

# Les bases du CSS

Ibrahim ALAME

3 décembre 2024

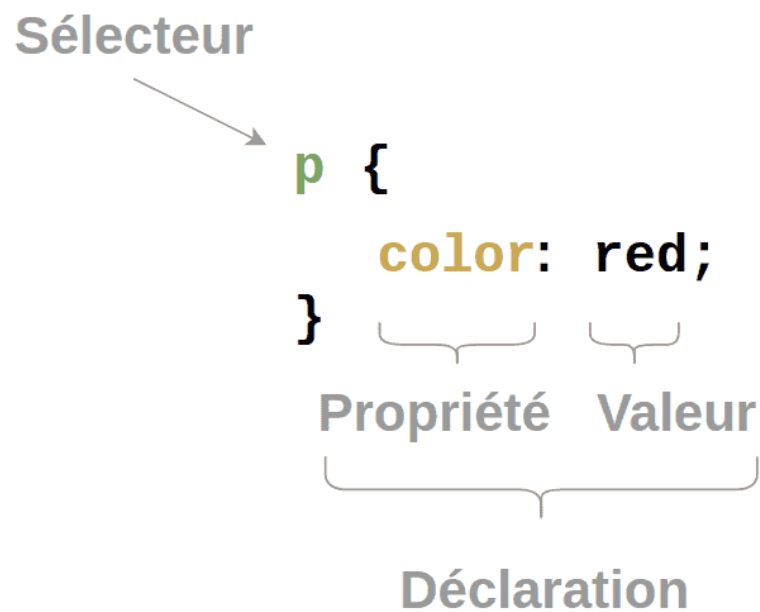
## 1 Présentation du chapitre

### 1.1 Rappels sur le CSS

Les feuilles de style en cascade, pour **CSS (Cascading Style Sheets)** est un langage qui décrit la présentation des documents **HTML**. Le principe du **CSS** est de sélectionner certains éléments pour leur appliquer un style particulier.

### 1.2 Les règles CSS

Tout le **CSS** se divise a minima de cette manière si il n'est pas appliqué côté **HTML** (ce qui est déconseillé comme nous le verrons) :



Le sélecteur permet de sélectionner le contenu **HTML** sur lequel la règle doit s'appliquer. Ici, par exemple, la règle s'appliquera à tous les éléments **HTML** p, c'est-à-dire à tous les paragraphes. Après le sélecteur, nous avons forcément une paire d'accolades qui contiennent la ou les propriétés à modifier pour les éléments sélectionnés.

- La propriété est ce que l'on souhaite définir sur l'élément. Par exemple ici sa couleur. Il faut toujours utiliser : pour séparer la propriété et la ou les valeurs à lui donner.

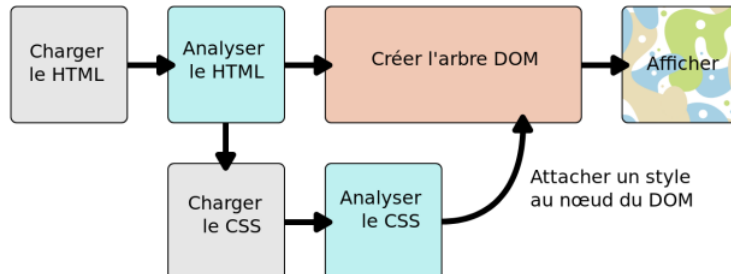
- La valeur est la mise en forme à appliquer pour une propriété donnée. Ici, de mettre la couleur en rouge. Il faut toujours terminer par ; pour passer à la modification suivante, par exemple :

```
p {
  color: red;
  width: 100px;
}
```

### 1.3 Comment le CSS fonctionne t-il ?

Pour afficher un document le navigateur va réaliser les principales étapes suivantes :

1. Il va charger le code HTML avec une requête HTTP.
2. Il va convertir le HTML en objet l'DOM (document object model). Il s'agit du modèle en arbre de tous les éléments HTML de la page qui est mis en mémoire vive pour pouvoir le modifier dynamiquement. Nous y reviendrons
3. Le navigateur va récupérer toutes les ressources contenues dans les liens indiqués dans le HTML dont les feuilles de style CSS. Il effectue donc des requêtes HTML supplémentaires.
4. Le navigateur analyse les règles CSS pour créer un arbre de rendu (ou render tree) qui contient tous les styles à appliquer aux différents sélecteurs.
5. Enfin le navigateur affiche le document en prenant donc en compte le HTML et le CSS (et éventuellement le JavaScript, nous y reviendrons plus tard). Il s'agit de la phase de peinture (painting).



## 2 Déclarer des règles CSS

Il existe trois manières d'appliquer du CSS à un document HTML. Cependant une seule est recommandée et nous la verrons en dernier.

### 2.1 Styles en ligne

La première méthode est déconseillée mais vous serez amené à la voir souvent. Il s'agit de mettre du CSS directement dans le HTML. Par exemple :

```
<p style="color:red;">Element paragraphe avec du style CSS en ligne.</p>
```

Ce n'est pas bon pour de nombreuses raisons :

- La première est qu'en programmation, un des concepts fondamentaux est la séparation des préoccupations. Cela signifie qu'il ne faut pas mélanger du code qui n'ont pas du tout les mêmes finalités. Ici le **HTML** est pour la structure du **document** et le **CSS** pour la mise en forme du **document**. Ils n'ont pas la même finalité et doivent donc être séparés.
- Deuxième raison, la lisibilité et la maintenabilité du code s'en trouve fortement réduite car la combinaison de règles devient compliquer à comprendre et à mettre à jour.
- Troisième raison, l'un des autres concepts très important en programmation est de rester **DRY (don't repeat yourself)**. Cela signifie qu'il faut éviter au maximum de répéter le même code à différents endroits.

Ici, si vous voulez appliquer le même style à plusieurs paragraphes qui sont dans différents fichiers **HTML** ou dans des parties différentes vous devrez vous répéter. Cela aura pour conséquence de diminuer la maintenabilité du code car lors d'une mise à jour il faudra se rappeler tous les endroits où vous avez mis des règles **CSS**.

La seule exception est pour les emails car les clients de messagerie sont plus nombreux et moins bien développés que les navigateurs. Pour s'assurer que le **CSS** soit bien compatible avec tous les clients de messagerie, il est commun de mettre les règles **CSS** en ligne.

## 2.2 Feuilles de style interne

Une feuille de style interne est l'utilisation de règles **CSS** à l'intérieur de balises `<style>` dans le fichier **HTML** dans la partie `<head>`. Pour les mêmes raisons, ce n'est pas recommandé.

Voici un exemple :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Appliquer des styles CSS</title>
    <style>
      div {
        color: blue;
      }
    </style>
  </head>
  <body>
    <div>Div avec feuille de style interne</div>
  </body>
</html>
```

## 2.3 Feuilles de style externe

Cette méthode permet l'utilisation de fichiers **CSS** externes. C'est la méthode recommandée. Vous pouvez facilement appliquer les mêmes styles dans plusieurs pages **HTML** en important la même feuille de styles dans les différents fichiers. Il suffit de créer un fichier **CSS** avec l'extension **.css** puis de l'inclure avec un élément **link** :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Feuille de styles externe</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <p>Un paragraphe avec du contenu.</p>
  </body>
</html>
```

Dans le fichier CSS, vous pouvez mettre directement les règles :

```
h1 {
  color: blue;
}

p {
  color: red;
}
Copier
```

Voici un exemple avec les trois méthodes pour appliquer des styles **CSS** :

## 3 Les classes, les ids et les sélecteurs

### 3.1 Les sélecteurs **CSS**

Nous avons déjà vu ce qu'était un sélecteur : il permet de sélectionner un ou plusieurs éléments **HTML** sur lesquels appliquer une ou plusieurs règles **CSS**.

Cette leçon va être plus détaillée que la vidéo pour vous donner un aperçu approfondi des règles. Nous vous conseillons de les survoler pour le moment, nous y reviendrons au fur et à mesure dans les chapitres suivants.

Il existe différents types de sélecteurs : le sélecteur universel, les sélecteurs de type, les classes, les ids, les sélecteurs d'attribut, les combinateurs et les pseudo-classes.

Nous allons tous les voir sauf les pseudo-classes que nous verront plus tard.

### 3.2 Le sélecteur universel \*

Le sélecteur universel permet d'appliquer des règles à tous les éléments de la page.

```
* {  
  color: red;  
}
```

Vous vous en servirez rarement.

### 3.3 Les sélecteurs de type

Les sélecteurs de type CSS des éléments en fonction du nom de leur type.

Lorsqu'un sélecteur de type est utilisé seul, il ciblera tous les éléments de ce type.

Ainsi pour mettre tous les paragraphes en rouge :

```
p {  
  color: red;  
}
```

### 3.4 Les classes

Le sélecteur classe commence par un point . suivi du nom de la classe.

Il sélectionnera tous les éléments HTML qui ont un attribut class qui inclut le nom de la classe.

Un élément ayant un attribut class peut avoir plusieurs classes qui sont alors séparées par des espaces.

Par exemple côté HTML si nous avons :

```
<p class="rouge">Paragraphe 1.</p>  
<p class="gras rouge">Paragraphe 2.</p>  
<p class="gras">Paragraphe 3.</p>  
<p>Paragraphe 4.</p>
```

Et côté CSS :

```
.rouge {  
  color: red;  
}  
.gras {  
  font-weight: bold;  
}
```

Alors seuls les deux premiers paragraphes qui ont la classe rouge seront sélectionnés et se verront appliquer la règle CSS `color : red`.

Les deuxième et troisième paragraphes qui ont la classe gras seront sélectionnés et se verront appliquer la règle CSS `font-weight : bold`.

### 3.5 Les ids

Le sélecteur **ID** (identifiant) permet de cibler un élément grâce à la valeur de son attribut **id**.  
Un sélecteur Id commence par **#** suivi du nom de l'identifiant.  
Par exemple côté **HTML** si nous avons :

```
<p id="par1">Paragraphe 1.</p>
```

Et côté **CSS** :

```
#par1 {  
  color: red;  
}
```

### 3.6 Les sélecteurs d'attribut

Le sélecteur d'attribut permet de cibler un élément selon la présence d'un attribut ou selon la valeur donnée d'un attribut.

Nous donnons quelques exemples mais n'entrons pas dans les détails car ils ne sont pas fréquemment utilisés :

```
/* Sélectionne les éléments a avec un attribut title */  
a[title] {  
  color: red;  
}
```

Cela sélectionnerait :

```
<a href="#" title="test">hello</a>
```

Autre exemple :

```
a[href="https://dyma.fr"] {  
  color: red;  
}
```

Sélectionnerait :

```
<a href="https://dyma.fr">Dyma</a>
```

### 3.7 Le groupement

Il est possible de grouper plusieurs sélecteurs en les séparant par une virgule pour appliquer les mêmes règles à un ensemble de sélecteurs.

Par exemple :

```
div, span {  
  color: red;  
}
```

Sélectionnerait les deux premiers éléments :

```
<div>Je suis rouge</div>
<span>Je suis aussi rouge</span>
<p>Je ne suis pas rouge</p>
```

### 3.8 Les combinateurs

Il existe des combinateurs qui permettent de définir finement les relation entre les sélecteurs pour un ensemble de règles.

#### 3.8.1 Le combinateur de descendance

Lorsque que l'on met un espace entre deux sélecteurs cela signifie que l'on veut sélectionner les éléments sélectionnés par le second sélecteur et qui sont imbriqués dans les éléments sélectionnés par le premier sélecteur.

Cela à l'air dur dit comme cela, mais c'est très simple ! Prenons quelques exemples :

```
div span {
  color: red;
}
```

Donnera :

```
<div>Je ne suis pas rouge</div>
<span>Je ne suis pas rouge</span>
<div>
  <span>
    Je suis rouge
  </span>
</div>
```

#### 3.8.2 Le combinateur enfant

Le combinateur `>` cible seulement les éléments correspondant au second sélecteur qui sont imbriqués directement dans les éléments ciblés par le premier sélecteur.

```
div > span {
  color: red;
}
```

Donnera :

```
<div>
  <span>Je suis rouge</span>
</div>
```

```

    <p><span>Pas rouge</span></p>
  </div>
</div>
<span>Pas rouge</span>

```

### 3.8.3 Le combinateur de voisin direct

Pour sélectionner un élément uniquement si celui-ci suit un élément donné et que les deux éléments sont imbriqués dans le même élément parent, alors on utilise un **+** entre les deux sélecteurs.

```

div + span {
  color: red;
}

```

Donnera :

```

<div>Je ne suis pas rouge</div>
<span>Je suis rouge</span>
<span>Je ne suis pas rouge</span>

```

### 3.8.4 Le combinateur de voisins généraux

Le combinateur cible seulement les éléments correspondant au second sélecteur qui sont précédés par un élément ciblé par le premier sélecteur.

Les éléments doivent être de même niveau, c'est-à-dire être imbriqués dans le même parent.

Vous trouverez le caractère **tilde** en faisant **Alt Gr + ^**.

## 4 Utiliser l'inspecteur des navigateurs

Dans cette leçon nous utiliserons les outils **Chrome DevTools**, mais il existe à peu près les mêmes outils sur Firefox.

### 4.1 Ouvrir l'inspecteur **Chrome**

Pour ouvrir l'inspecteur sélectionnez un élément puis faites clic droit puis inspecter.

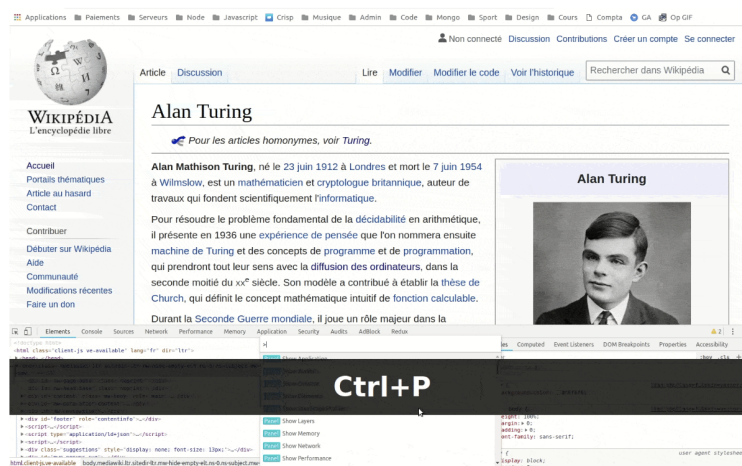
Vous pouvez également sélectionner l'élément et faire **Ctrl + Maj + i** :





## 4.2 Changer l'emplacement de l'inspecteur

Ouvrez le panel en faisant **Ctrl + Maj + p** et entrez la position : **right** ou **bottom** et pressez entrée :



Vous pouvez ensuite voir les styles appliqués à un élément dans l'onglet **Styles**.

Vous pouvez également utiliser l'onglet **Elements** pour sélectionner d'autres éléments et voir les styles qui leur sont appliqués.

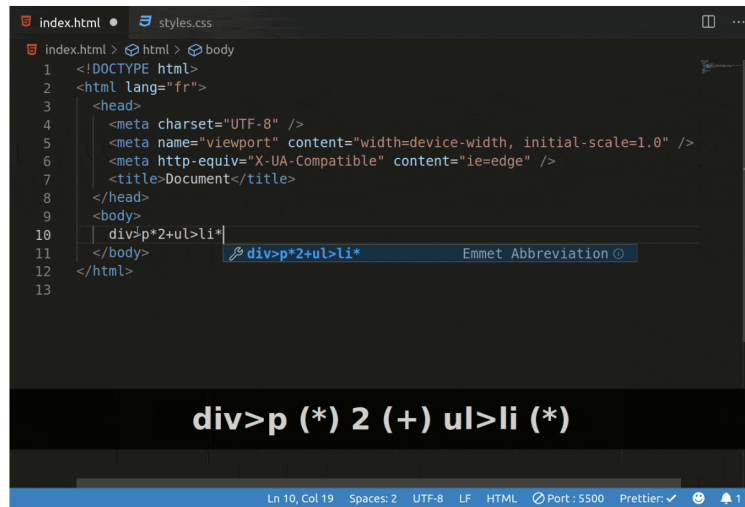
## 4.3 De nouveaux raccourcis Emmet

Nous avons vu **\*** et **;** avec **Emmet**.

Pour rappel le premier permet de multiplier l'élément précédent par le chiffre passé après.

Le second permet d'imbriquer un élément dans le premier.

Nous allons voir **+** qui permet d'ajouter un élément au même niveau :



## 5 Mise en forme du texte

Dans cette leçon, nous allons voir comment modifier le style des textes en utilisant du **CSS**.

### 5.1 Police de caractères avec **font-family**

En typographie, une fonte de caractères est un ensemble de représentations visuelles de caractères, d'une même police d'écriture, de même style, corps et graisse.

Le corps est la taille d'une fonte de caractères, mesurée en points typographiques.

La graisse est l'épaisseur d'un trait ou d'un caractère.

Une police de caractère est l'ensemble des déclinaisons d'un même caractère.

Une déclinaison est donc une fonte. Par exemple, la police de caractères **Times New Roman**, est constituée d'une fonte romaine, d'une fonte italique, d'une fonte grasse et d'une fonte grasse italiciée.

**font-family** permet de définir une police ou une liste de polices.

Le navigateur n'appliquera une police de caractères que si elle est disponible, sinon, il utilisera une police par défaut.

Les polices **Web** dites sûres sont celles qui sont disponibles sur tous les systèmes et dans tous les navigateurs : **Arial**, **Courier New**, **Georgia**, **Times New Roman**, **Trebuchet MS** et **Verdana** en sont des exemples.

Les polices se divisent en 5 familles génériques :

1. **serif** : Polices avec fioritures.
2. **sans-serif** : Les polices qui n'ont pas d'empattements
3. **monospace** : Les polices dans lesquelles chaque caractère a la même largeur.
4. **cursive** : Les polices qui ressemblent à de l'écriture manuscrite.
5. **fantasy** : Les polices décoratives.

Vous pouvez déclarer une liste de polices :

```
font-family: "Times New Roman", Times, serif;
```

Dans ce cas la première sera appliquée, et si elle n'est pas disponible la deuxième, et si elle n'est pas disponible la famille de polices génériques `serif`.

Vous pouvez trouver des polices gratuites sur <https://www.fontsquirrel.com/> et <https://www.dafont.com/fr/>.

## 5.2 Taille de police avec `font-size`

La taille des polices de caractères est définie par la propriété `font-size`.

Elle accepte la plupart des unités de valeurs que nous étudierons dans la prochaine leçon. Mais voyons la plus utilisée, les `pixels` :

```
body {  
  font-size: 20px;  
}
```

Ici nous fixons à 20 le nombre de pixels souhaités pour la hauteur de tous les textes dans `body`.

La propriété `font-size` d'un élément est héritée de son parent. C'est-à-dire de l'élément dans lequel il est imbriqué.

Par défaut, tous les éléments ont une taille de police de `16px`.

En effet, tous les éléments sont inclus dans l'élément `html`, et c'est la taille de la police qui est fixée par défaut pour cet élément dans tous les navigateurs.

Certains autres éléments ont des tailles par défaut en `HTML`, par exemple `h1` a une taille par défaut de `2 em`, qui comme nous le verrons, équivaut par défaut à `32px`.

## 5.3 Graisse de la police avec `font-weight`

Comme nous l'avons vu, la graisse en typographie est l'épaisseur d'un caractère.

Elle se définit avec `font-weight`.

Vous pouvez utiliser `bold` comme raccourci ou alors utiliser une valeur numérique entre `100` et `900`.

## 5.4 Style de la police avec `font-style`

Vous pouvez mettre la police en italique avec `font-style` :

```
<p style="font-style: italic">Italique</p>
```

## 5.5 Décoration du texte avec `text-decoration`

Vous pouvez décorer le texte avec `text-decoration`.

- `underline` permet de le souligner.
- `overline` de tracer une ligne au-dessus.
- `line-through` de le barrer.

A noter que `text-decoration` est un raccourci pour trois propriétés `text-decoration-line`, `text-decoration-style` et `text-decoration-color`.

Les valeurs possibles pour `text-decoration-line` sont celles que nous avons vues.

Les valeurs possibles pour `text-decoration-style` sont `solid`, `double`, `dotted`, `dashed` et `wavy`.

## 5.6 Transformer du texte avec `text-transform`

Vous pouvez transformer la police avec `text-transform`.

Les valeurs possibles sont : `uppercase` (tout en majuscule), `lowercase` (tout en minuscule) et `capitalize` (première lettre des mots en majuscule).

## 5.7 Hauteur de ligne

Il est possible de définir la hauteur de chaque ligne de texte avec `line-height`.

Vous pouvez passer une hauteur en pixels (ou toute autre unité de mesure que nous verrons) mais aussi juste un nombre qui sera alors un multiplicateur.

La hauteur de ligne recommandée est entre 1.5 et 2.

## 5.8 Espacement entre les lettres et les mots

Il est possible de définir l'espacement entre les lettres et les mots avec `letter-spacing` et `word-spacing`.

Vous pouvez définir la taille des espacements en `px` ou dans les autres unités de mesure que nous verrons dans la leçon suivante.

# 6 Les unités en CSS

## 6.1 Les unités absolues

Il existe de nombreuses unités absolues qu'il est possible d'utiliser en CSS : `cm` (centimètres), `mm` (millimètres), `Q` (quart de millimètres), `in` (pouces), `pc` (picas), `pt` (points) et `px` (pixels).

Mais seuls les pixels sont utilisés pour l'affichage Web .

## 6.2 Les unités relatives

Les unités relatives permettent de donner une mesure à un élément par rapport à la taille d'un autre élément.

Les principales sont les suivantes :

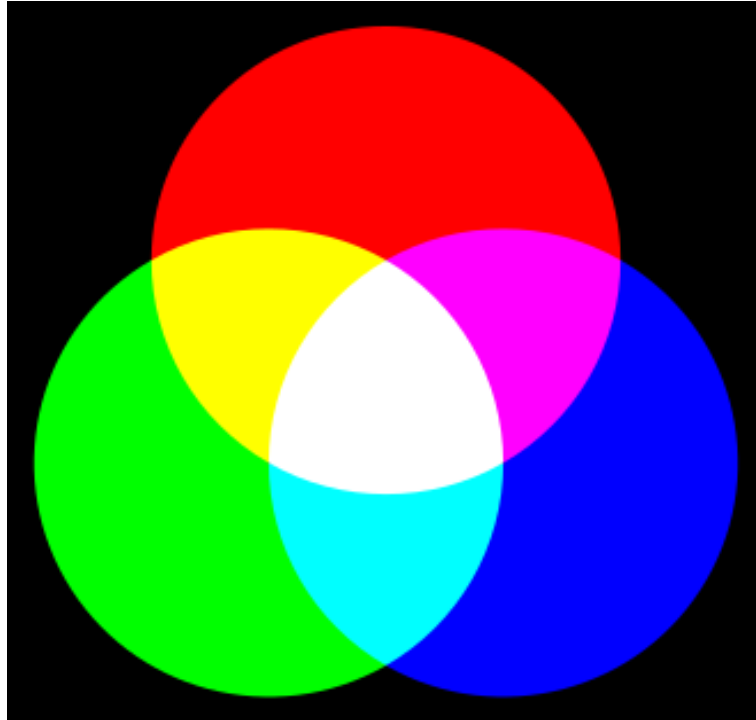
1. `em` est par rapport à la taille de la police de l'élément parent. Par exemple si l'élément parent fait `16px` alors `2em` fera `32px` .
2. `rem` est par rapport à la taille de la police de l'élément racine. Par défaut, ce sera `16px` qui est la taille de la police de l'élément `html` .
3. `vw` équivaut à `1%` de la largeur du `viewport` de l'utilisateur.
4. `vh` équivaut à `1%` de la hauteur du `viewport` de l'utilisateur.
5. `vmin` équivaut à `1%` de la plus basse dimension du `viewport` de l'utilisateur.
6. `vmax` équivaut à `1%` de la plus haute dimension du `viewport` de l'utilisateur.

La plupart du temps, pour des applications `responsive` (c'est-à-dire dont l'affichage s'adapte à la taille de l'écran de l'utilisateur), on utilisera les `rem` .

## 7 Les couleurs

### 7.1 Les couleurs

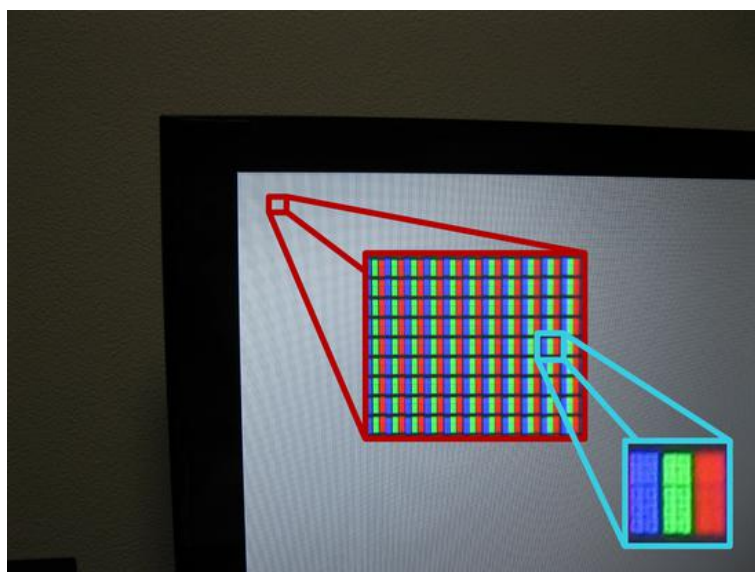
Une des méthodes pour obtenir une couleur est la *synthèse additive*.



La *synthèse additive* consiste à produire les couleurs par l'addition d'une proportion de trois couleurs primaires : le rouge, le vert et le bleu.

En informatique, les périphériques, par exemple les écrans d'ordinateur utilisent des **pixels**.

Chaque **pixel** peut afficher une couleur, et est décomposé en trois composants primaires : rouge, vert et bleu. Ces éléments lumineux sont beaucoup trop proches les uns des autres pour qu'on les distingue individuellement.



La machine peut contrôler pour chaque composant de chaque pixel l'intensité lumineuse. Une couleur est donc formée par une intensité spécifique pour chacun des trois composants d'un pixel : celui qui émet une teinte rouge, celui qui émet une teinte bleu et celui qui émet une teinte verte.

## 7.2 L'encodage des couleurs avec le système RGB :

Le système moderne de couleurs digitales utilise 24 **bits** pour encoder une couleur.

Ces 24 **bits** se décomposent en 3 fois 8 **bits** :

8 bits pour chaque teinte primaire : rouge, vert et bleu.

La valeur maximale possible avec 24 **bits** en encodant les nombres par **octet** (c'est-à-dire sur 8 **bits**) est de 255 255 255 . Cela permet d'encoder 16,7 millions de couleurs possibles.

En effet, le plus grand nombre sur 8 **bits** est 11111111 , ce qui correspond en base 2 à  $1 * 2^0 + 1 * 2^1 + 1 * 2^2 + 1 * 2^3 + 1 * 2^4 + 1 * 2^5 + 1 * 2^6 + 1 * 2^7$  ce qui est égal à 255 en base 10. Si vous voulez en savoir plus sur l'encodage des nombres et des caractères en informatique, donc en base 2, nous vous invitons à suivre la formation **Node.js** plus tard qui est très détaillée sur ce point. Pour le moment nous n'allons pas plus approfondi, car cela suffit à comprendre l'encodage des couleurs en **CSS** .

## 7.3 Le système **RGBA**

Il est également possible d'utiliser un dernier paramètre qui peut prendre pour valeur de 0 à 1.

Il s'agit de l'**Alpha** qui contrôle l'opacité.

Par exemple :

```
color: rgba(2, 121, 139, .3);
```

Créera un bleu (car quasiment pas de rouge) un peu clair (car avec du vert) et une opacité de 30

Le système hexadécimal

L'encodage hexadécimal commence par un **#** et est suivi par 6 nombres hexadécimaux. Un nombre hexadécimal a une valeur comprise entre 0 et 16 (d'où son nom hexa).

Les valeurs possibles sont 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f . a vaut 10, b vaut 11 et c jusqu'à f qui vaut 15.

Le nombre maximal en hexadécimal est donc #ffff .

Ce qui fait en base 16, en séparant par 2 les ff pour rester sur un octet :

$15 * 16^0 + 15 * 16^1$  ce qui vaut 255 en base 10.

## 8 Ordre d'application du CSS

Rappelez-vous de ce que veut dire CSS : Cascading Style Sheets . C'est le moment de voir pourquoi les feuilles de style sont dites en cascade.

### 8.1 La cascade

L'ordre de la déclaration des règles CSS importe beaucoup..

Ici les h1 seront bleu :

```
h1 {  
  color: red;  
}  
h1 {  
  color: blue;  
}
```

Et ici rouge :

```
h1 {  
  color: blue;  
}  
h1 {  
  color: red;  
}
```

Vous l'avez compris, pour un élément HTML c'est la dernière sélection qui l'emporte. Autrement dit, si l'élément HTML est sélectionné par plusieurs sélecteurs CSS , c'est le dernier sélecteur déclaré qui imposera ses règles.

Cependant, cela ne vaut que pour des sélecteur ayant la même spécificité.

### 8.2 La spécificité

Une autre règle est encore plus importante que l'ordre, c'est la spécificité. Plus le sélecteur est spécifique, plus il l'emporte et applique ses règles.

Par exemple, un sélecteur de classe .nomclasse a plus de poids qu'un sélecteur générique d'éléments par exemple p .

La spécificité est attribué avec une méthode de points :

Le CSS en ligne appliqué dans style gagne toujours car il n'y a pas de sélecteur vu qu'il est appliqué directement (score de 1000).

Si un id est présent dans le sélecteur le score augmente de 100 points Si un sélecteur de classe, d'attribut ou de pseudo-classe est présent le score augmente de 10 points.

Si un sélecteur d'élément est présent le score augmente de 1 point.

Quelques exemples :

`p` vaut 1.

`h1 + p` vaut 2.

`#identifiant` vaut 100.

### 8.3 L'importance

Il existe un terme spécial en **CSS** : `!important` .

Il permet de toujours gagner en spécificité pour une règle donnée.

Par exemple :

```
#millepoints {  
  color: red;  
}  
  
.jegagnetoujours {  
  color: orange !important;  
}
```

L'ordre d'application des règles est donc Importance > Spécificité > Cascade.

## 9 Définir la taille des éléments

### 9.1 La taille naturelle des éléments

Avant toute utilisation de **CSS** , les éléments **HTML** ont une taille dite naturelle.

Par exemple, une image a une hauteur et une largeur définies par le fichier de l'image. Cependant certains éléments **HTML** n'ont pas de taille naturelle comme par exemple une `div` ou une `span` vide.

Si vous ajoutez du contenu à une `div` , elle prendra la hauteur du contenu.

### 9.2 Définir une taille spécifique

Il est possible de définir une taille spécifique pour un élément **HTML** de bloc en utilisant les propriétés `height` et `width` (hauteur et largeur).

Fixer une taille absolue

Il est possible de donner une taille absolue à un élément en utilisant des **pixels** . On peut par exemple faire :

```
.fixe {  
  width: 150px;  
  height: 150px;  
}
```



## 9.3 Fixer une taille relative

Il est possible d'utiliser des pourcentages pour fixer la taille d'un élément d'une manière relative.

Dans ce cas, la taille fixée sera un pourcentage de la taille de son parent.

Par défaut, la propriété `width` des éléments de bloc a pour valeur `auto`.

Cela implique que si la `width` de l'élément parent n'est pas définie, sa valeur sera `auto`. Par défaut, l'élément occupera toute la largeur disponible.

Ce n'est pas le cas pour la hauteur, si la propriété `height` de l'élément parent n'est pas définie, alors la hauteur sera de 0 et il ne sera pas possible de fixer de hauteur relative pour l'élément enfant.

Définir une taille maximale ou minimale

Il est enfin possible de fixer une hauteur et / ou une largeur maximale et / ou minimale pour chaque élément.

Il suffit d'utiliser `min-width`, `min-height`, `max-height` et `max-width`.

## 10 Les arrière-plans

### 10.1 Le couleur d'arrière plan

Il est possible de fixer la couleur de l'arrière-plan d'éléments **HTML** en utilisant la propriété `background-color` :

```
div {  
  background-color: black;  
}
```

### 10.2 Utiliser une image

Il est également possible d'utiliser une image en arrière-plan en utilisant la propriété `background-image` :

```
.image {  
  background-image: url(chemin/vers/limage.jpg);  
}
```

Par défaut l'image n'est pas réduite en taille, si elle est trop grande seul un bout sera affiché.

Par défaut également, si l'image est plus petite que l'élément, elle se répétera et couvrira toute la surface.

### 10.3 Contrôler la répétition des images

Il est possible de contrôler la répétition des images.

Pour cela, il faut utiliser la propriété `background-repeat`.

Les valeurs possibles sont `no-repeat` (pas de répétition), `repeat-x` (répétitions uniquement sur l'axe horizontal), `repeat-y` (répétitions uniquement sur l'axe vertical) et `repeat` (par défaut, répétitions sur les deux axes).

## 10.4 Contrôler la taille des images

Il est possible de contrôler la taille des images avec `background-size`. Il est possible d'utiliser des unités absolues ( `px` ) ou relatives ( `em` , `rem` , `%` etc ) :

```
background-size: 12px;  
background-size: 12px 24px;
```

La première valeur est la largeur ( `width` ) et la deuxième valeur, optionnelle, est la hauteur ( `height` ).

Il existe aussi deux mots clés permettant soit d'agrandir l'image autant que possible sans la rogner ou l'étirer ( `contain` ), soit d'agrandir l'image autant que possible sans l'étirer, mais en la rognant éventuellement.

```
background-size: cover;
```

Contrôler la position des images

Il est également possible de contrôler la position de l'image avec `background-position`.

Par défaut la valeur est `0, 0`.

Le système de position utilise le système de coordonnées classique avec deux axes (horizontal et vertical).

La première valeur passée est pour l'axe horizontal et la seconde pour l'axe vertical :

```
background-position: 10px 50px;
```

Il est possible également d'utiliser des valeurs absolues ou relatives ou des mots clés. Les mots clés possibles sont `top` , `bottom` , `center` , `right` et `left` :

```
background-position: top center;
```

## 10.5 Contrôler le comportement lors du scroll

Il est possible de définir le comportement de l'image d'arrière-plan lors du scroll de l'utilisateur grâce à la propriété `background-attachment`.

Les valeurs possibles sont `fixed` , `local` et `scroll`.

- `fixed` : l'arrière-plan reste fixe par rapport à l'élément, même si l'élément sur lequel il est placé scroll.
- `local` : l'arrière-plan reste fixe par rapport au contenu de l'élément. Si l'élément scroll, alors l'arrière-plan scroll également avec l'élément.
- `scroll` : l'arrière-plan reste fixe par rapport à l'élément lui-même. Il ne scroll pas avec son contenu mais scroll lorsque la page scroll.

## 10.6 Utiliser la propriété raccourcie `background`

La propriété raccourcie est `background` : elle permet de fixer plusieurs propriétés en une ligne.

Les propriétés sont les suivantes : `background-clip`, `background-color`, `background-image`, `background-origin`, `background-position`, `background-repeat`, `background-size`, et `background-attachment`.

Nous verrons `background-clip` et `background-origin` plus tard car nous avons besoin de voir les marges et le remplissage avant.

Prenons l'exemple :

```
.raccourci {  
  height: 50px;  
  background: center right 50px / contain red  
    url("https://mdn.github.io/css-examples/learn/backgrounds-borders/star.png")  
    no-repeat;  
}
```

Ici nous commençons par fixer la **position** (mais l'ordre importe peu). L'étoile sera centrée sur l'axe vertical et à **50 px** de la droite sur l'axe horizontal.

Ensuite, nous devons mettre un **/** pour séparer la position et la taille !

Nous définissons la taille avec le mot clé **contain** que nous avons vu.

Ensuite nous passons à la couleur, à l'image puis enfin à la répétition.

## 11 Les bordures

### 11.1 Les bordures

Les bordures font partie du modèle de la boîte en **CSS** que nous étudierons en détails dans la leçon suivante.

Elles permettent de définir finement les bordures à appliquer à un élément **HTML** .

### 11.2 Les principales propriétés

Il existe trois propriétés principales pour définir une bordure.

1. **border-width** permet de définir la taille de la bordure.
2. **border-style** permet de définir le style de la ligne de la bordure : **dotted** (en pointillé avec des points), **dashed** (en pointillé avec des tirets), **solid** (ligne continue), **double** (deux lignes), **groove** (effet 3D de gravure), **ridge** (effet 3D de volume), **inset** (effet 3D donnant l'impression que l'élément est enfoncé) et **outset** (effet 3D donnant l'impression que l'élément ressort du document).
3. **border-color** permet de définir la couleur de la bordure.

### 11.3 L'arrondi des bordures

La propriété **border-radius** permet de définir des coins arrondis pour la bordure d'un élément.

La valeur passée à la propriété est le rayon de l'arrondi.

La courbure s'applique à l'arrière-plan même si l'élément n'a aucune bordure (voir le premier exemple).

### 11.4 Application à certains côtés

Il est possible de facilement appliquer un style, une taille ou une couleur à certains côtés.

## 11.5 Appliquer une propriété de bordure à un seul côté ou un seul coin

Pour appliquer une propriété à un seul côté il suffit d'ajouter le nom du côté dans le nom de la propriété.

Pour appliquer un arrondi à un seul coin il suffit également d'ajouter le nom :

```
border-top-width: 1px;
border-left-style: solid;
border-bottom-color: black;
border-right-color: red;
border-top-right-radius: 10px;
```

## 11.6 Appliquer des propriétés différentes suivants les côtés ou les coins

Pour appliquer des propriétés différentes suivants les côtés, il existe des raccourci plutôt que de devoir définir la propriété pour chaque côté ou coin.

Ainsi quelle que soit la propriété, si vous lui donnez :

1. une seule valeur : elle s'appliquera aux quatre côtés (ou à tous les coins pour le radius ).
2. deux valeurs : la première valeur s'applique aux côtés haut et bas et la seconde aux côtés gauche et droit (pour le **radius** , elle s'applique aux coins en haut à gauche et en bas à droite pour la première, et aux coins en haut à droite et en bas à gauche pour la seconde).
3. trois valeurs : la première valeur s'applique au côté haut, la seconde aux côtés gauche et droit et la troisième au côté bas (pour le **radius**, la première valeur s'applique au coin en haut à gauche, la deuxième aux coins en haut à droite et en bas à gauche, et la dernière pour le coin en bas à droite).
4. quatre valeurs : les valeurs s'appliquent dans le sens des aiguilles d'une montre en commençant par le haut (ou par le coin en haut à gauche pour le **radius** ).

## 11.7 Le raccourci **border**

Vous pouvez spécifier pour tous les côtés et pour les trois propriétés d'un coup :

```
border: medium dashed green;
```

**medium** étant un de mots clés pour l'épaisseur de la bordure (avec **thin** et **thick** ).

## 11.8 Nouveaux raccourcis **Emmet**

Nous allons voir deux nouveaux raccourcis **Emmet** .

Le premier **element.nomdelaclass** permet de créer l'élément en lui donnant une **class** avec le nom spécifié.

Vous pouvez en ajouter plusieurs avec **element.nomdelaclass1.nomdelaclass2** . Le second, **\$** permet la numérotation d'éléments qui se répètent.

Vous pouvez ajouter un arobase suivi d'un tiret pour inverser l'ordre de la numérotation : **\$@-** .

Vous pouvez ajouter un arobase suivi d'un nombre pour définir la valeur de départ pour la numérotation : **\$@3** .

Nous allons combiner les deux :

```
40 </head>
41 <body>
42   <p class="bordure1"></p>
43   <p class="bordure2"></p>
44   <p class="bordure3"></p>
45   <p class="bordure4"></p>
46   <p class="bordure5"></p>
47   <p class="bordure6"></p>
48   <p class="bordure7"></p>
49   <p class="bordure8"></p>
50   <p class="bordure9"></p>
51   <p class="bordure10"></p>
52 </body>
53 </html>
```

p.bordure\$ (\*) 10 ↵

## 12 Le modèle de la boîte CSS

### 12.1 Le modèles des boîtes CSS

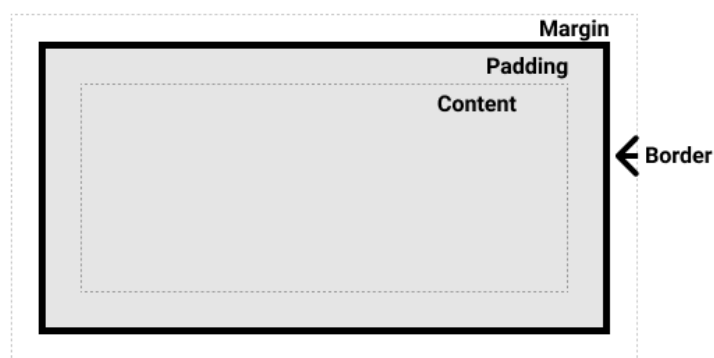
Nous allons voir une notion essentielle du CSS : les boîtes.

Tous les éléments qui utilisent du CSS ont une boîte autour d'eux.

Retenez dès maintenant qu'il y a des différences entre les éléments de bloc et les éléments en ligne pour l'application du modèle de la boîte.

### 12.2 Les éléments de la boîte

Une boîte CSS est constituée des éléments suivants :



- **content** : c'est la boîte du contenu. C'est là qu'est affiché le contenu de l'élément et sa taille est contrôlée par les propriétés **width** et **height** que nous avons vues.
- **padding** : c'est de l'espace vide qui est placé entre le contenu et la bordure. Il est contrôlé par la propriété **padding** que nous allons étudier.

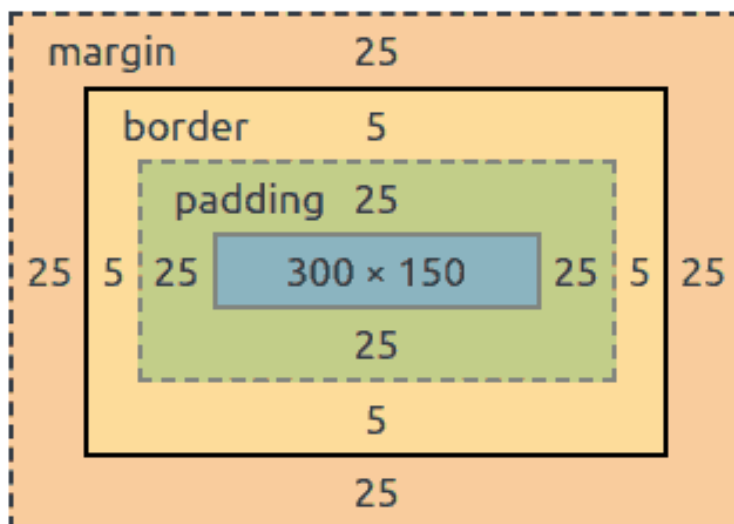
- **border** : c'est la bordure que nous avons étudiée dans la leçon précédente et qui se contrôle avec la propriété **border** .
- **margin** : c'est la marge qui est de l'espace vide entourant le bordure. La taille de la marge est contrôlée par la propriété **margin** .

Prenons par exemple un élément paragraphe et utilisons ce **CSS** :

```
.boite1 {
  width: 300px;
  height: 150px;
  margin: 25px;
  padding: 25px;
  border: 5px solid black;
}
```

Faites ensuite **Ctrl + Maj + i** après avoir sélectionné le rectangle.

Vous verrez la représentation de la boîte :



Vous retrouvez bien la boîte de contenu avec des dimensions de **300px** par **150px** , la boîte **padding** avec **25px** de tous les côtés, la boîte **border** qui a une épaisseur de **5px** et la boîte **margin** avec également **25px** de tous les côtés.

Ici l'élément prend donc en largeur totale :  $25 + 5 + 25 + 300 + 25 + 5 + 25 = 410\text{px}$  .

### 12.3 Le modèle alternatif **border-box**

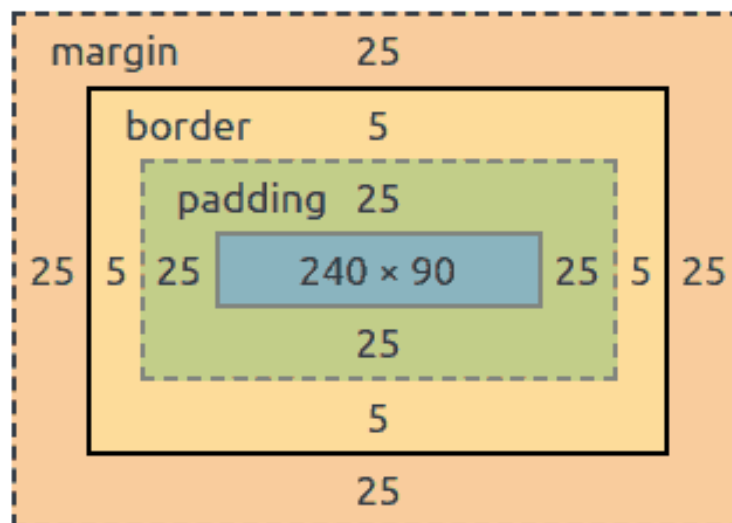
Il existe un modèle alternatif, plus récent, qui permet d'utiliser les propriétés **width** et **height** non pas pour définir la taille de la boîte de contenu, mais la taille de la boîte entière (à savoir contenu + **padding** + bordure). La marge est en effet considérée comme extérieure.

Pour utiliser ce modèle il faut utiliser **box-sizing : border-box** ; .

En reprenant le même exemple mais en changeant le modèle :

```
.boite2 {
  box-sizing: border-box;
  width: 300px;
  height: 150px;
  margin: 25px;
  padding: 25px;
  border: 5px solid black;
}
```

Nous obtenons :



Remarquez que les dimensions de la boîte de contenu ont été calculées automatiquement.

Le calcul est  $\text{width} - \text{padding} - \text{border}$  pour la largeur et  $\text{height} - \text{padding} - \text{border}$  pour la hauteur.

Cela donne pour la largeur :  $300 - 2 * (25+5) = 240\text{px}$ .

## 12.4 La marge avec **margin**

La marge est l'espace blanc autour de la boîte de bordure. Elle pousse les éléments autour de l'élément sur lequel elle est appliquée.

Pour fixer la marge, vous pouvez fixer la même pour tous les côtés en utilisant **margin** (c'est le même fonctionnement que pour **border**). Mais vous pouvez fixer individuellement la **margin-top**, la **margin-right**, la **margin-bottom** ou la **margin-left**.

Il est possible d'utiliser des valeurs négatives et dans ce cas la boîte sera décalée par rapport à sa position normale.

Pour appliquer des propriétés différentes suivant les côtés, il existe des raccourcis plutôt que de devoir définir la propriété pour chaque côté.

Ainsi si vous utilisez **margin** que vous lui donnez :

1. une seule valeur : elle s'appliquera aux quatre côtés.

2. deux valeurs : la première valeur s'applique aux côtés haut et bas et la seconde aux côtés gauche et droit.
3. trois valeurs : la première valeur s'applique au côté haut, la seconde aux côtés gauche et droit et la troisième au côté bas.
4. quatre valeurs : les valeurs s'appliquent dans le sens des aiguilles d'une montre en commençant par le haut.

## 12.5 La fusion des marges

Les marges haute et basse des blocs sont fusionnées en une seule marge dont la taille est la plus grande des deux marges fusionnées.

Ainsi si nous prenons les éléments **HTML** suivants :

```
<div class="container">
  <p class="p1">Je suis un paragraphe avec une marge en bas de 50px.</p>
  <p class="p2">Je suis un paragraphe avec une marge en haut de 30px</p>
</div>
```

Et que nous appliquons ce CSS :

```
.container {
  width: 400px;
  height: 200px;
  border: 5px solid red;
}
.p1 {
  margin-bottom: 50px;
  border: 1px solid black;
}
.p2 {
  margin-top: 30px;
  border: 1px solid black;
}
```

Alors la marge entre les deux paragraphes ne sera pas égale à la somme des marge basse et haute, mais à la marge la plus grande des deux.

## 12.6 Le centrage horizontale avec **margin**

Il est possible de centrer horizontalement un élément en faisant :

```
margin: auto;
```

Cela appliquera la marge appropriée à gauche et à droite pour centrer horizontalement l'élément.

La marge en haut et en bas de l'élément sera définie à zéro.



## 12.7 Le remplissage avec `padding`

Le remplissage est l'espace blanc autour de la boîte de contenu. Elle se définit en utilisant `padding`.

Pour fixer le `padding`, vous pouvez fixer la même valeur pour tous les côtés en utilisant `padding` (c'est le même fonctionnement que pour `border` et `margin`).

Mais vous pouvez fixer individuellement le `padding-top`, le `padding-right`, le `padding-bottom` ou le `padding-left`.

Pour appliquer des propriétés différentes suivant les côtés, il existe des raccourcis plutôt que de devoir définir la propriété pour chaque côté.

Ainsi si vous utilisez `padding` que vous lui donnez :

1. **une seule valeur** : elle s'appliquera aux quatre côtés.
2. **deux valeurs** : la première valeur s'applique aux côtés haut et bas et la seconde aux côtés gauche et droit.
3. **trois valeurs** : la première valeur s'applique au côté haut, la seconde aux côtés gauche et droit et la troisième au côté bas.
4. **quatre valeurs** : les valeurs s'appliquent dans le sens des aiguilles d'une montre en commençant par le haut.

## 13 La propriété `display`

### 13.1 La propriété `display`

La propriété `display` définit le type d'affichage utilisée pour le rendu d'un élément.

Comme nous le verrons plus tard dans la formation, elle permet également de définir la disposition utilisée pour les éléments enfants d'un élément. C'est ce qu'on appelle l'affichage intérieur qui permet d'organiser les éléments contenus dans un élément.

Il y a donc deux dimensions pour la propriété `display` : le type d'affichage extérieur (que nous allons étudier dans cette leçon) et le type d'affichage intérieur que nous étudierons dans les chapitres suivants.

### 13.2 Disposition des éléments par défaut

Nous avons vu pour l'instant que les éléments `HTML` peuvent être de deux types : de bloc ou en ligne.

Pour rappel, les éléments de bloc, comme par exemple `div`, ont comme spécificité : leur contenu prend toute la largeur disponible de son élément parent et sa hauteur est celle de son contenu. Chaque élément se met sur une nouvelle ligne. Les propriétés `width` et `height` sont respectées. Le `padding`, la `marge` et la `bordure` seront appliqués et pousseront les autres boîtes.

Les éléments en ligne, comme par exemple `span`, prennent la hauteur et la largeur de leur contenu. Chaque élément reste sur sa ligne si possible. Les propriétés `width` et `height` ne sont appliquées. Le `padding`, la `marge` et la `bordure` seront appliqués mais ne pousseront pas les autres boîtes.

### 13.3 Fixer la disposition des éléments

Vous pouvez changer la disposition des éléments en utilisant la propriété `display`. Par exemple :

```
p {
  display: inline;
}
span {
  display: block;
}
```

### 13.4 Utiliser la disposition **inline-block**

Cette disposition est à mi-chemin entre les dispositions de bloc et en ligne.

Les caractéristiques sont les suivantes :

1. Les propriétés **width** et **height** sont respectées (propriété des éléments de bloc).
2. Le **padding** , la marge et la bordure seront appliqués et pousseront les autres boîtes (propriété des éléments de bloc).
3. Chaque élément reste sur sa ligne si possible (propriété des éléments en ligne).
4. Chaque élément prend la taille de son contenu (propriété des éléments en ligne), sauf si les propriétés **width** et **height** sont spécifiées.

## 14 L'alignement horizontal et les boîtes flottantes

### 14.1 L'alignement horizontal

La propriété **text-align** définit l'alignement horizontal dans un élément de bloc. Les valeurs possibles sont **left** , **right** , **center** et **justify** .

Il vous suffit de regarder les exemples pour comprendre leur effet.

### 14.2 Les boîtes flottantes

La propriété **float** permet de faire flotter des images dans un bloc de texte.

Elle était utilisée également pour contrôler la disposition d'autres éléments mais depuis l'apparition des grilles et des boîtes flexibles que nous étudierons, elle n'a plus que ce cas d'utilisation.

L'élément sur lequel on applique la propriété **float** est retiré du cours normal de la disposition des éléments du document.

Il est collé sur le côté précisé (gauche ou droite) de son conteneur parent.

Tout contenu est ensuite disposé après ou avant l'élément flottant dans le cours normal de la mise en page.

Si vous voulez qu'un élément occupe sa place dans le cours normal sans être disposé autour de l'élément flottant, il suffit d'utiliser la propriété **clear** sur l'élément.

Les valeurs possibles sont **left** (pour dégager des éléments qui flottent à gauche), **right** (pour dégager des éléments qui flottent à droite) et **both** (pour dégager des éléments qui flottent à droite et à gauche).

## 15 La position

### 15.1 Le positionnement

Le positionnement permet de modifier la position d'un élément **HTML** .

Pour définir la façon dont un élément doit être positionné dans un document, il faut utiliser la propriété **position** .

Il existe également des propriétés **top** , **right** , **bottom** et **left** pour déterminer l'emplacement de l'élément.

### 15.2 La position static

Par défaut, les éléments **HTML** ont une propriété **position** définie à **static** . L'élément est positionné dans le flux normal en respectant sa disposition (de bloc, en ligne ou **inline-block** par exemple).

### 15.3 La position relative

La position **relative** permet de décaler un élément de sa position normale, c'est-à-dire **static** .

C'est pour cette raison que cette **position** s'appelle relative : la **position** est définie relativement à sa position normale.

Pour définir la **position** de l'élément il faut utiliser les propriétés **top** , **right** , **bottom** et **left** .

Par exemple :

```
position: relative;
top: 30px;
```

Ici, l'élément sera décalé de **30px** vers le bas à partir de la position qu'il aurait normalement eu.

### 15.4 La position absolute

La position **absolute** permet de retirer du flux normal un élément et de le positionner par rapport à son élément parent positionné.

Un élément est dit positionné si il a une propriété **position** fixée à **absolute** , **relative** ou **fixed** .

Si l'élément n'a pas de parent positionné, il sera positionné par rapport au **viewport** . Avec **absolute** , les propriétés **top** , **right** , **bottom** et **left** définissent la distance à laquelle l'élément doit être positionné par rapport aux côtés de l'élément parent positionné.

A noter que les éléments positionnés de manière **absolute** ne prennent plus d'espace lorsqu'il s'agit de positionner les autres éléments.

Par exemple :

```
position: absolute;
top: 10px;
```

Dans ce cas, l'élément positionné de manière absolue se placera à 10px du haut de l'élément positionné parent, ou à défaut du **viewport** .

## 15.5 La position `fixed`

La position `fixed` permet de positionner un élément de manière absolue par rapport à la fenêtre du navigateur lui-même.

Cela signifie que même en cas de défilement ( `scroll` ), l'élément restera à la même position par rapport à la fenêtre.

Par exemple :

```
position: fixed;
top: 650px;
left: 300px;
```

## 15.6 La position `sticky`

La position `sticky` permet de définir la position d'un élément de manière relative puis de passer en position fixe à partir d'un certain seuil.

Par exemple :

```
position: sticky;
top: 100px;
```

L'élément restera à sa position normale, puis lorsque le défilement aura atteint `100px` au dessus de l'élément, il passera en `position : fixed` et restera toujours à `100px` du `top` de la fenêtre lors du défilement.

# 16 Les pseudo-classes et les curseurs

## 16.1 Les pseudo-classes

Une pseudo-classe permet de sélectionner un état particulier d'un élément.

Il s'agit donc d'une extension à un sélecteur. Les pseudo-classes commencent par `:` suivi de l'état à cibler.

Par exemple :

```
:hover
```

Permet de cibler l'élément sélectionné lorsque le curseur survole l'élément. La syntaxe est la suivante :

```
sélecteur:pseudo-classe {
  propriété: valeur;
}
```

Nous allons voir brièvement la plupart des pseudo-classes.

## 16.2 Les pseudo-classes génériques

Certaines pseudo-classes sont liées à la position du curseur et aux clics.

- **:active** : permet d'indiquer la prise en compte de l'activation de l'élément par l'utilisateur. Généralement, un élément sera **active** entre le moment où l'utilisateur commence à cliquer et le relâchement du clic.
- **:hover** : permet de sélectionner un élément au moment où l'utilisateur le survole avec le curseur.
- **:disabled** / **:enabled** : permet de cibler un élément lorsqu'il est désactivé ou activé. Un élément est activé s'il peut être sélectionné, s'il est cliquable, si on peut y saisir du texte ou le focus.
- **:empty** : permet de cibler un élément lorsqu'il n'a pas d'élément enfant (par exemple une `div` vide).
- **:first-child** / **:last-child** : permet de cibler un élément qui est le premier / le dernier élément enfant par rapport à son élément parent.
- **:first-of-type** : permet de cibler un élément qui est le premier élément enfant du type donné (précédent la pseudo-classe) par rapport à son élément parent.
- **:focus** : permet de cibler un élément qui est **focus**. Par exemple le champ d'un formulaire, une liste d'options etc.
- **:focus-within** : permet de cibler un élément lorsque l'un des descendants a le **focus** (voir exemple).
- **:not()** : permet de cibler un élément qui n'est pas sélectionné par le sélecteur passé. Par exemple **:not(p)** : qui n'est pas un paragraphe.

## 16.3 Les pseudo-classes relatives aux liens

- **:link** : permet de cibler les éléments contenant un lien qui n'a pas encore été visité.
- **:visited** : permet de cibler les éléments contenant un lien qui a été visité.

## 16.4 Les pseudo-classes relatives aux champs de formulaire

- **:valid** / **:invalid** : permet de cibler un champ qui est valide ou invalide.
- **:optional** / **:required** : permet de cibler un champ qui est optionnel ou requis.
- **:read-only** : permet de cibler un champ qui est en lecture seul.

## 16.5 Les pseudo-classes relatives aux **input**

- **:in-range** : permet de cibler un élément **input** lorsque sa valeur est comprise entre les valeurs des attributs **min** et **max**.
- **:out-of-range** : permet de cibler un élément **input** lorsque sa valeur n'est pas comprise entre les valeurs des attributs **min** et **max**.

## 16.6 La pseudo-classe relatives aux boutons **radio**, aux listes d'option et aux cases à cocher

























**:checked** : l'élément est **checked** lorsqu'il est coché ou lorsqu'une **option** est sélectionnée dans une liste de valeurs.

Il y a d'autres pseudo-classes plus avancées, mais nous n'en aurons pas l'utilité dans les cours et elles sont très rarement utilisées.

## 16.7 Les curseurs

La propriété `cursor` permet de définir la forme du curseur lorsque le pointeur de la souris est au-dessus de l'élément.

Voici les principaux curseurs :

 auto	 move	 no-drop	 col-resize
 all-scroll	 pointer	 not-allowed	 row-resize
 crosshair	 progress	 e-resize	 ne-resize
 default	 text	 n-resize	 nw-resize
 help	 vertical-text	 s-resize	 se-resize
 inherit	 wait	 w-resize	 sw-resize

## 17 Visibilité, overflow et Z-index

### 17.1 Cacher un élément

Parfois, vous souhaitez rendre invisible un élément. Nous verrons des cas d'utilisation plus tard dans les cours.

Vous avez deux manières de cacher un élément `HTML` avec du `CSS` .

### 17.2 Utilisation de la propriété `visibility`

Nous pouvons utiliser la propriété `visibility` pour cacher l'élément :

```
visibility: hidden;
```

L'élément est invisible (totalement transparent) mais il continue d'avoir un impact sur la disposition des autres éléments et d'occuper de l'espace dans le flux normal.

### 17.3 Utilisation de la propriété `display`

Il est également possible d'utiliser la propriété `display` pour cacher un élément :

```
display: none;
```

Dans ce cas, l'élément sera transparent mais il n'occupera pas non plus d'espace. Les autres éléments seront disposés comme si l'élément n'existait pas.

### 17.4 Le `z-index`

La propriété `z-index` permet de gérer le chevauchement des éléments.

En effet, par défaut, les éléments `HTML` sont empilés dans l'ordre dans lequel ils sont déclarés.

Par défaut les éléments ont un `z-index` de 0. Les éléments avec un `z-index` de 1 seront au-dessus, ceux avec un `z-index` de 2 encore au-dessus, et ainsi de suite.

## 17.5 Gestion des dépassements

La gestion des dépassement se fait avec la propriété `overflow` qui définit comment gérer le dépassement du contenu d'un élément dans son bloc.

Les quatre valeurs possibles sont `visible` (par défaut), `hidden`, `scroll` ou `auto`.

1. `visible` : Le contenu n'est pas rogné et peut éventuellement dépasser de la boîte de contenu.
2. `hidden` : Le contenu est rogné si besoin pour ne pas dépasser de la boîte de contenu et aucune barre de défilement n'est affichée.
3. `scroll` : Le contenu est rogné si besoin pour ne pas dépasser de la boîte de contenu et les navigateurs affichent des barres de défilement horizontale et verticale.
4. `auto` : Le contenu est rogné si besoin pour ne pas dépasser de la boîte de contenu et les navigateurs affichent la ou les barres de défilement nécessaire(s).