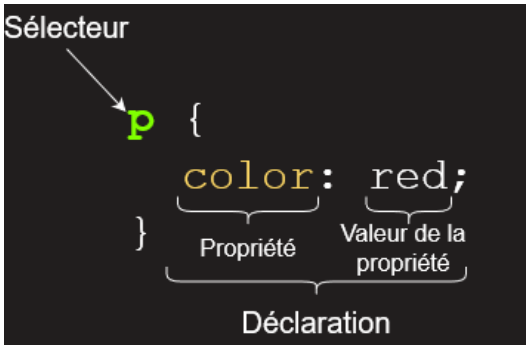
Fiche Révision CSS

Table des matières

Le CSS sert à modifier l'apparence d'une page en appliquant des styles au contenu HTML.



I. Organisation générale du code CSS

Le code CSS s'écrit préférentiellement dans un fichier CSS à part, à enregistrer dans le même dossier que la page html. Il faut rajouter dans le <head></head> de la page html :

```
<link rel="stylesheet" type="text/css" href="style.css">
```

1. Sélecteur

Il détermine l'élément ou le type d'élément HTML à qui appliquer le style.
Ex : "p" sélecteur pour les paragraphes, "h1" pour un titre. On peut aussi appliquer le style à un attribut class (".class") ou à un id ("#id").
Rappel : une *class* peut être utilisée plusieurs fois alors qu'un *id* ne peut être utilisé qu'une seule fois.

2. Propriété

Il sert à choisir le critère particulier auquel on va modifier le style.
Ex *: font-style* (italique), *color*(couleur du texte), *background-color*(couleur de fond)

3. Valeur

Complète la propriété et détermine son comportement. Ex : nom de la couleur.

Ce qui donne :
/*Un commentaire en CSS*/

Elément	class	id
`p{ color : blue; font-size : 16px; }`	`.nomClass{ color : blue; font-size : 16px; }`	`#nomID{ color : blue; font-size : 16px; }`

4. Les sélecteurs avancés

* : sélecteur universel : il sélectionne tous les éléments

<table>
<tr><th>HTML</th><th>CSS</th></tr>
<tr><td>Un élément peut avoir deux classes :
<pre><h1 class='title uppercase'>Top Vacation Spots</h1></pre></td><td><pre>.title {color: teal;}
.uppercase{text-transform: uppercase;}</pre></td></tr>
<tr><td>Pour sélectionner deux éléments :
A, B sélectionne les deux éléments</td><td><pre>Ul, li {
    list-style: none;
}</pre></td></tr>
<tr><td>Pour sélectionner un contenu dans un autre :
A B sélectionne un elt B contenu dans un élément A
<pre><h3>Titre avec texte important</h3></pre>Variante A > B : ne prend que les B enfants directs de A.</td><td><pre>h3 em{
    color : blue;
}</pre></td></tr>
<tr><td>Pour sélectionner un élément qui suit un autre :
A + B sélectionne le 1er elt B suivant un élément A
<pre><h3>Titre</h3>
Paragraphe </pre>A ~ B sélectionne tous les B qui suivent A.</td><td><pre>h3 + p{
    color : blue;
}</pre></td></tr>
<tr><td>Elément avec un attribut particulier :
A[attribut] sélectionne les elts A ayant cet attribut
<pre><a href="http://site.com" title="Infobulle"></pre></td><td><pre>a[title]{
    color : blue;
}</pre></td></tr>
<tr><td>Elément avec un attribut qui comprend une valeur :
A[attribut*= "valeur"] sélectionne les éléments A ayant "valeur" inclus dans le nom de l'attribut.
<pre><a href="http://site.com" title="Quelque part par ici"></pre></td><td><pre>a[title*="ici"]{
    color : blue;
}
span[class~="sr-only"]</pre></td></tr>
<tr><td>Elément avec attribut et valeur exacte :
A[attribut= "valeur"] sélectionne les éléments A possédant cet attribut avec une valeur précise.
<pre><a href="http://site.com" title="Cliquez ici"></pre></td><td><pre>a[title="Cliquez ici"]{
    color : blue;
}</pre></td></tr>
<tr><td>Elément avec attribut qui commence par :
<pre>a[href^="http"]</pre></td><td>Elément avec attribute qui finit par :
<pre>a[href$=".jpg"]</pre></td></tr>
<tr><td colspan="2">Multiples règles :
<pre>h1[rel="handsome"][title^="Important"] { color: red; }</pre></td></tr>
<tr><td>Un sélecteur qui comprend une classe :
selecteur.classe ne sélectionne que les sélecteurs ayant cette classe.
<pre><h2 class='destination'></pre></td><td><pre>h2.destination{
  font-family: Tahoma;
}</pre>Si un élément a la même classe il ne sera pas changé.</td></tr>
<tr><td>Sélectionner un élément à deux classes :
<pre> <h2>1</h2> </pre></td><td>Pour le cibler précisément :
<pre>.middle.side {
}</pre></td></tr>
<tr><td>Sélectionner un élément avec classe et id :
<pre> This is </pre></td><td><pre>.sub#preview {
    color: blue;
}</pre></td></tr>
</table>

https://www.w3schools.com/cssref/css_selectors.asp
https://www.freecodecamp.org/news/css-selectors-cheat-sheet/

5. Les pseudos classes

https://css-tricks.com/almanac/selectors/
https://www.w3schools.com/css/css_pseudo_classes.asp
https://developer.mozilla.org/fr/docs/Web/CSS/Pseudo-classes
Pour cela on rajouter la syntaxe *:pseudo-class* derrière le sélecteur. Ex : *:focus, :link, :visited, :disabled, :active, :hover, :checked, :not(selector), :valid, :invalid,*

:hover Permet d'affecter un style au survol de l'élément. Ici un lien.	`a:hover{` `  color: darkorange` `}`
:first-of-type :last-of-type :nth-of-type(nb), :only-of-type,	:first-child, :last-child, :empty(no children), :only-child, :nth-child(nb, even, odd)

6. Les pseudos selecteurs

::before, ::after, ::first-letter, ::marker, ::selection, ::first-line
https://developer.mozilla.org/fr/docs/Web/CSS/Pseudo-elements

7. Héritage

Tout élément HTML va hériter automatique du style de ses parents.
On peut annuler cet effet en attribuant directement un style propre à un élément enfant.
Le CSS appliquera le style le plus proche de l'élément en question.
Une bonne pratique consiste donc à commencer avec les sélecteurs les moins spécifiques (types d'éléments : balise) puis les classes et enfin les IDs.

8. !important

On peut rajouter *!important* à une règle propriété spécifique. Il permet d'outrepasser les autres règles. Il faut éviter de l'utiliser car il est compliqué de le changer ensuite.
Une justification de l'utilisation est si l'on travaille avec de multiples feuilles de styles. Ex : on utilise Bootstrap CSS et on souhaite écraser le style d'un élément HTML spécifique.

II. Gestion de la couleur

https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

1. Nom de la couleur

La couleur de l'élément (foreground) s'atteint avec la propriété *color* alors que la couleur de l'arrière-plan s'atteint avec la propriété *background-color*.
Les valeurs peuvent avec des noms communs comme : *red, blue, lightgreen, midnightblue* par exemple. Mais tous les choix ne sont donc pas possibles.

2. Code Hexadécimal

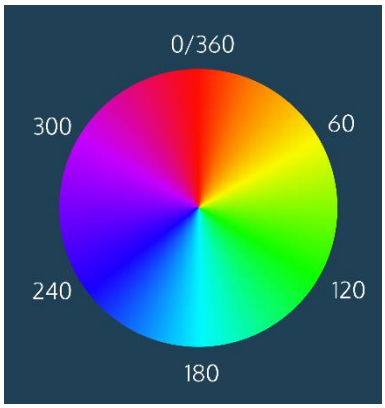
```
darkseagreen: #8FBC8F
sienna:       #A0522D
saddlebrown:  #8B4513
brown:        #A52A2A
black:        #000000 or #000
white:        #FFFFFF or #FFF
aqua:         #00FFFF or #0FF
```

Chacune des trois composantes RGB est coté par 2 caractères qui peuvent prendre 16 valeurs possibles. Comme il n'y a que 10 chiffres, on rajouter 6 lettres : [0-9]+[A-F].

3. Code RGB

On peut également utiliser le code rgb avec pour chacune des composantes une valeur entre 0 et 255.
Ex : color : rgb(127, 56, 236) ;

4. Teinte, Saturation et luminosité (Hue, Saturation, Lightness) :



color: hsl(120, 60%, 70%).

1ère valeur : *hue*. C'est l'angle sur la roue colorée. 0° : red ; 120° : green ; 240° : blue.

2ème valeur : *saturation*. C'est l'intensité de la couleur de 0% (gris) à 100%.

3e valeur : *luminosité*. De 0% (foncé-noir) à 100%(clair-blanc)). 50% est la luminosité normale.

HSL permet de mieux ajuster les couleurs. En hexa ou RGB pour éclaircir ou foncer une couleur, il faut jouer sur les trois paramètres. En HSL, un seul.

HSL permet également de créer un ensemble de couleur qui fonctionne bien ensemble en choisissant différentes couleurs qui ont les mêmes luminosité et saturation.

5. Opacité

Il faut utiliser les propriétés *rgba* ou *hsla* qui prennent une 4ème valeur dans la parenthèse : un chiffre décimal entre 0 (transparent) et 1 (opaque) :

a. rgba(12, 256, 14, 0.33)
b. hsla(34, 100%, 50%, 0.12)

Même si c'est peu conventionnel on peut rajouter deux caractères à l'écriture hexadécimale à la fin `#52BC8280`

6. Linear Gradient, repeating-linear-gradient, radial-gradient

Cela permet de faire des dégradés de couleurs fluide. La syntaxe de base est :

```
background-image: linear-gradient(direction, color-stop1, color-stop2, ...);
```

Par défaut la direction est : *to bottom.* Il peut donc y avoir *to top, to right, top left, to top left, to bottom right*. Elle peut prendre une valeur en degré égalment : *10deg, 60deg etc…*

On peut mettre autant de couleur que l'on veut ensuite comme argument.

Pour qu'une couleur prenne plus de place on peut ajouter un % juste après : *linear-gradient(45deg, red 70%, blue)*.

Pour faire des transitions de couleurs brutes et non graduelle on peut utiliser cette syntaxe :

```
body {
    background: linear-gradient(
        var(--first-color) 0%,
        var(--first-color) 40%,
        var(--second-color) 40%,
        var(--second-color) 80%
      );
}
```

On peut également répéter un certain motif :

```
body {
    background: repeating-linear-gradient(
        var(--first-color) 0%,
        var(--first-color) 5%,
        var(--second-color) 5%,
        var(--second-color) 10%
      );
}
```

Avec radial-gradient, la transition est définie par son centre :

```
.sky {
    background: radial-gradient(closest-corner circle at 15% 15%,
        #ccc,
        #ccc 20%,
        #445 21%,
        #223 100%
      );
  }
```

III. Formater du texte

1. Les propriétés de type font-

<table>
<tr><th>font-style</th><th>font-size</th></tr>
<tr><td>Elle accepte 4 valeurs différentes :
- Normal
- Italic
- Oblique
- Inherit

Italic est un état d'une police (non dispo sur toute).
Oblique est un été penché forcé.</td><td>Pour modifier la taille d'un texte. Elle accepte des valeurs en :
- Absolu (px ou pt)
- Relatif (em, ex, %)

Relatif : s'adapte aux préférences utilisateurs.
Absolu : fixe : contrôle précis du rendu.</td></tr>
<tr><th>Font-weight</th><th>font-family</th></tr>
<tr><td>Elle accepte 6 valeurs :
- normal (défaut)
- lighter
- bold
- bolder
- une centaine entre 1 et 1000 (du + léger au + gras)
- inherit</td><td>Permet de choisir la police du texte.
Il est de coutume de mettre plusieurs types de police en cas de problème avec la 1ère, le navigateur essaiera la 2nd etc…
Il est conseiller de mettre une moins une police "web safe font" dans les valeurs. Ex : Arial, Times New Roman, etc…
Si la police est composée de plusieurs mots il faut mettre des guillemets, sinon ce n'est pas nécessaire.
Il est conseillé de terminer par serif ou sans-serif.</td></tr>
<tr><th>line-height</th><th>color</th></tr>
<tr><td>Sert à fixer l'écartement entre 2 lignes du texte.
En général 1,5 fois la taille du texte en pixel</td><td>Permet de choisir la couleur. La valeur peut être :
- son nom (limité)
- code hexa : #AF33AF (violet)
- code rgb : rgb(25, 152, 65)</td></tr>
<tr><th>opacity</th><th></th></tr>
<tr><td>Propriété qui prend une valeur entre 0 (totalement transparent) et 1 (totalement opaque).
Si on utilise le code rgb pour la couleur, on peut utiliser rgba qui prend pour 4e valeur l'opacité :
- color : rgba(25, 142, 56, 0.5)
opacité à 50%</td><td>

```
p{
    font-style: normal;
    font-size: 1.8em;
    font-weight: bold;
    font-family: Impact, "Arial Black", Verdana, sans-serif;
    color: rgba(25, 152, 65, 0.8);
}
```

</td></tr>
</table>

https://www.cssfontstack.com/ https://fonts.google.com/ https://fonts.adobe.com/

Pour ajouter une police qui vient du web :

HTML :

```
<head>
  <!-- Add the link element for Google Fonts along with other metadata -->
  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap" rel="stylesheet">
</head>
```

CSS :

```
p {
  font-family: 'Roboto', sans-serif;
}
```

Inherit fonctionne dans un cas particulier :
Soit un font-style : italic à tous les <p>. Et un font-style normal à un . Si un <p> est dans le div, il sera en italic. Si on veut que ce <p> soit en normal, on utilise inherit pour forcer l'héritage du .

Pour rajouter une police qui n'est pas présente par défaut :
https://developer.mozilla.org/en-US/docs/Web/CSS/@font-face
https://www.fontsquirrel.com/tools/webfont-generator

```
@font-face { /* Définition d'une nouvelle police nommée CAC Champagne */
    font-family: 'cac_champagneregular';
    src: url('cac_champagne-webfont.eot');
    src: url('cac_champagne-webfont.eot?#iefix') format('embedded-opentype'),
         url('cac_champagne-webfont.woff') format('woff'),
         url('cac_champagne-webfont.ttf') format('truetype'),
         url('cac_champagne-webfont.svg#cac_champagneregular') format('svg');
}

h1 /* Utilisation de la police qu'on vient de définir sur les titres */
{
    font-family: 'cac_champagneregular', Arial, serif;
}
```

2. Les propriétés de type text-

<table>
<tr><th>text-align</th><th>text-decoration</th></tr>
<tr><td>Pour aligner un texte, prend 5 valeurs différentes :
- Left (par défaut)
- Center
- Right
- Justify
- Inherit

Centrage ou alignement vis-à-vis du plus proche parent.</td><td>6 valeurs sont possibles :
- Underline
- Overline
- Line-through
- Blink
- Inherit
- None (défaut)</td></tr>
<tr><th>text-indent</th><th>text-transform</th></tr>
<tr><td>Permet un décalage vers la droite (valeur > 0) ou vers la gauche (valeur < 0).
Elle accepte des valeurs en :
- Absolu (px ou pt)
- Relatif (em, ex, %)

Relatif : s'adapte aux préférences utilisateurs.
Absolu : fixe : contrôle précis du rendu.</td><td>Modifie l'aspect d'un texte. 5 valeurs sont possibles :
- uppercase
- lowercase
- capitalize (1ère lettre en majuscule)
- None
- Inherit</td></tr>
<tr><th>letter-spacing et word-spacing</th><th rowspan="2">text-shadow</th></tr>
<tr><td>Permet d'ajuster l'espace entre les lettres ou les mots
Elle accepte des valeurs en :
- Absolu (px ou pt)
- Relatif (em, ex, %)</td></tr>
<tr><th>text-overflow</th><td rowspan="2">Permet de rajouter un effet d'ombre à un texte. Elle a besoin de 4 valeurs :
- 1ère : déplacement horizontal de l'ombre
- 2ème : déplacement vertical de l'ombre
- 3ème : flou de l'ombre
- 4ème : couleur

p{
text-shadow: 2px 2px 4px black;
}
On peut en mettre plusieurs séparés par des virgules.</td></tr>
<tr><td>Si le texte est trop grand par rapport au cadre :
- Clip : tronque le texte
- Ellipsis : met "…" à la fin du dernier mot
- "-" rajoute "-" à la fin du dernier mot
- "string" rajoute "string" à la fin du dernier mot
- fade(lenght ou %) : tronque avec un dégradé</td></tr>
</table>

IV. Gestion du background

1. Couleur de fond

Pour mettre une couleur de fond sur toute la page :

```
body{
    background-color: lightgoldenrodyellow;
}
```

On peut également appliquer à un background à une balise spécifique :

```
h1{
    background-color: aquamarine;
}
```

2. Image de fond

Pour mettre une image de fond :

```
body{
    background-image: url("img/back.jpg");
    height: 150px;
    width: 100px;
}
```

On peut modifier la taille de l'image.
Cela ne redimensionne pas. Cela crop à partir du coin supérieur gauche.

La propriété *background-repeat* gère la répétition du fond. Elle accepte 4 valeurs :

- Repeat : se répète sur x et y (défaut)
- Repeat-x : se répète sur x (horizontal)
- Repaet-y : se répète sur l'y (vertical)
- No-repeat : ne se répète pas
- Space : répétition sans rognage : espace entre
- Round : répétition sans rognage : étirage

Pour gérer la position de l'image de fond, on utilise la propriété *background-position* qui prend deux valeurs : les coordonnées verticale et horizontale :

```
body{
    background-position: 100px 50px;
}
```

Image décaler de 100px vers la droite et 50px vers le bas par rapport au coin supérieur gauche de la balise.
On peut également utiliser les valeurs suivantes :

- Top, bottom, left, center, right
- Deux valeurs en %
- background-position: bottom 50px right 100px

On peut choisir de ficher le fond ou de le faire défiler avec la page avec *background-attachment*. Cette propriété accepte deux valeurs :

- Fixed
- Scroll

```
body{
    background-image: url("img/back.jpg");
    background-attachment: fixed;
    background-repeat: no-repeat;
    background-position: center;
}
```

La propriété *background-size* peut prendre les valeurs suivantes :

- Auto
- Cover
- Contain
- Px ou % : 1 valeur (x), 2 valeurs (x, y)

La propriété *background-clip* indique comment s'étend le background :

- border-box : jusque sous la border
- padding-box : jusqu'au padding
- content-box : que sous le contenu

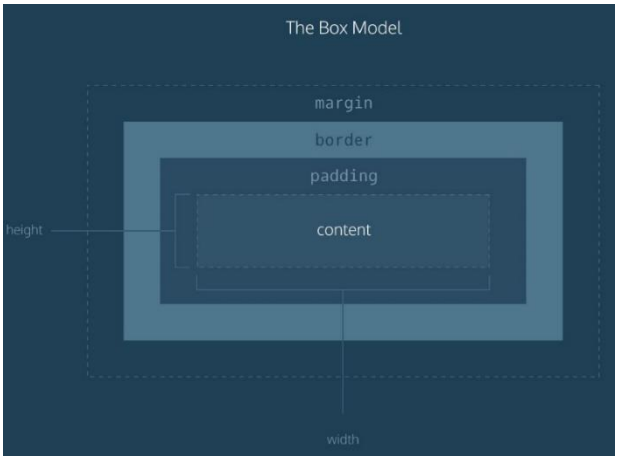
La propriété *background-origin* indique l'origine à partir de laquelle sera calculée la propriété *background-position*. Elle peut prendre comme valeur *border-box, padding box, content-box*.

On peut regrouper toutes les valeurs dans une seule propriété background *shorthand* dont l'ordre est :
Background-color/image/repeat/attachement/position

```
body{
    background:
        url("image1.jpg") fixed no-repeat 300px 0px,
        url("image2.jpg") fixed no-repeat top left,
        url("image3.jpg") fixed no-repeat top right;
}
```

Les couleurs de fond et images de fond peuvent s'appliquer à n'importe quelle balise.

V. Les modèles de boites



Tout élément d'une page web, de type block ou inline, est en réalité une boite rectangulaire qui peut avoir une bordure, des marges intérieures et extérieures. On va pouvoir utiliser les propriétés suivantes en CSS : *width, height, padding, border, margin, overflow…*

1. Hauteur et largeur d'un élément avec *width et height*

La hauteur d'un élément est déterminée par son contenu. Mais peut être modifiée par la propriété height à laquelle on attribue une valeur en px, % ou auto.

Par défaut un élément de type block occupe toute la largeur disponible. La largeur d'un élément de type inline dépend de son contenu. La largeur d'un élément peut être modifiée par la propriété width à laquelle on attribue une valeur en px, % ou auto.

```
p
{
    width: 50%;
    min-width: 400px;
}
```

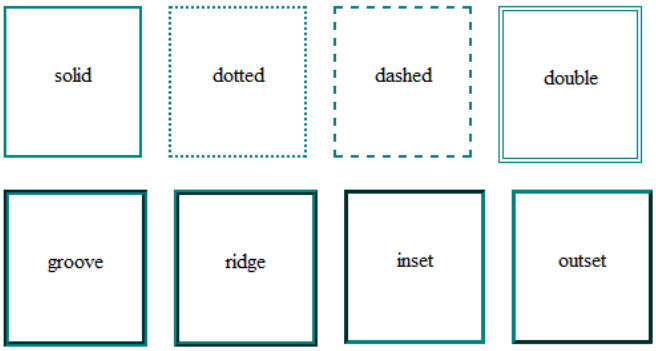
On peut donc utiliser également min-width, min-height, max-width, max-height.

La valeur auto est utilisé pour conserver les proportions tout en s'adaptant à l'écran de l'utilisateur.

2. Les bordures avec *border*

https://developer.mozilla.org/en-US/docs/Web/CSS/border-style#values

3 propriétés permettent de créer et personnaliser les bordures :
- Border-width : épaisseur de la bordure en px (1, 2 (top bottom) ou 4 valeurs(top clockwise)
- Border-style : style de la bordure(cf illus.) : 1, 2, 3 ou 4 valeurs possibles).
- Border-color : couleur. Accepte les mêmes valeurs que color.



Pour aller plus vite on peut utiliser une seule propriété border : [border-width] [border-style] [border-color]
- Border : 5px groove red

```
h1
{
    border: 3px blue dashed;
```

On peut appliquer les caractéristiques des bordures spécifiquement à un côté avec :
- Border-top-…
- Border-bottom-…
- Border-left-…
- Border-right-…

```
p
{
    border-left: 2px solid black;
    border-right: 4px double red;
}
```

On a la possibilité d'arrondir les coins avec :
- Border-radius : valeur de l'angle en px des 4 coins
- Border-top-left-radius, border-bottom-right-radius etc… pour viser un seul coin.

```
p{
    border-radius: 10px 5px 10px 5px;
}
```

3 valeurs :
- Top-left (topright+bottom-left) bottom-right

2 valeurs :
- (top-left+bottom-right) (top-right+bottom-left)

3. Les marges intérieures avec *padding*

Permet de définir la marge entre le contenu d'un élément et sa bordure (présente même si invisible) à l'aide de la propriété padding qui reçoit une valeur en px.

- padding : 10px; pour tous les côtés.
- padding-left, padding-top etc… uniquement pour un côté précis.
- On peut affecter 4 valeurs différentes au 4 côtés en une seule ligne : padding : 12px 10px 15px 20px. Dans l'ordre top, right, bottom, left
- Si les bordures sont les mêmes à droite est à gauche on ne met que 3 valeurs : padding : 12px 10px 15px. La 2e valeur sert pour les marges droite et gauche.
- Si on ne met que deux valeurs : padding: 12px 10px. 1ère valeur top et bottom. 2e valeur right et left.

```css
p
{
    padding: 12px 10px 15px 20px;
}
```

```css
p
{
    padding: 12px;
}
```

4. Les marges extérieures avec *margin*

Permet de définir l'espace à l'extérieur des bordures d'un élément à l'aide de la propriété margin qui reçoit une valeur en px ou en %.

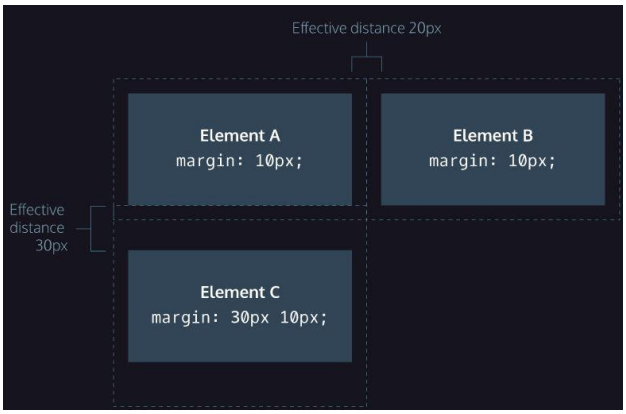
- margin : 10px; pour tous les côtés.
- margin-left, margin-bottom etc… uniquement pour un côté précis.
- Pour aller plus vite on peut utiliser une seule propriété margin : [margin-top] [margin-right] [margin-bottom] [margin-left]. Ex : margin : 5px 10px 5px 10px ;

```css
p
{
    width: 350px;
    padding: 12px;
    margin-bottom: 20px;
}
```

Pour être sûr que le rendu soit le même quel que soit le navigateur de l'utilisateur, on peut faire un reset CSS pour padding et margin. Il faut mettre des valeurs égales à 0 à l'élément *.

Pour center un élément horizontalement : margin: 0 auto. 1er valeur 0 pour top et bottom. 2e valeur auto pour droite et gauche.

Attention les marges horizontales s'additionnent. Mais au niveau des marges verticales c'est la plus grande des deux qui est effective.



5. Quand ça dépasse : overflow

Si le texte d'un paragraphe dépasse des dimensions de la boîte, on utilise la propriété overflow qui accepte comme valeur :

- Visible (défaut) : reste visible et dépasse du block… et peut donc empiéter sur le suivant.
- Hidden : coupé. Devient donc invisible.
- Scroll : texte coupé mais navigateur mettra en place une barre de défilement.
- Auto : navigateur décide. (à préférer).

```
p
{
    width: 250px;
    height: 110px;
    text-align: justify;
    border: 1px solid black;
    overflow: auto;
}
```

Si un mot est très long et dépasse du bloc en largeur (comme une url : pas d'espace, de tiret) on peut utiliser la propriété word-wrap.

```
p
{
    word-wrap: break-word;
}
```

A utiliser dès qu'un bloc peut contenir du texte saisi par des utilisateurs. Sinon cela peut "casser" le design d'un site si l'utilisateur écrit une longue suite de lettre come "aaaaaa……."

6. Visibility

On peut cacher des éléments avec la propriété *visibility* qui prend les valeurs :

- Hidden
- Visible
- collapse

```
p{
  visibility: hidden;
}
```

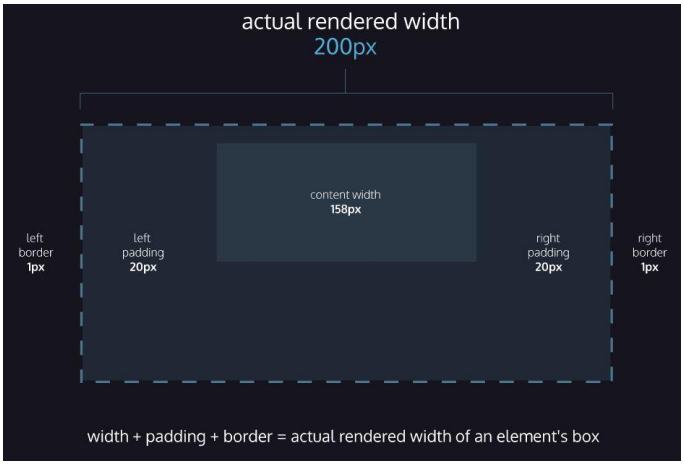
La différence entre *visibility: hidden* et *display: none* est que dans la première méthode, l'espace restera présent.

Box-Sizing

Avec les bordures et les marges, il peut donc être difficile de gérer la taille d'une boite et rendre donc le contenu d'une page web difficile à gérer. Pour cela on peut utiliser la propriété *box-sizing*. Par défaut elle a la valeur *content-box* :

- box-sizing: content-box

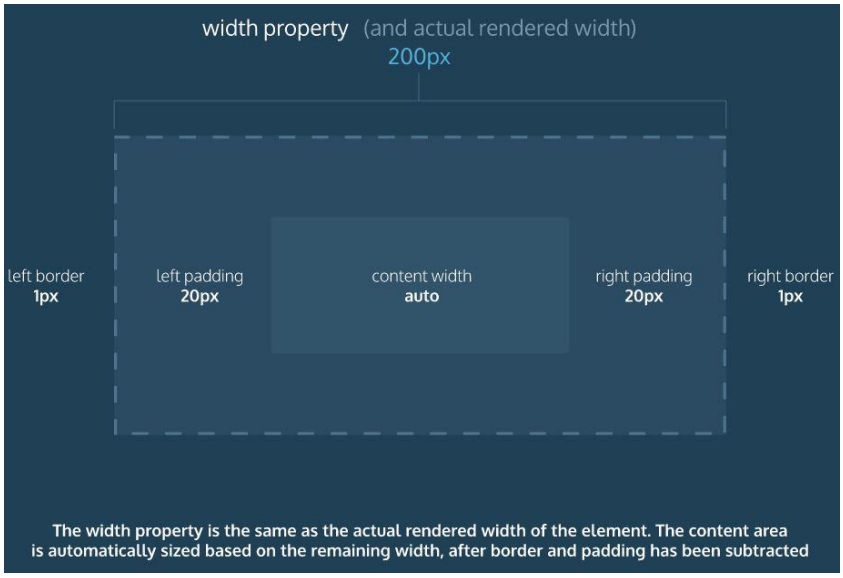
La taille de la boite sur l'écran est donc la taille définie par la propriété *width* à laquelle on doit ajouter les *padding* et *border* de chaque côté.



- box-sizing; border-box

Ce code reset le modèle de boîte pour tous les éléments HTML. Il évite les problèmes dimensionnels qui existent dans le modèle de boîtes vu précédemment. Dans ce modèle, la largeur et la hauteur resteront fixes. La bordure et le padding seront inclus à l'intérieur de la boîte ➔ Dimensions globales ne changent pas.

```
* {
  box-sizing: border-box;
}
h1 {
  border: 1px solid black;
  height: 200px;
  width: 300px;
  padding: 10px;
}
```



7. Les ombres des boites

On utilise la propriété box-shadow qui fonctionne comme text-shadow :
Permet de rajouter un effet d'ombre à une boite. Elle a besoin de 4 valeurs :

- 1ère : déplacement horizontal de l'ombre
- 2ème : déplacement vertical de l'ombre
- 3ème : rayon flou de l'ombre (optionnel)
- 4ème : propagation de l'ombre (optionnel)
- 5ème : couleur

```
p
{
    box-shadow: 6px 6px 6px black;
}
```

Ex : box-shadow : 2px 3px 2px red;
On peut rajouter une valeur inset à la fin si on veut que l'ombre soit à l'intérieur.

```
p
{
    box-shadow: 6px 6px 6px black inset;
}
```

A AJOUTER : PROPRIETE ASPECT RATIO

Propriété *object-fit* concerne les images. Valeurs possibles :

- fill, défaut : image redimensionnée pour s'adapter au conteneur (image étirée ou écrasée pour cela)
- contain, image conserve son ratio mais est redimensionnée pour s'adapter à la dimension
- cover, image conserve son ratio et sera couper pour s'adapter à la dimension
- none, image non redimensionnée
- scale-down image réduite à la plus petite version de *none* ou *contain*.

Permet de redimensionner un <img> ou <video> à son conteneur.

VI. Positionner des éléments (Anciennes méthodes)

1. Faire flotter un élément

Pour aligner les éléments les uns par rapport aux autres, on peut utiliser la propriété float qui peut prendre les valeurs suivantes : left, right, none.
Si on fait flotter un élément, les suivants vont se mettre à côté de celui-ci.

Général c'est plutôt pour des éléments de type inline.
Pour annuler cet effet à un élément suivant, on met la propriété clear avec les valeurs left, right, both possible.

```
<p><img src="flash.gif"
class="imageflottante" alt="Image
flottante" /> Ceci est un texte normal de
paragraphe, écrit à la suite de l'image et
qui l'habillera car l'image est
flottante.</p>
<p class="dessous">Ce texte est écrit sous
l'image flottante.</p>
```

```
.imageflottante
{
    float: left;
}
.dessous
{
    clear: both;
}
```

On peut ensuite "jouer" avec les marges pour affiner le positionnement.

```
nav
{
    float: left;
    width: 150px;
    border: 1px solid black;
}

section
{
    margin-left: 170px;
    border: 1px solid blue;
}
```

Déconseillé car si on a un site complexe, on va devoir souvent recourir à la propriété clear qui complexifient rapidement le code de la page.

2. La propriété display

La propriété *display* permet de changer le type d'un élément : *block* à *inline* ou *d'inline* à *block* par exemple. Cette propriété peut prendre 4 valeurs différentes : *inline*, *block*, *inline-block*, *none*.
Rappel :

Type	Exemples	Comportement
Inline	<a>, , , …	Eléments d'une ligne, se placent les uns à côté des autres.
Block	<p>, , <section>, …	Eléments en forme de block, se placent les uns en dessous des autres et peuvent être redimensionnés.
Inline-block	<select>, <input>	Eléments positionnés les uns à côté des autres (comme inline) mais peuvent être redimensionnés (comme block)
None	<head>	Eléments non affichés

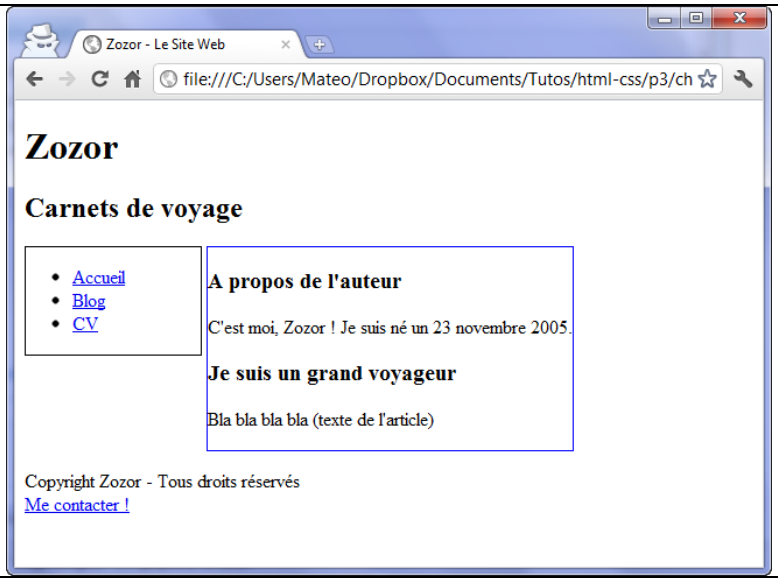
```css
nav
{
    display: inline-block;
    width: 150px;
    border: 1px solid black;
}

section
{
    display: inline-block;
    border: 1px solid blue;
}
```

Par défaut les éléments de type inline-block se positionnent sur une même ligne de base (baseline) en bas.

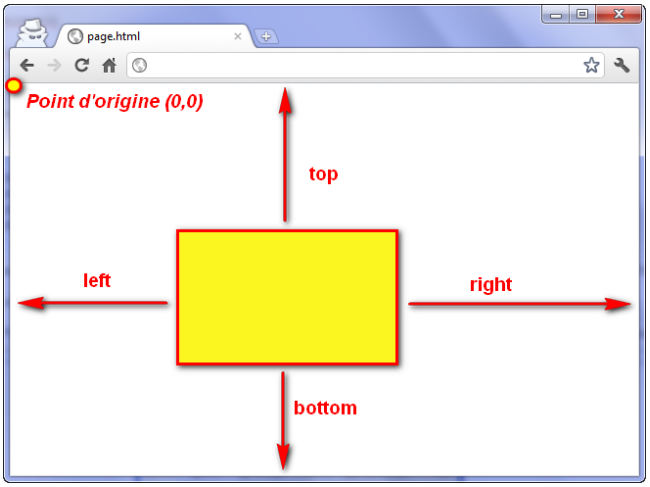
Donc on peut utiliser une nouvelle propriété : vertical-align. Cette propriété permet de modifier l'alignement vertical des éléments qui prend comme valeur :

- Baseline : aligne la base (défaut)
- Top : aligne en haut
- Middle : centre verticalement
- Bottom : aligne en bas
- Valeur (px ou %) : aligne à une certaine distance de la ligne de base.

```css
nav
{
    display: inline-block;
    width: 150px;
    border: 1px solid black;
    vertical-align: top;
}

section
{
    display: inline-
block;
    border: 1px solid blue;
    vertical-align: top;
}
```



3. Positionnement absolu, fixe et relatif



Pour cela on utilise la propriété *position* qui peut prendre comme valeur :

- *static* : par défaut
- *absolute* : positionnement absolu. On peut placer l'élément n'importe où sur la page.
- *fixed* : positionnement fixe malgré scroll. Sors du flux HTML.
- *relative* : positionnement relatif par rapport par rapport à sa position normale
- *sticky* : similaire à fixed mais sa place est gardée dans le flux HTML

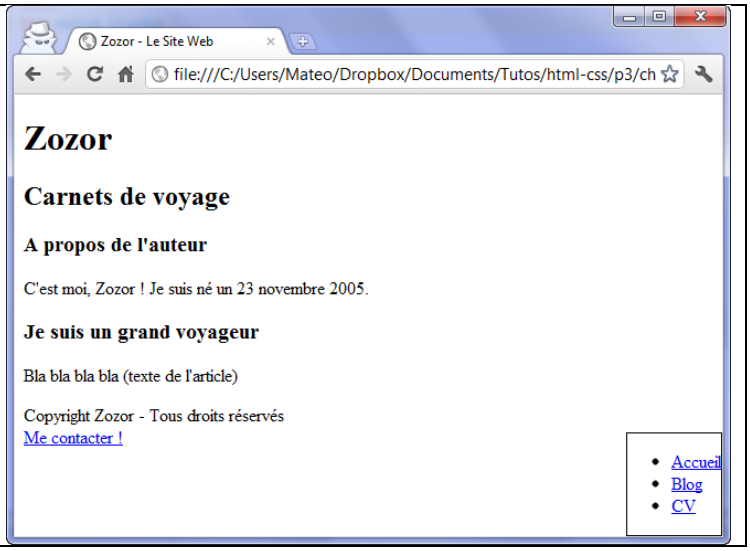
Le positionnement est complété par les propriétés left, right, top, bottom qui prennent des valeurs en px ou en %. Le décalage est compté par rapport au point d'origine en haut à gauche de la fenêtre.
Attention : si le block se situe à l'intérieur d'un autre, le point d'origine est donc le coin supérieur gauche du bloc parent.

Pour placer un élément en bas à droite :

```
element
{
    position: absolute;
    right: 0px;
    bottom: 0px;
}
```

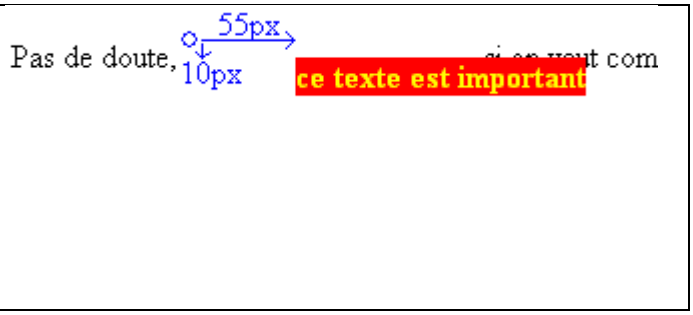
Si on veut le fixer :

```
element
{
    position: fixed;
    right: 0px;
    bottom: 0px;
}
```



Le positionnement relatif est plus délicat à utiliser. L'élément est simplement décalé par rapport à sa position initiale. Cette fois-ci le point d'origine ne se situe pas au haut à gauche de la fenêtre mais en haut à gauche de sa position initiale.

```
strong
{
   background-color: red;
   color: yellow;
   position: relative;
   left: 55px;
   top: 10px;
}
```



4. Le z-index

Si des éléments se chevauchent, le z index permet de savoir qui est au-dessus. Cela ne fonctionne pas avec des éléments en static (cf au-dessus).
Le z-index prend une valeur sans unité. Celui qui a la valeur la plus élevée sera au-dessus.

VII. FLEXBOX

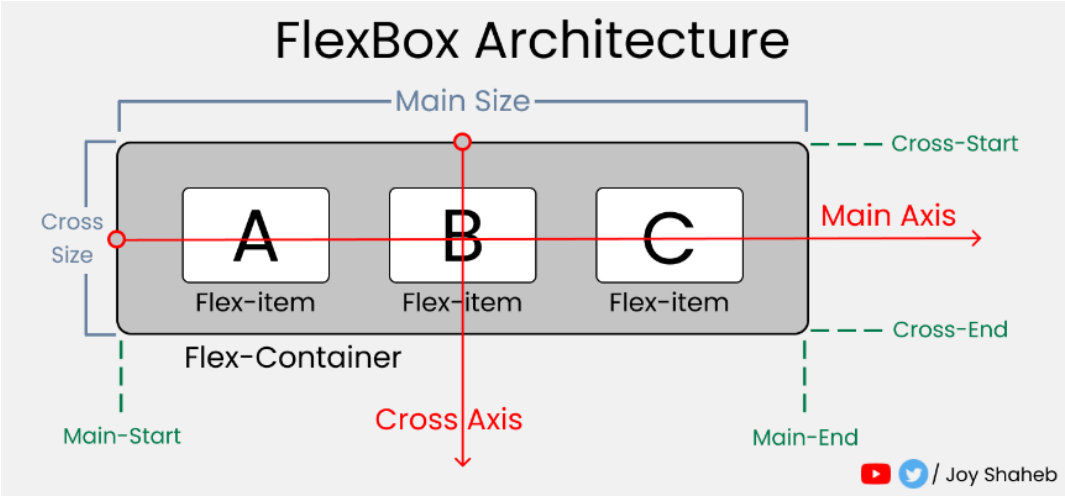
1. Présentation de Flexbox

https://developer.mozilla.org/fr/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox
Flexbox est une méthode permettant de disposer les éléments en lignes ou en colonnes en s'adaptant automatiquement à l'espace disponible.

Il y a les flex-containers (élément parent) qui est là pour contrôler le positionnement des flex-items (éléments fils). On peut contrôler les flex-items individuellement mais c'est surtout le rôle du flex-container.

a. display : flex ;

Pour utiliser les flexbox il faut donner la valeur *flex* à la propriété *display* au conteneur : d*isplay : flex ;*
Si on met cette propriété à un bloc (div par exemple), cela va permettre de gérer le positionnement des éléments enfants. Les éléments enfants se mettront les uns à côté des autres (par défaut) au lieu de se mettre sur une nouvelle ligne. L'élément parent (ici le div) prendra bien toute la largeur de la fenêtre par défaut.



b. *display: inline-flex*

Même chose que flex, mais transforme l'élément parent en inline : d'autres éléments pourront se mettre à côté de lui. Dans ce cas, si la taille du conteneur parents est définie, les éléments enfants vont se rétrécir pour rentrer dans l'élément parent.

2. flex-direction

On peut agencer les éléments dans le sens que l'on veut avec la propriété *flex-direction* qui peut prendre les valeurs suivantes :

- Row : organisés en une ligne (par défaut)
- Column : organisés en colonne
- Row-reverse : en ligne mais ordre inversé
- Column-reverse : organisé en colonne ordre inversé

```
<h1>Flex</h1>
    <div id="conteneur">
        <div class="element 1">Element </div>
        <div class="element 2">Element </div>
        <div class="element 3">Element </div>
    </div>
```

```
#conteneur{
    display:flex;
    flex-direction: row-reverse;
}
```

Row et *row-reverse* vont définir l'axe horizontal comme axe principal et l'axe vertical comme axe secondaire (cross-axis).
Column et *column-reverse* vont définir l'axe vertical comme axe principal et l'axe horizontal comme axe secondaire (cross-axis).

Les propriétés suivantes vont agir sur des axes déterminés :

- *justify-content, flex-wrap, flex-grow, flex-shrink, flex-basis* vont agir sur l'axe principal
- *align-items, align-content* vont agir sur l'axe secondaire

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Aligning_Items_in_a_Flex_Container

3. justify-content

Cette propriété permet d'aligner, d'organiser différemment les éléments à l'intérieur d'un flex-container sur l'axe principal. La propriété *justify-content* peut prendre comme valeur :

- flex-start : les flex-items sont sur la même ligne collés à gauche.
- flex-end : les flex-items sont sur la même ligne collés à droite.
- center : les flex-items sont sur la même ligne au centre.
- space-between : les flex-items sont répartis sur la même ligne. Les 1er et dernier sont collés aux extrémités.
- space-around : les flex-items sont répartis sur la même ligne avec un espace identique autour de chaque item.
- space-evenly : flex-items répartis sur la même ligne avec le même espace entre chaque et sur les côté.

```
#conteneur{
    display:flex;
    flex-wrap: wrap;
    justify-content: space-around;
}
```

NB : dans space-around l'espace entre les items et donc deux fois plus grand que l'espace entre le 1er item et la border (ou le dernier).

Sur l'axe horizontal, le positionnement dépend par défaut de la taille de la fenêtre :

```
#conteneur{
    display:flex;
    flex-wrap: wrap;
    justify-content: space-around;
}
```

Pour l'axe vertical, il faut obligatoire indiquer une hauteur au bloc :

```
#conteneur{
    display:flex;
    height: 500px;
    flex-direction: column;
    justify-content: center;
}
```

4. align-items

Cette propriété permet d'aligner, d'organiser différemment les éléments à l'intérieur d'un flex-container sur l'axe secondaire. La propriété *align-items* peut prendre comme valeur :

- stretch : éléments étirés sur tout l'axe (par défaut, sauf si *height* est définie)
- flex-start : alignés au début
- flex-end : alignés à la fin
- center : alignés au centre
- baseline : alignés sur la ligne de base (semblable flex-start)

Pour centrer sur une page on peut donc faire cela :

```
#conteneur{
    display:flex;
    height: 500px;
    flex-direction: column;
    justify-content: center;
    align-items: center;
}
```

Et plus simple encore :

```
#conteneur{
    display: flex;
}
.element{
    margin: auto;
}
```

Pour aligner un seul élément on utilise la propriété align-self directement dans l'élément visé :

```
#conteneur{
    border : 2px solid black;
    display:flex;
    height: 500px;
    justify-content: space-around;
}
.element:nth-child(2){
    background-color:green;
    align-self: flex-end;
}
```

5. flex-grow, flex-shrink, flex-basis

Attention, ces 3 propriétés se mettent sur les **flex-items**

a. flex-grow

Si le conteneur est (ou devient) plus grand que les objets qu'il contient, cette propriété qui permet de faire grossir les éléments et prendre tous l'espace disponible du bloc parent sur l'axe principal.
Les éléments grossissent dans la limite d'une taille maximale éventuellement définie sur l'axe principal.
Par défaut *flex-grow* vaut zéro. Les éléments respectent leur taille.

b. flex-shrink

Si le conteneur est (ou devient) plus petit que les objets qu'il contient, cette propriété permet de rétrécir les éléments et les adapter à l'espace disponible dans le bloc parent sur l'axe principal.
Les éléments rétrécissent dans la limite d'une taille minimale éventuellement définie sur l'axe principal.
Par défaut *flex-shrink* vaut 1: tous les éléments rétrécissent de la même manière, dans la même proportion.
Les marges ne sont pas affectées par flex-grow et flex-shrink
Les largeurs et hauteurs, minimales et maximales prévaudront sur flex-grow et flex-shrink

c. flex-basis

flex basis permet de spécifier la largeur d'un élément avant qu'il ne rétrécisse ou ne s'élargisse.

d. flex

C'est une propriété *shorthand* qui reprend flex-grow, flex-shrink, flex-basis. Elle prend donc trois valeurs :

```css
.big {
    flex-grow: 2;
    flex-shrink: 1;
    flex-basis: 150px;
  }

.small {
    flex-grow: 1;
    flex-shrink: 2;
    flex-basis: 100px;
}
```

Equivalent shorthand :

```css
.big {
    flex: 2 1 150px;
}

.small {
    flex: 1 2 100px;
}
```

La classe big va grossir deux fois plus que la classe small qui le conteneur est plus grand que 250px. Et la classe small va se réduire deux fois plus que la classe big si le conteneur est plus petit que 250px.

```css
.big {
    flex: 2 1;
}
```

Ici on ne déclare que *flex-grow* et *flex-shrink*

```css
.small {
    flex: 1;
}
```

Ici on ne déclare que *flex-grow*.

```css
.small {
    flex: 1 20px;
}
```

Ici on ne déclare que *flex-grow* et *flex-basis.*

Il n'y a donc pas moyen de ne déclarer que *flex-shrink* et *flex-basis.*

Faire grossir ou maigrir des éléments
On peut faire grossir des éléments pour occuper tout l'espace restant.

Pour que l'élément 2 prenne tout l'espace :

```css
.element:nth-child(2){
    flex: 1;
}
```

Pour que le 1er soit deux fois plus grand que le 2e :

```css
.element:nth-child(1){
    flex: 2;
}
.element:nth-child(2){
    flex: 1;
}
```

6. flex-wrap

Par défaut les flex-items essaient de rester sur la même ligne, et vont donc rétrécir (jusqu'à *min*-height ou min-width si définie) ce qui peut provoquer des bugs de design (trop petit, sort du container par exemple). Pour éviter cela, on peut forcer les flex-items à aller à la ligne on utilise la propriété *flex-wrap* qui peut prendre comme valeur :

- Nowrap : pas retour à la ligne (défaut)
- Wrap : à la ligne quand il n'y a plus de place
- Wrap-reverse : va à la ligne mais en sens inverse

```
#conteneur{
    display:flex;
    flex-wrap: wrap;
}
```

Cette propriété, lorsqu'elle agit, va donc créer une deuxième ligne.

7. align-content

Dans le cas où il y a plusieurs lignes dans la flexbox, on peut choisir comment elles seront réparties à l'aide de la propriété align-content. Cela n'aura aucun effet s'il n'y a qu'une seule ligne. Elle pourra prendre les valeurs :

- Flex-start
- Flex-end
- Center
- Space-between
- Space-around
- Stretch (par défaut)

8. flex-flow

shorthand pour déclarer *flex-direction* puis *flex-wrap* sur la même ligne.

```
#container {
  flex-flow: row wrap;
}
```

Exemple : alignement en bas à gauche :

```
#conteneur{
    border : 2px solid black;
    display:flex;
    flex-wrap:wrap-reverse;
    flex-direction:row;
    justify-content:flex-start;
    align-items:flex-end;
    height:400px;
    align-content:flex-start;

}
```

9. Rappel à l'odre

Sans changer le code HTML on peut modifier l'ordre des éléments grâce à la propriété order. Il faut simplement indiquer un nombre en valeur et les éléments seront triés du plus petit au plus grand.

```
#conteneur{
    display: flex;
}
.element:nth-child(1){
    order: 3;
}
.element:nth-child(2){
    order: 1;
}
.element:nth-child(3){
    order: 2;
}
```

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Typical_Use_Cases_of_Flexbox

VIII. GRIDS

1. Présentation

Flex Box est utilisé pour des parties de pages, mais pour agencer une page web entière, on utilise plutôt grid.
Pour cela on utilise la propriété *display* sur le conteneur, qui admet deux valeurs possibles :

- *Grid*
- *inline-grid*

`` .grid { border: 2px blue solid; width: 500px; height: 500px; display: grid; } ```	Par défaut la grille ne comporte qu'une seule colonne. Les éléments enfants viennent se rajouter en dessous en créant une nouvelle ligne. La hauteur des éléments enfants, si non précisée, s'adapte pour prendre toute la hauteur de l'élément parent.

2. grid-template-columns & grid-template-rows & grid-template

a. *grid-template-columns*

Pour rajouter des colonnes, on utilise la propriété *grid-template-columns* qui prend en argument des tailles de colonnes en px, en %...

``grid-template-columns: 100px 50% 100px;```	Si la somme des largeurs est plus petite que l'élément parent, il restera de l'espace. Si c'est trop grand, cela va dépasser.
``grid-template-columns: 1fr 60px 1fr;```	On peut mêler les *fr* et les autres unités comme *px*. Dans ce cas, les *fr* représentent la fraction de l'espace restant disponible. Ici si le parent fait 100px de large. 60px sont réservés par la 2e colonne. La 1ère et la 3e font 20px chacune.

Les propriétés qui gèrent le nombre de lignes et de colonnes peuvent avoir une fonction comme valeur. La fonction *repeat()* va répéter les spécifications des lignes et des colonnes un certain nombre de fois.

``grid-template-columns: repeat(3, 1fr);```	Remplace : ```grid-template-columns: 1fr 1fr 1fr;```

Le second paramètre peut avoir plusieurs valeurs si on veut respecter un certain pattern :

```
grid-template-columns: repeat(2, 20px 50px)
```

b. *Grid-template-rows*

Par défaut la taille des lignes est définie pour rentrer dans la grille. On peut les changer manuellement avec la propriété *grid-template-rows*.

``grid-template-rows: 40% 50% 50px;```	Même comportement que *grid-template-rows*
``grid-template-rows: repeat(3, min-content)```	Pour s'adapter au contenu.

c. *Grid-template*

``grid-template: 200px 300px / 20% 10% 70%;```	La shorhand *grid-template* remplace les deux propriétés *rows / columns*.
``grid-template: 2fr 1fr 1fr / 1fr 3fr 1fr;```	Pour éviter que les colonnes ou les lignes dépassent de la grille on utilise la terminaison *fr* derrière les valeurs de taille.

Si la grille s'adapte à la taille de l'écran/la fenêtre, on peut vouloir mettre une condition sur la taille d'une colonne (ex : image avec une taille minimum). Pour cela on utilise la fonction *minmax()* :

```
.grid {
    display: grid;
    grid-template-columns: 100px minmax(100px, 500px) 100px;
}
```

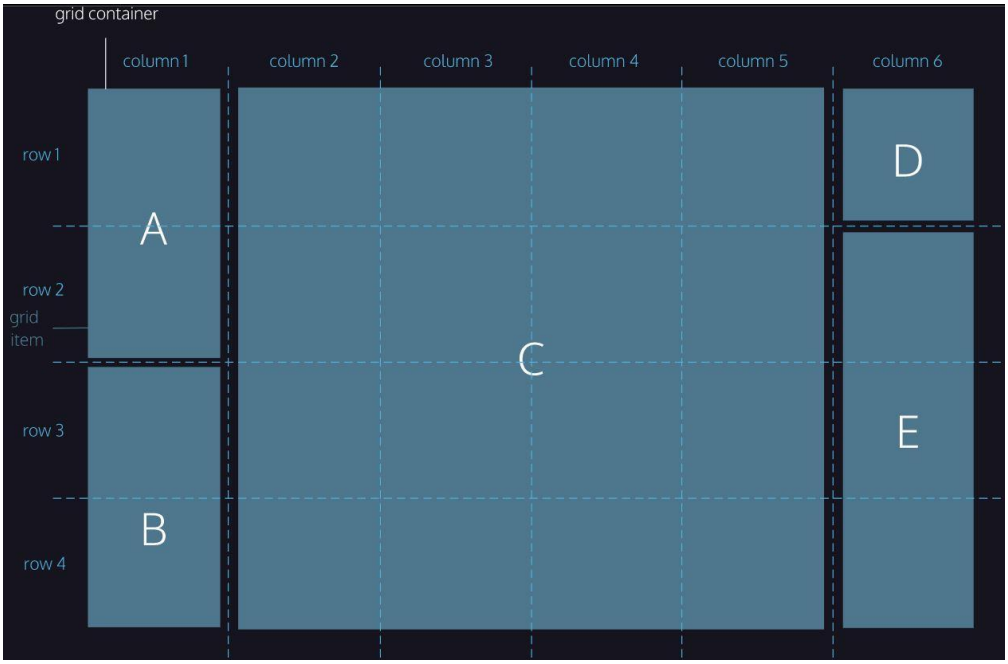
La fonction *minmax()* ne peut être utiliser qu'avec les grilles.
Pour dimensionner les lignes et les colonnes, on peut aussi utiliser les fonctions *min(*args), max(*args)* et *clamp(min-size, ideal-size, max-size)*.

3. Column-gap, row-gap, gap

Les propriétés *column-gap* et *row-gap* permettent de mettre des espaces entre les lignes et les colonnes. La propriété shorthand *gap* permet de regrouper les deux propriétés : valeurs séparées par un espace, *row* d'abord, *column* ensuite.
Cela n'ajoute pas d'espace au début ou à la fin mais bien entre les éléments. Cet espace est pris sur la taille allouée aux éléments.

``.grid { display: grid; width: 320px; grid-template-columns: repeat(3, 1fr); gap: 20px 10px; }```	Ici la largeur est de 320px. Il a 3 colonnes et un espace de 10px entre chaque colonne donc les colonnes se partagent 300px de largeur.

4. Etaler les éléments sur plus d'un bloc



Jusqu'à présent tous les éléments prenaient l'espace d'un bloc dans la grille. On va pouvoir maintenant étaler les objets de la grille sur plusieurs lignes ou plusieurs colonnes.

Pour cela il va falloir ajouter des propriétés aux éléments enfants de la grille

a. *grid-row-start & grid-row-end &grid-row*

Pour un élément qui est une plusieurs lignes on utilise les propriétés *grid-row-start* and *grid-row-end*. La numérotation pour *grid-row-start* est le numéro de la ligne où cela commence (numérotation à partir de 1) et le numéro de *grid-row-end* doit être un numéro plus haut que la ligne de fin.

`.item {` `  grid-row-start: 5;` `  grid-row-end: 7;` `}`	Ici l'objet a va prendre les 5e et 6e ligne de la grille.
`.item {` `  grid-row: 4 / 7;` `  }` `//Ou alors` `.item {` `  grid-row: 4 / span 2;` `  }`	Version shorthand de la propriété précédente. *span* précise l'étendu du nombre de ligne sur lesquelles l'objet doit s'étendre. Si on ne met pas de deuxième valeur, l'élément prendra une seule case.

b. *grid-column-start* & *grid-column-end* & *grid-column*

Même fonctionnement avec les propriétés *grid-column-start* et *grid-column-end*.

`.item {` `  grid-column-start: 4;` `  grid-column-end: span 2;` `}`	`.item {` `  grid-column: 4 / 6;` `}`	`.item {` `  grid-column: 4 / span 2;` `}`

c. *grid-area*

La propriété *grid-area* est la version shorthand de *grid-column* et *grid-row*. Elle prend donc 4 valeurs séparées par des "/". L'ordre est donc important :
grid-area: grid-row-start / grid-column-start /grid-row-end / grid-row-end / grid-column-end;

`#living-room {` `  grid-column-start: 1;` `  grid-column-end: 6;` `  grid-row-start: 1;` `  grid-row-end: 3;` `}`	`#living-room {` `  grid-area: 1 / 1 / 3 / 6;` `}` `.item {` `  grid-column: 1 / -1; /*Permet de prendre le reste de la ligne*/` `}`

5. grid-template-areas

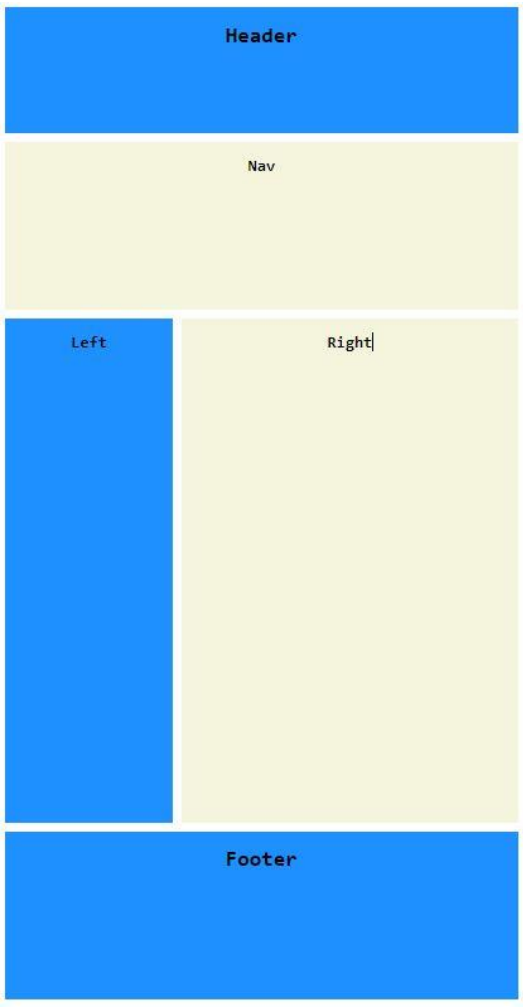
Propriété qui permet de donner un nom aux sections de la page pour les réutiliser dans les propriétés grid area des éléments enfants de la grille :

```
.container {
    display: grid;
    max-width: 900px;
    position: relative;
    margin: auto;
    gap: 10px;
    grid-template-areas: "header header"
"nav nav" "left right" "footer footer";
   }
```

La propriété va créer une mise en page avec 4 lignes et deux colonnes.

```
header {
    background-color: dodgerblue;
    grid-area: header;
}

nav {
    background-color: beige;
    grid-area: nav;
}

.left {
    background-color: dodgerblue;
    grid-area: left;
}

.right {
    background-color: beige;
    grid-area: right;
}

footer {
    background-color: dodgerblue;
    grid-area: footer;
}
```

Les valeurs données à *grid-area* permettent d'associer les éléments et classes aux noms indiqués dans *grid-template-area*.



On peut ensuite redimensionner les colonnes et les lignes :

```
grid-template-columns : 200px 400px;
  grid-template-
rows: 150px 200px 600px 200px;
```

On peut signifier une case vide avec un point "."

6. Positionnement dans la grille

a. justify-items

C'est une propriété qui positionne les éléments horizontalement (c-a-d dans leur ligne) dans leur colonne (fixe). Elle accepte 4 valeurs :

- start : aligne les éléments sur la gauche
- end : aligne les éléments sur la droite
- center : centre les éléments
- stretch : étend les éléments

```html
<main>
    <div class="card">Card 1</div>
    <div class="card">Card 2</div>
    <div class="card">Card 3</div>
</main>
```

```css
main {
    display: grid;
    grid-template-columns: repeat(3, 400px);
    justify-items: center;
}
```

Dans l'exemple au-dessus, la grille fait 3 colonnes de 400px chacune. La taille des éléments de la grille n'est pas précisée. Par défaut les éléments prennent la totalité des 400px. Avec la propriété *justify-items: center* se centrent dans leur colonne.
Voir propriété *place-items* pour déterminer les proprités align-items et justify-items en même temps.

b. justify-content

Cette propriété permet de positionner la grille dans son élément parent : elle positionne les colonnes horizontalement. Si la taille des colonnes est plus petite que la taille du conteneur, on va pouvoir déplacer les colonnes horizontalement. Cette propriété est donc déclarée sur le conteneur de grille. Elle accepte les valeurs suivantes :

- start : la grille est alignée à gauche
- end : alignée à droite
- center : placée au centre
- stretch : la grille est étirée pour prendre toute la place
- space-around : même espace à gauche et à droite de chaque élément donc deux fois plus d'espace entre les éléments qu'aux extrémités.
- space-between : même espace entre les éléments. Aucun aux extrémités
- space-evenly : même espace entre les éléments et aux extrémités.

c. align-items

C'est une propriété qui positionne les éléments dans leur colonne verticalement dans sa ligne (fixe). Elle accepte 4 valeurs :

- start : aligne les éléments sur la gauche
- end : aligne les éléments sur la droite
- center : centre les éléments
- stretch : étend les éléments

d. align-content

Cette propriété permet de positionner la grille dans son élément parent : elle positionne les lignes verticalement. Si la taille des lignes est plus petite que la taille du conteneur, on va pouvoir déplacer les lignes verticalement (dans leur colonne). Cette propriété est donc déclarée sur le conteneur de grille. Elle accepte les valeurs suivantes :

- start : la grille est alignée à gauche
- end : alignée à droite
- center : placée au centre
- stretch : la grille est étirée pour prendre toute la place
- space-around : même espace à gauche et à droite de chaque élément donc deux fois plus d'espace entre les éléments qu'aux extrémités.
- space-between : même espace entre les éléments. Aucun aux extrémités
- space-evenly : même espace entre les éléments et aux extrémités.

e. justify-self et align-self

Ces propriétés spécifie individuellement comment un élément doit se comporter. Elles se mettent donc dans les objets enfants de la grille. Elles écrasent les consignes de *justify-items* et *align-items*.
justify-self spécifie comment un élément doit se positionner horizontalement (*start, end, center, stretch)*
align-self spécifie comment un élément doit se positionner verticalement (*start, end, center, stretch)*

f. grid-auto-rows & grid-auto-columns

Lorsqu'il y a plus d'éléments que prévu dans le CSS. Exemple recherche où on ne sait pas combien d'éléments vont sortir, on utilise les grilles implicitent qui vont prendre le relais si la grille de départ est saturée.
Spécifie la hauteur des lignes ajoutées implicitement. Valeur en px, %, fr, fonction repeat().
Spécifie la largeur des colonnes ajoutée implicitement. Valeur en px, %, fr, fonction repeat().

```html
<body>
    <div>Part 1</div>
    <div>Part 2</div>
    <div>Part 3</div>
    <div>Part 4</div>
    <div>Part 5</div>
</body>
```

```css
body {
    display: grid;
    grid: repeat(2, 100px) / repeat(2, 150px);
    grid-auto-rows: 50px;
}
```

Dans cet exemple, il y a 5 et le tableau d'origine prévoit 2 lignes et 2 colonnes soit 4 cellules. La 5e cellule va donc être ajoutée dans une ligne implicite de 50px de hauteur. Si on ne précise pas, sa taille sera ajustée en fonction de la hauteur de son contenu.

g. grid-auto-flow

Spécifie si les nouveaux éléments doivent être ajoutés aux lignes ou aux colonnes. Déclaré sur les conteneurs.

- row : les éléments créent de nouvelles lignes (par défaut)
- column : les éléments créent de nouvelles colonnes
- dense : algorithme qui tente de combler les trous dans la disposition si des éléments plus petits sout ajoutés

IX. Apparence dynamique

1. Au survol : hover

```css
a /* Liens par défaut (non survolés) */
{
   text-decoration: none;
   color: red;
   font-style: italic;
}
```

```css
a:hover /* Apparence au survol des liens */
{
   text-decoration: underline;
   color: green;
}
```

Cela fonctionne sur les liens mais aussi sur n'importe quel élément.

2. Lors du clic : active

En pratique cela n'est utilisé que sur les liens :
Le lien gardera cette apparence très peu de temps : juste lorsque le bouton de la souris est enfoncé.

```css
a:active /* Quand le visiteur clique sur le lien */
{
    background-color: #FFCC66;
}
```

3. Lors du focus (élément sélectionné) : focus

Lorsqu'on a cliqué sur le lien il reste sélectionné. Sur chrome et safari, cela ne se voit que si l'on appuie sur la touche Tab.

```css
a:focus /* Quand le visiteur sélectionne le lien */
{
    background-color: #FFCC66;
}
```

4. Lorsque le lien a été consulté : visited

Pas défaut le navigateur colore le lien en violet. On peut changer cela :

```css
a:visited /* Quand le visiteur a déjà vu la page concernée */
{
    color: #AAA; /* Appliquer une couleur grise */
}
```

Autres :
https://www.w3schools.com/cssref/css3_pr_column-width.asp

X. CSS TRANSITION

```
a {
    background-color: orange;
    transition-property: background-color;
    transition-duration: 2s;
}
a:hover {
    background-color: red;
    transition-property: background-color;
    transition-duration: 2s;
}
```

Permet de créer un lien pour un lien dont le background apparaît progressivement.

Lorsqu'on passe la souris dessus, le background change de couleur progressivement également.

transition-property peut être *color, background-color, font-size, width, height*
transition-duration est spécifier en secondes ou millisecondes : *3s, 0.75s, 500ms*.

La propriété *transition-timing-function* décrit le rythme de la transition. Elle prend comme valeur :

- ease (défaut) : démarre doucement, accélère au milieu, ralenti à la fin
- ease-in : commence doucement, accélère et stop brusquement
- ease-out : commence brusquement, ralenti et fini doucement
- ease-in-out démarre doucement, rapide au milieu et fini doucement
- linear : vitesse constante.

La propriété *transition-delay* spécifie le temps avant de commencer la transition. Par défaut 0s.

```
a:hover {
    background-color: red;
    transition: background-color 1.5s
linear 0.5s;
}
```

La propriété shorthand *transition* permet de regrouper les 4 propriétés précédentes en une. Elles doivent être spécifiées dans l'ordre suivant : *transition-property, transition-duration, transition-timing-function, transition-delay*.

```
.item(s) {
    transition: width 750ms ease-in 200ms,
    left 500ms ease-out 450ms,
    font-size 950ms linear;
}
```

Ou encore :

```
.item {
    transition: all 1.2s ease-out 0.2s;
  }
```

Grâce à cette propriété shorthand *transition*, il est aussi possible de combiner plusieurs transitions différentes séparées par des virgules.

Pour affecter la même transition à toutes les propriétés.

XI. Animation

1. La propriété *transform*

Avec la propriété *transform*, il existe des fonctions qui redimensionnent, pivotent, déforment des éléments.

- scale(x, y) redimensionne un élément. Si un argument est donné, il agit sur les deux axes de la même manière.

Par exemple *scale(2)* va doubler la taille sur les deux axes. Sinon on donne deux arguments, le premier selon les x, le 2e selon les y. Si on ne veut toucher qu'à un axe on utilise *scaleX()* ou *scaleY()*. Pour les objets en 3D il y a également *scaleZ()*.

- rotate() produit la rotation d'un élément dans le plan. Elle prend un argument : l'angle en degré.

Il faut préciser "deg". Ex : *rotate(180deg)* pour faire une rotation de 180° dans le sens horaire. Pour les objets en 3D il y a les fonctions *rotateX(), rotateY(), rotateZ().*

- translate(x, y) déplace un élément de sa position initiale en fonction des arguments passés.

Si un argument est donné, la translation se fera selon l'axe des x. Si deux arguments sont donnés, le deuxième servira à la translation selon l'axe des y. Ex : *translate(0px, 100px)*. Il existe également les fonctions *translateX(), translateY(), translateZ()*.

- skew(x-angle, y-angle) : produit une transformation oblique des angles précisés.

Il existe aussi les fonctions *skewX(), skewY()*

- matrix(scaleX(), skewY(), skewX(), scaleY(), translateX(), translateY())

Fonction qui regroupe toutes les transformations 2D en une seule propriété.

2. La propriété transform-origin

Elle est utilisée pour définir le point autour duquel la transition CSS sera appliquée. Quand on a envie de faire une rotation avec la fonction *rotate* la fonction *transform-origin* déterminera autour de quel point l'élément tournera.

```
div {
    transform: rotate(45deg);
    transform-origin: 20% 40%;
}
```

La syntaxe de base est :

- transform-origin: *x-axis y-axis z-axis*|initial|inherit;

Par défaut, les valeurs sont 50% 50% 0 ; soit le centre de la figure.
Les valeurs possibles de x-axis sont *left, center, right, length ou %*
Les valeurs possibles de y-axis sont *top, center, bottom, length ou %*

3. Les animations

a. *animation-name*

Donne le nom à l'animation que l'on crée

b. *animation-duration*

Préciser la durée de l'animation. Unité à préciser.

c. *animation-iteration-count*

Indique le nombre de fois que l'animation doit se faire : *infinite* si l'on veut que cela ne s'arrêt pas.

d. Animation-direction

Permet de changer la direction de l'animation avec les paramètres suivants : *reverse, alternate, alternate-reverse*

e. Animation-delay

Défini le délai de départ de l'animation, unité à préciser.

f. Animation-play-state

Permet notamment de mettre l'animation en pause avec la valeur *paused* ou la démarrer avec la valeur *running*.

g. *animation-timing-function*

Définie le timing du déroulement de l'animation :

- linear : même vitesse du début à la fin
- ease : démarre doucement, accélère et ralenti à la fin
- ease-in : animation démarre doucement
- ease-out : animation fini doucement
- ease-in-out : animation démarre et fini doucement.

h. *animation*

Version *shorthand* des propriétés précédentes avec la syntaxe suivante :
animation: *name duration timing-function delay iteration-count direction fill-mode play-state*;

4. @keyframes Rule

Le @keyframe permet de spécifier le code de l'animation. Durant l'animation on peut changer plusieurs fois le CSS.

```css
div {
    width: 100px;
    height: 100px;
    background: red;
    position: relative;
    animation: mymove 5s infinite;
}
```

```css
@keyframes mymove {
    from {top: 0px;}
    to {top: 200px;}
}
```

Rotation d'une roue :

```css
.wheel {
    border: 2px solid black;
    border-radius: 50%;
    height: 55vw;
    width: 55vw;
    animation-name: wheel;
    animation-duration: 10s;
    animation-iteration-count: infinite;
    animation-timing-function: linear;
}
```

```css
@keyframes wheel {
    0% {
      transform: rotate(0deg);
    }
    100% {
      transform: rotate(360deg);
    }
}
```

Rotation d'un carré par rapport au milieu de la face supérieure :

```css
.cabin {
    background-color: red;
    width: 20%;
    height: 20%;
    position: absolute;
    border: 2px solid;
    transform-origin: 50% 0%;
    animation: cabins 10s ease-in-out
infinite;
}
```

```css
@keyframes cabins {
    0% {
      transform: rotate(0deg);
    }
    25% {
      background-color: yellow;
    }
    50% {
      background-color: purple;
    }
    100% {
      transform: rotate(-360deg);
    }
}
```

Carré qui se déplace dans la page :

```css
div {
    width: 100px;
    height: 100px;
    background: red;
    position: relative;
    animation: mymove 5s
infinite;
}
```

```css
@keyframes mymove {
    0%   {top: 0px; left: 0px; background: red;}
    25%  {top: 0px; left: 100px; background: blue;}
    50%  {top: 100px; left: 100px; background: yellow;}
    75%  {top: 100px; left: 0px; background: green;}
    100% {top: 0px; left: 0px; background: red;}
}
```

XII. Responsive Design avec les Media Queries

Ce sont des règles à appliquer pour changer le design d'un site en fonction des caractéristiques de l'écran. On va pouvoir créer un design qui va s'adapter automatiquement à l'écran de chaque visiteur.

Pour le responsive design il est plus intéressant d'utiliser des mesures relatives.
Par exemple pour les tailles de police :

- Le *em* : 1 em est la taille de police par défaut de l'élément dans lequel on se trouve. (16px pour la majorité des navigateurs).

Exemple. Un <p> a une taille de 2em. Un dans ce <p> a automatiquement une taille de 2em. Si on affecte une *font-size* de 2em au , cela fera 4em.

- Le *rem* : comme le *em* mais en référence la taille de l'élément racine soit <html>

Conclusion :
L'unité *rem* peut être pratique car la taille est toujours comparée à la même valeur. Cela peut être intéressant pour dimensionner les éléments de manière cohérente sur tout un site web. Si on veut dimensionner les éléments par rapport aux éléments à proximité *em* est mieux adapté.

On utilise également le % pour définir les tailles des boîtes comme *width, height, padding, border, margin*.
height et *width* en % sont calculées sur hauteur et largeur de l'élément parent.
Pour padding et *margin*, le % est calculer par rapport à *width* de l'élément parent.
Avec les %, il peut y avoir des rendus non souhaités avec des écrans trop grand ou trop petit. Pour palier à cela, il peut être intéressant de définir *min-width, max-width, min-height, max-height*

Mise en place des media queries :
Ce ne sont pas de nouvelles propriétés, mais des règles :

- Augmenter la taille du texte
- Changer la couleur de fond
- Positionner différemment le menu

Que l'on va appliquer dans certaines conditions comme :

- La largeur de l'écran du visiteur
- Type de l'écran : smartphone, télévision, projecteur…
- Nombre de couleur
- Orientation de l'écran

Pour les appliquer on peut procéder de deux manières :

- En chargeant une feuille de style différente en fonction de la règle. Ex : si la résolution est inférieure à 1280px de large, charge le fichier petite_resolution.css

Pour charge une feuille de style différente, on va utiliser l'attribut media (dans la balise <link />) dans lequel on va écrire la règle qui doit s'appliquer pour que le fichier soit chargé.
Le code HTML peut donc proposer plusieurs fichiers CSS : un par défaut (chargé dans tous les cas) et un ou deux autres qui seront chargés en supplément, uniquement si la règle s'applique :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <link rel="stylesheet" href="style.css" /> <!-- Pour tout le monde -->
```

```
        <link rel="stylesheet" media="screen and (max-width: 1280px)"
href="petite_resolution.css" /> <!-- Pour ceux qui ont une résolution inférieure à 1280px
-->
        <title>Media queries</title>
    </head>
```

`@media screen and (max-width: 1280px)` `{` `  /* Rédigez vos propriétés CSS ici */` `}`	En écrivant directement dans le fichier .css habituel
`@media only screen and (min-width: 320px)` `and (max-width: 480px) {` `  /* ruleset for 320px - 480px */` `}`	On peut également mettre plusieurs conditions sur la même ligne.
`@media only screen and (min-width: 320px) {` `  /* ruleset for >= 320px */` `}` `@media only screen and (min-width: 480px) {` `  /* ruleset for >= 480px */` `}`	Ou faire des règles différentes en fonction des conditions.

```
@media only screen and (min-resolution: 300dpi) {
    /* CSS for high resolution screens */
}
```

Exemple tablette en paysage :

```
@media only screen and (min-width: 768px) and (max-width: 1024px) and (orientation:
landscape) {
    /* CSS ruleset */
}
```

Les règles disponibles :

- color : gestion de la couleur en bits/pixel
- height : hauteur de la zone d'affichage
- width : largeur de la zone d'affichage
- device-height : hauteur du périphérique
- device-width : largeur du périphérique
- resolution
- orientation : portrait ou landscape
- media : type d'écran de sortie
    - screen : écran classique
    - handheld : mobile
    - print : impression
    - Tv : télévision
    - projection : projecteur
    - all : tout types.

https://developer.mozilla.org/fr/docs/Web/CSS/Media_Queries/Using_media_queries

On peut rajouter le préfixe min- ou max- devant la plupart de ces règles.
La différence entre width et device-width se perçoit surtout sur les navigateurs mobiles des smartphones.
Les règles peuvent être combinées à l'aide des mots suivants :

- only : uniquement
- and : et
- not : non

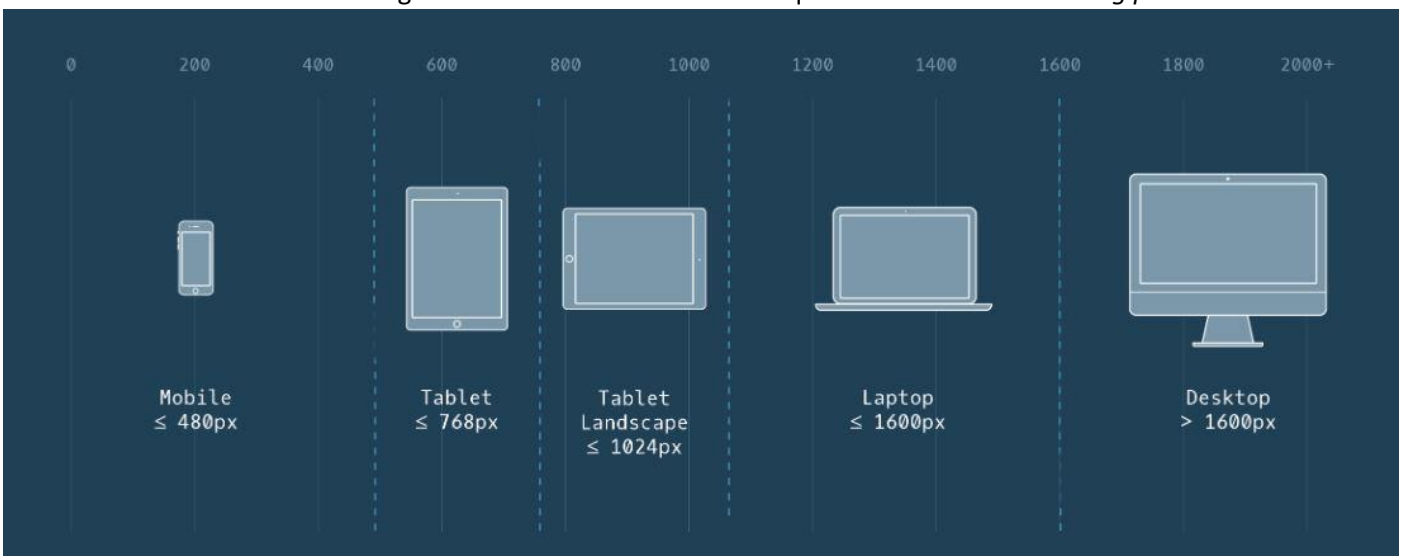
Voici quelques exemples :

```
/* Sur les écrans, quand la largeur de la fenêtre fait au maximum 1280px */
@media screen and (max-width: 1280px)

/* Sur tous types d'écran, quand la largeur de la fenêtre est comprise entre 1024px et 1280px */
@media all and (min-width: 1024px) and (max-width: 1280px)

/* Sur les téléviseurs */
@media tv

/* Sur tous types d'écrans orientés verticalement */
@media all and (orientation: portrait)
```

Si on souhaite faire plusieurs règles mais qu'on a besoin qu'une seule se réalise au moins, on sépare les règles par des virgules.

```
@media only screen and (min-width: 480px), (orientation: landscape) {
    /* CSS ruleset */
}
```

On appelle *breakpoint* les points auxquels des *media queries* sont définies. Ces *breakpoints* sont les tailles d'écran où notre site ne s'affiche pas bien.
Cependant, définir des *media queries* pour chaque taille d'écran serait incroyablement difficile tant il en existe et que chaque année de nouveaux apparaissent.
Plutôt que de définir des *breakpoint* en fonction d'appareils, il est plus efficace de redimensionner le navigateur pour voir où le site web s'affiche étrangement. Ce sont ces dimensions qui deviennent nos *breaking points*.



Les media queries sont surtout utilisées pour adapter le design du site aux différentes largeurs d'écran. Les règles suivantes permettent de la couleur et la taille du texte si la fenêtre fait moins de 1024pixels de large.

```
/* Paragraphes en bleu par défaut */
p
{
    color: blue;
}

/* Nouvelles règles si la fenêtre fait au plus
1024px de large */
@media screen and (max-width: 1024px)
{
    p
    {
        color: red;
        background-color: black;
        font-size: 1.2em;
    }
}
```

Media Queries et navigateur mobile :

Les écrans de smartphone sont beaucoup moins larges les écrans habituels. Pour s'adapter, les navigateurs mobiles affichent le site en dézoomant. : ils simulent une zone d'affichage appelée viewport.
On pourrait donc cibler l'écran avec max-width avec les media queries. Mais le viewport change en fonction du navigateur mobile.
Il faut donc recourir à max-device-width : la largeur du périphérique. Ils ne dépassent pas 480px de large :

```
@media all and (max-device-width: 480px)
{
    /* Vos règles CSS pour les mobiles ici */
}
```

Malheureusement on ne peut pas utiliser la règle handheld car seul opera mobile la reconnaît.
Pour modifier la largeur du viewport du navigateur mobile comme cela :

```
<meta name="viewport" content="width=320" />
```

Ou on peut demander à ce que le viewport soit le même que la largeur de l'écran :

```
<meta name="viewport" content="width=device-width" />
<meta name="viewport" content="width=device-width, initial-scale=1">
```

XIII. Variables et fonction en CSS

1. Les variables

Si on fait un site et que le client veut par exemple changer une couleur, cela peut être long à faire tant cette couleur peut être utilisé beaucoup de fois dans le CSS. Les variables servent à résoudre ce problème.

Pour déclarer une variable :

``h1 { --main-header-color : #DADECC; }```	Un nom de variable commence toujours par "-- ". Les noms sont sensibles à la casse dont on évite les majuscules. Par convention on nomme une variable en gardant à l'esprit où on l'a déclaré et on séparer les mots avec "-".

Pour utiliser une variable :

``h1 { --main-background-color: #DADECC; background-color: var(--main-background-color); }```	Pour utiliser une variable, on utilise la fonction var() qui prend en argument la variable que l'on a défini.
``html { --custom-purple: #FF64ED; --main-color: var(--custom-purple); } body { background: var(--main-color); }```	Les variables peuvent être basée sur d'autres de manière à garder une race des règles et les rendre modulable.

Les variables ont une portée locale. Elles sont valables dans l'élément dans lequel elles sont déclarées et dans les éléments enfants. C'est l'héritage. Certaines propriétés s'héritent d'autres non. Rappel : les *width, height, margin, padding, border* ne s'héritent pas par exemple.
Comme les variables ont une porte locale, il n'y a aucun problème à donner le même nom pour deux variables différentes dans deux endroits différents.
Les variables globales sont déclarées dans la pseudo classe *:root*. Elle pointe la racine du document, donc l'élément <html>. Elles peuvent donc être utilisées dans tout le document HTML.
On peut sans problème redéclarer la même variable à un autre endroit. Cela va écraser localement la valeur de la variable mais pas ailleurs dans le document.

``:root { --menu-color-blue: blue; } #menu-items a { color: var(--menu-color-blue); }```	C'est assez commun de déclarer les variables dans *:root* mais pas obligatoire. Si on fait un très gros site web, c'est plus propre de déclarer les variables là où on en a besoin que de toutes les déclarer dans *:root.*

En cas de problème lors de la déclaration de variable :

- Variable qui n'existe pas/plus
- Valeur incorrecte. Ex : 20px pour un *background*

Les valeurs de secours ne sont pas obligatoires mais cela assure qu'un style soit appliqué en cas d'erreur.

`` body { background: var(--main-background-color, #F3F3F3); } ``` ou ``` body { font-color: var(--main-color, var(--favorite-orange, red)); } ```	On utilise une valeur de secours. Elle se place dans le deuxième argument de la fonction var. Ce 2e argument peut être également qui accepte lui aussi une valeur de secours etc…

Les variables sont très utiles avec les *media queries*. Ils permettent de redéfinir uniquement les variables pour changer l'apparence d'un site sans devoir réécrire un grand nombre de propriétés CSS.

```
@media screen and (min-width: 600px) {
    :root {
      /* Light Color Theme */
      --body-background: lightblue;
      --inner-margin: 6px;
      --body-text-color: black;
      --font-size: 18px;
    }
}

@media screen and (max-width: 600px) {
    :root {
      /* Dark Color Theme */
      --body-background: #000;
      --inner-margin: 12px;
      --body-text-color: #fff;
      --font-size: 12px;
    }
}
```

2. Les fonctions

On a déjà vu les fonctions var() et repeat() jusqu'à présent.
Également la fonction url() permet mettre un *background* par exemple.
Les fonctions sont des blocks de code qui peuvent donc être modulés et réutilisés.
On ne peut pas déclarer ses propres fonctions mais on a accès à une grande variété de fonctions prédéfinies :
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Functions

a. Calc()

La fonction calc prend en argument une expression mathématique et la calcule pour obtenir une valeur qui sera appliquée à la propriété en question.

`` .item { margin: 15px calc(1.5vw + 5px); } ```	La marge de droite est égale à 1.5% du viewport + 5px.

b. Min() et max()

https://developer.mozilla.org/en-US/docs/Web/CSS/min
https://developer.mozilla.org/en-US/docs/Web/CSS/max
Ces fonctions vont prendre le minimum ou le maximum parmi une plage de valeur. C'est intéressant pour le côté responsif d'un site internet.

```
.content {
    width: 50vw;
    max-width: 500px;
}
```

```
.content {
    width: min(500px, 50vw);
}
```

Les deux écritures sont équivalentes. 50vw signifie 50% du viewport.
On peut passer plus de 2 arguments, séparés par des virgules et dans des unités différentes.

c. clamp()

clamp() prend 3arguments et permet de maintenir une valeur spécifiée entre deux bornes min et max.

```
.main-text{
    font-size: clamp(12px, 1.5vw, 48px);
}
```

Elle prendre 3 arguments :
- La valeur minimale
- La valeur préférée 1.5% du viewport ici
- La valeur maximale.

Cela permet d'utiliser une valeur relative mais avec des valeurs minimales et maximales.

d. Rgd(), rgb(a), hsl(), hsla()

Voir paragraphe couleur.

e. Fonction de filtrage

Pour les propriétés *filter* et *backdrop-filter* il existe les fonctions :

- blur()

Applique un flou gaussien à l'élément spécifié. Un seul argument pour le rayon du flou spécifié. Il faut une unité, sauf si zero.

- *brightness()*

Prend un seul argument en %. Pour un nombre inférieur à 1.0 (ou 100%) cela assombrit l'élément. Au-dessus de 1.0 (ou 100%), cela éclaircit l'élément.

- dropshadow()

Applique un effet d'ombre portée.
`drop-shadow(offset-x offset-y blur-radius color)`
Les deux premiers arguments sont obligatoires : ils déterminent le décalage horizontal et vertical de l'ombre. Ensuite viennent le rayon de flou et la couleur, eux, facultatifs.

- contrast()
- grayscale()
- hue-rotate()
- saturate()

Permet avec un argument en % de changer la saturation.

- Invert()
- Opacity()
- Sepia()
- url()

https://developer.mozilla.org/en-US/docs/Web/CSS/filter

f. Fonction de transformation

Cf partie Animation et transformation

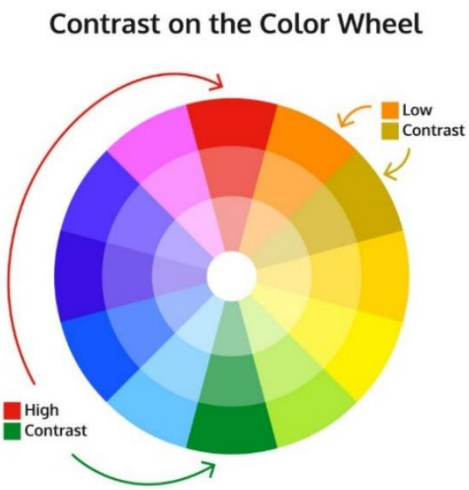
g. linear-gradient()

Accessibilité :
https://www.w3.org/TR/2008/REC-WCAG20-20081211/#visual-audio-contrast-visual-presentation

h. Taille d'écriture : police à 18-20px pour les petits écrans. *Line-height* (1.2-1.5 minimum) et *letter-spacing*. Ne pas trop espacer les lettres sinon les lettres d'écran lisent les lettres individuellement.

i. Structure : avec *text-align*. Largeur paragraphe entre 45 et 85 caractères. Idéalement 65. Unité de largeur *"ch"* qui correspond à la largeur d'un zéro 0. Elle évolue donc en fonction de la police et de sa taille.

j. La couleur : attention au manque de contraste. La couleur ne doit pas se perdre dans l'arrière-plan.



https://webaim.org/resources/contrastchecker/

`a:visited { color: purple; }`	Pour changer le couleur d'un lien déjà visité.
`input:focus { border-color: blue; }`	Pour indiquer le focus sur un champ à remplir.

k. Intéractivité : la balise :

```
<abbr title='Cascading Style Sheets'>CSS</abbr>
```

Mettre des liens., des boutons en valeur. Utiliser la propriété *cursor*. Exemple : *cursor: pointer;* affiche une main.

l. Cacher pour un lecteur d'écran :

```
<h3 aria-hidden='true'>Codecademy Survey</h3>
```

XIV. Préprocesseur SASS

1. Structurer le CSS

Le CSS manque de structure donc il est intéressant de nommer les classes en fonction de la structure du document HTML. Si le container a la classe *form*, on peut donc avoir les classes *form__heading, form__field*.

```css
.btn {
    background-color: #001534;
    color: #15DEA5;
    padding: 1.5rem;
}
.btn-wide {
    width: 100%;
}
```

Lorsque l'on veut créer des boutons avec différents style, pour éviter de se répéter, il vaut mieux créer un bouton avec un style de base et créer ensuite des autres classes que l'on rajoutera aux boutons.

```css
.btn-rounded {
    border-radius: 3rem;
}
```

Cela permet d'appliquer le principe du DRY : Don't Repeat Yourself. Il faut également faire attention à la spécificité des sélecteurs CSS :

- Inline > ID > class > Element

2. Créer des sélecteurs HTML avec la méthodologie BEM

a. Introduction : Bloc, Elément, Modificateur

Il est important de créer des bloc dans une page qui peuvent fonctionner indépendamment les uns des autres comme un header, un footer, un conteneur, un menu ou encore un bouton.

```css
.proj-prev {
    color: #fff;
    margin-bottom: .25rem;
}
.proj-prev__heading {
    font-size: 4rem;
    padding-left: 2.5rem;
    margin: 0;
    line-height: 6rem;
}
.proj-prev--mint {
    color: #15DEA5;
}
```

On nomme un bloc en décrivant sa fonction : *.proj-prev* comme preview project dans un portfolio.

Ensuite on crée des éléments à l'intérieur de ce bloc, comme un titre, un contenu.

Le nom d'un élément doit indiquer son bloc parent suivi d'un double underscore suivi de la fonction de l'élément.

On peut ensuite rajouter des modificateurs qui peuvent modifier l'apparence d'un bloc ou d'un élément. Ils peuvent servir à différencier un bloc/un élément parmi les autres. Pour rajouter un modificateur, on reprend le nom de l'élément et on rajouter deux tirets "--" suivi du nom du modificateur.

b. Exemple barre de navigation BEM

```html
<nav class="nav">
    <ul>
        <li class="nav__link nav__link--active">work</li>
        <li class="nav__link"><a href="/about.html">about</a></li>
        <li class="nav__link"><a href="/contact.html">contact</a></li>
    </ul>
</nav>
```

```css
.nav {
    padding-right: 6rem;
    text-align: right;
}
.nav__link {
    display: inline;
    font-size: 3rem;
    padding-left: 1.5rem;
}
```

```css
.nav__link a {
    text-decoration: none;
    color: #D6FFF5;
}
.nav__link--active {
    color: #001534;
}
.nav__link a:hover {
    color: #fff;}
```

3. Syntaxe des préprocesseurs

Il existe plusieurs plusieurs préprocesseurs. Les plus connus sont Sass, Less, Stylus. Il existe quelques différences de syntaxe mais ils font tous la même chose de manière très similaire. Ici on va parler de Sass et de sa syntaxe.

Sass doit ensuite être compilé pour être transformer en CSS afin d'être compréhensible par le navigateur.

Il y a deux manières d'écrire du Sass. Soit dans un fichier .sass ou .scss. On peut très bien écrire du CSS classique dans un fichier .sass.

a. Le Nesting

Les préprocesseurs sont des outils qui vont permettre de générer du CSS à partir de fichiers écrits dans la syntaxe du préprocesseur que l'on a choisi. Grâce aux préprocesseurs, on peut écrire du code d'une manière plus cohérente visuellement en utilisant le nesting par exemple.

Syntaxe css standard

```css
.nav {
    padding-right: 6rem;
    flex: 2 1 auto;
    text-align: right;
}
.nav nav__link {
display: inline;
font-size: 3rem;
padding-left: 1.5rem;
}
.nav nav__link nav__link--
active {
color: #001534;
}
```

scss

```scss
.nav {
  padding-right: 6rem;
  flex: 2 1 auto;
  text-align: right;
  .nav__link {
      display: inline;
      font-size: 3rem;
      padding-left: 1.5rem;
      .nav__link--active {
        color: #001534;;
      }
  }
}
```

css

```css
.nav {
  padding-right: 6rem;
  flex: 2 1 auto;
  text-align: right;
}
.nav .nav__link {
  display: inline;
  font-size: 3rem;
  padding-left: 1.5rem;
}
.nav .nav__link .nav__link--active {
  color: #001534;
}
```

La syntaxe .scss s'appuie sur la sytaxe CSS standard. Elle permet donc aux développeurs de ne pas être perdus lorsqu'ils écrivent du CSS tout en bénéficiant de toutes les fonctionnalités Sass. Elle ressemble beaucoup à du CSS normal alors que le .sass a une syntaxe plus condensée et concise : plus d'accolades ni de points-virgules.

Il est assez rare de tomber sur du .sass. En général quand on parle du Sass, on parle quasi systématiquement de .scss. On va donc ici parler majoritairement de .scss.

Il faut faire attention à ne pas abuser du nesting. Il ne faut pas répliquer intégralement la structure du HTML sans Sass. Plus on imbrique des sélecteurs, plus on augmente la spécificité, et plus cela sera difficile de les outrepasser.

Il est préférable d'imbriquer les sélecteurs uniquement au sélecteur racine. En pratique il faut éviter d'aller au-delà de deux niveaux de profondeur.

b. L'esperluette

Si on peut ajouter une pseudoclasse *li:hover*. Si on utilise la techniquement précédente, il va se créer un espace entre la classe et la pseudoclasse. Ce qui ne va pas fonctionner. Pour cela il faut donc utiliser l'esperluette "&".

```scss
li {
    display: inline;
    font-size: 3rem;
    color: #D6FFF5;
    &:hover {
        color: #001534;
    }
}
```

```css
li {
    display: inline;
    font-size: 3rem;
    color: #D6FFF5;
}
li:hover {
    color: #001534;
}
```

c. BEM + Nesting

On peut donc combiner le BEM et le nesting :

```scss
.btn {
    display: inline-block;
    margin: 0 auto;
    background: #15DEA5;
    padding: 1rem;
    &--disabled {
        background: grey;
    }
    &--outline {
        background: transparent;
        border: 2px solid #15DEA5;
        &.btn--disabled{
            border: 2px solid grey;
        }
    }
}
```

```scss
.btn {
    display: inline-block;
    margin: 0 auto;
    background: #15DEA5;
    padding: 1rem;
}
.btn--disabled {
    background: grey;
}
.btn--outline {
    background: transparent;
    border: 2px solid #15DEA5;
}
.btn--outline.btn--disabled {
    border: 2px solid grey;
}
```

Le sélecteur *.btn—disabled* marche avec *.btn—outline* et il fonctionne aussi seul, ce qui en fait un système modulaire et réutilisable.

4. Créer du code maintenable à l'aide des techniques Sass intermédiaires

a. Les variables

Les variables permettent de stocker des valeurs comme des couleurs ou des mesures que l'on va réutiliser à travers le code. Grâce aux variables, si on fait le changement une seule fois, il sera répercuté partout où la variable a été utilisée.

Sass compile les variables en CSS et remplace l'instance de la variable directement par sa valeur.

Les variables Sass fonctionnent comme les variables CSS classiques à ceci près que la syntaxe Sass est beaucoup moins compliquée.

Il est plus pratique de nommer une variable en fonction de son rôle. Ici si on décide de changer la couleur le mot *mint* n'aura plus vraiment beaucoup de sens. Et si on change le nom de la variable, il faut la changer partout dans le fichier .scss. Ici il aurait fallu la nommer *$color-primary* par exemple.

Les variables ne servent pas que pour les couleurs. Elles servent aussi pour :

- Les chaînes de caractères
- Les nombres
- Les listes et maps
- Les booléens, les nulls et les fonctions

b. Les mixins

Les mixins sont un ensemble de règle. C'est comme une variable mais qui stocke des blocs entiers de code.

```scss
@mixin heading-shadow{
    text-shadow: .55rem .55rem #15DEA5;
}
.form {
    &__heading {
        @include heading-shadow;
    }
}
```

```css
.form__heading {
    text-shadow: .55rem .55rem #15DEA5;
}
```

On peut bien sûr répliquer le *@include* autant de fois qu'on le souhaite.

Les mixins ne sont pas forcément figées. Elles peuvent accueillir des arguments pour les rendre customisable :

```scss
@mixin heading-shadow($colour){
    text-shadow: .55rem .55rem $colour;
}
.heading{
    &__header {
    @include heading-shadow(#fff);
    }
}
```

Ce qui donne en CSS :

```css
.heading__header {
    text-shadow: 0.55rem 0.55rem #fff;
}
```

On peut également donner une valeur par défaut, par exemple la couleur la plus utilisée pour ne pas devoir la préciser à chaque fois :

```scss
@mixin heading-shadow($colour: $colour-primary){
    text-shadow: .55rem .55rem $colour;
}
.heading{
    &__header {
        @include heading-shadow($colour-white);
    }
}
.form{
    &__heading {
        @include heading-shadow;
    }
}
```

Ce qui donne en CSS :

```css
.heading__header {
    text-shadow: 0.55rem 0.55rem #fff;
}
.form__heading {
    text-shadow: 0.55rem 0.55rem #15DEA5;
}
```

On peut aller encore plus loin en stockant dans une variable la taille de la bordure :

```scss
$heading-shadow-size: 0.55rem;
@mixin heading-shadow($colour: $colour-primary, $shadow-size: $heading-shadow-size){
    text-shadow: $shadow-size solid $colour;
}
```

c. Les extensions

Une mixin peut contenir un ensemble de règle :

```scss
@mixin .typography {
    color: $colour-primary;
    font-size: 2rem;
    font-weight: 100;
    line-height: 1.7;
}
```

```scss
h1 {
    @include .typography;
}
textarea {
    @include .typography;
}
button {
    @include .typography;
}
input {
    @include .typography;
}
```

Cela peut paraître intéressant mais le CSS compilé sera très répétitif et il sera difficile de s'y retrouver au milieu de tous les blocs qui se ressemblent.

On pourrait alors penser à créer un sélecteur *.typography* et l'appliquer aux éléments HTML qui en ont besoin. Mais cela va à l'encontre des conventions de nomenclature BEM vues précédemment. Pour cela on va utiliser des extensions Sass.

```scss
.typography {
    color: $color-primary;
    font-size: 2rem;
    font-weight: 100;
    line-height: 1.7;
}
h1 {
  @extend .typography;
}
textarea {
    @extend .typography;
}
button {
    @extend .typography;
}
input {
    @extend .typography;
}
```

Ce qui va donner en CSS :

```scss
.typography, input, button, textarea, h1 {
    color: #15DEA5;
    font-size: 2rem;
    font-weight: 100;
    line-height: 1.7;
}
```

Ce qui est beaucoup plus court que la méthode précédente.

Avec les *@mixin*, Sass inclut le contenu du mixin un peu partout où il est appelé, ce qui a pour conséquence un tas de répétitions. Mais avec *@extend* sur h1 par exemple, Sass va plutôt étendre les règles de *.typography* à h1 et va l'ajouter à la liste.

Le problème ici c'est que finalement le sélecteur *.typography* n'est pas utilisé. Il sert ici de placeholder. Mais avoir dans son CSS des sélecteurs non utilisés est une mauvaise idée. Ils augmentent la taille du fichier et ajoutent de la confusion inutilement. Sass vient donc à la rescousse avec son placeholder intégré et prêt à l'emploi : il se matérialise par le préfixe % :

```scss
%typography {
    color: $colour-primary;
    font-size: 2rem;
    font-weight: 100;
    line-height: 1.7;
}
```

```scss
h1 {
    @extend %typography;
}
```

Ainsi le placeholder n'apparaît pas dans le fichier CSS final

Mixins ou Extensions ?

Les mixins et les extensions sont très semblables. Pour discerner leurs différentes fonctions, il faut prêter attention au code compilé, ce qui complique la tâche pour déterminer la situation dans laquelle utiliser l'un ou l'autre.

Si on veut des arguments, pas le choix ici, il faudra obligatoirement utiliser les mixins. Les extensions ne tolèrent pas les arguments.

A part les arguments la différence principale est que *@mixin* duplique les règles alors que *@extend* duplique le sélecteur.

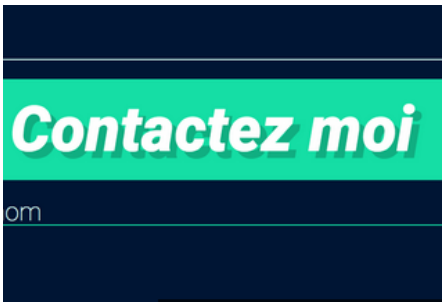
Dans le cas de notre *.typography* cela a pour conséquence que lorsque l'on cherche le sélecteur h1 par exemple dans le CSS, on a accès aux propriétés de h1 mais aux propriétés typographiques. On les trouvera dans une liste aux côtés des autres sélecteurs. Cela signifie que l'on peut risquer d'oublier de les avoir étendues.

Alors même si les mixins génèrent des tas de code qui se répètent, ils n'affectent pas l'organisation générale du CSS. Les extensions démolissent l'ordre et la prédictibilité de notre codebase uniquement pour épargner du code répétitif. Le jeu n'en vaut donc pas la chandelle. Avec les mixins, le code sera donc au final plus propre et plus simple à maintenir.

Il est tout de même important de connaître les extensions car il n'y a pas de règles concrète sur l'architecture du CSS et que d'autres gens peuvent penser différemment. Être développeur ce n'est pas seulement écrire son code, c'est aussi être capable de lire le code des autres et de travailler avec.

d. Améliorer les mixins avec les fonctions

Si on veut faire l'ombre d'un texte d'une couleur plus foncée que la couleur du background. On pourrait stocker cette couleur dans une variable, mais si on décide de changer de couleur, il faudra ne pas oublier de changer la couleur de l'ombre également.



Sass dispose de beaucoup de fonction donc des fonctions pour désaturer, inverse, foncé, éclaircir une couleur.

La fonction *darken* prend deux arguments : une valeur de couleur et une valeur qui indique à quel point on veut assombrir la couleur.

```
@mixin heading-shadow($colour:$colour-primary, $size: $heading-shadow-size){
    text-shadow: $size $size darken($colour, 10%);
}
```

Ici, dès qu'on changera la valeur de *$colour*, l'ombre changera automatiquement.

La fonction *darken* va modifier la composante "l" du code hsl de la couleur (ici enlever 10%).

La fonction *adjust-hue(color, angle)* modifie la couleur donnée en argument en changeant l'angle dans le *hsl*.

La fonction complement(color) retourne la couleur complémentaire.

Document des fonctions de Sass :
https://sass-lang.com/documentation/modules

e. Améliorer les mixins grâce aux conditions

On peut améliorer l'exemple précédent grâce aux conditions. En effet si on utilise une couleur foncée, on verra moins l'ombre car l'œil humain distingue mal les couleurs sombres. Il serait donc logique, quand la couleur est trop foncée de faire plutôt une ombre plus claire.

A l'aide d'une condition on pourrait indiquer que toute couleur ayant une luminosité de moins de 25% sera éclaircie, sinon elle sera foncée.

```scss
@mixin heading-shadow($colour: $colour-primary, $size: $heading-shadow-size){
    @if ( lightness($colour) < 25% ) {
        $colour: lighten($colour, 10%);
    }@else{
        $colour: darken($colour, 10%);
    }
    text-shadow: $size $size $colour;
}
```

Ici on utilise la fonction *lightness()* qui retourne la luminosité d'une couleur.

Si on veut que la couleur soit éclaircie, on utilise la fonction *lighten()* sinon *darken()*.

Il existe 6 opérateurs de comparaison : ==, !=, >, <, >=, <=.

On peut enchaîner plusieurs conditions en utilisant des opérateurs logiques comme *and, or*.

L'instruction *else* n'est pas obligatoire.

f. Créer ses propres fonctions

Dans l'exemple précédent, si on a déjà choisi la couleur de l'ombre, elle sera modifiée quoi qu'il arrive par le bloc de condition. Également si on veut utiliser le même procéder ailleurs, on va devoir retaper le bloc de condition. Il est donc intéressant de mettre ce bloc dans une fonction pour le réutiliser si et quand on en a envie.

```scss
@function lightness-shift($colour){
    @if ( lightness($colour) < 25% ) {
        @return lighten($colour, 10%);
    }@else{
        @return darken($colour, 10%);
    }
}
```

La fonction est créée avec le mot clé *@function*. On y place ensuite le bloc de condition et le mot clé *@return* retourne la valeur modifiée que l'on va pouvoir utiliser ensuite. Dans la *@mixin* précédente par exemple :

```scss
@mixin heading-shadow($colour: lightness-shift($colour-primary), $size: $heading-shadow-size){
    color: $colour;
    text-shadow: $size $size $colour;
}
```

Ainsi on peut utiliser cette *@mixin* avec n'importe quelle couleur de notre choix qui ne sera pas modifiée.

5. Optimiser son code à l'aide des techniques Sass avancées

a. Le système 7-1 pour une codebase plus facile à gérer

Les fichiers Sass sont bien organisés mais il peut être fastidieux de scroller de haut en bas pour chercher les variables, les mixins, les sélecteurs etc… Le système 7-1 c'est créer 7 dossiers dans lequel on va ranger nos fichiers au lieu de tout mettre dans un seul :

1. Base

Il va contenir les fichiers qui définissent les fondations du site par exemple la police, les normes que l'on veut appliquer à tout le site comme le box-sizing par exemple

2. Utils

On y range les variables, les fonctions, les mixins, les %placeholders pour les extensions si on en utilise.

3. Layout

Dossier où l'on range les blocs BEM qui contiennent ce qui est réutilisable comme par exemple un header pour les mises en page de grande taille.

4. Composants

Utilisé pour ranger les blocs BEM qui sont plus indépendants comme les boutons.

5. Pages

Contient les blocs de code qui ne s'appliquent qu'à une seule page et qui ne sont pas réutiliser ailleurs.

6. Themes

Ici on stocke le code thématique comme un style customisé pour Noël, pâques, l'été etc…

7. Vendors (tiers)

Un dossier pour les feuilles de style externe comme Bootstrap ou jQuery. En gros pour tous les CSS venant de l'extérieur. Des Framework comme Bootstrap permettent d'accélérer le développement d'un site car ils contiennent des feuilles de style prédéfinies comme pour des formulaires ou des boutons.

Marche à suivre :

On va créer un fichier "*_variables.scss*". Les fichiers individuels font tous partie d'une codebase que Sass appelle les partiels. Pour indiquer à Sass qu'un fichier est un partiel, on lui ajouter le préfixe underscore(_).

Ensuite il faut préciser dans le fichier scss principal où se trouve les variables que l'on vient de déplacer. Pour cela on ajoute la ligne :

```
@import "./utils/variables";
```

Ici le dossier utils est dans le même répertoire que le fichier main.scss.

Lors de l'importation d'un partiel, Sass sait déjà qu'il faut chercher un fichier qui commence par un underscore et qu'il s'agit d'un fichier Sass, donc pas besoin de préciser ces éléments.

De la même manière on peut créer un fichier *_mixins.scss* puis *_functions.scss* dans ce même dossier puis les fichiers *_form.scss, _nav.scss* et *_header.scss* dans le dossier layout etc…

Il faudra alors faire attention à l'ordre d'importation. En effet si on importe le partiel du formulaire avant les variables, on aura un erreur de compilation car les variables utilisées dans le formulaire ne seraient pas encore définies.

D'une manière générale pour évier les erreurs il faut importer les fichiers dans l'ordre suivant :

1. utils
    a. variables
    b. fonctions
    c. mixins
    d. placeholders
2. feuilles de style tiers (vendors) si on en a
3. base
4. composants
5. layout
6. pages
7. theme

```scss
@import "./utils/variables";
@import "./utils/functions";
@import "./utils/mixins";
@import "./utils/extensions";
@import "./base/base";
@import "./base/typography";
@import "./components/buttons";
@import "./layouts/header";
@import "./layouts/nav";
@import "./layouts/container";
@import "./layouts/form";
@import "./pages/work";
@import "./pages/about";
@import "./pages/project";
```

On n'utilise pas les 7 dossiers dans tous les projets mais il est important de connaître l'ordre.

On fois que l'ensemble du code a bien été séparé en partiels importés, le fichier main.scss ne doit contenir que des imports. Les ensembles des règles sont eux aussi rangés dans leurs propres partiels.

6. Les listes et les maps

- Les listes

Dans le cas des padding, il ne serait pas pertinent de créer 4 variables pour les 4 valeurs. On peut alors procéder comme cela :

SCSS :

```scss
$padding-dimensions: 1rem 2rem 3rem 4rem;
.block {
  padding: $padding-dimensions;
}
```

CSS compilé :

```css
.block {
    padding: 1rem 2rem 3rem 4rem;
}
```

Cette variable *$padding-dimensions* s'appelle une liste pour Sass. La syntaxe est assez flexible, on peut séparer par des espaces, des virgules, mettre toutes les valeurs dans des parenthèses avec ou sans virgule.

On peut créer une liste avec différentes tailles de police d'écriture et affecter les tailles à différents éléments :

```scss
$font-size: 7rem 5rem 4rem 2rem;
.form{
  &__field {
      & label {
        font-size: nth($font-size, 4);
      }
  }
}
```

Ce qui permet de stocker toutes les tailles de police dans une seule variable. Mais cela peut être compliqué à comprendre si l'on relit le code quelques semaines plus tard.

Attention dans les listes Sass, les indices commencent à 1.

- Les maps

Les maps sont semblables aux listes sauf qu'elles assignent à chaque valeur un nom sous la forme de paire clé/valeur.

```
$font-size: (logo:7rem, heading:5rem, project-heading:4rem, label:2rem);
```

Cela permet d'associer chaque taille de police à une partie de la page de manière plus compréhensible.

```
$font-size: (logo:7rem, heading:5rem, project-heading:4rem, label:2rem);
.form{
  &__field {
    & label {
        font-size: map-get($font-size, label);
      }
  }
}
```

Pour accéder à une valeur d'une map c'est un peu différent. On utilise la fonction *map-get()* qui nécessite 2 arguments : le 1er est le nom de la map et le 2e est le nom de clé.

En CSS compilé :

```
.form__field label {
    font-size: 2rem;
}
```

Cas pratique :

On peut créer une map *$input-txt-palettes* qui va contenir les palettes de couleurs pour tous les input de texte. On va regrouper les palettes de couleurs qui seront classées par état (active, focus, invalid) grâce au nesting :

```
$colour-primary: #15DEA5;
$colour-secondary: #001534;
$colour-accent: #D6FFF5;
$colour-white: #fff;
$colour-invalid: #DB464B;
```

```
$txt-input-palette: (
  active: (
      bg: $colour-primary,
      border: $colour-primary,
      txt: $colour-white,
  ),
  focus: (
      bg: $colour-primary,
      border: $colour-primary,
      txt: $colour-white,
  ),
  invalid: (
      bg: $colour-invalid,
      border: $colour-white,
      txt: $colour-white,
  )
);
```

- Utiliser les mixins avec les maps

Ici une mixin va nous permettre de déployer les palettes :

```
@mixin txt-input-palette($state) {
    $palette: map-get($txt-input-palette, $state);
    border: .1rem solid map-get($palette, border);
    background-color: map-get($palette, bg);
    color: map-get($palette, txt);
}
```

On crée donc la mixin qui va prendre en argument l'état du bouton (active, focus ou invalid).

On va stocker dans la variable *$palette* la map qui est la palette correspond à l'état indiqué.

On réutilise donc la fonction *map-get()* pour extraire les différentes couleurs et générer les règles CSS :

```
.form {
    &__field {
        & input {
            @include txt-input-
palette(focus);
        }
    }
}
```

```
.form__field input {
    border: 0.1rem solid #15DEA5;
    background-color: #001534;
    color: #15DEA5;
}
```

7. Les boucles dans Sass

Avec l'exemple précédent, il faut encore créer un sélecteur pour chaque état. Sauf que c'est un travail répétitif. Les boucles vont permettre d'éviter cela.

On va itérer dans la palette précédente à l'aide de la syntaxe suivante :

```
@each $state, $palette in $palettes
```

$state va représenter le nom de l'état *active, focus* et *invalid* et *$palette* la palette de couleur associé. Suivra ensuite les actions à faire à chaque itération de la boucle.

Cette boucle va être intégré dans la mixin et son rôle va être d'écrire les sélecteurs imbriqués. C'est pour cela que le code généré par la boucle commencera par "&" :

```
@mixin txt-input-palette($palettes) {
  @each $state, $palette in $palettes{
      &:#{$state}{
        border: .1rem solid map-get($palette, border);
        background-color: map-get($palette, bg);
        color: map-get($palette, txt);
      }
  }
}
```

Cette syntaxe : `&:#{$state}` avec le # et les {} permet de faire une interpolation de la variable et créer le sélecteur.

Cela générera le code suivant :

```
.form__field input:active {
    border: 0.1rem solid #15DEA5;
    background-color: #15DEA5;
    color: #fff;
}
.form__field input:focus {
    border: 0.1rem solid #15DEA5;
    background-color: #15DEA5;
    color: #fff;
}
.form__field input:invalid {
    border: 0.1rem solid #fff;
    background-color: #DB464B;
    color: #fff;
}
```

Un autre exemple de boucle :

scss

```scss
$colours: (
  mint: #15DEA5,
  navy: #001534,
  seafoam: #D6FFF5,
  white: #fff,
  rust: #DB464B
);

@each $colour, $hex in $colours {
  .btn--#{$colour} {
      background-color: $hex;
  }
}
```

scss

```scss
.btn--mint {
  background-color: #15DEA5;
}

.btn--navy {
  background-color: #001534;
}

.btn--seafoam {
  background-color: #D6FFF5;
}

.btn--white {
  background-color: #fff;
}

.btn--rust {
  background-color: #DB464B;
}
```

8. Les medias queries et les breakpoints avec Sass

En CSS classique, on doit placer les sélecteurs au sein de la media query, ce qui a pour conséquences qu'ils ne feront partie de leur bloc BEM nestés. Cela complique la tâche pour maintenir le code. Sass va donc permettre de faciliter cela en autorisant le fait de placer les media queries directement dans les sélecteurs.

En SCSS :

```scss
.proj-grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  @media (max-width: 599px) {
      grid-template-columns: 1fr;
  }
}
```

En CSS :

```css
.proj-grid {
display: grid;
grid-template-columns: repeat(3, 1fr);
}
@media (max-width: 599px) {
    .proj-grid {
      grid-template-columns: 1fr;
    }
}
```

Pour éviter de devoir recopier la media queries à la suite de chacun des sélecteurs concernés, on va d'abord créer une map avec les différents breakpoints :

```scss
$breakpoints: (
  mobile: 599px
);
```

Pour éviter de devoir écrire cela :

```scss
@media screen and (max-width: map-get($breakpoints, mobile)
```

Qui serait contre-productif. On va plutôt s'aider d'une mixin :

```scss
@mixin mobile-only {
  @media screen and (max-width: map-get($breakpoints, mobile)){
      grid-template-columns: 1fr;
  }
}
```

Sauf que le contenu de la media query sera différent en fonction du sélecteur concernés… Il faudrait donc écrire une mixin pour chacune des media queries, ce qui serait encore une fois contreproductif. On va donc utiliser la directive *@content* qui va permettre de remplacer *@content* par le code que l'on aura placé au moment de l'appel de la mixin à l'aide d'accolage juste après *@include mobile-only{}* :

SCSS :

```scss
@mixin mobile-only {
  @media screen and (max-width: map-get($breakpoints, mobile)){
      @content;
  }
}
.proj-grid {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  @include mobile-only{
      grid-template-columns: 1fr;
  }
}
```

CSS :

```css
.proj-grid {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
}
@media screen and (max-width: 599px) {
    .proj-grid {
      grid-template-columns: 1fr;
    }
}
```

9. Autoprefixer

```css
.header {
    display: -webkit-box;
    display: -moz-box;
    display: -ms-flexbox;
    display: flex;
}
```

Les équipes qui conçoivent les navigateurs n'attendent pas forcément que les nouveaux outils soient standardisés pour les intégrer, il crée leur propre version en ajoutant un préfixe pour la distinguer des autres navigateurs. Il faut utiliser des préfixes dans le code :

Il y a donc *webkit* pour chrome et safari, *ms* pour microsoft et *moz* pour Firefox.

"Autoprefixer" est un plugin qui nous éviter de noter les préfixes pour les propriétés où cela est nécessaire.

- npm install autoprefixer postcss postcss-cli -g
- on rajoute dans le fichier package.json

```
"prefix": "postcss ./public/css/style.css --use autoprefixer -d ./public/css/prefixed/"
```

On indique le nom du nouveau package *postcss*, où se trouve le fichier CSS compilé, d'utiliser autoprefixer puis l'endroit où mettre la nouvelle feuille de style préfixée.

Il nous reste encore à préciser jusqu'à quand autoprefixer doit remonter pour s'assurer de la compatibilité avec tous les browers. Par défaut il n'y vérifier que la précédente version. Pour modifier cela, il faut rajouter une clé dans le fichier .json :

```
"browserslist": "last 4 versions",
```

Cela lui préciser qu'il va remonter et prendre en compte les 4 versions avant celle actuelle.

Il ne reste plus qu'à lancer le script :

- npm run prefix

10. Installer Sass

Pré requis : installer nodeJS.

- npm -g install sass
- contrôle avec node -v et npm -v
- npm -g install sass
- installation de l'extension VScode Live Sass Compiler puis
    - npm package.json (peut ne pas fonctionner)
    - npm init
    - donner un nom : package.json par exemple etc… puis "yes" pour valider
    - npm install -g sass
    - contrôle avec sass –version
    - dans le fichier package.json au niveau de "scripts":
        - `sass --watch ./sass/main.scss:./public/css/style.css`

indique où trouver le fichier sass et où compiler le résultat.

    - npm run sass

Il existe plusieurs façons de compiler le CSS :

- compressed

```
"sass": "sass --watch ./sass/main.scss:./public/css/style.css --style compressed"
```

Va mettre tout le CSS en une seule ligne

- nested
- expanded
- compact

Il faut quitter le watch en cours (Ctrl-C) et relancer sass si on change le fichier package.json.

Pour Aller Plus Loin

I. Du site Web à l'application web : JavaScript, AJAX

JavaScript est un langage qui existe depuis de nombreuses années et que l'on utilise sur le web en plus de HTML et CSS.
On peut faire beaucoup de choses avec HTML et CSS mais dès qu'on veut rendre sa page interactive, un langage comme JavaScript devient indispensable.
Exemple d'utilisation de JavaScript :

- On peut l'utiliser pour modifier des propriétés CSS sans recharger la page. Ex : on pointe sur une image et le fond du site change de couleur. Impossible à faire avec :hover car cela concerne deux balises différentes. Limite du CSS
- On peut modifier le code HTML de la page sans avoir à la recharger, pendant que le visiteur consulte la page.
- Afficher des boites de dialogue à l'écran du visiteur
- Modifier la taille de la fenêtre.

JavaScript est un langage qui se rapproche des langages de programmation comme C, C++, Python, Ruby à l'inverse du HTML et CSS qui sont davantage des langages de description : ils décrivent comme doit apparaître la page mais ne donnent pas d'ordres directs à l'ordinateur.
JavaScript est utilisé pour faire de l'AJAX (Asynchronous JavaScript And XML). Cette technique permet de modifier une page web pendant le visiteur la consulte en échangeant des données avec le serveur. Cela donne l'impression que les pages sont plus dynamiques et plus réactives. Plus besoin de recharger la page.
Les navigateurs sont de plus en plus efficaces dans leur traitement du JavaScript. Les pages l'utilisant sont de plus en plus réactives et on peut créer des sites qui deviennent de véritables applications web : l'équivalent de logiciels mais disponibles comme des sites web. Ex : google docs.

II. Technologies liées à HTML5 : Canvas, SVG, Web Sockets…

- Canvas : permet de dessiner au sein de la page web à l'intérieur de la balise <canvas>. On peut dessiner des formes, mais aussi ajouter des images, les manipuler, appliquer des filtres graphiques. Ce qui permet de réaliser aujourd'hui des jeux et applications graphiques directement dans des pages web.
- SVG : permet de créer des dessins vectoriels au sein des pages web. A la différence Canvas, ces dessins peuvent être agrandis à l'infini.
- Drag & Drop : permet de faire du glisser-déposer dans une page web. Ex : Gmail pour les pièces jointes.
- File API : permet d'accéder aux fichiers stockés sur la machine du visiteur (avec son autorisation). Utilisation en combinaison avec Drag & Drop
- Géolocalisation : Pour localiser les visiteurs et lui proposer des services liés à sa localisation. Localisation pas toujours précise mais permet de repérer un visiteur à quelques km près (avec son accord).
- Web Storage : permet de stocker des infos sur la machine du visiteur. Alternative, plus puissante que les cookies. Infos hiérarchisées comme dans une base de données.
- AppCache : permet de demander au navigateur de mettre en cache certains fichiers. Plus besoin de téléchargement systématique. Dans le cas d'une application web : permet une utilisation hors ligne.
- Web Sockets : permet des échanges en temps réel entre le navigateur du visiteur et le serveur. Avenir des applications web qui pourront devenir aussi rapide que des programmes.
- WebGL : permet d'introduire de la 3D dans les pages web en utilisant le standard 3D OpenGL. Scènes 3D directement gérées par la carte graphique.

La plupart de ces technos s'utilisent avec JavaScript.

III. Les sites web dynamiques : PHP, JEE, ASP.NET…

Ce sont des langages de programmation mais qui au lieu de s'exécuter sur la machine des visiteurs, ils s'exécutent côté serveur.
Le serveur étant souvent bien plus puissant que l'ordinateur du visiteur, cela va permettre d'effectuer des calculs plus complexes. Il y a également plus de contrôle côté serveur possible. Mais le JavaScript reste indispensable pour certains choses que l'on ne peut faire que côté visiteur.
Les langages serveurs permettent de générer la page web qui est ensuite envoyé au visiteur. Chaque visiteur peut donc obtenir une page web personnalisée.
Ces langages ne servent pas à la même chose, ils se complètent. En combinant HTML CSS JavaScript PHP, on peut faire de l'AJAX, effectuer des calculs, stocker des informations dans des bases de données soit de vrais sites web dynamiques.

- PHP : l'un des plus connus. Facile à utiliser et puissant. Il est utilisé sur Facebook ou OpenClassrooms par exemple.
- JavaEE : très utilisé dans le monde pro, c'est une extension du langage Java qui permet de réaliser des sites web dynamiques, puissants et robustes. Au début il est un peu plus complexe à prendre en main que PHP.
- ASP .NET (C#) : assez semblable à JEE. C'est le langage de Microsoft. On l'utilise en combinaison avec d'autres technos Microsoft (windows server…). Il utilise le puissant framework .NET : véritable couteau suisse des développeurs qui offre de nombreuses fonctionnalités.
- Django (Python) : une extansion du langage Python qui permet de réaliser rapidement et facilement des sites web dynamiques. Il est connu pour générer des interfaces d'administration prêtes à l'emploi.
- Ruby on Rails : une extension du langage Ruby, assez similaire à Django qui permet de réaliser des sites web dynamiques facilement et avec une grande souplesse.

Connaître ces langages est indispensable et on veut traiter le résultat des formulaires HTML.
Grâce à ces langages on peut réaliser ces choses sur notre site web :

- Forums
- Newsletter
- Compteur de visiteurs
- Système de news automatisé
- Gestion de membres
- Jeux web
- Etc…