



CASCADING STYLE SHEETS

CASCADING STYLE SHEETS.....	0
Un peu d’histoire.....	3
1. Syntaxe CSS.....	4
2. UN SÉLECTEUR CSS ?	6
Types de sélecteurs CSS	6
2.1 Sélecteurs de type	6
2.2 Sélecteurs de class.....	6
2.3 Sélecteurs d'identifiant	7
2.4 Sélecteurs d'attributs	7
2.5 Sélecteurs combinés.....	7
2.6 Conclusion	9
3. Les propriétés	9
4. Le texte	12
5. Couleurs	14
A. Avantages et limitations des valeurs de type « nom de couleur ».....	23
B. Les notations de type RGB	23
A. Avantages et limitations des valeurs de type « nom de couleur ».....	25
6. Images	32
A. Comment mettre une image dans une forme circulaire en Css ?.....	32
B. Comment faire des images qui s'adaptent à son conteneur en Css?	33
C. Comment faire une image en arrière-plan ?	34
7. Bordures	36
8. Box-Model	38
9. margin	44
10. Padding	46
11. Pseudo-class	47
12. Positions.....	47

13.	Z-Index.....	50
14.	Flexbox / Grid	51
15.	EXERCICES	51

Un peu d'histoire

Le standard définissant CSS a évolué au fil des versions :

- **CSS1** publié en décembre 1996,
- **CSS2** publié en mai 1998,
- **CSS3** dont un premier essai a été publié en 1999, est encore en train d'évoluer avec des modules ajoutés en **2021**,
- **CSS4** dont la définition est en cours depuis 2010.

CSS (*Cascading Style Sheets*, feuilles de style en cascade) est un langage informatique essentiel utilisé pour décrire la présentation des documents HTML (et XML).

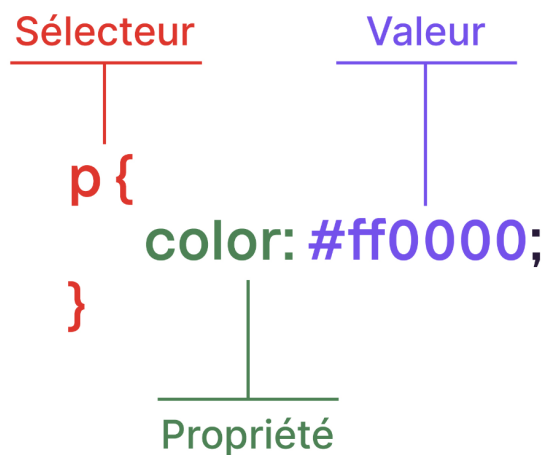
*Il permet de contrôler la mise en page et l'apparence des pages web
aka the **look and feel**.*

*(Style Sheets), La feuille de style permet construire le design de
l'interface en ciblant balise ou class en leur appliquant de
nombreuse propriétés qui interagissent
par héritage (Cascading)*

1. SYNTAXE CSS

La syntaxe de CSS est basée sur des règles, où chaque règle cible un ou plusieurs éléments HTML.

Par exemple, ici le sélecteur est `p` et la propriété suivante change la couleur d'un paragraphe :



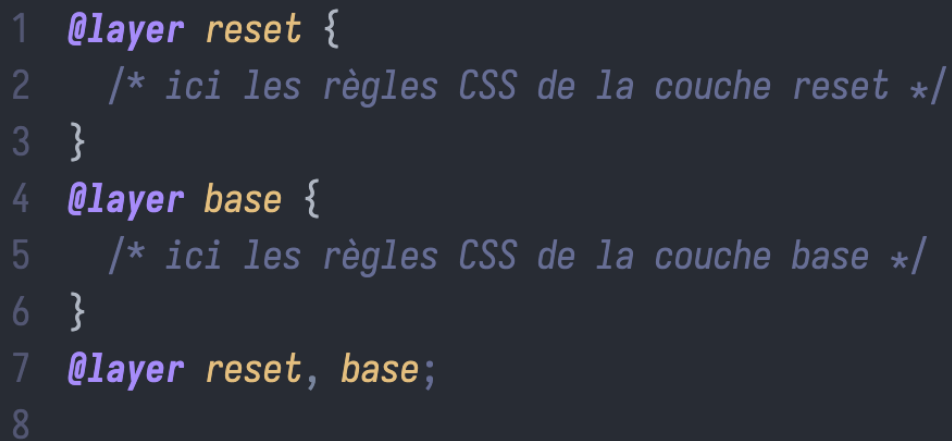
2. CUSTOMS LAYERS

Les styles CSS doivent leur nom à la Cascade, qui est un bien joli et complexe algorithme tenant compte de nombreux paramètres tels que l'origine des styles, la spécificité des sélecteurs ainsi que leur ordre d'apparence.

La Cascade, c'est l'essence même de CSS. C'est ce qui fait son utilité, sa beauté... et c'est aussi le pire cauchemar des intégratrices et intégrateurs.

2.1 DÉCLARER DES LAYERS

Il existe plusieurs moyens de créer des couches de styles : la règle `@layer` avec styles associés, la même sans styles, ou l'import de fichiers de styles externes.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It contains CSS code for declaring layers.

```
1 @layer reset {  
2     /* ici les règles CSS de la couche reset */  
3 }  
4 @layer base {  
5     /* ici les règles CSS de la couche base */  
6 }  
7 @layer reset, base;  
8
```

Import de styles externes

La notion de layer peut être associée à la règle `@import` :

```
@import url("reset.css") layer(reset);
```

En plus de l'import `via @import`, le W3C travaille sur une version importée avec l'élément `<link>`.

Concrètement cela représente un moyen très simple de pouvoir :

Importer les styles d'un framework tel que Bootstrap (au hasard) au sein de `layers`, donc avec une spécificité moindre

Pouvoir redéfinir ses propres styles sans se soucier du poids des sélecteurs Bootstrap ni même des `!important`... et si j'évoque ce framework en particulier, ce n'est pas tout à fait anodin (oui, il y a bien 1307 `!important` dans ce seul fichier CSS)

```
@import url("bootstrap.css") layer/framework);
```

3. UN SÉLECTEUR CSS ?

Un sélecteur CSS est une expression qui permet de cibler un ou plusieurs éléments HTML pour leur appliquer des règles de style.

Chaque règle CSS commence par un sélecteur qui indique quels éléments seront affectés par les déclarations de style qui suivent.

3.1 SÉLECTEURS DE TYPE

Le sélecteur de type cible tous les éléments d'un même type (balise HTML). Par exemple, pour cibler tous les éléments `<p>`

```
p {  
  color: blue; /* Applique la couleur bleue à tous les paragraphes */  
}
```

3.2 SÉLECTEURS DE CLASS

Les sélecteurs de classe ciblent les éléments ayant une classe spécifique. On utilise un point (.) avant le nom de la classe. Par exemple, pour cibler tous les éléments avec la classe `introduction`

```
.introduction {  
  font-weight: bold;  
}
```

```
/* Met en gras tous les éléments avec la classe "introduction" */  
}
```

3.3 SÉLECTEURS D'IDENTIFIANT

Les sélecteurs d'identifiant ciblent un élément unique ayant un identifiant spécifique. On utilise un dièse (#) avant le nom de l'identifiant. Par exemple, pour cibler un élément avec l'identifiant `menu` :

```
#menu {  
  
    background-color: gray;  
    /* Applique un fond gris à l'élément avec l'identifiant "menu" */  
}
```

3.4 SÉLECTEURS D'ATTRIBUTS

Ces sélecteurs permettent de cibler des éléments en fonction de leurs attributs. Par exemple, pour sélectionner tous les liens (`<a>`) ayant un attribut `target` :

```
a[target] {  
  
    text-decoration: underline;  
    /* Souligne les liens ayant un attribut "target" */  
}
```

3.5 SÉLECTEURS COMBINÉS

Il est possible de combiner plusieurs sélecteurs pour appliquer les mêmes styles à plusieurs éléments. Par exemple :

```
h1, .special {  
  
    color: red;  
    /* Applique la couleur rouge à tous les h1 et aux éléments avec la classe  
    "special" */  
}
```


3.6 LE SÉLECTEUR UNIVERSEL

Ce sélecteur permet de cibler tous les nœuds d'un document.

Exemple : `*` permettra de cibler tous les éléments du document.

3.7 LES COMBINATEURS

Les sélecteurs de voisin direct

Le combineur '`+`' permet de sélectionner les nœuds qui suivent immédiatement un élément donné. **Syntaxe :** `A + B` **Exemple :** `div + p` permettra de cibler n'importe quel élément `<p>` qui suit immédiatement un élément `<div>`.

Les sélecteurs de voisins

Le combineur '`~`' permet de sélectionner les nœuds qui suivent un élément et qui ont le même parent. **Syntaxe :** `A ~ B` **Exemple :** `p ~ span` permettra de cibler les éléments `` qui suivent (immédiatement ou non) un élément `<p>` et qui ont le même élément parent.

Les sélecteurs d'éléments enfants

Le combineur '`>`' permet de sélectionner les nœuds qui sont des enfants directs d'un élément donné. **Syntaxe :** `A > B` **Exemple :** `ul > li` permettra de cibler tous les éléments `` qui sont directement situés sous un élément ``.

Les sélecteurs d'éléments descendants

Le combineur '' (**espace**) permet de sélectionner les nœuds qui sont des descendants (pas nécessairement des enfants directs) d'un élément donné.

Syntaxe : A B **Exemple :** `div span` permettra de cibler n'importe quel élément `` situé à l'intérieur d'un élément `<div>`.

3.8 CONCLUSION

Les sélecteurs CSS sont variés et permettent de cibler les éléments de manière précise.

En maîtrisant les sélecteurs de base comme ceux de type, de classe et d'identifiant, vous pouvez commencer à styliser vos pages web efficacement.

Plus vous vous familiariserez avec les différents types de sélecteurs, plus vous pourrez créer des mises en page complexes et attrayantes.

4. LES PROPRIÉTÉS

Les propriétés CSS sont des éléments fondamentaux qui permettent de contrôler l'apparence et le comportement des éléments HTML sur une page web. Voici un aperçu des types de propriétés CSS, leurs usages et quelques exemples.

```
background : #f30 url(monimage.png) no-repeat top center;
```

Les propriétés raccourcies permettent de définir plusieurs valeurs en une seule déclaration, ce qui rend le code plus concis et lisible.

Par exemple :

`background` : regroupe les propriétés `background-color`, `background-image`, `background-repeat`, et `background-position`.

```
background : #f30 url(monimage.png) no-repeat top center;
```

`border` : combine `border-width`, `border-style`, et `border-color`.

```
border: 1px solid #000;
```

`margin` : permet de définir les marges en une seule déclaration.

```
margin: 10px 5px 10px 5px; /* haut, droit, bas, gauche */
```

Nous n'avons pas les mêmes valeurs !

pour chaque **propriété** CSS, on doit indiquer une **valeur**.

Par exemple :

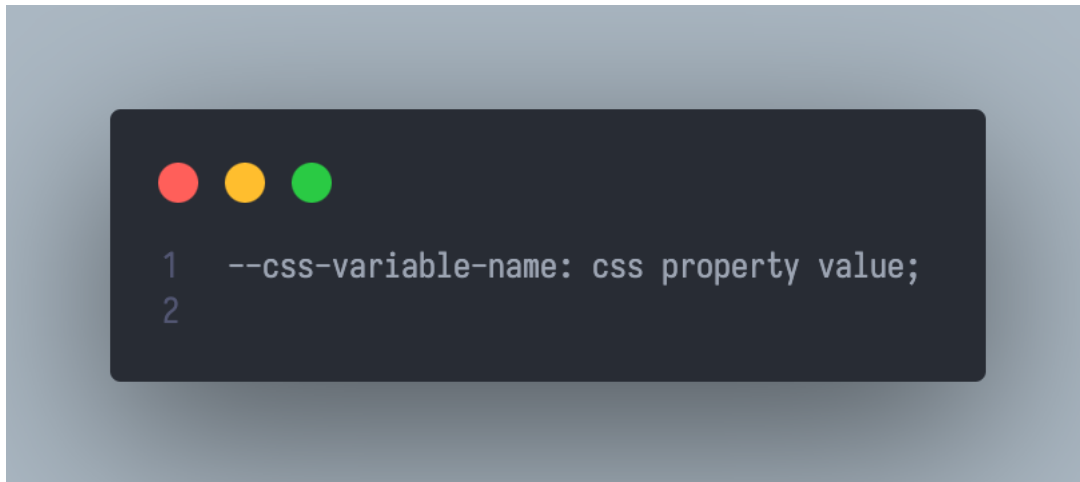
```
color : red ;
```

```
font-size : 16px ;
```

4.1 LES CUSTOMS PROPERTIES AKA VARIABLES

Je vais vous montrer comment créer des variables CSS sur le pseudo-sélecteur `:root` et comment y accéder à l'aide de la fonction `var()`.

La meilleure pratique consiste à définir **toutes vos variables** au début de votre feuille de style. Pour les projets plus importants, il est courant de créer **un fichier séparé** pour vos variables de couleurs personnalisées afin de pouvoir les réutiliser dans d'autres feuilles de style.



Dans cet exemple, je souhaite créer des variables de couleur d'arrière-plan et de texte personnalisées que je pourrai réutiliser dans la feuille de style. Je vais nommer ces variables `--main-bg-color` et `--main-text-color`.



Je vais placer ces variables à l'intérieur du pseudo-sélecteur `:root` qui représente l'élément racine de mon document HTML.

```
:root {  
  --main-bg-color: #000080;  
  --main-text-color: #fff;  
}
```

Dans mon sélecteur `body`, je vais référencer ces variables en utilisant la fonction `var()`.

```
body {  
  background-color: var(--main-bg-color);  
  color: var(--main-text-color);  
}
```

```
:root {  
  --main-bg-color: #000080;  
  --main-text-color: #fff;  
}
```

Exemple: <https://codepen.io/jessica-wilkins/pen/LYeoOmP>

5. LE TEXTE

Les propriétés CSS dédiées au texte permettent de contrôler l'apparence et l'alignement du texte dans une page web.

Voici un aperçu de quelques une des principales propriétés :

`font` : regroupe les propriétés relatives à la police, comme `font-family`, `font-size`, `font-weight`, interlignage. C'est la forme raccourci pour les props Font.

`font: italic bold 14px verdana;`

`color` : permet de définir la couleur du texte
(exemple : `color: red;`)

`font-size` : définit la **taille** du texte en pixels ou en rem (1) ⁱ
(exemple : `font-size: 1rem;`)

`font-family` : permet de définir le nom de police à utiliser pour le texte
(exemple : `font-family: Arial, sans-serif;`)

`font-weight` : définit la graisse du texte (*normal*, *bold*, *lighter*, etc.) (exemple : `font-weight: bold;`)

text-align : permet d'aligner le texte à gauche (left), au centre (center) ou à droite (right). Il peut aussi prendre la valeur justify pour aligner le texte en ligne entière à gauche et à droite (exemple : ``text-align: center;``)

line-height : définit la hauteur de la ligne (lignes d'écriture) (exemple : ``line-height: 1.5;``)

letter-spacing : permet d'ajouter un espacement entre les lettres (exemple : ``letter-spacing: 2px;``)

text-decoration : permet de définir la décoration du texte, tels que sous-ligné (*underline*), sur-ligné (*overline*) ou barré (*strikethrough*). Il peut également prendre les valeurs *none* (aucune décoration), *initial* (héritage par défaut) ou *inherit* (héritage de la propriété parent) (exemple : ``text-decoration: underline;``)

text-transform : permet de transformer le texte, comme en majuscules (*uppercase*), minuscules (*lowercase*) ou en capitalisations alternées (*capitalize*). Il peut également prendre les valeurs *none* (aucune transformation) ou *inherit* (héritage de la propriété parent) (exemple : ``text-transform: uppercase;``)

word-spacing : permet d'ajouter un espacement entre les mots (exemple : ``word-spacing: 2px;``)

word-break : permet de définir comment casser un mot pour l'afficher sur plusieurs lignes (exemple : `word-break: break-all;``)

text-indent : permet d'ajouter une indentation au début du texte, cela peut être utile pour indiquer une indentation de texte (exemple : ``text-indent: 20px;``)

6. COULEURS

Cette propriété est à la fois très *simple* à utiliser et relativement *complexe* à parfaitement maîtriser 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) ;
- La plus courante, une notation hexadécimale, #FFFFFF ;
- Une notation RGB ou RGBa (alpha) ;
- Une notation HSL ou HSLa (alpha) ;

Les couleurs fondamentales

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	#008000	
Lime	#00FF00	
Maroon	#800000	
Navy	#000080	
Olive	#808000	
Purple	#800080	
Red	#FF0000	
Silver	#C0C0C0	
Teal	#008080	
White	#FFFFFF	

Yellow

#FFFF00

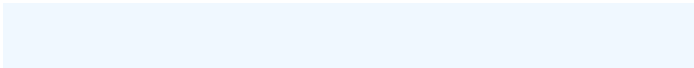
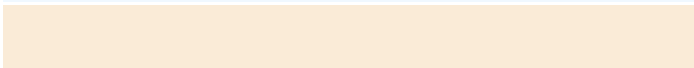




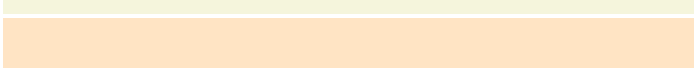








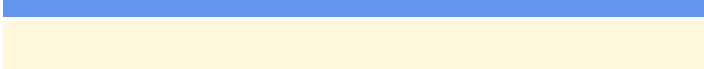





Les autres couleurs nommées

Avec l'évolution des performances et des langages, le support pour de nouvelles couleurs a progressivement été ajouté.



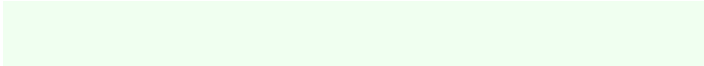



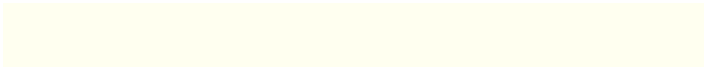








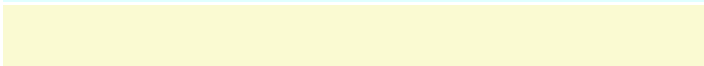



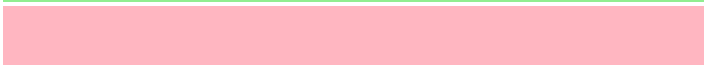


Ainsi, aujourd'hui, les navigateurs reconnaissent et supportent l'utilisation de plus de **140 noms** de couleurs différents. Nous allons donc pouvoir passer chacune de ces valeurs à la propriété `color` pour définir une nouvelle couleur pour un texte.

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	

6.1 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 voudrions 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**, **RGBA**, **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.

6.2 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.

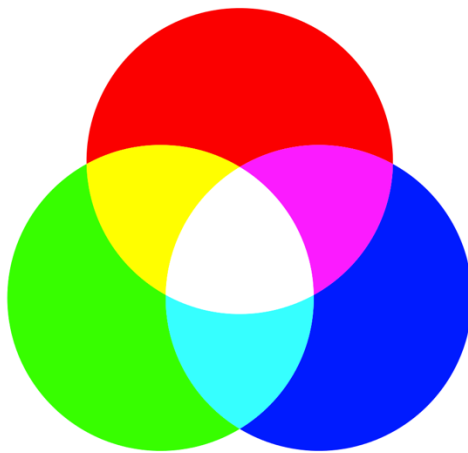
Chaque niveau d'intensité qu'on va renseigner va être compris entre 0 (intensité nulle ou absence de la couleur en question) et 255 (intensité maximale ou couleur pure).

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.

Par exemple : `h1{color : rgb(255, 180, 0);}`

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.



Exemple : <https://codepen.io/pierregiraud/pen/BrLvqP>

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 maîtriser 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, `#FFFFFF` ;
- Une notation RGB ou RGBA (alpha (transparence));
- 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.

6.3 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 voudrions 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.

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, reprenez 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 sorte de 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 du 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	Equivalent 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 devons faire précéder cette notation par un dièse (symbole #).

Prenons immédiatement quelques exemples pour illustrer cela :

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 du 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.

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.

Lien : <https://www.w3.org/wiki/CSS3/Color/HSL>

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.

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-transparentes (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).

Comme vous pouvez le tester, 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.

7. IMAGES

Dans ce chapitre nous allons découvrir par l'exemple comment le CSS intervient sur la balise ``. Pour la balise `<picture>` se référer au cours.

A. COMMENT METTRE UNE IMAGE DANS UNE FORME CIRCULAIRE EN CSS ?

```
.image-ronde{  
  width : 150px; height : 150px;  
  border: none;  
  border-radius : 75px;  
}
```

Exemple une balise image dans un cercle

```
.image-ronde-bg{  
  background-size : contain;  
  background-position : 50% 50%;  
  background-image : url(/img/exemple/filter-image.jpg);  
  display : inline-block;  
  width : 150px; height : 150px;  
  border: none;  
  border-radius : 75px;  
}
```

Exemple une image en arrière-plan dans un cercle

7.1 COMMENT FAIRE DES IMAGES QUI S'ADAPTENT À SON CONTENEUR EN CSS?

Vous pouvez en Css, faire en sorte qu'une image (`img`) s'adapte en largeur à son conteneur parent. Pour cela vous devez utiliser la propriété Css `width` à `100%` sur l'image et préciser la propriété Css `height` à `auto`.

Les proportions de l'image seront respectées. L'image peut être plus grande en hauteur que son conteneur, si ce dernier à une hauteur spécifiée. Dans ce cas vous pouvez utiliser sur le conteneur la propriété Css `overflow` à `hidden`.

```
.responsive-w{  
  width : 100%; height : auto;  
}
```

Exemple d'écriture Css d'une image adaptative en largeur en fonction de son conteneur

7.2 COMMENT FAIRE UNE IMAGE EN ARRIÈRE-PLAN ?

La propriété background en CSS permet de contrôler l'arrière-plan d'un élément. Il s'agit d'une propriété abrégée, ce qui signifie qu'elle vous permet d'écrire ce qui serait plusieurs propriétés CSS en une seule.

Voici un exemple :

```
body {  
  background:  
    url(sweettexture.jpg) /* image */  
    top center / 200px 200px /* position / size */  
    no-repeat /* repeat */  
    fixed /* attachment */  
    padding-box /* origin */  
    content-box /* clip */  
    red; /* color */  
}
```

Background est composé de huit autres propriétés :

- background-image
- background-position
- background-size
- background-repeat
- background-attachment
- background-origin
- background-clip
- background-color

Vous pouvez utiliser n'importe quelle combinaison de ces propriétés, dans presque n'importe quel ordre (bien que l'ordre recommandé dans la spécification soit ci-dessus).

Il y a cependant un problème : tout ce que vous ne spécifiez pas dans la propriété `background` est automatiquement défini par défaut.

Ainsi, si vous faites quelque chose comme ceci :

```
body {  
  background-color: red;  
  background: url(sweettexture.jpg);  
}
```

L'arrière-plan sera transparent, au lieu d'être rouge.

La solution est pourtant simple : il suffit de définir la couleur d'arrière-plan après l'arrière-plan, ou d'utiliser l'abréviation (par exemple, `background : url(...) red ;`).

Plusieurs background en un

CSS3 a ajouté la prise en charge de **plusieurs arrière-plans**, qui se **superposent** les uns aux autres.

Toute propriété relative aux arrière-plans peut prendre la forme d'une liste séparée par des virgules, comme ceci :

```
body {  
  background: url(sweettexture.jpg), url(texture2.jpg) black;  
  background-repeat: repeat-x, no-repeat;  
}
```

Chaque valeur de la liste **séparée par des virgules** correspond à une couche : la première valeur est la couche supérieure, la deuxième valeur est la deuxième couche et la couleur d'arrière-plan est toujours la dernière couche.

Exemple : <https://codepen.io/webinspect/pen/emBzRd>

8. BORDURES

La propriété border est une syntaxe abrégée en CSS qui accepte plusieurs valeurs pour dessiner une ligne autour de l'élément auquel elle s'applique.

```
.element{  
  border: 3px solid red;  
  width: 200px;  
  aspect-ratio: 1;  
}
```

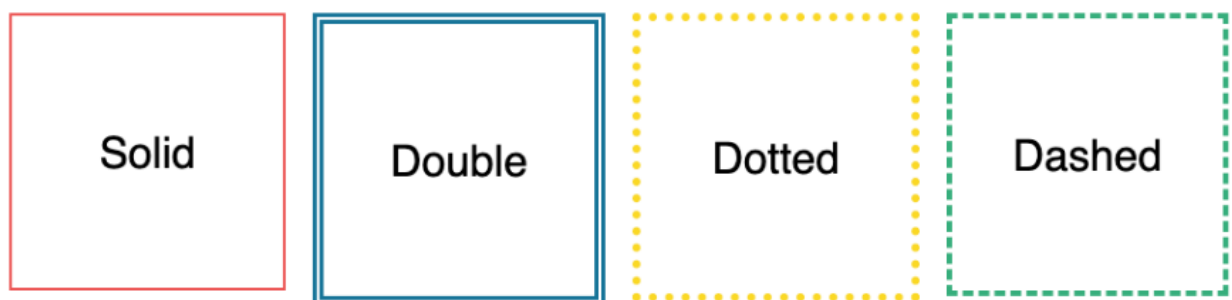
Syntax

`border: <border-width> <border-style> <border-color>;`

La propriété border accepte une ou plusieurs des valeurs suivantes en combinaison :

- **border-width** : spécifie l'épaisseur de la bordure.
 - px, em, rem, vh et vw : valeur numérique mesurée en unités px, em, rem, vh et vw.
- **border-style** (style de bordure) : Spécifie le type de ligne tracée autour de l'élément :
 - solid : une ligne solide et continue.
 - none (par défaut) : Aucune ligne n'est tracée.
 - hidden (caché) : Une ligne est tracée, mais elle n'est pas visible. Cela peut être pratique pour ajouter un peu de largeur à un élément sans afficher de bordure.
 - dashed (pointillé) : Une ligne composée de tirets.

- pointillé : une ligne composée de points.
- double : Deux lignes sont tracées autour de l'élément.
- rainure : Ajoute un biseau basé sur la valeur de la couleur de manière à ce que l'élément semble enfoncé dans le document.
- crête : Semblable à la rainure, mais en inversant les valeurs de couleur de manière à ce que l'élément paraisse surélevé.
- inset (incrustation) : Ajoute un ton divisé à la ligne qui donne l'impression que l'élément est légèrement enfoncé.
- outset (départ) : Semblable à l'incrustation, mais en inversant les couleurs de manière à ce que l'élément apparaisse légèrement en relief.



- **border-color** : spécifie la couleur de la bordure et accepte toutes les valeurs de couleur valides.
- **Border-radius** : valeur des coins arrondis

```
.element {  
  border: 3px solid #f8a100;  
}
```

...la même chose que:

```
.element {  
  border-top: 3px solid #f8a100;  
  border-right: 3px solid #f8a100;  
  border-bottom: 3px solid #f8a100;  
  border-left: 3px solid #f8a100;  
}
```

Exemple : <https://codepen.io/geoffgraham/pen/gxvKbQ>

9. Box-MODEL

Tout élément de conception web est une boîte rectangulaire.

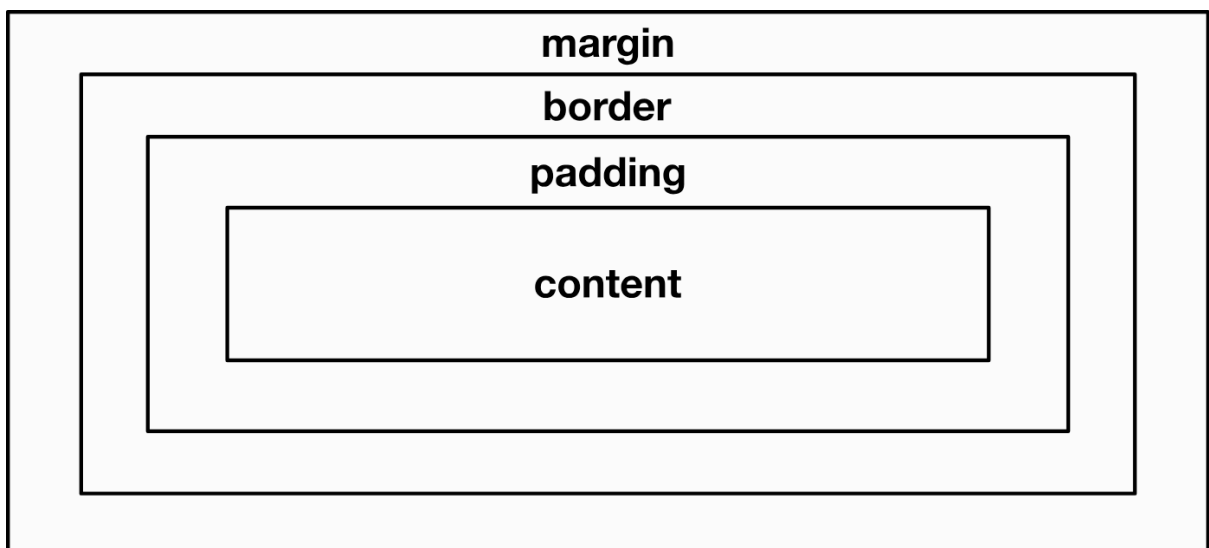
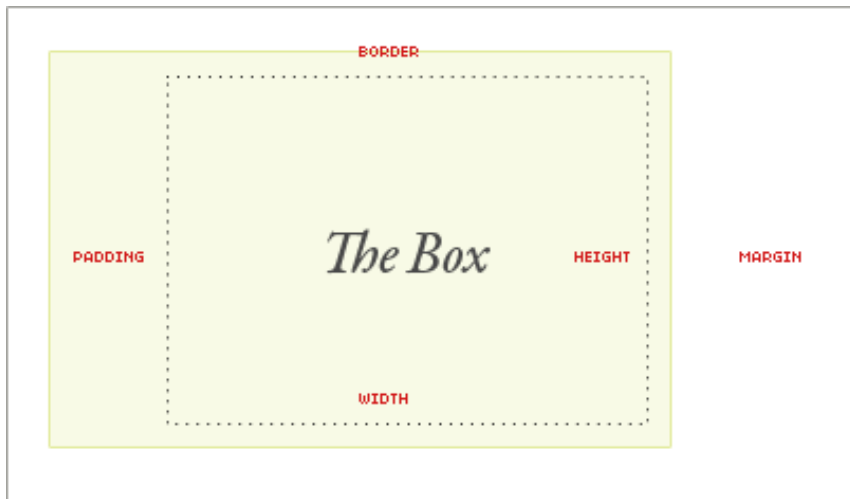
C'est ce qui m'a permis de commencer à comprendre la conception web basée sur les CSS et de réaliser les mises en page que je souhaitais.

Nous parlerons plus tard du positionnement de ces boîtes et de leur comportement.

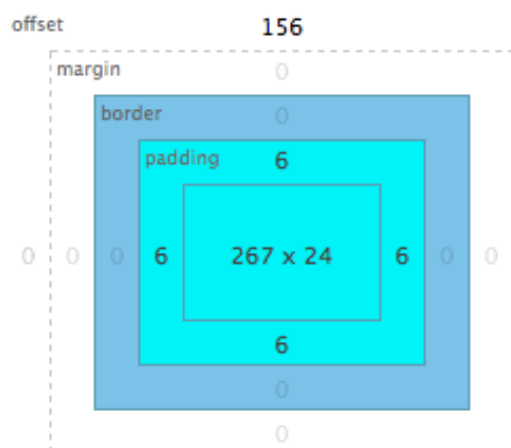
Ce dont nous n'avons pas beaucoup parlé, c'est de la boîte elle-même.

Comment la taille de la boîte est-elle calculée exactement ?

Voici deux schémas :



Si vous êtes un utilisateur de Firefox, vous êtes peut-être habitué à ce type de diagramme, qui permet de visualiser les chiffres affectant n'importe quelle case de la page :



Firebug's Box Model Display
(not to scale, but very useful)

Remarquez que dans les deux exemples, la marge est dans le blanc. La marge est unique en ce sens qu'elle n'affecte pas la taille de la boîte elle-même, mais qu'elle affecte les autres contenus qui interagissent avec la boîte, et qu'elle est donc un **élément important** du modèle de boîte CSS.

La taille de la boîte elle-même est calculée comme suit :

Width width + padding-left + padding-right + border-left + border-right

Height height + padding-top + padding-bottom + border-top + border-bottom

Largeur par défaut des boîtes

Si vous ne déclarez pas de largeur et que la boîte a un positionnement statique ou relatif, la largeur restera de 100 % et le padding et le border pousseront vers l'intérieur au lieu de l'extérieur. Mais si vous définissez explicitement la largeur de la boîte à 100 %, le padding poussera la boîte vers l'extérieur comme à l'accoutumée.



La leçon à en tirer est que la largeur par défaut d'une boîte n'est pas vraiment de 100 %, mais plutôt de « *ce qui reste* », ce qui est moins tangible.

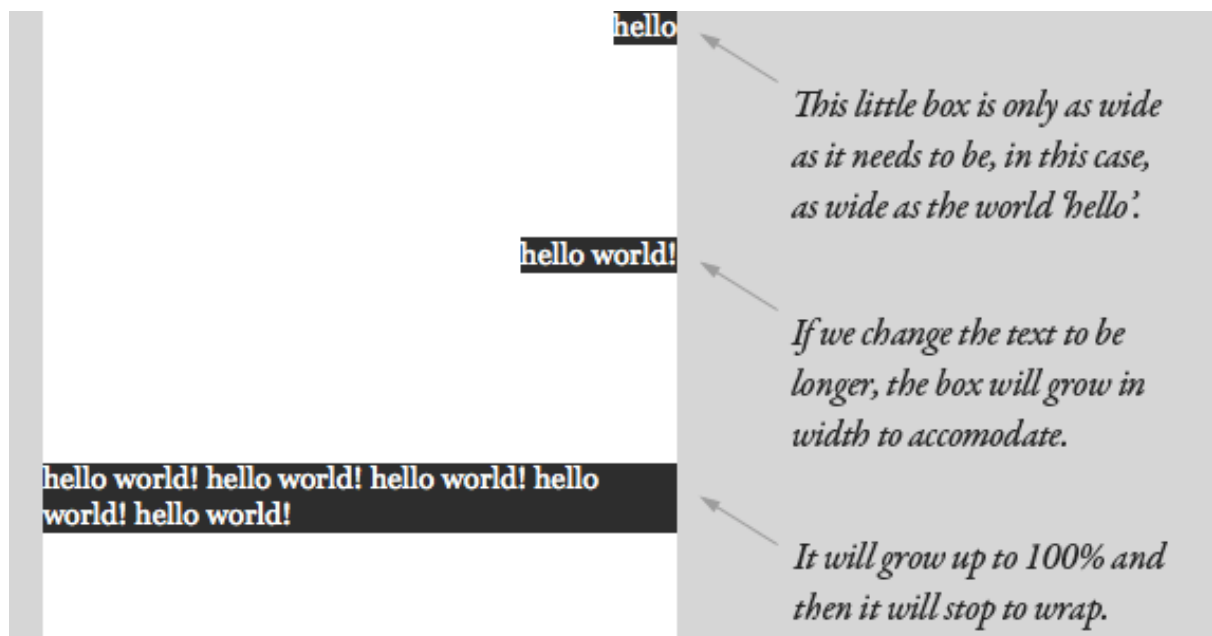
Il est particulièrement utile de le savoir, car il existe de nombreuses circonstances dans lesquelles il est extrêmement utile de définir ou non une largeur.

Le plus gros problème que je rencontre souvent est celui des éléments `textarea`, qui ont besoin d'une largeur définie pour respecter l'attribut « `cols` » requis, et qui ne peuvent pas avoir d'éléments enfants. Il faut donc souvent que la zone de texte soit explicitement fixée à 100 %, mais qu'elle soit également dotée d'un padding, ce qui la rend trop grande pour tenir dans la page. Dans un environnement à largeur

statique, nous avons souvent recours à des largeurs en pixels qui conviennent, mais ce n'est pas le cas dans les environnements à largeur fluide (%age).

Boîtes absolues sans largeur

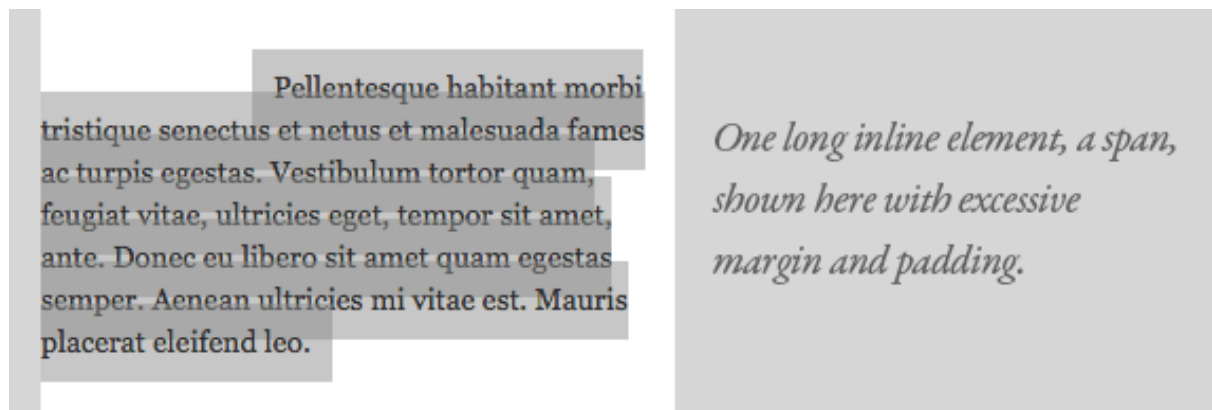
Les boîtes positionnées de manière **absolute** qui n'ont pas de largeur définie se comportent de manière un peu étrange. Leur largeur ne dépasse pas celle nécessaire pour contenir le contenu. Ainsi, si la boîte ne contient qu'un seul mot, sa largeur ne dépasse pas celle du mot en question. S'il y a deux mots, la boîte sera large de deux mots, etc ...



Les éléments en ligne sont aussi des boîtes

Nous nous sommes concentrés sur les boîtes en tant qu'éléments de **niveau bloc**. Il est facile de considérer les éléments de niveau bloc comme des boîtes, mais les **éléments en ligne sont également des boîtes**.

Il s'agit de rectangles très longs et très minces qui s'enroulent sur chaque ligne. Ils peuvent avoir des margin, un padding et des border, comme n'importe quelle autre boîte.



C'est l'habillage qui est source de confusion.

Une marge gauche comme celle montrée ci-dessus pousse la boîte vers la droite, comme on peut s'en douter, mais seulement la première ligne, car c'est le début de la boîte. Le remplissage est appliqué au-dessus et au-dessous du texte comme il se doit, et lorsqu'il s'enroule, il ignore la ligne au-dessus de son remplissage et commence à l'endroit où la hauteur de ligne indique qu'il doit commencer. L'arrière-plan a été appliqué en transparence pour que son fonctionnement soit plus clair.

— TIPS : Vous voulez voir toutes les "boîtes" qui composent une page ? Essayez de mettre ceci dans la feuille de style temporairement :

```
* {  
  outline: 1px solid red !important;  
}
```

10. MARGIN

La propriété `margin` définit la partie la plus extérieure du modèle de boîte, créant un espace autour d'un élément, en dehors de toute bordure définie.

Les marges sont définies à l'aide de longueurs, de pourcentages ou du mot-clé `auto` et peuvent avoir des valeurs négatives.

Voici un exemple :

```
.box {  
  margin: 0 3em 0 3em;  
}
```

`margin` est une propriété abrégée qui accepte jusqu'à quatre valeurs, comme indiqué ici :

```
.box {  
  margin: <margin-top> || <margin-right> || <margin-bottom> ||  
  <margin-left>  
}
```

Chacune des marges individuelles peut être déclarée à la main, auquel cas vous ne définirez qu'une seule valeur par propriété :

```
.box {  
  margin-top: 20px;  
  margin-right: 10px;  
  margin-bottom: 20px;  
  margin-left: 10px;  
}
```

auto et centrage

Chacune des propriétés de marge peut également accepter la valeur `auto`.

La valeur `auto` indique au navigateur de définir la marge pour vous.

Dans la plupart des cas, une valeur `auto` sera équivalente à une valeur 0 (qui est la valeur initiale de chaque propriété `margin`) ou à l'espace disponible de ce côté de l'élément.

Cependant, `auto` est pratique pour le **centrage horizontal** :

```
.container {  
  width: 980px;  
  margin: 0 auto;  
}
```

Dans cet exemple, deux choses sont faites pour centrer cet élément horizontalement dans l'espace disponible :

La `width` de l'élément est spécifiée

Les `margin` gauche et droite sont réglées sur `auto`

Sans la largeur spécifiée, les valeurs `auto` n'auraient pratiquement aucun effet, fixant les marges gauche et droite à 0 ou à l'espace disponible à l'intérieur de l'élément parent.

Il convient également de souligner que la valeur `auto` **n'est utile que pour le centrage horizontal**, de sorte que l'utilisation de la valeur `auto` pour les marges supérieure et inférieure ne permet pas de centrer un élément verticalement, ce qui peut être source de confusion pour les débutants.

Marges négatives

Comme on peut s'en douter, alors qu'une marge positive éloigne les autres éléments, une marge négative tire l'élément lui-même dans cette direction ou attire d'autres éléments vers lui.

Voici un exemple d'un conteneur avec un padding, et l'en-tête h2 a des marges négatives qui le ramènent vers les bords à travers le padding :

Exemple : <https://codepen.io/chriscoyier/pen/GRJvEVV>

11. PADDING

La propriété padding en CSS définit la partie la plus intérieure du modèle de boîte, créant un espace autour du contenu d'un élément, à l'intérieur de toutes les marges et/ou bordures définies.

Les valeurs de padding sont définies à l'aide de longueurs ou de pourcentages et ne peuvent **pas** accepter de valeurs **négatives**. La valeur initiale, ou valeur par défaut, de toutes les propriétés de remplissage est 0.

Syntax

```
.box {  
  padding: 0 1.5em 0 1.5em;  
}
```

```
.box {  
  padding: <padding-top> || <padding-right> || <padding-bottom> ||  
  <padding-left>  
}
```

```
.box {  
  padding-top: 20px;  
  padding-right: 20px;  
  padding-bottom: 20px;  
  padding-left: 20px;  
}
```

Calculs de la largeur des éléments et du padding

Si un élément a une largeur spécifiée, tout padding ajouté à cet élément s'ajoutera à la largeur totale de l'élément.

Il s'agit souvent d'un résultat indésirable, car la largeur de l'élément doit être recalculée à chaque fois que le padding est ajusté. (Notez que cela se produit également avec la hauteur, mais dans la plupart des cas, il est préférable de ne pas donner de hauteur fixe à un élément).

Par exemple :

```
.box {  
  padding: 20px;  
  width: 400px;  
}
```

La largeur de .box est donc de 440px

12. PSEUDO-CLASS

Voir cours pseudo-class

13. POSITIONS

La propriété position peut vous aider à manipuler l'emplacement d'un élément, par exemple :

```
.element {  
  position: relative;  
  top: 20px;  
}
```

Par rapport à sa position d'origine, l'élément du dessus sera maintenant décalé de 20px vers le bas.

Si nous animons ces propriétés, nous pouvons voir à quel point nous avons le contrôle (bien que ce ne soit pas une bonne idée pour des raisons de performance) :

<https://codepen.io/team/css-tricks/pen/RPBqoQ/fcdf9b19b6bed8da6af791d7433116b0>

Valeurs

- **static**: every element has a static position by default, so the element will stick to the normal page flow. So if there is a [left/right/top/bottom/z-index](#) set then there will be no effect on that element.
- **relative**: an element's original position remains in the flow of the document, just like the **static** value. But now [left/right/top/bottom/z-index](#) will work. The positional properties "nudge" the element from the original position in that direction.
- **absolute**: the element is removed from the flow of the document and other elements will behave as if it's not even there whilst all the other positional properties will work on it.
- **fixed**: the element is removed from the flow of the document like absolutely positioned elements. In fact they behave almost the same, only fixed positioned elements are always relative to the document, not any particular parent, and are unaffected by scrolling.
- **sticky**: the element is treated like a **relative** value until the scroll location of the viewport reaches a specified threshold, at which point the element takes a **fixed** position where it is told to stick.
- **inherit**: the **position** value doesn't cascade, so this can be used to specifically force it to, and **inherit** the positioning value from its parent.

Absolute

Si un élément enfant a une valeur **absolute**, l'élément parent se comportera comme si l'élément enfant n'existait pas :

Exemple : <https://codepen.io/team/css-tricks/pen/aOjeNE/7291a601af02608d928b1232d6456ec9>

Et lorsque nous essayons de définir d'autres valeurs telles que `left`, `bottom` et `right`, nous constatons que l'élément enfant ne répond pas aux dimensions de son parent, mais à celles du document :

```
.element {  
  position: absolute;  
  left: 0;  
  right: 0;  
  bottom: 0;  
}
```

Exemple : <https://codepen.io/team/css-tricks/pen/XbBvpe/e7d3b934a8ce213384dc119106b71b7d>

Pour que l'élément enfant soit positionné de manière absolue par rapport à son élément parent, nous devons définir ce paramètre sur l'élément parent lui-même :

```
.parent {  
  position: relative;  
}
```

Fixed

La valeur fixe est similaire à la valeur absolue, car elle permet de positionner un élément n'importe où par rapport au document, mais cette valeur **n'est pas affectée par le défilement**.

Sticky

La valeur collante est un compromis entre la valeur relative et la valeur fixe. À l'heure où nous écrivons ces lignes, il s'agit d'une valeur expérimentale, ce qui signifie qu'elle ne fait pas partie de la spécification officielle et qu'elle n'est que partiellement adoptée par certains navigateurs. En d'autres termes, ce n'est probablement **pas la meilleure idée** de l'utiliser sur un site web **en production**.

Qu'est-ce que cela fait ? Elle vous permet de positionner un élément par rapport à n'importe quel élément du document et, une fois que l'utilisateur a défilé au-delà d'un certain point de la fenêtre, de fixer la position de l'élément à cet emplacement afin qu'il reste affiché de manière persistante comme un élément à valeur fixe.

L'élément sera relativement bien positionné jusqu'à ce que l'emplacement de défilement de la fenêtre atteigne un point où l'élément sera à 50 px du haut de la fenêtre. À ce moment-là, l'élément devient collant et reste à une position fixe de 50 px en haut de l'écran.

La démo suivante illustre ce point, où la navigation supérieure a un positionnement relatif par défaut et où la deuxième navigation est collée tout en haut de la fenêtre de visualisation.

Veillez noter que la démo ne fonctionne que dans Chrome, Safari et Opera au moment de la rédaction de ce document.

Exemple : <https://codepen.io/geoffgraham/pen/ybVzeX>

14. Z-INDEX

```
div {  
  z-index: 1 ;  
  position : absolute;  
}
```

La propriété z-index en CSS contrôle l'ordre d'empilement des enfants *absolute* par rapport au premier parent *relatif*.

Les éléments (enfants) peuvent se chevaucher pour diverses raisons, par exemple, le positionnement relatif a fait passer l'élément au-dessus d'un autre.
Une marge négative a fait passer l'élément au-dessus d'un autre.

Des éléments positionnés de manière absolue se chevauchent.
Toutes sortes de raisons.



15. FLEXBOX / GRID

⚠ Voir cours Flexbox and Grid

16. EXERCICES

ⁱ EM et REM (emphemeral unit / root empheral unit) sont toutes deux des unités de mesure flexibles et scalables traduites par le navigateur en pixel, qui dépendent de la taille de police utilisée par votre système (en pixel, couramment 16px).

La différence entre les unités de mesure em et rem réside uniquement dans la manière dont le navigateur va traduire leur valeur en pixels. (source : <https://medium.com/codeshake/unit%C3%A9s-de-mesures-em-vs-rem>)