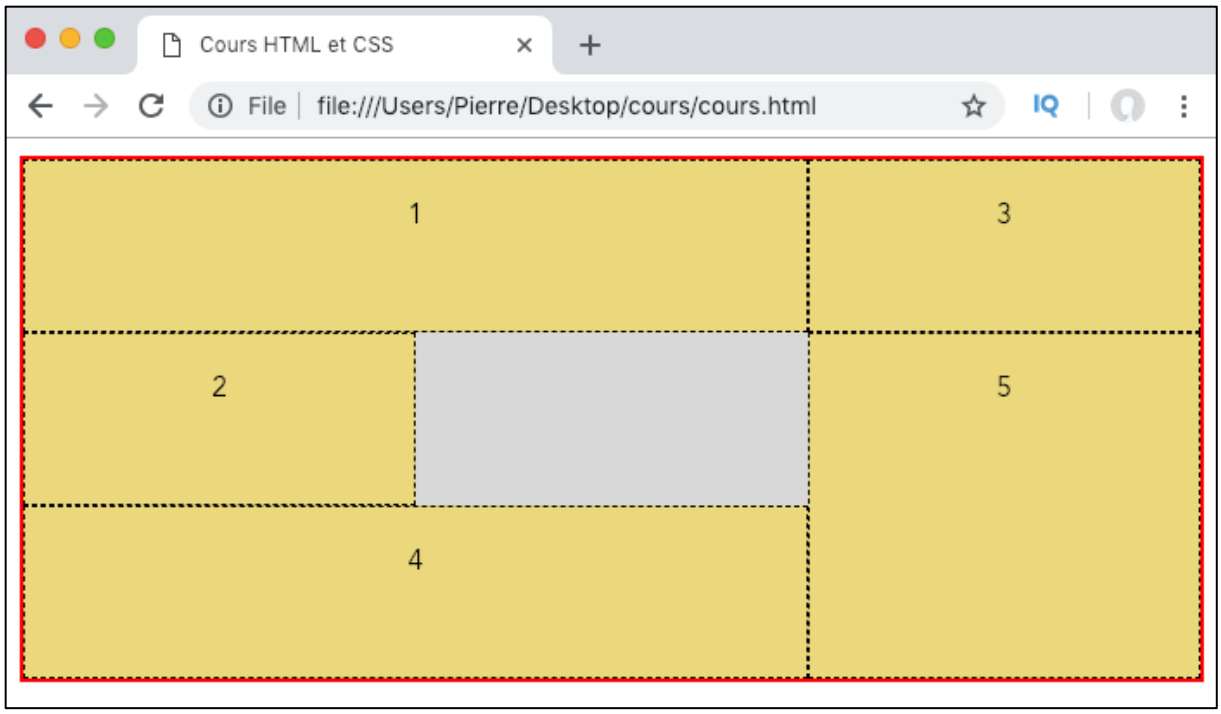
La première valeur correspond au numéro de la ligne horizontale où la zone doit commencer, la deuxième valeur correspond au numéro de la ligne verticale où la zone doit commencer, la troisième valeur correspond au numéro de la ligne horizontale où la zone se termine et la quatrième valeur correspond au numéro de la ligne verticale où la zone se termine.

L'ordre des valeurs (pour un langage qui s'écrit de gauche à droite) est donc un ordre antihoraire : haut, gauche, bas, droite.

La propriété grid-area va s'appliquer aux éléments de grille et ainsi également définir leur taille. Elle peut ainsi remplacer les propriétés grid-column-start, grid-column-end, grid-row-start et grid-row-end.

Illustrons cela en reprenant à nouveau notre grille précédente et en utilisant plutôt grid-area pour placer les éléments dans la grille.

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(3,minmax(100px, 1fr));
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
.g1{
    grid-area: 1 / 1 / 2 / 3;
}
.g2{
    grid-area: 2 / 1 / 3 / 2;
}
.g3{
    grid-area: 1 / 3 / 2 / 4;
}
.g4{
    grid-area: 3 / 1 / 4 / 3;
}
.g5{
    grid-area: 2 / 3 / 4 / 4;
}
```



Nommer des zones pour positionner les éléments

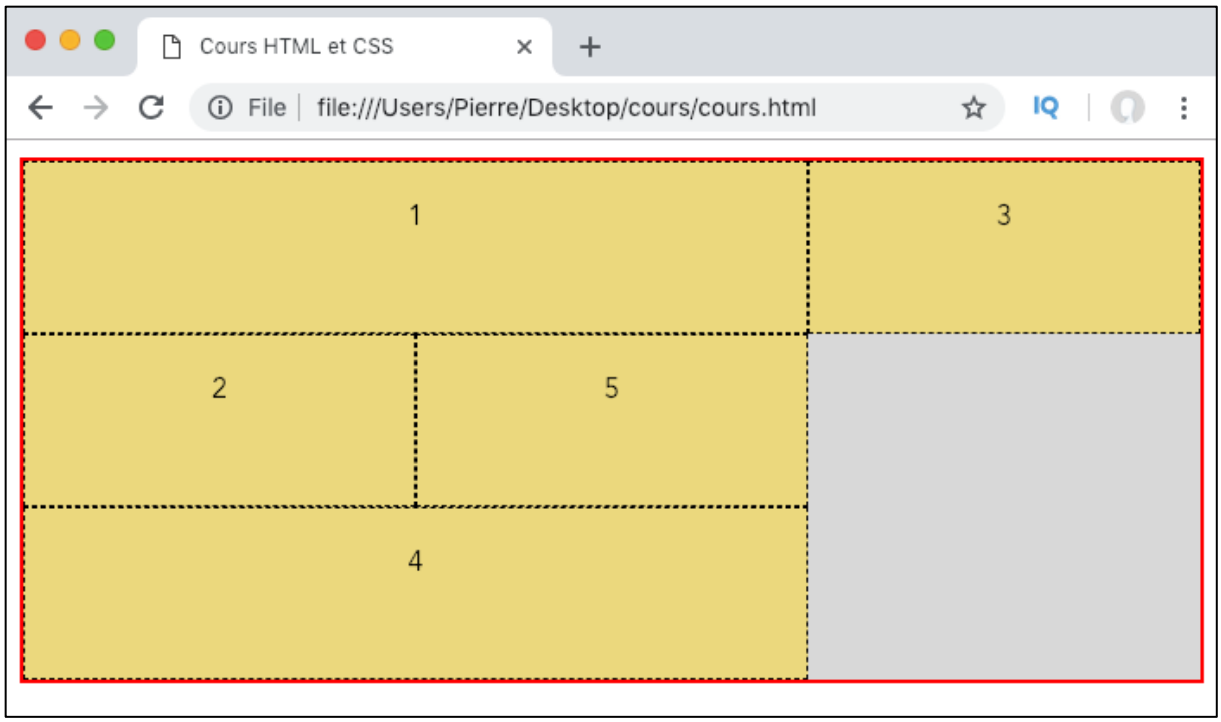
Nous allons également pouvoir nommer nos zones de grille pour ensuite placer les éléments. Pour cela, il va falloir procéder en deux étapes.

Nous allons déjà passer les différents noms de zones à `grid-area` puis ensuite les utiliser avec la propriété `grid-template-areas` pour définir les zones en soi dans notre grille.

Nous allons passer à `grid-template-areas` le nom de nos zones en définissant pour chaque cellule la zone à laquelle elle appartient. Nous allons entourer les valeurs pour chaque rangée de la grille avec un couple de guillemets `""`. Pour laisser une cellule hors zone, nous indiquerons un point `.` en valeur.

Notez que les zones crées doivent toujours être rectangulaires. D'autres formes comme une forme en « L » ne sont à l'heure actuelle pas permises.

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(3,minmax(100px, 1fr));
  grid-template-areas: "hg hg hd" "mg . bd" "bg bg bd";
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
.g1{
    grid-area: hg;
}
.g2{
    grid-area: mg;
}
.g3{
    grid-area: hd;
}
.g4{
    grid-area: bg;
}
.g5{
    grid-area: bd;
}
```



Ici, la zone `hg` recouvre les deux premières cellules de la première rangée de notre grille tandis que la zone `hd` recouvre la troisième cellule de cette rangée.

La zone `mg` recouvre la première cellule de la deuxième rangée de notre grille.

La zone `bg` recouvre les deux premières cellules de la troisième rangée de la grille.

Finalement, la zone `bd` recouvre la dernière cellule de la deuxième rangée de la grille ainsi que la dernière cellule de la troisième rangée de la grille.

La deuxième cellule de la deuxième rangée de notre grille n'appartient à aucune zone définie explicitement.

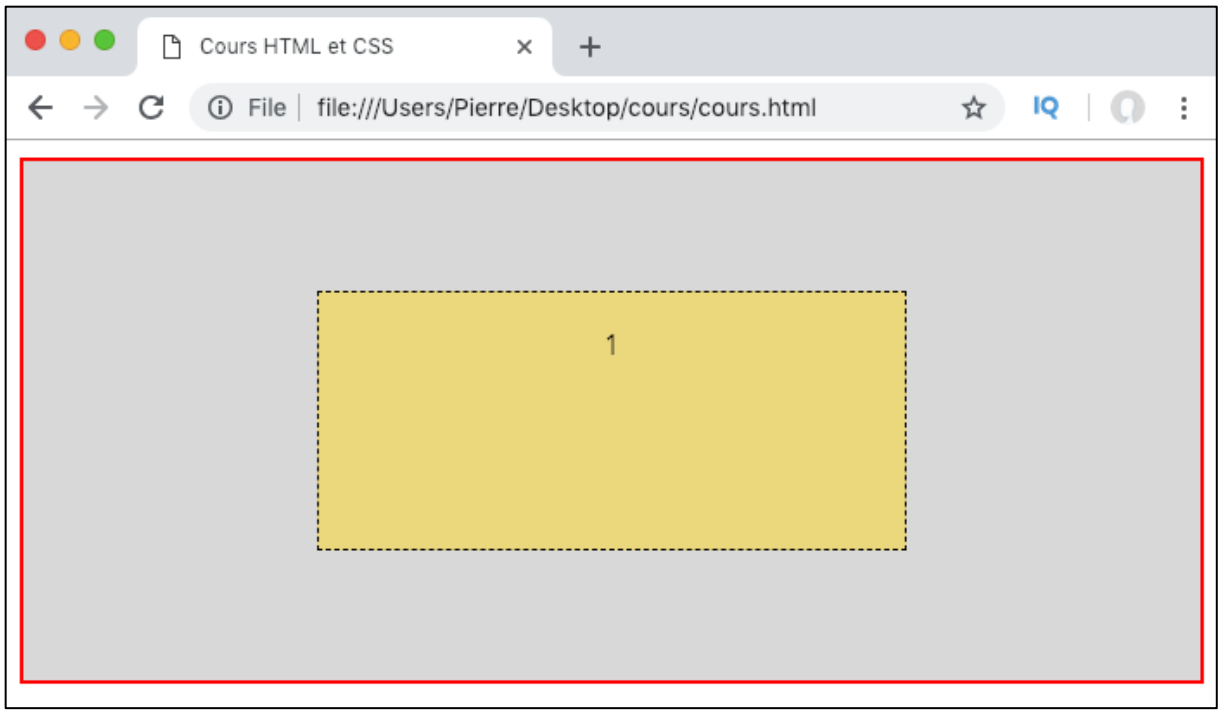
Lignes nommées et zones implicites, zones nommées et lignes implicites

Nous allons pouvoir donner n'importe quel nom à nos lignes et à nos zones. Cependant, certains noms de lignes vont déclencher des mécanismes de création de zones nommées et à l'inverse la création d'une zone nommée va automatiquement définir des noms pour les lignes qui la définissent.

Par exemple, si on utilise un même nom avec les suffixes `-start` et `-end` pour nommer les lignes de départ et d'arrivée qui peuvent définir une zone, alors la grille va automatiquement créer la zone en question et va lui donner le même nom sans préfixe. Nous allons ensuite pouvoir utiliser ce nom pour placer nos éléments de grille.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid">
            <div class="g1">1</div>
        </div>
    </body>
</html>
```

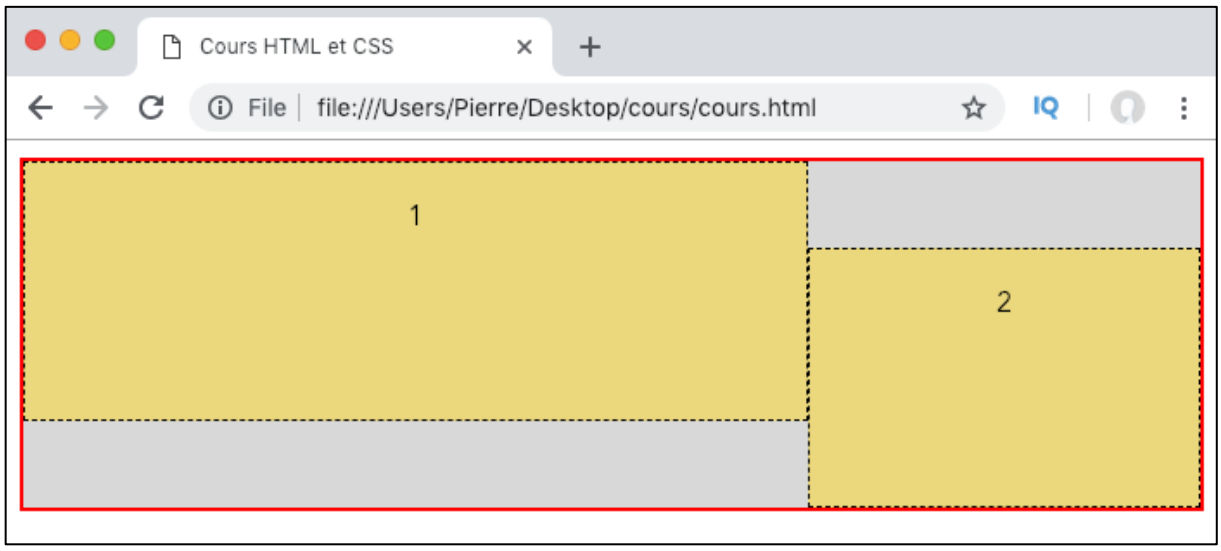
```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: [ext-start] 1fr [int-start] 2fr [int-end] 1fr [ext-end];
  grid-template-rows: [ext-start] 1fr [int-start] minmax(150px, 2fr) [int-end] 1fr [ext-end];
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
.g1{
    grid-area: int;
}
```



Ici, la grille va automatiquement créer une zone `ext` et une zone `int` à partir de la définition de nos lignes. On se sert de la zone `int` pour positionner notre élément de grille.

A l'inverse, en créant une zone nommée, la grille va automatiquement nommer les lignes qui servent à la définir en leur ajoutant les suffixes `-start` pour la ligne de départ et `-end` pour la ligne d'arrivée.

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3, [col] 1fr);
  grid-template-rows: [ran] 1fr [ran] minmax(100px,2fr) [ran] 1fr;
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
.g1{
    grid-column: col / col 3;
    grid-row: ran / ran 3;
}
.g2{
    grid-column: col 3 / col 4;
    grid-row: ran 2 / ran 4;
}
```



Ici, on définit une zone gauche qui recouvre la première cellule des première et deuxième rangées de la grille. Les lignes gauche-start et gauche-end sont alors automatiquement créées par la grille. On s'en sert pour positionner notre deuxième élément de grille.

Notez qu'il n'est pas gênant qu'une ligne verticale possède le même nom qu'une ligne horizontale ou même que deux lignes verticales ou deux lignes horizontales (ou plus)

possèdent le même nom. Cela est parfaitement autorisé et peut faire gagner du temps de développement si on souhaite créer des pistes similaires.

En cas d'ambiguïté sur le nom de la ligne, la grille utilisera toujours la première des lignes qui portent le même nom. Pour lever l'ambiguïté, on peut préciser le numéro de la ligne derrière son nom pour la cibler en particulier.

Ici, on crée une grille possédant à nouveau 3 colonnes et 3 rangées. Toutes les lignes verticales vont porter le nom `col` (observez bien la syntaxe dans `repeat()` tandis que toutes les lignes horizontales vont porter le nom `ran`.

Pour placer un élément dans la grille à partir ou jusqu'à une ligne qui n'est pas la première, il faut préciser le numéro de ligne après son nom pour lever l'ambiguïté.

Les propriétés raccourcies grid-template et grid

Notez qu'il existe deux autres propriétés raccourcies qui peuvent servir à positionner les éléments dans une grille : les propriétés `grid-template` et `grid`.

La propriété `grid-template` est la version raccourcie des propriétés `grid-template-rows`, `grid-template-columns` et `grid-template-areas`.

La propriété `grid` est une version qui condense encore plus de propriété puisque c'est la version raccourcie de `grid-template-rows`, `grid-template-columns`, `grid-template-areas`, `grid-auto-rows`, `grid-auto-columns` et `grid-auto-areas`.

Personnellement, je vous déconseille pour des raisons évidentes de lisibilité d d'utiliser ces deux versions raccourcies qui rendre le code très compliqué à comprendre.

Contrôler le positionnement automatique des éléments dans la grille

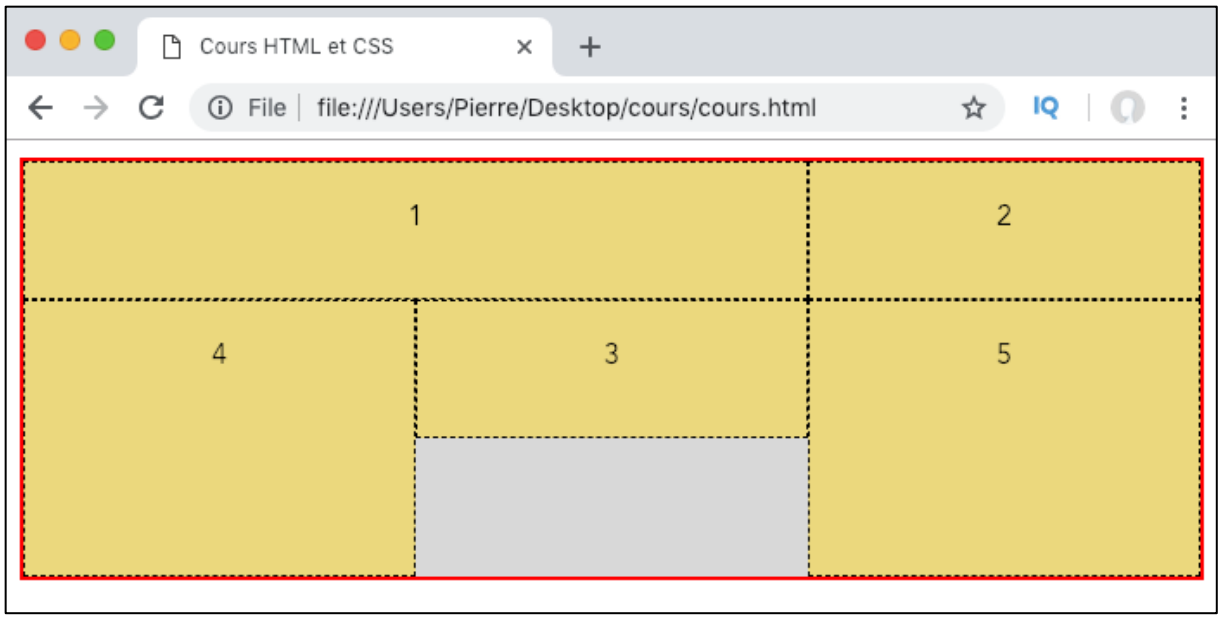
Mélange de positionnement explicite et implicite

On ne va pas être obligé de positionner explicitement (c'est-à-dire en utilisant une des propriétés vues précédemment) tous les éléments d'une grille.

Dans le cas où certains éléments sont positionnés explicitement et d'autres non, la grille va commencer par placer les éléments positionnés explicitement puis placera ensuite les autres éléments en utilisant le positionnement par défaut.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(3,minmax(80px,1fr));
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
.g1{
    grid-column: 1 / 3;
    grid-row: 1;
}
.g4{
    grid-column: 1;
    grid-row: 2 / 4;
}
.g5{
    grid-column: 3;
    grid-row: 2 / 4;
}
```



Ici, on ne positionne explicitement que nos éléments 1, 4 et 5. Ce seront donc les premiers éléments positionnés dans la grille. Ensuite, la grille va positionner automatiquement les éléments 2 et 3 en utilisant le positionnement par défaut dès qu'il y aura un espace suffisant dans la grille.

L'élément n°2 est le premier élément non positionné explicitement et va donc se positionner dans le premier espace disponible. L'élément n°3 est le deuxième élément non positionné et va donc se positionner dans le premier espace disponible suivant l'élément n°2.

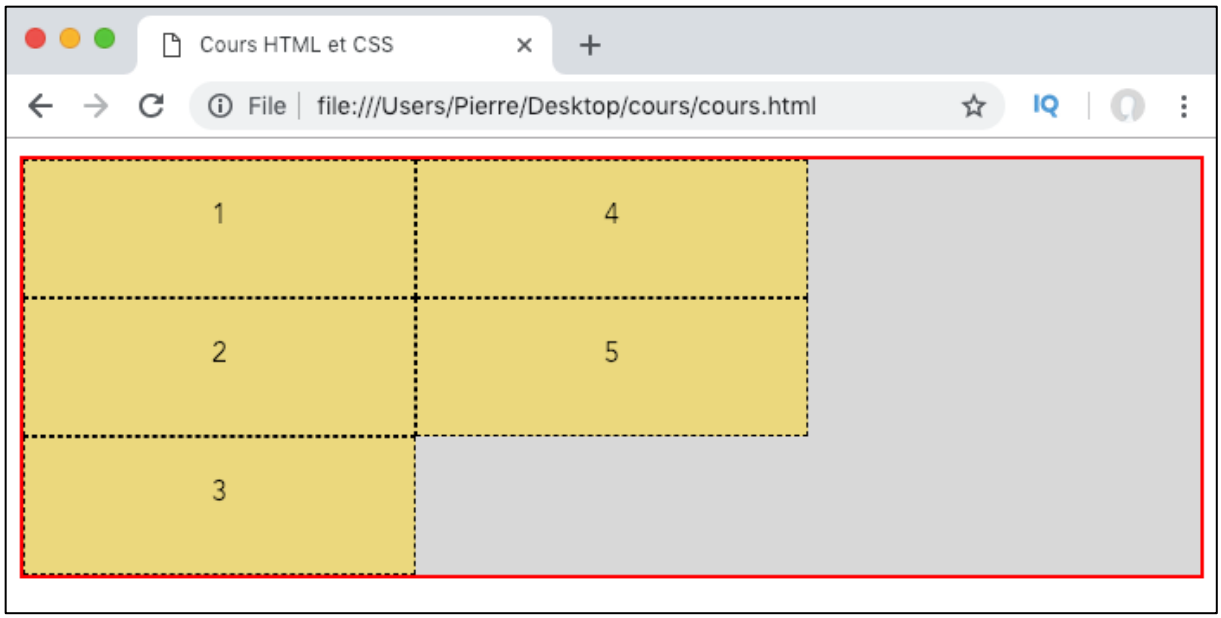
Placer les éléments automatiquement en colonne

Le placement automatique des éléments est défini par la propriété `grid-auto-flow`. La valeur par défaut est `row` qui signifie que les éléments positionnés automatiquement viendront se placer à côté des autres tant que possible.

Nous allons également pouvoir définir un placement automatique en colonne en utilisant la valeur `column` de cette même propriété.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  text-align: center;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(3,minmax(80px,1fr));
  grid-auto-flow: column;
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  padding: 20px 0px;
  background-color: #ED8;
  border: 1px dashed black;
}
```



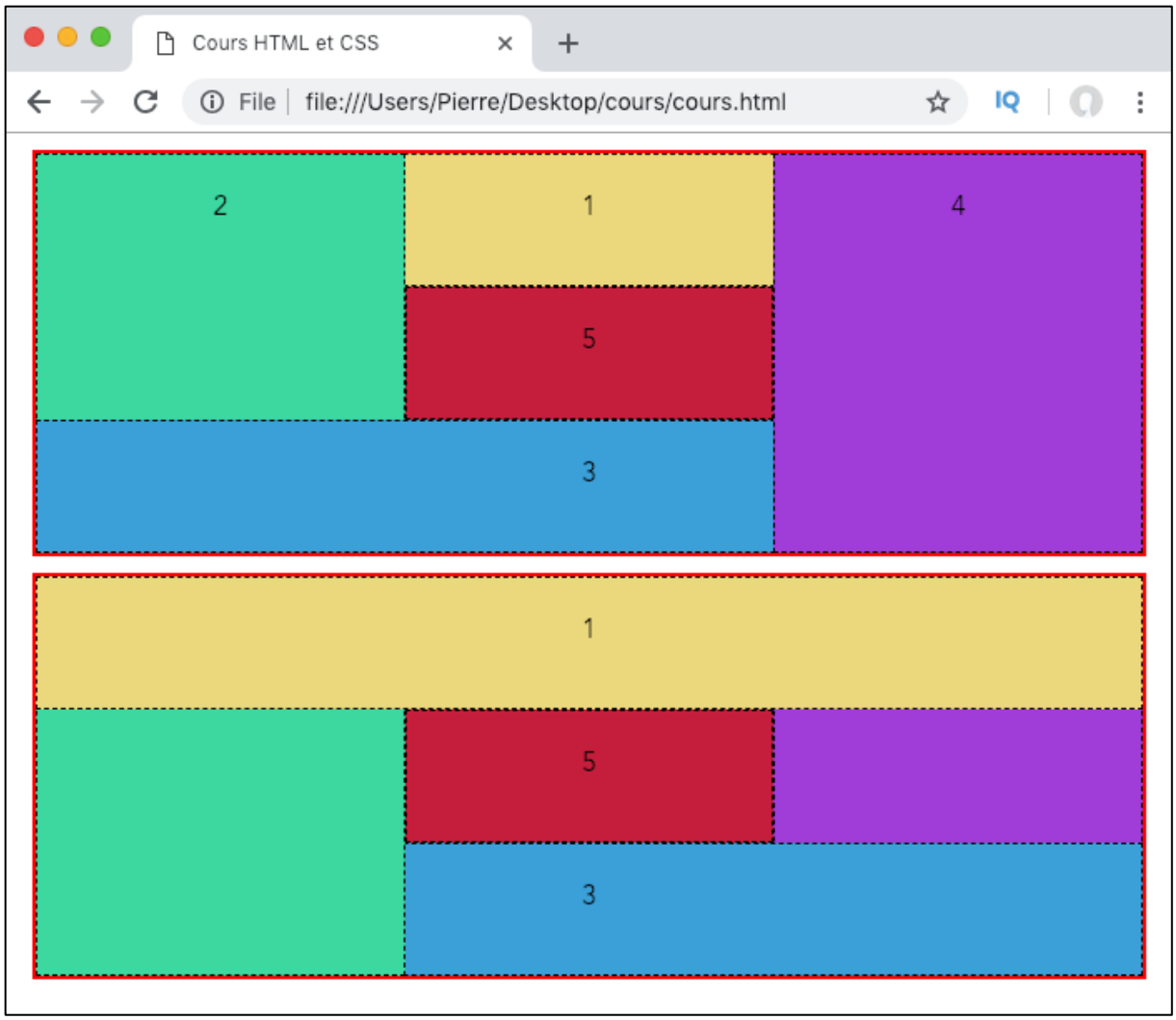
Contrôler le chevauchement des éléments dans une grille

Notez également que nous allons pouvoir positionner plusieurs éléments de grille dans les mêmes cellules. Par défaut, le dernier élément déclaré se placera au-dessus des autres.

Nous allons alors pouvoir modifier ce comportement et choisir quel élément devra apparaitre au-dessus de quel autre en définissant un `z-index` pour les différents éléments se chevauchant.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>

        <div class="conteneur-grid grille2">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif; font-size: 1em; text-align: center;
  margin: 0px; padding: 0px; box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(3,minmax(80px,1fr));
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  padding: 20px 0px;
  border: 1px dashed black;
}
.g1{
    grid-column: 1 / 4;
    grid-row: 1 / 2;
    background-color: #ED8; /*Jaune*/
}
.g2{
    grid-column: 1 / 2;
    grid-row: 1 / 4;
    background-color: #4DA; /*Vert*/
}
.g3{
    grid-column: 1 / 4;
    grid-row: 3 / 4;
    background-color: #4AD; /*Bleu*/
}
.g4{
    grid-column: 3 / 4;
    grid-row: 1 / 4;
    background-color: #A4D;/*Violet*/
}
.g5{
    background-color: #C24;/*Rouge*/
}
.grille2 .g1{z-index: 4;}
.grille2 .g2{z-index: 3;}
.grille2 .g3{z-index: 2;}
.grille2 .g4{z-index: 1;}
```



Ici, nous créons deux grilles identiques. Les éléments 1, 2, 3 et 4 des deux grilles vont se chevaucher. La première grille suit le comportement par défaut des éléments : le deuxième élément déclaré dans le code viendra se positionner au-dessus du premier, le troisième au-dessus du deuxième et etc.

Nous ajoutons ensuite des `z-index` aux éléments de notre deuxième grille afin de modifier l'ordre d'empilement des éléments.

Aligner et espacer les éléments de grille

Dans la leçon précédente, nous avons vu comme positionner ou placer des éléments dans une grille.

Il nous reste plus que deux choses à voir pour être totalement opérationnel avec nos grilles : comment aligner les éléments et comment les espacer.

En effet, par défaut, les éléments de grille vont s'étirer pour remplir tout l'espace dans lequel on les a positionnés quel que soit leur contenu. Nous allons pouvoir modifier ce comportement grâce aux propriétés d'alignement.

La plupart des propriétés d'alignement que nous allons voir ici devraient vous rappeler celles déjà vues dans la partie liée au modèle des boites flexibles et c'est tout à fait normal puisque le module CSS relatif à l'alignement (dans la spécification officielle) est commun à ces deux modèles.

Les axes des grilles

Le modèle des boites flexibles était un modèle unidimensionnel : c'est un modèle qui fonctionne avant tout selon un axe principal.

Le modèle des grilles est lui un modèle bidimensionnel : il va fonctionner selon deux axes qu'on va pouvoir manipuler de façon égale. On ne parlera donc plus d'axe « principal » et d'axe « secondaire ». A la place, nous parlerons d'axe de bloc et d'axe en ligne.

L'axe de bloc est l'axe selon lequel les éléments de type `block` sont disposés, c'est-à-dire l'axe vertical puisque les éléments de type `block` se placent par défaut les uns en dessous des autres.

L'axe en ligne est l'axe selon lequel les éléments de type `inline` sont disposés, c'est-à-dire l'axe horizontal puisque les éléments de type `inline` vont par défaut se placer les uns à côté des autres.

Avec les grilles, nous allons pouvoir gérer indifféremment l'alignement des éléments selon l'axe de bloc et l'axe en ligne.

Aligner les éléments ou les pistes d'une grille par rapport à l'axe de bloc

Commençons avec l'alignement des éléments ou des pistes selon l'axe de bloc. Nous allons pouvoir gérer cet alignement en utilisant les propriétés `align-items`, `align-self` et `align-content`.

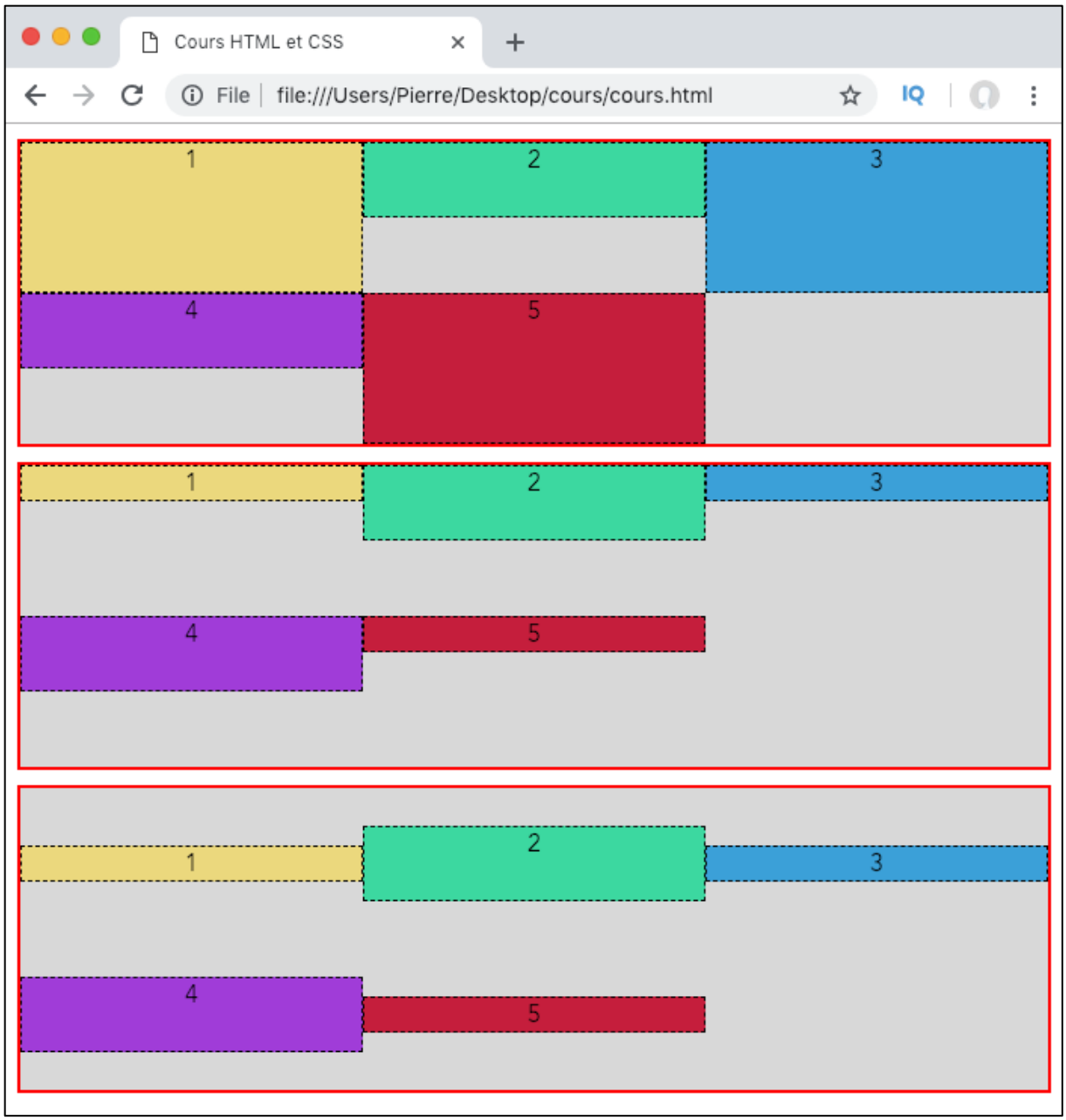
La propriété `align-items` permet de contrôler l'alignement des éléments de grille le long de l'axe de bloc.

On va devoir appliquer cette propriété à l'élément conteneur de grille et nous allons pouvoir lui passer les valeurs suivantes :

- stretch : Valeur par défaut. Les éléments de grille vont s'étirer pour occuper toute la hauteur des cellules dans lesquelles ils sont placés ;
- start : La hauteur des éléments va être déterminée par leur contenu et les éléments vont s'aligner à partir du départ (en haut) de l'espace dans lequel ils sont définis ;
- end : La hauteur des éléments va être déterminée par leur contenu et les éléments vont s'aligner à partir de la fin (en bas) de l'espace dans lequel ils sont définis ;
- center : La hauteur des éléments va être déterminée par leur contenu et les éléments vont s'aligner au milieu de l'espace dans lequel ils sont définis ;
- baseline : Les éléments vont être alignées de telle sorte à ce que leurs lignes de base soient alignées les unes par rapport aux autres.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid alignitemsstretch">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid  alignitemsstart">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid alignitemscenter">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif; font-size: 1em; text-align: center;
  margin: 0px; padding: 0px; box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(2,100px);
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  border: 1px dashed black;
}
.g1{background-color: #ED8;} /*Jaune*/
.g2{background-color: #4DA; height: 50px} /*Vert*/
.g3{background-color: #4AD;} /*Bleu*/
.g4{background-color: #A4D; height: 50px}/*Violet*/
.g5{background-color: #C24;}/*Rouge*/

.alignitemsstretch{align-items: stretch;}
.alignitemsstart{align-items: start;}
.alignitemscenter{align-items: center;}
```



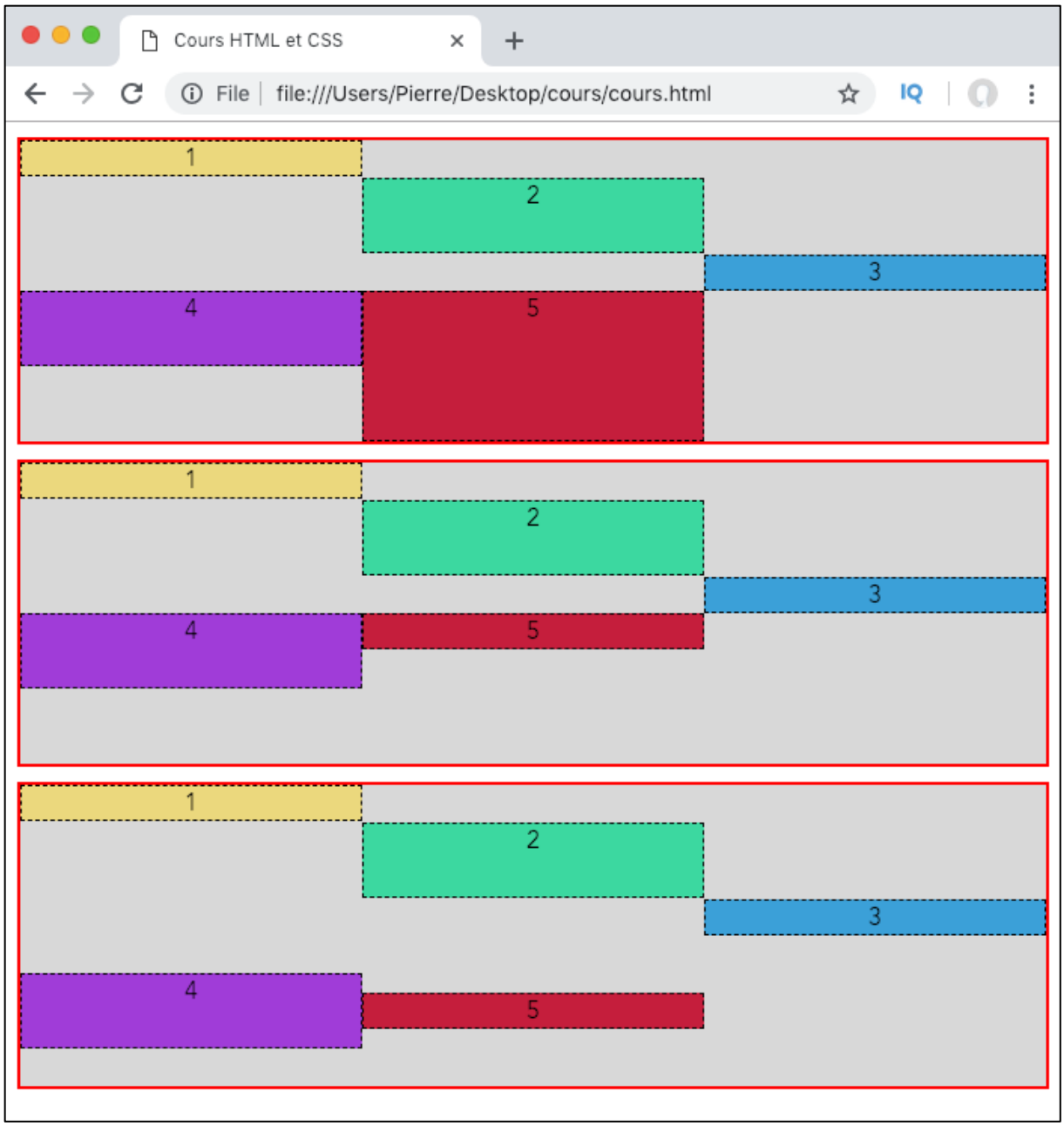
Ici, on impose à nos rangées de faire 100px de haut. La hauteur intrinsèque de nos éléments de grille, qui ne contiennent qu'un chiffre, est inférieure à la hauteur des rangées. On va donc pouvoir utiliser `align-items` pour aligner les éléments en hauteur.

On définit également des éléments de hauteurs différentes afin de bien voir l'effet des propriétés d'alignement que nous étudions. Notez ici que la valeur `stretch` n'a pas la capacité d'étirer un élément qui possède une taille explicite.

La propriété `align-self` permet de contrôler l'alignement d'un élément de grille en particulier dans l'axe de bloc. On va devoir cette fois-ci l'appliquer à l'élément qu'on souhaite aligner. Cette propriété va pouvoir prendre les mêmes valeurs que la propriété `align-items`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid alignitemsstretch">
            <div class="g1 alignselfstart">1</div>
            <div class="g2 alignselfcenter">2</div>
            <div class="g3 alignselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
          </div>

          <div class="conteneur-grid alignitemsstart">
            <div class="g1 alignselfstart">1</div>
            <div class="g2 alignselfcenter">2</div>
            <div class="g3 alignselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
          </div>

          <div class="conteneur-grid alignitemscenter">
            <div class="g1 alignselfstart">1</div>
            <div class="g2 alignselfcenter">2</div>
            <div class="g3 alignselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif; font-size: 1em; text-align: center;
  margin: 0px; padding: 0px; box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(2,100px);
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  border: 1px dashed black;
}
.g1{background-color: #ED8;} /*Jaune*/
.g2{background-color: #4DA; height: 50px} /*Vert*/
.g3{background-color: #4AD;} /*Bleu*/
.g4{background-color: #A4D; height: 50px}/*Violet*/
.g5{background-color: #C24;}/*Rouge*/

.alignitemsstretch{align-items: stretch;}
.alignitemsstart{align-items: start;}
.alignitemscenter{align-items: center;}

.alignselfstart{align-self: start;}
.alignselfcenter{align-self: center;}
.alignselfend{align-self: end;}
```



Si les pistes d'une grille n'occupent pas toute la hauteur dans la grille, alors on va pouvoir les aligner dans le conteneur selon l'axe de bloc, c'est-à-dire choisir comment distribuer l'espace restant en hauteur. Nous allons pouvoir faire cela en utilisant la propriété `align-content`.

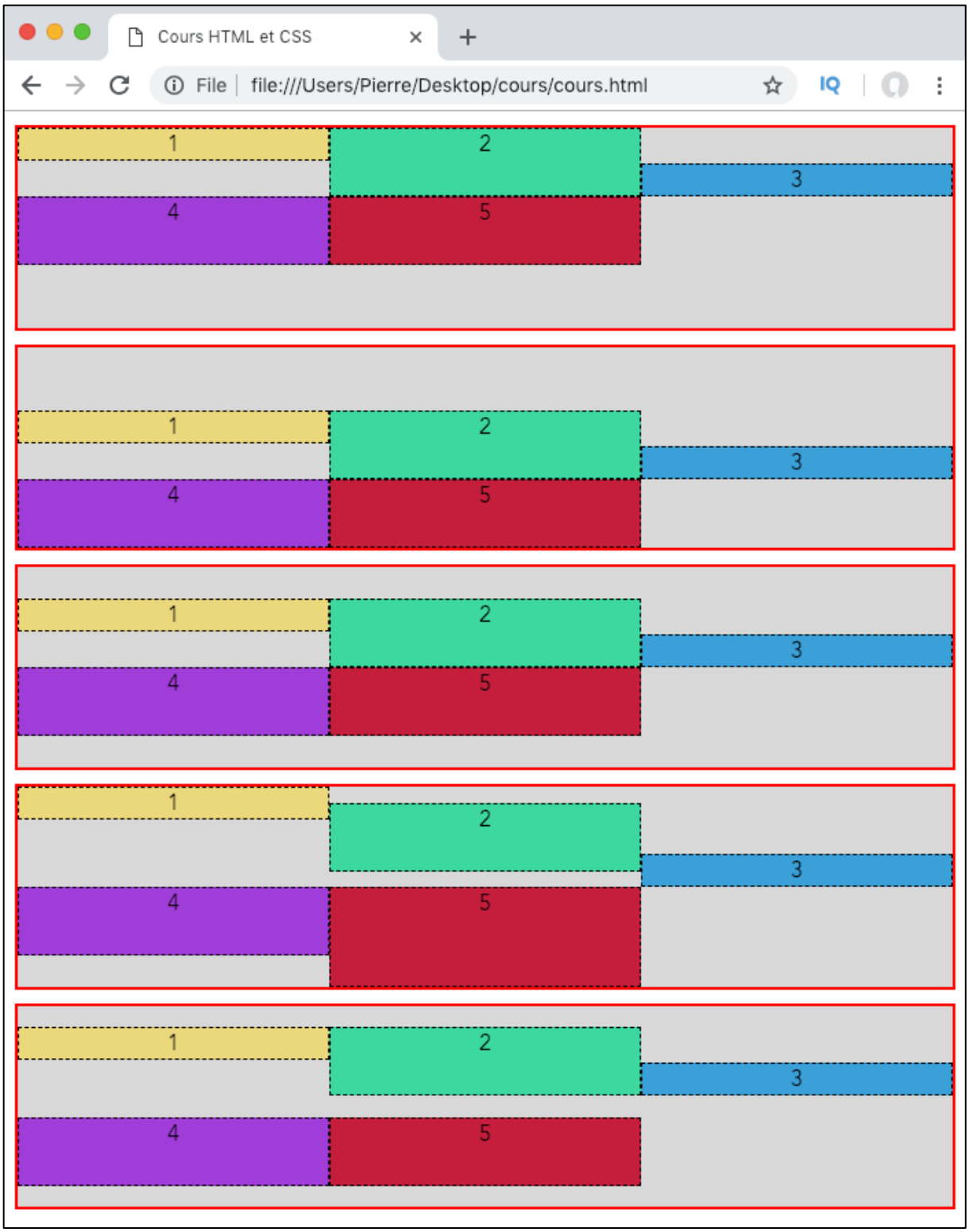
Cette propriété va être appliquée à l'élément conteneur et nous allons pouvoir lui passer les valeurs suivantes :

- `start` : La hauteur des pistes va être définie par la hauteur des éléments et les pistes vont s'empiler au début du conteneur selon l'axe de bloc c'est-à-dire en haut pour un langage dont l'écriture se fait de haut en bas ;
- `end` : La hauteur des pistes va être définie par la hauteur des éléments et les pistes vont s'empiler à la fin du conteneur selon l'axe de bloc c'est-à-dire en bas pour un langage dont l'écriture se fait de haut en bas ;

- `center` : La hauteur des pistes va être définie par la hauteur des éléments et les pistes vont s'empiler au centre du conteneur selon l'axe de bloc ;
- `baseline` : Les pistes vont être alignées de telles sortes à ce que les lignes de base soient alignées selon l'axe de bloc ;
- `stretch` : Les pistes vont s'étirer pour prendre toute la hauteur disponible dans le conteneur (sauf si une hauteur autre que `auto` a été explicitement définie) ;
- `space-between` : L'espace disponible dans le conteneur va être reparti équitablement entre les différentes pistes selon l'axe de bloc. La première et la dernière pistes vont être collées contre les bords du conteneur ;
- `space-around` : L'espace disponible dans le conteneur va être reparti équitablement entre les différentes pistes selon l'axe de bloc. L'espace entre les bords du conteneur et la première et la dernière piste sera deux fois moins important qu'entre deux pistes ;
- `space-evenly` : L'espace disponible dans le conteneur va être reparti équitablement entre les différentes pistes et entre une piste et un bord du conteneur selon l'axe de bloc.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid aligncontentstart">
            <div class="g1 alignselfstart">1</div>
            <div class="g2 alignselfcenter">2</div>
            <div class="g3 alignselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid aligncontentend">
            <div class="g1 alignselfstart">1</div>
            <div class="g2 alignselfcenter">2</div>
            <div class="g3 alignselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid aligncontentcenter">
            <div class="g1 alignselfstart">1</div>
            <div class="g2 alignselfcenter">2</div>
            <div class="g3 alignselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid aligncontentstretch">
            <div class="g1 alignselfstart">1</div>
            <div class="g2 alignselfcenter">2</div>
            <div class="g3 alignselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid aligncontentevenly">
            <div class="g1 alignselfstart">1</div>
            <div class="g2 alignselfcenter">2</div>
            <div class="g3 alignselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif; font-size: 1em; text-align: center;
  margin: 0px; padding: 0px; box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(2,auto);
  height: 150px;
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  border: 1px dashed black;
}
.g1{background-color: #ED8;} /*Jaune*/
.g2{background-color: #4DA; height: 50px} /*Vert*/
.g3{background-color: #4AD;} /*Bleu*/
.g4{background-color: #A4D; height: 50px}/*Violet*/
.g5{background-color: #C24;}/*Rouge*/

.alignitemsstretch{align-items: stretch;}
.alignitemsstart{align-items: start;}
.alignitemscenter{align-items: center;}

.alignselfstart{align-self: start;}
.alignselfcenter{align-self: center;}
.alignselfend{align-self: end;}

.aligncontentstart{align-content: start;}
.aligncontentend{align-content: end;}
.aligncontentcenter{align-content: center;}
.aligncontentstretch{align-content: stretch;}
.aligncontentevenly{align-content: space-evenly;}
```



Ici, on définit des conteneurs de 150px de haut et une hauteur `auto` pour nos rangées qui vont donc adapter leur hauteur à leur contenu.

J'utilise la valeur `auto` ici car c'est la seule valeur que `align-content : stretch` va pouvoir surcharger et car je sais que la hauteur de mes éléments est inférieure à la hauteur du conteneur et qu'on pourra donc bien voir l'effet de la propriété `align-content`.

J'ai également conservé les propriétés d'alignement des éléments dans les pistes et les éléments de différentes tailles afin que vous puissiez bien voir l'effet de chaque propriété. Je vous conseille ici d'inspecter vos conteneurs avec les outils pour développeur de votre navigateur afin de bien voir la hauteur et l'alignement de nos différentes pistes.

Aligner les éléments ou les pistes d'une grille par rapport à l'axe en ligne

De façon similaire à l'alignement selon l'axe de bloc, nous allons pouvoir gérer l'alignement ou plus exactement la justification selon l'axe en ligne des éléments et des pistes en utilisant trois propriétés qui sont `justify-items`, `justify-self` et `justify-content`.

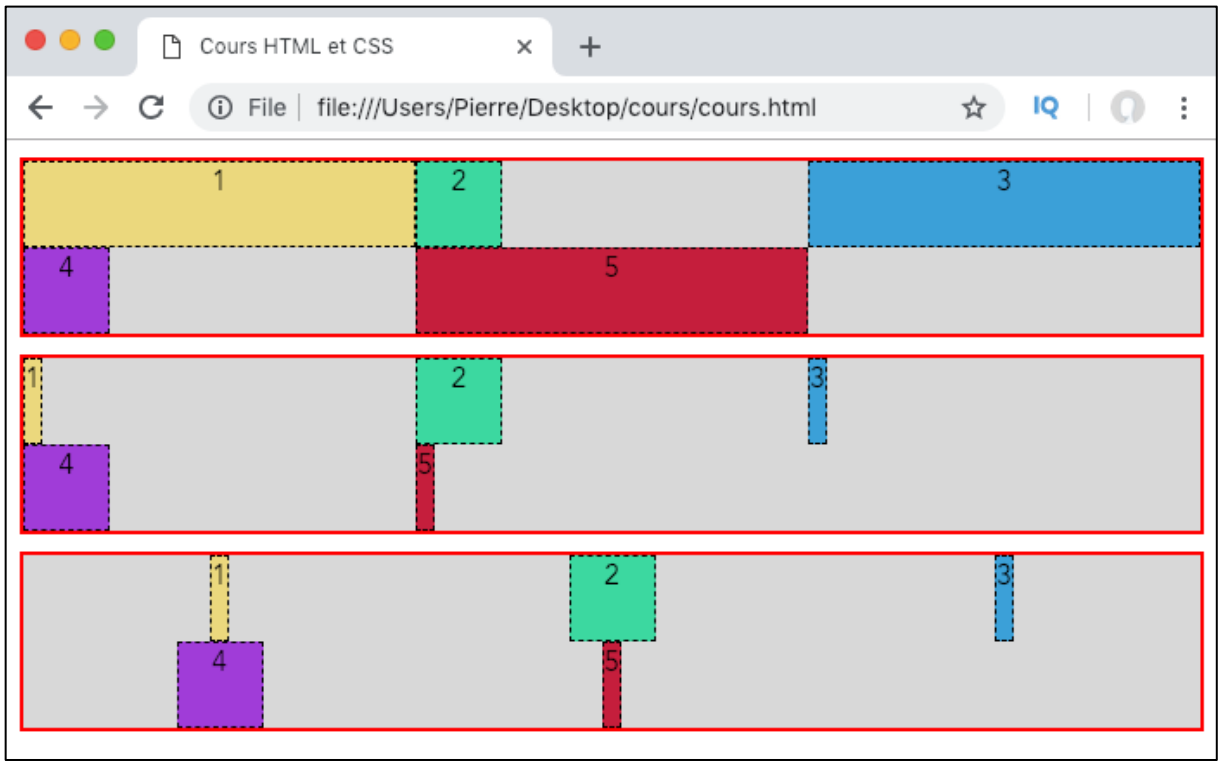
La propriété `justify-items` va nous permettre de contrôler l'alignement de tous les éléments dans le conteneur selon l'axe en ligne.

Nous allons devoir appliquer cette propriété à notre élément grille conteneur et allons pouvoir lui passer les mêmes valeurs qu'à la propriété `align-items` à savoir :

- `stretch` : Valeur par défaut. Les éléments de grille vont s'étirer pour occuper toute la largeur des cellules dans lesquelles ils sont placés ;
- `start` : Les éléments vont se placer au début du conteneur selon l'axe en ligne (à gauche pour un langage qui va de gauche à droite) ;
- `end` : Les éléments vont se placer à la fin du conteneur selon l'axe en ligne (à droite pour un langage qui va de gauche à droite) ;
- `center` : Les éléments vont être centrés dans le conteneur selon l'axe en ligne ;
- `baseline` : La ligne de base des éléments va être alignée.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid justifyitemsstretch">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid justifyitemsstart">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid justifyitemscenter">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif; font-size: 1em; text-align: center;
  margin: 0px; padding: 0px; box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(2,50px);
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  border: 1px dashed black;
}
.g1{background-color: #ED8;} /*Jaune*/
.g2{background-color: #4DA; width: 50px} /*Vert*/
.g3{background-color: #4AD;} /*Bleu*/
.g4{background-color: #A4D; width: 50px}/*Violet*/
.g5{background-color: #C24;}/*Rouge*/

.justifyitemsstretch{justify-items: stretch;}
.justifyitemsstart{justify-items: start;}
.justifyitemscenter{justify-items: center;}
```



Cette fois-ci, afin de bien vous montrer l'effet des propriétés d'alignement dans l'axe en ligne, je définis des éléments de grille de largeurs diverses. Notez qu'une nouvelle fois la valeur `stretch` ne va pas pouvoir surcharger une largeur définie explicitement.

La propriété justify-self va elle nous permettre de contrôler l'alignement selon l'axe en ligne d'un élément en particulier. Nous allons appliquer cette propriété à l'élément de grille qu'on souhaite aligner et allons pouvoir lui passer les mêmes valeurs qu'à justify-items.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid justifyitemsstretch">
            <div class="g1 justifyselfstart">1</div>
            <div class="g2 justifyselfcenter">2</div>
            <div class="g3 justifyselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid justifyitemsstart">
            <div class="g1 justifyselfstart">1</div>
            <div class="g2 justifyselfcenter">2</div>
            <div class="g3 justifyselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid justifyitemscenter">
            <div class="g1 justifyselfstart">1</div>
            <div class="g2 justifyselfcenter">2</div>
            <div class="g3 justifyselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif; font-size: 1em; text-align: center;
  margin: 0px; padding: 0px; box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(2,50px);
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.conteneur-grid > div{
  border: 1px dashed black;
}
.g1{background-color: #ED8;} /*Jaune*/
.g2{background-color: #4DA; width: 50px} /*Vert*/
.g3{background-color: #4AD;} /*Bleu*/
.g4{background-color: #A4D; width: 50px}/*Violet*/
.g5{background-color: #C24;}/*Rouge*/

.justifyitemsstretch{justify-items: stretch;}
.justifyitemsstart{justify-items: start;}
.justifyitemscenter{justify-items: center;}

.justifyselfstart{justify-self: start;}
.justifyselfcenter{justify-self: center;}
.justifyselfend{justify-self: end;}
```

En, cas de conflit avec justify-items, la valeur de justify-self va être prioritaire puisque la sélection est plus précise.

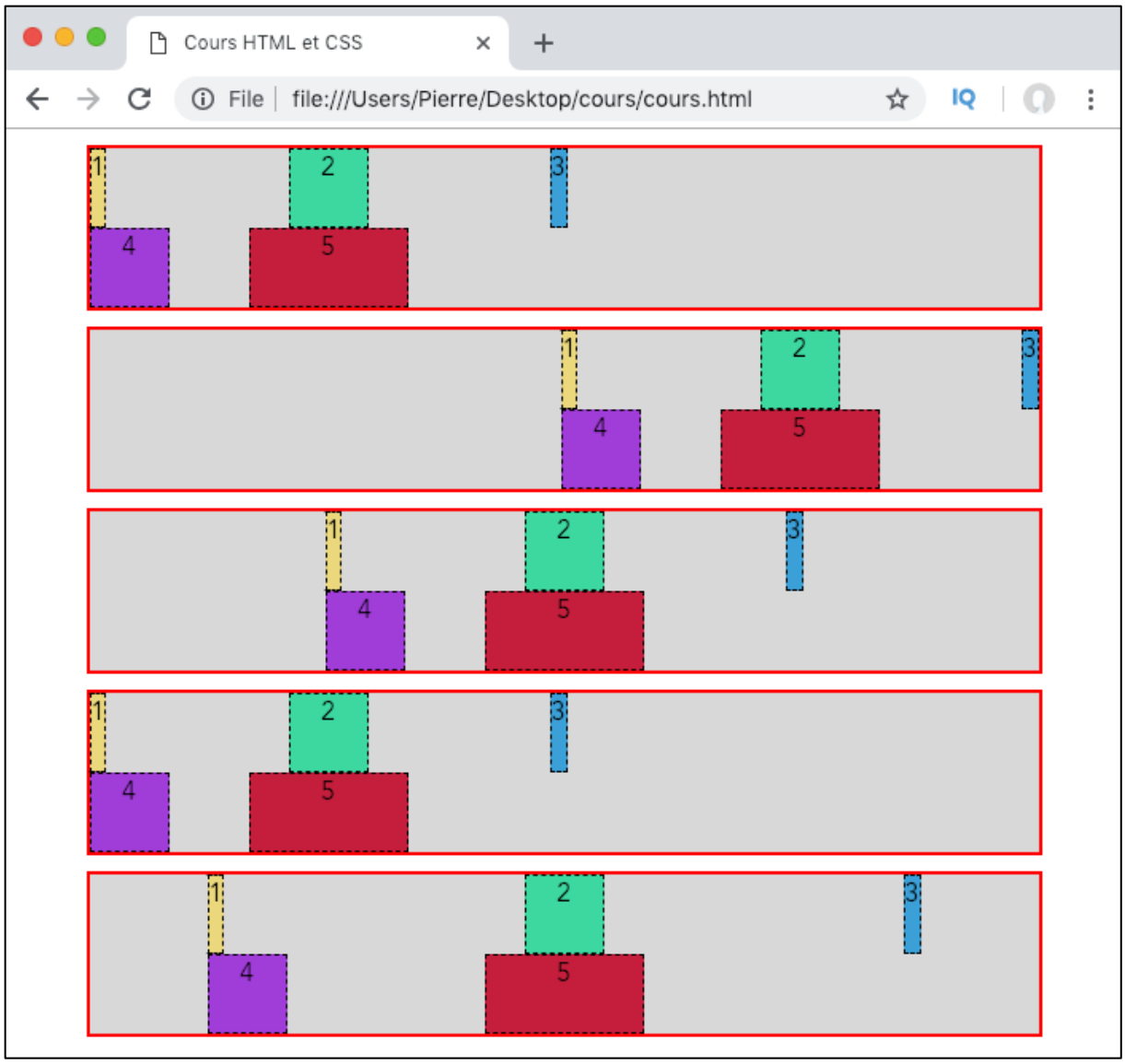
Finalement, si les pistes d'une grille n'occupent pas tout l'espace en largeur dans le conteneur, alors nous allons pouvoir les aligner c'est-à-dire choisir comment distribuer l'espace restant selon l'axe en ligne.

Nous allons pouvoir faire cela en utilisant la propriété justify-content qu'on va devoir appliquer à l'élément grille conteneur. Nous allons pouvoir lui passer les mêmes valeurs qu'à la propriété align-content à savoir :

- stretch : Les pistes vont s'étendre pour occuper toute la largeur disponible dans le conteneur ;
- start : Les pistes vont se grouper en début de conteneur (à gauche pour un langage dont l'écriture se fait de gauche à droite) ;
- end : Les pistes vont se grouper en fin de conteneur (à droite pour un langage dont l'écriture se fait de gauche à droite) ;
- center : Les pistes vont se grouper au centre du conteneur selon l'axe en ligne ;
- baseline : Les pistes vont s'aligner de telle sorte à ce que leurs lignes de bases soient alignées selon l'axe en ligne ;
- space-between : L'espace disponible dans le conteneur va être reparti équitablement entre les différentes pistes selon l'axe en ligne. La première et la dernière pistes vont être collées contre les bords du conteneur ;
- space-around : L'espace disponible dans le conteneur va être reparti équitablement entre les différentes pistes selon l'axe en ligne. L'espace entre les bords du conteneur et la première et la dernière piste sera deux fois moins important qu'entre deux pistes ;
- space-evenly : L'espace disponible dans le conteneur va être reparti équitablement entre les différentes pistes et entre une piste et un bord du conteneur selon l'axe en ligne.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid justifycontentstart">
            <div class="g1 justifyselfstart">1</div>
            <div class="g2 justifyselfcenter">2</div>
            <div class="g3 justifyselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid justifycontentend">
            <div class="g1 justifyselfstart">1</div>
            <div class="g2 justifyselfcenter">2</div>
            <div class="g3 justifyselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid justifycontentcenter">
            <div class="g1 justifyselfstart">1</div>
            <div class="g2 justifyselfcenter">2</div>
            <div class="g3 justifyselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid justifycontentstretch">
            <div class="g1 justifyselfstart">1</div>
            <div class="g2 justifyselfcenter">2</div>
            <div class="g3 justifyselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid justifycontentevenly">
            <div class="g1 justifyselfstart">1</div>
            <div class="g2 justifyselfcenter">2</div>
            <div class="g3 justifyselfend">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif; font-size: 1em; text-align: center;
  margin: 0px; padding: 0px; box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,100px);
  grid-template-rows: repeat(2,50px);
  width: 600px;
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px auto;
}
.conteneur-grid > div{
  border: 1px dashed black;
}
.g1{background-color: #ED8;} /*Jaune*/
.g2{background-color: #4DA; width: 50px} /*Vert*/
.g3{background-color: #4AD;} /*Bleu*/
.g4{background-color: #A4D; width: 50px}/*Violet*/
.g5{background-color: #C24;}/*Rouge*/

.justifyselfstart{justify-self: start;}
.justifyselfcenter{justify-self: center;}
.justifyselfend{justify-self: end;}

.justifycontentstart{justify-content: start;}
.justifycontentend{justify-content: end;}
.justifycontentcenter{justify-content: center;}
.justifycontentstretch{justify-content: stretch;}
.justifycontentevenly{justify-content: space-evenly;}
```



Pour voir l'effet de `justify-content`, il faut que les pistes de notre grille n'occupent pas toute la largeur du conteneur. Pour cela, on définit ici une grille de 3 colonnes faisant chacune 100px de large ainsi qu'une taille fixe de 600px pour le conteneur.

Ensuite, on utilise `justify-content` pour distribuer l'espace restant dans le conteneur. La valeur `start` va regrouper nos différentes colonnes au début du conteneur par exemple, c'est-à-dire à gauche pour une écriture de gauche à droite.

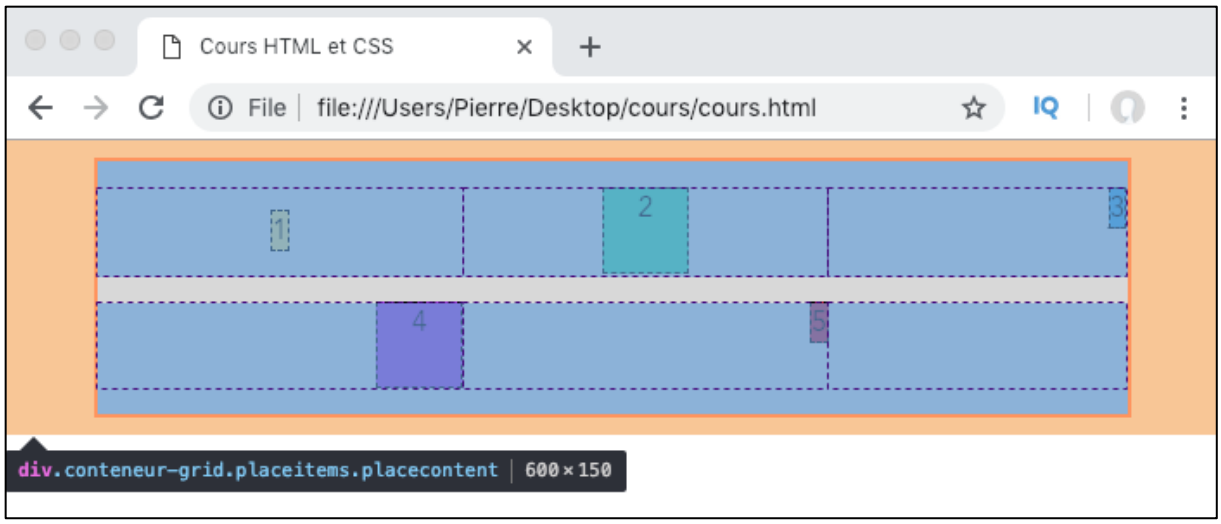
Propriétés d'alignement raccourcies

On va également pouvoir préciser l'alignement des éléments ou des pistes sur les deux axes en une seule fois grâce aux propriétés d'alignement raccourcies suivantes :

- `place-items` : notation raccourcie de `align-items` et `justify-items` ;
- `place-self` : notation raccourcie de `align-self` et `justify-self` ;
- `place-content` : notation raccourcie de `align-content` et `justify-content`.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid placeitems placecontent">
            <div class="g1 placeself">1</div>
            <div class="g2 placeself">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif; font-size: 1em; text-align: center;
  margin: 0px; padding: 0px; box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,auto);
  grid-template-rows: repeat(2,auto);
  width: 600px;
  height: 150px;
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px auto;
}
.conteneur-grid > div{
  border: 1px dashed black;
}
.g1{background-color: #ED8;} /*Jaune*/
.g2{background-color: #4DA; width: 50px; height: 50px;} /*Vert*/
.g3{background-color: #4AD;} /*Bleu*/
.g4{background-color: #A4D; width: 50px; height: 50px;}/*Violet*/
.g5{background-color: #C24;}/*Rouge*/

.placeitems{place-items: start end;}
.placeself{place-self: center center;}
.placecontent{place-content: space-evenly stretch;}
```



Ici, on définit une grille de 3 colonnes et 2 rangées avec des tailles `auto`. Notre conteneur de grille a des dimensions de 600 * 150px.

Les éléments vont être placés en haut à droite dans leur cellule grâce à `place-items: start end` à l'exception des éléments auxquels on applique `place-self: center center` qui vont eux être centrés. L'espace en hauteur va être réparti équitablement entre les pistes tandis que l'espace en largeur va être absorbé grâce à la valeur `stretch`.

Ci-dessus, j'ai affiché le découpage de ma grille et donc les pistes (délimitées par des tirets noirs) en utilisant les outils pour développeurs de Chrome (clic droit > inspecter l'élément) afin que vous puissiez bien voir où sont alignés les éléments dans les pistes et les pistes dans le conteneur.

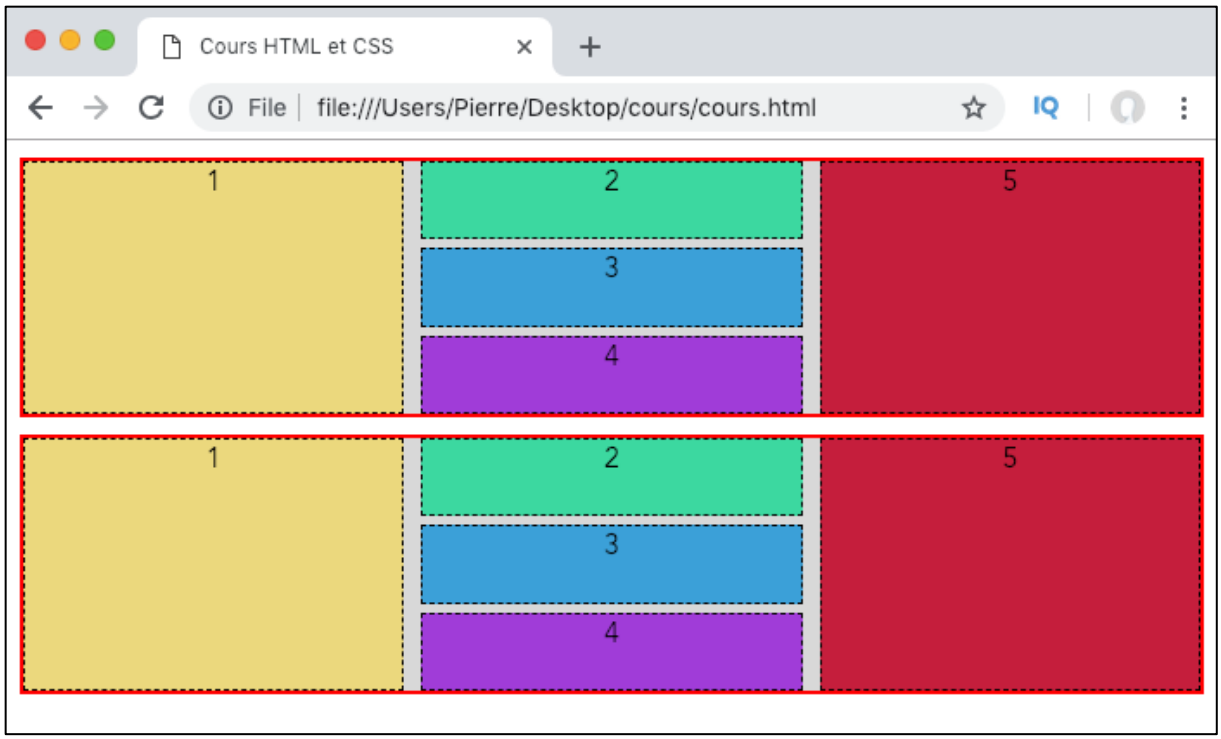
Définir les tailles des gouttières pour espacer les éléments de grille

Finalement, le modèle des grilles nous offre des propriétés pour éloigner les éléments de grille les uns des autres sans que les éléments ne s'éloignent des bords du conteneur de la grille.

On appelle cet espace entre les éléments des gouttières. La dimension des gouttières des cellules d'une grille va pouvoir être définie à l'aide des propriétés `column-gap` et `row-gap` ou avec la propriété raccourcie `gap`. Il faudra appliquer ces propriétés au conteneur.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <div class="conteneur-grid">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
        <div class="conteneur-grid grille2">
            <div class="g1">1</div>
            <div class="g2">2</div>
            <div class="g3">3</div>
            <div class="g4">4</div>
            <div class="g5">5</div>
        </div>
    </body>
</html>
```

```css
*{
  font-family: Avenir, sans-serif; font-size: 1em; text-align: center;
  margin: 0px; padding: 0px; box-sizing: border-box;
}
.conteneur-grid{
  display: grid;
  grid-template-columns: repeat(3,auto);
  grid-template-rows: repeat(2,auto);
  row-gap: 5px;
  column-gap: 10px;
  height: 150px;
  border: 2px solid red;
  background-color: #DDD;
  margin: 10px;
}
.grille2{
    gap: 5px 10px; /*row-gap column-gap*/
}
.conteneur-grid > div{
  border: 1px dashed black;
}
.g1{grid-row: 1 / 4;
    background-color: #ED8;} /*Jaune*/
.g2{grid-column: 2;
    background-color: #4DA;} /*Vert*/
.g3{grid-column: 2;
    background-color: #4AD;} /*Bleu*/
.g4{grid-column: 2;
    background-color: #A4D;}/*Violet*/
.g5{grid-column: 3;
    grid-row: 1 / 4;
    background-color: #C24;}/*Rouge*/
```



Comme vous pouvez le voir, les gouttières sont des espaces qui n'apparaissent qu'entre deux éléments. On peut ainsi espacer les éléments de grille les uns des autres sans les éloigner des bords du conteneur de grille.

EXERCICE #7 : Création d'une page à 3 colonnes avec des éléments flexibles

Le but de cet exercice est de créer un design de page complexe en réutilisant le maximum de concepts vus jusqu'ici : éléments structurants, flexbox, responsive design, grilles, etc. afin de voir une dernière fois comment les différentes notions vues jusqu'ici peuvent fonctionner ensemble en pratique.

Je vous propose donc de créer à nouveau une page complète qui va suivre deux dispositions différentes selon qu'elle soit affichée sur petits écrans (sur mobile) ou sur un écran de bureau.

Comme nous en avons désormais l'habitude, nous allons commencer par créer la version mobile de la page qui sera la version standard puis utiliserons les Media Queries pour créer une disposition différente pour l'affichage sur bureau.

La page va contenir les éléments suivants :

- 1 en-tête ou `header` qui va comprendre un menu ;
- 4 blocs de type `aside` ;
- 1 bloc de contenu de type `article` qui contiendra lui-même une structure complète ;
- 1pied de page ou `footer`.

Notre page suivra un design vertical avec une seule colonne pour l'affichage sur mobile et un design sur 3 colonnes pour l'affichage sur bureau.

Structure HTML de notre page

Comme d'habitude, commençons avec la structure HTML de notre page et notamment avec la structure minimale d'une page HTML et sans oublier la `meta name="viewport"`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <meta name="viewport"
              content="width=device-width,initial-scale=1.0,user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>

    </body>
</html>
```

Ensuite, nous allons ajouter un titre principal dans notre page et récupérer la structure du menu créé précédemment qui va très bien nous convenir. Nous allons placer le menu (mais pas le titre) dans un élément structurant header.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours HTML et CSS</title>
        <meta charset="utf-8">
        <meta name="viewport"
              content="width=device-width,initial-scale=1.0,user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Une page utilisant les grilles et le flexbox ? Facile !</h1>
        <header>
            <nav>
                <div class="conteneur-nav">
                <label for="mobile">Afficher / Cacher le menu</label>
                <input type="checkbox" id="mobile" role="button">
                  <ul>
                    <li class="deroulant"><a href="#">Cours Complets  </a>
                      <ul class="sous">
                        <li><a href="#">Cours HTML et CSS</a></li>
                        <li><a href="#">Cours JavaScript</a></li>
                        <li><a href="#">Cours PHP et MySQL</a></li>
                      </ul>
                    </li>
                    <li class="deroulant"><a href="#">Articles  </a>
                      <ul class="sous">
                        <li><a href="#">CSS display</a></li>
                        <li><a href="#">CSS position</a></li>
                        <li><a href="#">CSS float</a></li>
                      </ul>
                    </li>
                    <li><a href="#">Contact</a></li>
                    <li><a href="#">A propos</a></li>
                  </ul>
                </div>
            </nav>
        </header>
    </body>
</html>
```

Sous le header, nous allons ajouter nos quatre éléments aside et notre élément article et allons placer tout cela dans un élément section.

Notre élément article va lui-même contenir un header, un menu de navigation interne, un corps qu'on va à nouveau placer dans un élément article et un footer.

```
    </header>

    <section class="conteneur">
        <aside class="g1"><p>Aside n°1</p></aside>
        <aside class="g2"><p>Aside n°2</p></aside>
        <article class="g3 sousconteneur">
            <header>
                <h1>Titre de l'article</h1>
                <p>Temps de lecture : 3mn</p>
                <nav>
                    <ul>
                        <li><a href="#">Chapitre 1</a></li>
                        <li><a href="#">Chapitre 2</a></li>
                        <li><a href="#">Chapitre 3</a></li>
                    </ul>
                </nav>
            </header>
            <article>
                <p>Le sujet de cet exercice est d'apprendre à créer
                une page avec un design complexe et flexibles</p>
            </article>
            <footer>
                <p>A propos de l'auteur : Pierre Giraud...</p>
            </footer>
        </article>
        <aside class="g4"><p>Aside n°3</p></aside>
        <aside class="g5"><p>Aside n°4</p></aside>
    </section>
```

Finalement, nous allons ajouter notre élément `footer` sous notre élément `section`.

```
    <footer>
        <p>Page HTML / CSS avec grid et flexbox</p>
        <p>© Pierre Giraud / https://www.pierre-giraud.com</p>
    </footer>
</body>
```

Pour cette partie HTML, nous n'allons pas nous attarder sur le contenu comme on a pu le faire dans l'exercice de création d'un CV puisque ce n'est pas ce qui compte ici. Ce qui nous importe ici va être de créer des dispositions différentes. Libre à vous ensuite de placer le contenu que vous voulez dans les différents blocs HTML et de le mettre en forme !

Mise en forme CSS de notre page version mobile

Notre page en version mobile ne va posséder qu'une seule colonne. Les différents éléments vont donc être disposés selon un seul axe. Dans cette situation, on préfèrera utiliser le modèle des boites flexibles plutôt que les grilles.

Commençons déjà par notre traditionnel reset CSS et par ajouter quelques styles globaux.

```css
*{
  font-family: Avenir, sans-serif;
  font-size: 1em;
  margin: 0px;
  padding: 0px;
  box-sizing: border-box;
}
```

Ensuite, nous allons reprendre le code de notre menu créé précédemment et allons faire quelques ajustements.

Tout d'abord, étant donné que notre élément `nav` est désormais dans un élément `header`, nous allons plutôt appliquer la `position : sticky` au `header` afin que le menu continue à rester collé lorsqu'on défile dans la page.

Nous allons également ici modifier les sélecteurs contenant `nav` en préférant l'écriture `body > header > nav` pour bien différencier les styles de notre menu de navigation principal et celui interne à notre élément `article`.

```css
body > header{
    position: sticky;
    top: 0px;
}
body > header > nav{
    width: 100%;
    height: 42px;
    margin: 0 auto;
}
.conteneur-nav{
    position: absolute;
    width: 100%;
}
body > header > nav input[type=checkbox]{
    display: none;
}
body > header > nav label{
    display: inline-block;
    width: 100%;
    padding: 10px 0px;
    text-align: center;
    background-color: RGBa(220,220,220,0.5);
}
body > header > nav ul{
    display: none;
    list-style-type: none;
    background-color: #555;
}
body > header > nav input[type=checkbox]:checked + ul{
    display: flex;
    flex-flow : column wrap;
}
body > header > nav ul li{
    flex: 1 1 auto;
    text-align: center;
}
body > header > nav > div > ul > li > a{
    color: white;
}
body > header > nav a{
    display: block;
    text-decoration: none;
    color: black;
    padding: 10px 0px;
}
```

Voilà tout pour les ajustements relatifs au menu. Passons maintenant au corps de page qui va utiliser le modèle des boites flexibles.

Ici, nous allons déjà commencer par appliquer un `display : flex` et un `flex-flow : column wrap` à notre élément `section class="conteneur"` pour obtenir une orientation en colonne et transformer ses enfants directs en éléments flexibles.

```css
.conteneur{
    display: flex;
    flex-flow: column wrap;
}
```

Ensuite, nous allons simplement nous contenter d'ajouter une couleur de fond et une hauteur minimale à nos différents éléments `aside`.

```css
.g1, .g4{background-color: RGBa(120,205,225,0.4);}
.g2, .g5{background-color: RGBa(225,100,175,0.4);}
.g1, .g2, .g4, .g5{min-height: 100px;}
```

On va maintenant mettre en forme notre élément `article class="g3 sousconteneur"`. Nous allons déjà transformer cet élément en conteneur flexible et allons à nouveau choisir l'axe vertical comme axe principal.

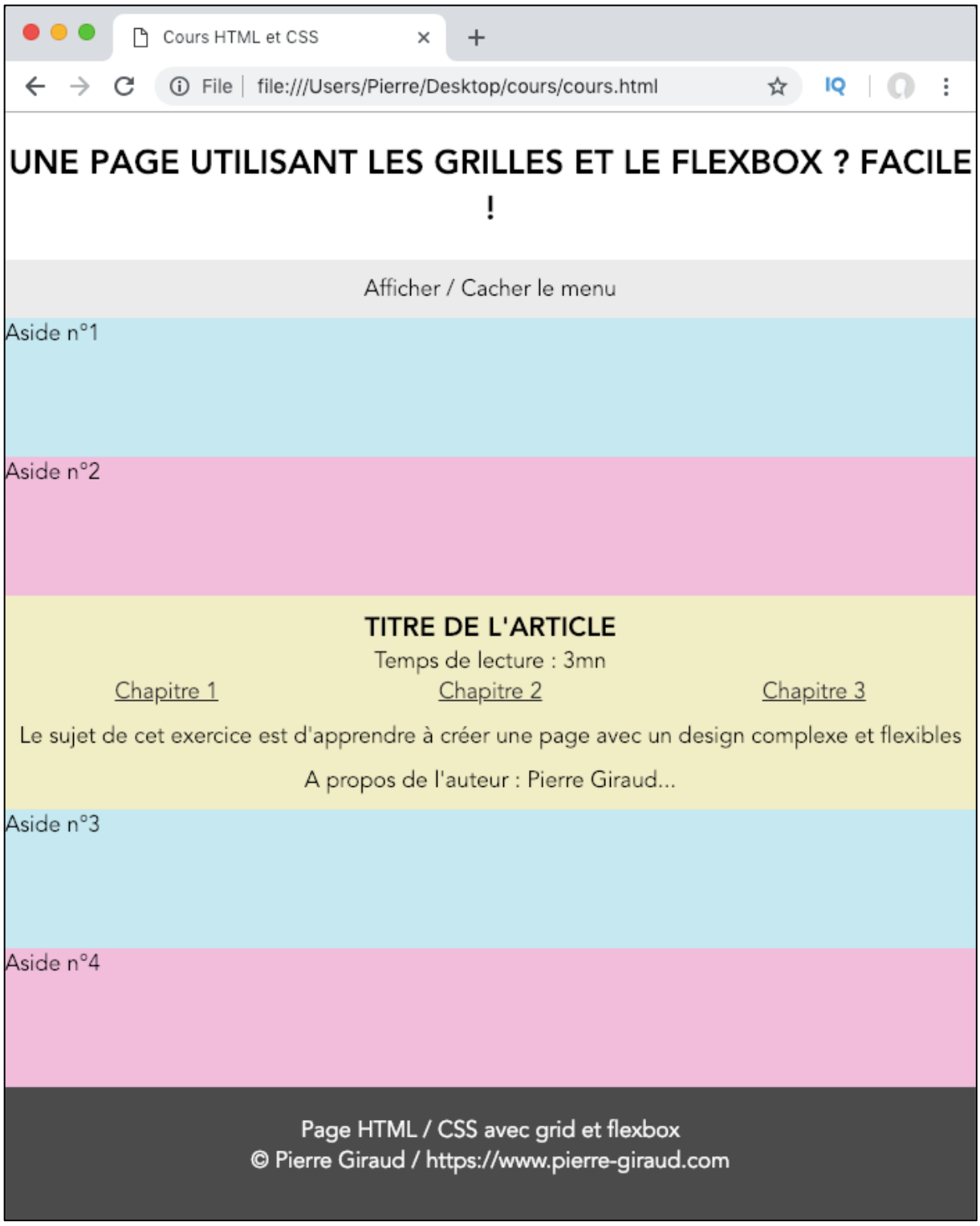
On choisit d'afficher la navigation interne de l'article en ligne. Pour cela, on applique à nouveau un `display : flex` à notre élément de liste `ul` qui fait office de navigation et on choisit cette fois-ci l'axe horizontal comme axe principal. On espace régulièrement les éléments dans l'axe principal avec `justify-content: space-around`.

```css
.sousconteneur{
    display: flex;
    flex-flow: column wrap;
    text-align: center;
    background-color: RGBa(225,215,120,0.4);
}
.sousconteneur h1{
    font-size: 1.2em;
    margin: 10px 0px 0px 0px;
}
.sousconteneur nav ul{
    display: flex;
    flex-flow: row wrap;
    list-style-type: none;
    justify-content: space-around;
}
.sousconteneur nav a{
    display: block;
    color: #333;
}
.sousconteneur article{
    margin: 10px;
}
.sousconteneur footer{
    margin-bottom: 10px;
}
```

Finalement, on applique une mise en forme très basique à notre `footer` principal qui ne va nous servir ici qu'à afficher une notice de copyright.

```css
body > footer{
    width: 100%;
    height: 100px;
    background-color: #555;
    color: white;
    text-align:center;
    padding: 20px;
}
```

Voilà tout pour la partie mobile. Passons maintenant à la version bureau sur trois colonnes de notre page.



Mise en forme CSS de notre page version bureau

Pour la version bureau de notre page, nous allons vouloir positionner et aligner les éléments sur les deux axes. Nous allons donc préférer l'utilisation des grilles au modèle des boites flexibles pour les blocs de la page.

Nous utiliserons cependant également le flexbox à l'intérieur de certains blocs pour créer des éléments de grille flexibles.

Commençons déjà par définir une règle `@media screen and (min-width: 980px)` et par remplacer les sélecteurs de notre menu principal `nav` comme précédemment par `body > header > nav`.

```
@media screen and (min-width: 980px){
    .conteneur-nav{
        position: static;
    }
    body > header > nav label, nav input{
        display: none;
    }
    body > header > nav input[type=checkbox]:checked + ul, body > header > nav ul{
        display: flex;
        flex-flow: row wrap;
        background-color: RGBa(220,220,220,0.5);
    }
    body > header > nav ul li{
        position: relative;
    }
    body > header > nav > div > ul > li > a{
        color: black;
    }
    body > header > nav a{
        border-bottom: 2px solid transparent;
    }
    body > header > nav a:hover{
        color: orange;
        border-bottom: 2px solid gold;
    }
    .sous{
        display: none;
        box-shadow: 0px 1px 2px #CCC;
        background-color: white;
        position: absolute;
        width: 100%;
    }
    body > header > nav > div > ul li:hover .sous{
        display: flex;
        flex-flow: column wrap;
    }
```

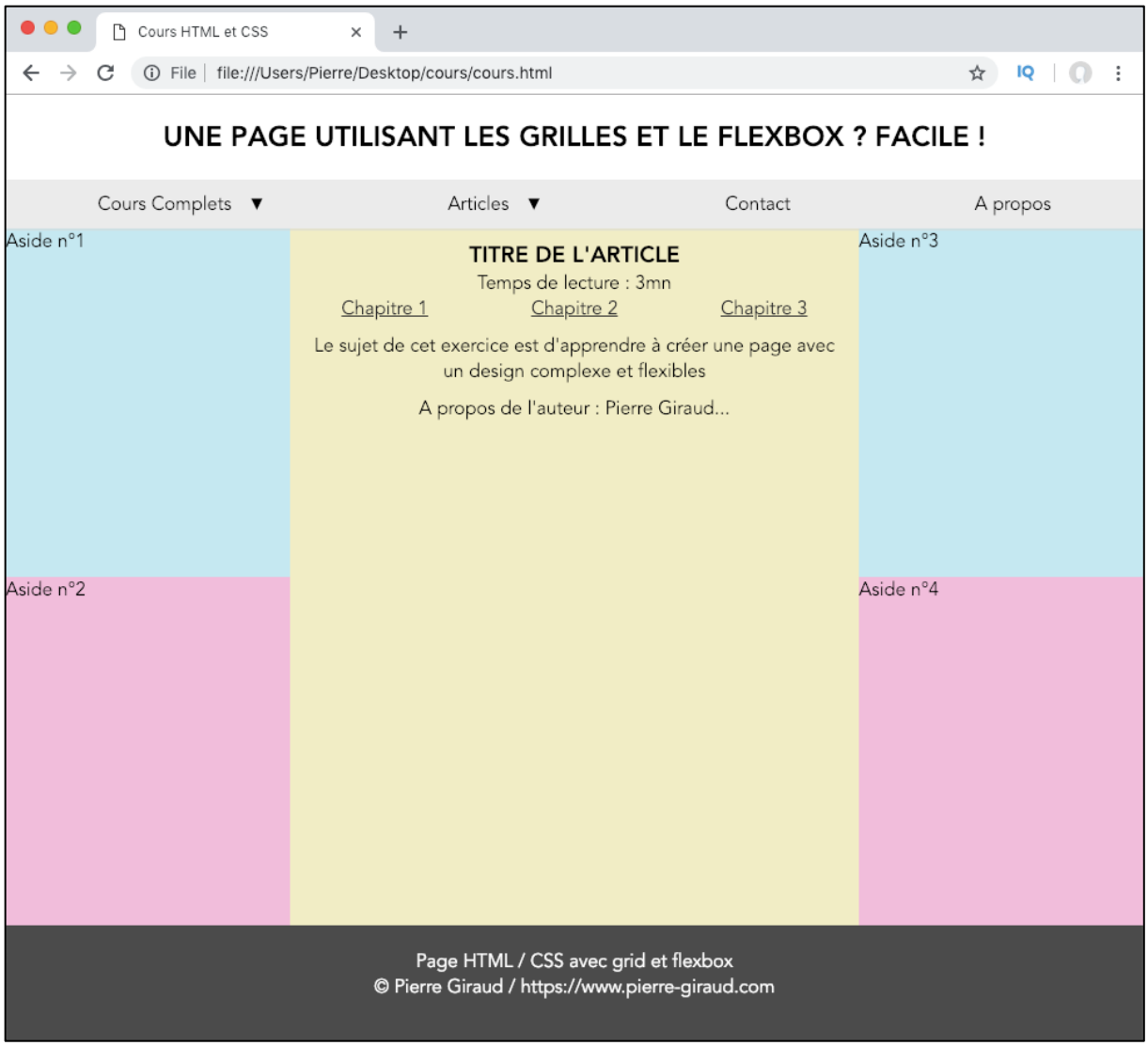
Ensuite, nous allons cette fois-ci appliquer un `display : grid` à notre élément `section class="conteneur"`. On va donc ici créer une grille à trois colonnes avec une colonne centrale deux fois plus large que les colonnes gauche et droite. Notre grille va également posséder deux rangées qui devront faire à minima 300px de haut.

```css
.conteneur{
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
    grid-template-rows: repeat(2,minmax(300px, 1fr));
}
```

On va ensuite positionner nos différents éléments dans notre grille. On va ici vouloir que l'élément `article` occupe toute la colonne centrale et allons positionner les éléments `aside` de chaque côté.

```css
.g1{
    grid-column: 1 / 2;
}
.g2{
    grid-column: 1 / 2;
    grid-row: 2 / 3;
}
.g3{
    grid-column: 2 / 3;
    grid-row: 1 / 3;
}
.g4{
    grid-column : 3 / 4;
}
.g5{
    grid-column : 3 / 4;
    grid-rows: 2 / 3;
}
```

Et voilà tout ! Notre élément `article` est toujours un conteneur flexible et les propriétés définies pour l'affichage mobile s'appliquent toujours ici et nous conviennent très bien pour l'affichage sur bureau. Pas besoin d'aller plus loin donc.



Nous avons donc créé une page totalement responsive et avec une structure d'affichage complexe en utilisant certaines des notions les plus récentes du HTML et du CSS !

PARTIE XVII

Evolution et futur du CSS

Évolution du CSS : vers un langage de programmation ?

Ces dernières années, le CSS a évolué de plus en plus vite et dans une direction le rapprochant de plus en plus d'un langage de programmation à part entière à l'inverse d'un simple langage de présentation. L'introduction des concepts de fonctions et de variables ou "propriétés personnalisées" ("custom properties") comme le CSS les appelle en est la preuve.

Certaines des fonctionnalités présentées dans cette dernière partie sont encore en développement ou instables, mais il me semblait tout de même très intéressant de vous présenter le futur du CSS et la direction dans laquelle le langage progresse.

Cette dernière partie va être pour moi l'opportunité de vous présenter certaines technologies très récentes ou à venir afin que vous ayez une vue véritable vue d'ensemble sur l'état du CSS aujourd'hui et sur la direction dans laquelle le langage avance. Nous allons donc discuter des sujets suivants :

- Les propriétés personnalisées ou "variables" CSS ;
- Les fonctions CSS ;
- L'imbrication des sélecteurs CSS ;
- L'héritage étendu des propriétés en CSS

Les fonctions CSS

Qu'est-ce qu'une fonction ?

Une fonction est un bloc de code cohérent dont le but est d'effectuer une tâche précise et qui renvoie un résultat.

Dans la plupart des langages informatiques utilisant les fonctions, on peut distinguer deux grands types de fonctions :

- Les fonctions prêtes à l'emploi, c'est-à-dire des fonctions dont la tâche et le code sont déjà définis. Nous n'avons qu'à appeler ces fonctions pour utiliser leurs fonctionnalités ;
- Les fonctions personnalisées ou conçues par l'utilisateur, c'est-à-dire nous, afin d'effectuer les tâches qui nous intéressent.

Nous avons déjà utilisé des fonctions dans ce cours sans vraiment le savoir, comme les fonctions `rgb()` ou `linear-gradient()` par exemple.

En CSS, nous allons nous contenter d'utiliser des fonctions prédéfinies par le langage. Nous passerons le résultat de ces fonctions en valeur de nos propriétés. Par exemple, lorsqu'on écrit `color : rgb(255,255,255)`, vous devez bien comprendre que derrière `rgb()` se cache tout un code complexe qui va s'exécuter afin de calculer et de renvoyer une couleur exploitable par la propriété `color`.

Dans ce cas précis, pour que la fonction `rgb()` fonctionne correctement, nous allons devoir lui fournir trois données qu'on appelle également des arguments. Ici, ces données sont des chiffres compris entre 0 et 255 et qui servent à indiquer les niveaux d'intensité de rouge, de vert et de bleu qui doivent être mélangées par `rgb()` afin que la fonction détermine la couleur finale.

Vous devez imaginer l'intérieur d'une fonction comme une série cohérente d'instructions qui vont être exécutées dès qu'on va appeler la fonction. Pour appeler ou exécuter une fonction, il suffit de mentionner son nom suivi d'un couple de parenthèses et éventuellement d'une ou plusieurs arguments dans ces parenthèses selon la fonction.

Quel est l'intérêt d'utiliser des fonctions ?

Le grand intérêt des fonctions est d'éviter la réécriture du code et d'avoir à inventer la roue : lorsqu'on utilise un langage, on effectue souvent les mêmes opérations. Certains créateurs de langages s'en sont rendus compte et ont donc commencé à intégrer à leur langage des fonctions dont le but était précisément d'effectuer telle ou telle opération. Plutôt que d'avoir à créer le script pour effectuer telle opération, les utilisateurs n'ont alors plus qu'à utiliser la fonction correspondante pour parvenir au même résultat.

Dans certains autres cas, nous allons avoir des besoins plus spécifiques en tant que développeurs mais allons nous rendre compte que nous effectuons plusieurs fois le même type d'opérations pour un projet. Dans ce cas, si aucune fonction prédéfinie répondant à

notre besoin n'existe, il va être intéressant de créer nos propres fonctions (si le langage utilisé nous le permet) pour ne pas avoir à réécrire l'ensemble du code lié à la fonction à chaque fois et ainsi gagner du temps et créer un code plus facilement maintenable.

Liste des fonctions CSS disponibles

Durant ce cours, nous avons déjà eu l'opportunité de rencontrer une bonne partie des fonctions CSS. Complétons la liste !

Les fonctions liées à la couleur

Nous avons déjà rencontré les fonctions `rgb()`, `rgba()`, `hsl()`, `hsla()` qui permettent de créer des couleurs.

En plus de celles-ci, la fonction `hwb()` peut également être utilisée pour fournir une valeur de couleur. Pour cela, on devra spécifier les composants de teinte, de blancheur et de noirceur de la couleur, ainsi qu'une valeur alpha.

Les fonctions liées aux images

- La fonction CSS `image()` permet d'inclure une image dans une page. Cette fonction est similaire à la fonction `url()` mais fournit des options supplémentaires comme la possibilité de spécifier des images de secours (au cas où le navigateur ne pourrait pas afficher l'image préférée) ou encore de découper une partie d'une image ;
- La fonction `blur()` est utilisée pour flouter une image. On l'utilisera généralement avec la propriété `filter` ;
- La fonction CSS `brightness()` est utilisée avec la propriété `filter` pour ajuster la luminosité d'une image ;
- La fonction CSS `saturate()` est utilisée avec la propriété `filter` pour ajuster le niveau de saturation d'une image ;
- La fonction CSS `hue-rotate()` est utilisée avec la propriété `filter` pour appliquer un changement ou une "rotation" de teinte à une image. On précisera la rotation en deg. ;
- La fonction CSS `sepia()` est utilisée avec la propriété `filter` pour convertir une image en sépia ;
- La fonction `grayscale()` permet de transformer une image en noir et blanc. On va lui passer un nombre entre 0 et 1 en valeur qui va représenter la proportion des couleurs qui devront être transformées en noir et blanc ;
- La fonction `invert()` est utilisée avec la propriété `filter` pour inverser les couleurs d'une image ;
- La fonction `opacity()` est utilisée avec la propriété `filter` pour appliquer une transparence aux échantillons d'une image. A ne pas confondre avec la propriété `opacity` ;
- La fonction `contrast()` est utilisée avec la propriété `filter` pour ajuster le contraste d'une image ;
- La fonction `drop-shadow()` est utilisée avec la propriété `filter` pour ajouter un effet d'ombre portée à une image.

Les fonctions liées aux dégradés

Nous avons déjà étudié les fonctions de création de dégradés qui sont les suivantes :

- La fonction linear-gradient() ;
- La fonction radial-gradient() ;
- La fonction repeating-linear-gradient() ;
- La fonction repeating-radial-gradient().

Les fonctions liées aux listes

Le CSS nous fournit trois fonctions pour modifier le comportement des listes ou d'autres éléments HTML pour qu'ils ressemblent à des listes.

- La fonction symbols() est utilisée pour définir un style de puce. On peut utiliser son résultat en valeur de la propriété list-style-type par exemple ;
- La fonction counter() permet d'afficher le compteur généré par un élément ;
- La fonction counters() permet d'afficher des compteurs imbriqués générés par un élément et ses parents.

Les fonctions liées aux transformations

Nous avons déjà travaillé avec des fonctions CSS permettant de créer des rotations, des mises à l'échelle etc.

Le CSS met à notre disposition les fonctions suivantes pour créer des transformations :

- Les fonctions permettant de créer des translations translate(), translateX(), translateY(), translate3d(), translateZ() ;
- Les fonctions permettant de créer des rotations rotate(), rotateX(), rotateY(), rotate3d(), rotateZ() ;
- Les fonctions permettant de créer des mises à l'échelle scale(), scaleX(), scaleY(), scale3d(), scaleZ() ;
- Les fonctions permettant de créer des torsions skew(), skewX(), skewY() ;
- Les fonctions matrix() et matrix3d() ;
- La fonction cubic-bezier() ;
- La fonction CSS perspective() définit la profondeur afin de donner une perspective à l'élément positionné en 3D.

Les fonctions liées aux figures géométriques

Le CSS dispose d'un module de spécification appelé "shape" qui décrit la façon dont on peut créer des formes géométriques en CSS.

Les fonctions circle(), ellipse(), polygon() et inset() retournent des valeurs de formes dites "basiques". On va pouvoir utiliser ces fonctions avec des propriétés comme shape-outside pour contrôler l'aspect extérieur (la forme) du contenu autour de l'élément ou avec clip-path pour découper le contenu de l'élément selon la forme définie par la valeur de la fonction.

Attention : ces fonctionnalités sont très récentes et le support par les navigateurs n'est pas encore complet.

Autres fonctions CSS

- La fonction `attr()` retourne la valeur d'une attribut d'un élément HTML. On utilisera généralement le résultat de fonction comme valeur pour la propriété `content` ;
- La fonction `calc()` permet d'utiliser le résultat de calculs dans les valeurs de propriétés CSS. Elle peut être utilisée à la place d'autres types d'unités lors de la définition de la largeur ou de la hauteur d'un élément ou encore pour des angles, des fréquences, etc. ;
- La fonction `url()` permet de spécifier la localisation d'un fichier. On peut l'utiliser pour indiquer l'emplacement d'une image ou d'une police ;
- La fonction `var()` permet de passer le contenu d'une propriété personnalisée (variable) à une propriété CSS. Nous allons reparler de cette fonction dans la prochaine leçon.

Les propriétés personnalisées ou variables CSS

Qu'est-ce qu'une propriété personnalisée ou variable CSS ?

Une variable en informatique est un conteneur nommé qui va pouvoir stocker différentes valeurs durant le script. Vous pouvez imaginer une variable comme une boite qui ne va pouvoir contenir qu'un objet à la fois mais dont on va pouvoir changer l'objet stocké lorsqu'on le souhaite.

Le principe d'utilisation des variables est relativement simple à comprendre : on affecte une valeur à un nom (à notre variable donc) pour ensuite utiliser le nom de la variable plutôt que la valeur partout dans notre code. Lors de l'analyse du code, le nom de la variable sera remplacé par la valeur avant d'afficher la page à l'utilisateur.

Le CSS a introduit il y a peu de temps ce concept de variable dans son langage. Les variables CSS sont également appelées "propriétés personnalisées". Notez que les "variables" CSS ne sont pas exactement des variables au même sens que dans la plupart des langages de programmation car certaines propriétés inhérentes aux variables sont manquantes en CSS et c'est la raison pour laquelle on les appelle plutôt "propriétés personnalisées".

Pourquoi utiliser des propriétés personnalisées ou variables CSS ?

Les principaux intérêts liés à l'utilisation des propriétés personnalisées en CSS sont la création d'un code plus clair et plus facilement maintenable. En effet, imaginez que vous travailliez sur un gros projet avec des feuilles de styles de plusieurs centaines ou milliers de lignes.

Si le projet possède une charte graphique cohérente, vous aurez à réutiliser les mêmes styles (couleur, taille de police, marges, etc.) à plusieurs endroits dans la feuille de style.

Dans le cas d'une feuille de style longue, il est facile de rapidement se perdre et donc l'approche sans utilisation de variable est peu adaptée. De plus, si un jour vous deviez modifier la couleur principale du thème par exemple, vous devrez modifier la valeur liée à cette couleur à travers toute la feuille de style ce qui rend le code difficilement maintenance au final.

Les variables résolvent ces deux problèmes : en effet, en utilisant les variables, nous allons affecter une valeur à une variable une bonne fois pour toutes puis ensuite utiliser le nom de notre variable pour gaffer référence à la valeur dans toute notre feuille de style.

Ainsi, si on doit un jour changer une valeur, nous n'aurons qu'à mettre à jour la valeur affectée à la variable dans le code et n'aurons pas à modifier le reste du code puisque le nom de la variable n'a pas à changer.

Ensuite, on peut donner n'importe quel nom à une variable. Cela permet de s'y retrouver beaucoup plus facilement dans une feuille de style puisqu'on va généralement donner des noms qui font du sens à nos variables comme "main-color" (couleur principale) pour stocker la couleur principale du site par exemple.

Apprendre à utiliser les variables ou propriétés personnalisées CSS

La mise en place des propriétés personnalisées en CSS va se faire en deux temps : on va commencer par déclarer notre propriété personnalisée puis on va ensuite utiliser la variable créée.

On déclare une variable en CSS en utilisant la notation `--nom-de-la-variable : valeur`. On préfixe donc le nom de la variable choisi par un double tiret.

On déclarera généralement nos propriétés personnalisées à l'intérieur de sélecteurs très larges comme `:root` (qui est équivalent au sélecteur `html` mais qui possède un niveau de précision plus fort) afin de pouvoir les réutiliser partout.

Pour utiliser ensuite notre variable dans le code, il nous suffit de passer une fonction `var()` avec le nom de notre variable (toujours préfixé avec deux tirets) en argument en valeur des propriétés concernées.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Cours HTML et CSS</title>
        <link rel="stylesheet" href="cours.css">
    </head>
    <body>
        <header>
            <p>Logo, menu, slogan du site...</p>
        </header>
        <section>
            <h1>Propriétés personnalisées CSS</h1>
            <p>On utilise la fonction var() pour utiliser une variable CSS</p>
        </section>
        <footer>
            <p>Liens de bas de page...</p>
        </footer>
    </body>
</html>
```

```css
:root{
    --main-color: #333;
    --sec-color: #fff;
    --third-color: orange;
    --bg-color: #555;
    --m-0: 0;
}

body{
    color: var(--main-color);
    margin: var(--m-0);
}

header, footer{
    color: var(--sec-color);
    background-color: var(--bg-color);
    min-height: 50px;
}

/*Marges gauche et droite de 10px pour les section et pour les éléments
 *directement contenus dans header et dans footer*/
section, header > *, footer > *{
    margin: var(--m-0) 10px;
}

h1, a{
    color: var(--third-color);
}
```

Notez que l'héritage se fait de la même manière pour les propriétés personnalisées que pour les propriétés classiques. Ainsi, si un ne définit pas de valeur spécifique pour élément enfant, il héritera de la valeur de la propriété de son parent.

Contrôle de la validité des valeurs et gestion des erreurs

Il est impossible pour un navigateur de valider la valeur d'une propriété personnalisée à priori puisque lorsque la valeur est lue, il ne sait pas encore où elle va être utilisée c'est-à-dire avec quelles propriétés on va l'utiliser.

Cela oblige donc le navigateur à considérer toute valeur (ou presque) comme valide et cela permet donc d'utiliser certaines valeurs dans des contextes ou cela ne fait aucun sens (comme passer une valeur en px à une propriété `color` par exemple).

Dans ces cas là, c'est-à-dire lorsque le navigateur détecte une utilisation non valide d'une propriété personnalisée (ou plus exactement lorsqu'il analyse une substitution avec `var()` invalide), il va plutôt utiliser la valeur initiale ou la valeur héritée de la propriété en question.

Les règles CSS arobase (@)

Nous avons déjà rencontré la syntaxe @ dans ce cours notamment dans la leçon sur les media queries avec la règle @media.

Les règles @ ou “règles at” ou encore “règles arobase” permettent de fournir des instructions au CSS en tant que langage sur la façon dont il doit se comporter et sur la façon dont il doit réaliser certaines opérations.

Les règles @ CSS sont toutes construites de la même manière : elles sont composées du signe @ suivi d’un nom (un mot clef ou encore un identifiant) qui indique le type de règle pour lequel on va donner des instructions suivi par la règle en soi. La forme est donc la suivante : @type-de-regle valeur-de-la-regle.

Liste des règles @ CSS et définitions

Les règles globales

Le CSS possède à ce jour 3 règles @ globales qui sont les suivantes :

- La règle @charset permet de définir le jeu de caractères utilisé par la feuille de style ;
- La règle @import permet d’importer une feuille de styles externe dans la feuille courante ;
- La règle @namespace permet principalement d’indiquer que le contenu doit être pris en compte comme s’il était préfixé pour un espace de noms XML.

```
/*Ensemble de caractères utilisés : utf-8*/
@charset "UTF-8";

/*Syntaxe pour inclure un fichier nommé styles.css*/
@import "styles.css";

/*Namespace pour xhtml*/
@namespace url(http://www.w3.org/1999/xhtml);

/*Namespace pour svg*/
@namespace svg url(http://www.w3.org/2000/svg);
```

Les règles imbriquées

En plus de ces trois règles globales, le CSS possède également de nombreuses règles appelées “règles imbriquées” car elles nous permettent d’inclure des déclarations imbriquées supplémentaires.

Parmi ces règles imbriquées, nous disposons notamment d’un ensemble de 4 règles qu’on groupe généralement sous l’appellation “règles de groupe conditionnelles” car elles ont un

mode de fonctionnement similaire : elles définissent toutes trois une certaine condition qui, selon qu'elle est évaluée à vrai ou à faux, permettre d'appliquer les instructions imbriquées du groupe. Ces trois règles sont les suivantes :

- La règle `@media` permet d'appliquer des styles aux éléments seulement si l'appareil utilisé répond aux conditions fournies dans la règle ;
- La règle `@supports` permet d'appliquer des styles si le navigateur respecte une condition donnée ;
- La règle `@document` permet d'appliquer des styles à une page ou à un ensemble de pages spécifiques, c'est-à-dire à un document s'il respecte une condition donnée ;
- La règle `@viewport` permet d'appliquer certains styles si la zone d'affichage (viewport) respecte une condition donnée.

```
@document url("https://www.pierre-giraud.com/home.php"){
    /*Règles pour cette page en particulier*/
}
@document url-prefix("https://www.pierre-giraud.com/html-css-cours/"){
    /*Règles pour toutes les pages qui commencent par cette url*/
}
@document domain("pierre-giraud.com"){
    /*Règles pour toutes les pages du domaine*/
}

@supports (display: grid){
    /*Règles pour les navigateurs qui supportent display: grid*/
}
@supports not (display: grid){
    /*Règles pour les navigateurs qui ne supportent pas display: grid*/
}
@supports (display: flex) and (display: grid){
  /*Règles pour les navigateurs qui supportent display:flex et display:grid*/
}

@viewport {
    min-width: 680px;
    max-width: 960px;
    zoom: 1;
    min-zoom: 0.5;
    max-zoom: 1.5;
    orientation: auto;
}
```

Les autres règles imbriquées sont les suivantes :

- La règle `@keyframes` permet de définir les étapes d'une animation ;
- La règle `@font-face` permet de télécharger des polices externes dans une page ;
- La règle `@page` permet d'indiquer une disposition à appliquer pour l'impression de la page ;
- La règle `@counter-style` permet de définir des styles de compteurs personnalisés pour les listes ;
- Les règles `@font-feature-values`, `@swash`, `@ornaments`, `@annotation`, `@stylistic`, `@styleset` et `@character-`

variant permettent de définir des noms d'usages pour la propriété font-variant-alternates qui contrôle l'utilisation de glyphes alternatifs.

Le futur du CSS : imbrication et héritage étendu ?

Pour conclure ce cours, nous allons discuter de l'avenir du CSS, de la direction prise par ce langage et des fonctionnalités les plus attendues par les développeurs et notamment l'imbrication des sélecteurs et l'héritage étendu des styles.

L'imbrication des sélecteurs CSS

La façon dont le CSS est construit et fonctionne aujourd'hui nous pousse à souvent déclarer de multiples fois certains sélecteurs afin de leur appliquer des styles.

Dans le cas de page modérément complexe, cela peut amener à de nombreuses duplications et à des feuilles de styles inutilement lourdes et complexes à lire.

Cela fait des années que les développeurs militent pour une imbrication des sélecteurs CSS pour résoudre ce genre de problème. Le groupe de travail gérant l'évolution du CSS se penche sur ce problème depuis longtemps et un premier brouillon de travail a vu le jour en 2019 avec deux méthodes qui devraient nous permettre d'imbriquer nos sélecteurs dans le futur :

- L'imbrication directe avec l'utilisation du sélecteur d'imbrication `&` ;
- L'utilisation de la règle `@nest`.

Ces deux outils sont toujours en phase de développement et ne sont pas encore reconnus comme standards mais il y a peu de chance qu'ils soient abandonnés à ce stade. Il me semble donc très intéressant de vous les présenter.

Le sélecteur d'imbrication &

Lorsqu'on déclare une règle CSS imbriquée, nous avons besoin de faire référence au parent puisque c'est tout l'intérêt de l'imbrication. Le sélecteur d'imbrication `&` permet de faire référence à ce sélecteur parent.

Cette première méthode fonctionne à condition que le sélecteur d'imbrication soit le premier sélecteur simple contenu dans le premier sélecteur complexe du sélecteur parent. Grosso-modo, cela signifie qu'il faudra que nos sélecteurs imbriqués commencent par `&`. Par exemple, `& .foo` est autorisé mais `.foo &` ne l'est pas.

Voici la syntaxe qui sera utilisée en CSS avec ce sélecteur :

```
/*Syntaxe d'utilisation du sélecteur &*/
ul{
    & li{
        /*Styles CSS*/
    }
}
```

La règle CSS @nest

La règle CSS `@nest` produit exactement le même effet que l'utilisation du sélecteur `&` mais est moins restrictive dans son utilisation (les situations d'imbrication invalides sont moins nombreuses). En contrepartie, elle est un peu plus "lourde" dans son écriture.

La syntaxe devrait être la suivante :

```
/*Syntaxe d'utilisation de la règle @nest*/
ul{
    @nest li{

    }
}
```

Vous pourrez donc utiliser `&` ou `@nest` en fonction de la situation et de vos préférences personnelles.

L'héritage étendu

L'héritage étendu des styles est une autre fonctionnalité très attendue par les développeurs. L'idée ici serait qu'un sélecteur puisse hériter de l'ensemble des styles définis pour un autre sélecteur non apparenté.

Les préprocesseurs CSS comme Sass et Less proposent cette fonctionnalité depuis longtemps via des règles comme `@extend`. Pour le moment, cependant, il n'existe rien de tel en CSS.

La seule chose possible aujourd'hui en CSS est l'importation de l'ensemble des styles d'une feuille de style dans une autre via la règle `@import`.

PARTIE XVIII

Conclusion du cours

Conclusion du cours

Nous avons désormais fait le tour des notions à connaitre en HTML et en CSS. Vous connaissez désormais le rôle de chacun de ces deux langages et savez les utiliser à bon escient. Vous êtes capables d'utiliser les bons éléments HTML pour donner le plus de sens possible à vos différents contenus et savez créer des design complexes et adapter vos pages selon l'écran de vos visiteurs.

Il ne vous reste désormais qu'une chose à faire : pratiquer le plus possible. En effet, je suis de ceux qui sont persuadés qu'on n'apprend véritablement à coder qu'en pratiquant et qu'en étant confronté aux difficultés et cela est un processus long.

Par de panique donc si vous ne vous sentez pas encore tout à fait prêt pour gérer un projet de A à Z : c'est tout à fait normal. Il faut du temps et il faut surtout répéter les mêmes opérations plusieurs fois pour créer des automatismes.

De plus, si vous voulez devenir un développeur à part entière, vous n'allez pas pouvoir vous contenter de ne connaitre que le HTML et le CSS mais allez également devoir découvrir, comprendre et apprendre à utiliser d'autres langages comme le JavaScript par exemple si vous souhaitez vous orienter plutôt sur du développement « front side » ou « côté client » ou le PHP et un langage de gestion de bases de données si vous êtes plutôt attiré par le développement « back side » ou « côté serveur ».

Connaitre et comprendre un ensemble varié de langages de programmation différents vous permettra également de bien comprendre comment fonctionnent les différents langages entre eux et le rôle de chacun dans un site web, vous donnant par la même une bien meilleure vue d'ensemble et une bien meilleure approche du code dans vos projets futurs.