CSS avancé

I) Structurer une page HTML

1) L'en-tête du site

```
<header>
   <!-- Placez ici le contenu de l'en-tête de votre page (logo, une bannière, le slogan de votre site, etc.) -->
</header>
```

Rq : L'en-tête peut contenir tout ce que vous voulez : images, liens, textes…
Il peut y avoir plusieurs en-têtes dans votre page. Si celle-ci est découpée en plusieurs sections, chaque section peut en effet avoir son propre<header>.

2) Le pied de page

```
<footer>
   <!-- Placez ici le contenu du pied de page(liens de contact, le nom de l'auteur, les mentions légales, etc.)-->
</footer>
```

3) Les principaux liens de navigation

La balise <nav> doit regrouper tous les principaux liens de navigation du site comme par exemple le menu principal de votre site.
Généralement, le menu est réalisé sous forme de liste à puces à l'intérieur de la balise<nav>:

```
<nav>
   <ul>
      <li><a href="index.html">Accueil</a></li>
      <li><a href="forum.html">Forum</a></li>
      <li><a href="contact.html">Contact</a></li>
   </ul>
</nav>
```

4) Section de page

```
<section>
   <h1>Ma section de page</h1>
   <p>Bla bla bla bla</p>
</section>
```

Rq : La balise <section> sert à regrouper des contenus en fonction de leur thématique. Elle est généralement au centre de la page.

5) Les informations complémentaires

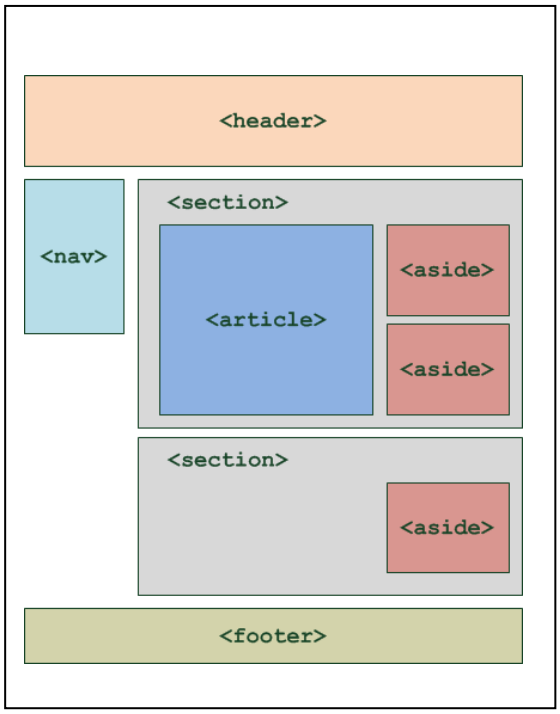
```
<aside>
   <!-- Placez ici des informations complémentaires -->
</aside>
```

Rq : La balise <aside> est conçue pour contenir des informations complémentaires au document que l'on visualise. Elle est généralement placées sur le côté.

6) Les articles indépendants

```
<article>
   <h1>Mon article</h1>
   <p>Bla bla bla bla</p>
</article>
```

Rq : La balise <article> sert à englober une portion généralement autonome de la page. C'est une partie de la page qui pourrait ainsi être reprise sur un autre site (exemple : articles de journaux ou de blogs).



7) Schéma récapitulatif

Ce schéma n'est qu'un exemple.

Ces balises peuvent être imbriquées les unes dans les autres. Ainsi, on peut imaginer une seconde balise <header>, placée cette fois à l'intérieur d'une <section>.

Ces balises seules ne changent pas l'apparence du site. Elles sont là pour organiser le code HTML et mieux s'y retrouver dans le codage.

II) Le modèle des boites

1) Les balises de type block et inline

En HTML, la plupart des balises peuvent se ranger en deux catégories :

- Les balises block
  une balise de type block crée automatiquement un retour à la ligne avant et après. La page web sera constituée d'une série de blocs les uns à la suite des autres. Mais, il est possible de mettre un bloc à l'intérieur d'un autre.
  Ex : <p>, <footer>, <h1>, <article>, etc.
- Les balises inline
  une balise de type inline se trouve obligatoirement à l'intérieur d'une balise block. Le texte qui se trouve à l'intérieur s'écrit donc à la suite du texte précédent, sur la même ligne.
  Ex : , , <a>, <img/>, <mark>, etc.

Les balises universelles sont des balises qui n'ont aucun sens particulier, mais l'intérêt est que l'on peut leur appliquer une « class » quand aucune autre balise ne convient.

-   (inline) ;
-   (block).

Attention : ces balises universelles sont pratiques dans certains cas, mais attention à ne pas en abuser. Quand on peut, il faut utiliser les balises les plus adaptés afin d'être mieux comprises par l'ordinateur et mieux référencé sur les moteurs de recherche.

2) Les dimensions

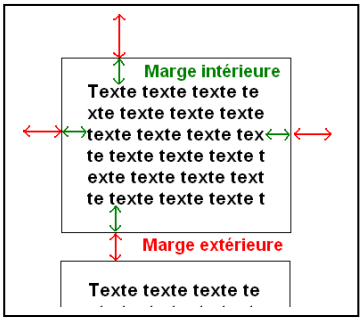
Par défaut, un bloc prend 100% de la largeur disponible, mais il est possible d'en définir les dimensions dans CSS
width: largeur du bloc (sans les marges) exprimée en pixels (px) ou en pourcentage (%)
height: hauteur du bloc (sans les marges) exprimée en pixels (px) ou en pourcentage (%)

Rq : Les pourcentages seront utiles pour créer un design qui s'adapte automatiquement à la résolution d'écran du visiteur.

On peut demander à ce qu'un bloc ait des dimensions minimales et maximales. Cela permet de définir des dimensions limites pour que le site s'adapte aux différentes résolutions d'écran des visiteurs :
min-width: largeur minimale ;
min-height: hauteur minimale ;
max-width: largeur maximale ;
max-height: hauteur maximale.

Exemple : on peut demander à ce que les paragraphes occupent 50% de la largeur et exiger qu'ils fassent au moins 400 pixels de large dans tous les cas :
p

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



3) Les marges

Tous les blocs possèdent des marges. Il existe deux types de marges :
padding : marges intérieures exprimées en général en pixels (px)
margin : marges extérieures exprimées en général en pixels (px)

Rq : les marges par défaut ne sont pas les mêmes pour toutes les balises. Par défaut, <p> n'a pas de marge intérieure, mais a une marge extérieure qui donne l'impression de saut de ligne.

Ces propriétés CSS s'utilisent aussi pour les balises de type inline qui possèdent également des marges.

Attention : les marges s'appliquent aux 4 côtés du bloc. Pour contrôler la marge de chaque côté, on utilise :
margin-top, margin-bottom, margin-left, margin-right, padding-top, padding-bottom, padding-left, padding-right

- plus simplement en utilisant : margin: 2px 0 3px 1px; (haut/droite/bas/gauche)
- une autre notation raccourcie margin: 2px 1px (haut et bas / gauche et droite)

Il est tout à fait possible de centrer horizontalement des blocs, ce qui est très pratique pour réaliser un design centré quand on ne connaît pas la résolution du visiteur.
Pour cela, il faut respecter les règles suivantes :

- donner une largeur au bloc avec la propriété width
- indiquer que vous voulez des marges extérieures automatiques avec margin: auto

Exemple :

```
p
{
   width: 350px; /* On a indiqué une largeur (obligatoire) */
   margin: auto; /* On peut donc demander à ce que le bloc soit centré avec auto */
   border: 1px solid black;
   text-align: justify;
   padding: 12px;
   margin-bottom: 20px;
}
```

4) Quand ça dépasse…

Lorsque les dimensions d'un bloc est fixé, il arrive qu'ils deviennent trop petits pour un long texte.

Overflow: couper un bloc

- visible (par défaut) : si le texte dépasse les limites, il reste visible et sort du bloc.
- hidden : si le texte dépasse les limites, il sera coupé et ne pourra être lu entièrement.
- scroll : le texte sera coupé s'il dépasse les limites. Mais, le navigateur mettra en place une barre de défilement pour qu'on puisse lire l'ensemble du texte.
- auto (conseillé) : le navigateur met une barre de défilement si cela est nécessaire.

Rq : il existe une ancienne balise HTML, <iframe>, qui donne à peu près le même résultat. Mais, son usage est déconseillé aujourd'hui. Elle permet de charger tout le contenu d'une autre page HTML au sein de votre page.

word-wrap: break-word;: coupe les mots trop larges pour le bloc (comme les URL) en forçant la césure.

Rq : il est conseillé d'utiliser cette fonctionnalité dès qu'un bloc est susceptible de contenir du texte saisi par des utilisateurs (sur les forums par exemple).

III) La mise en page avec flexbox

1) Un conteneur et des éléments

Le principe de la mise en page avec Flexbox est simple : on définit un conteneur et à l'intérieur on place plusieurs éléments.

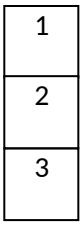
Le conteneur est une balise HTML, et les éléments sont d'autres balises HTML à l'intérieur :

```
<div id="conteneur">
   <div class="element">Elément 1</div>
   <div class="element">Elément 2</div>
   <div class="element">Elément 3</div>
</div>
```

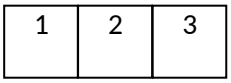
Sur une même page web, on peut avoir plusieurs conteneurs.

2) La propriété flex

Par défaut les blocs se placent les uns au-dessous des autres. Mais si l'on applique la propriété flex à conteneur dans CSS, alors les blocs se placent maintenant par défaut côte à côte.




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

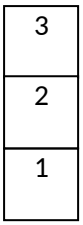


flex-direction : permet de positionner les blocs comme on le souhaite

- row : organisés sur une ligne (par défaut)
- column : organisés sur une colonne
- row-reverse : organisés sur une ligne, mais en ordre inversé
- column-reverse : organisés sur une colonne, mais en ordre inversé

Exemple :

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



flex-wrap : permet de demander que les blocs aillent à la ligne lorsqu'ils n'ont plus la place (pour éviter des bugs de design)

- nowrap : pas de retour à la ligne (par défaut)
- wrap : les éléments vont à la ligne lorsqu'il n'y a plus la place
- wrap-reverse : les éléments vont à la ligne lorsqu'il n'y a plus la place en sens inverse

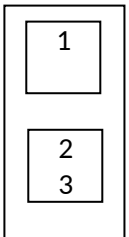
3) Alignement des éléments

Les éléments sont organisés soit horizontalement (par défaut), soit verticalement. Cela définit ce qu'on appelle l'axe principal. Il y a aussi un axe secondaire (cross axis) :

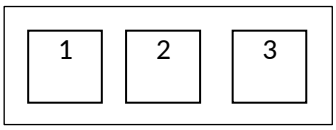
- Si vos éléments sont organisés horizontalement, l'axe secondaire est l'axe vertical.
- Si vos éléments sont organisés verticalement, l'axe secondaire est l'axe horizontal.

justify-content : permet de choisir l'alignement des éléments suivant l'axe principal (horizontal par défaut) :

- flex-start : alignés au début (par défaut)
- flex-end : alignés à la fin
- center : alignés au centre
- space-between : les éléments sont étirés sur tout l'axe (il y a de l'espace entre eux)
- space-around : idem, les éléments sont étirés sur tout l'axe, mais ils laissent aussi de l'espace sur les extrémités



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



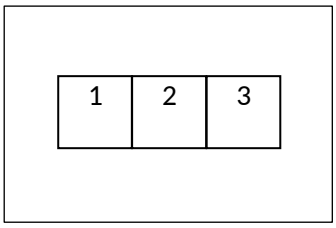
Rq : si on ajoute la propriété flex-direction: column, l'axe vertical deviendra l'axe principal.

align-items : permet de choisir l'alignement des éléments suivant l'axe secondaire (vertical par défaut) :

- stretch : les éléments sont étirés sur tout l'axe (valeur par défaut)
- flex-start : alignés au début
- flex-end : alignés à la fin
- center : alignés au centre
- baseline : alignés sur la ligne de base (semblable à flex-start)

Exemple :

```
#conteneur
{
   display: flex;
   justify-content: center;
   align-items: center;
}
```



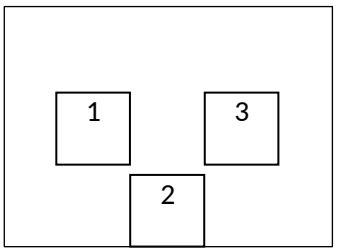
Rq : Le centrage vertical et horizontal peut être obtenu encore plus facilement avec le code ci-dessous.

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

align-self : permet de faire une exception d'alignement sur l'axe secondaire pour un seul des éléments

```
#conteneur
{
   display: flex;
   flex-direction: row;
   justify-content: center;
   align-items: center;
}

.element:nth-child(2) /* On prend le deuxième bloc élément */
{
   align-self: flex-end; /* Seul ce bloc sera aligné à la fin */
}
```



4) Répartir plusieurs lignes d'éléments

align-content : permet de choisir comment seront reparti les lignes d'élément, lorsqu'il y a trop d'élément pour qu'ils tiennent sur une ligne (cette propriété n'a aucun effet si tous les éléments tiennent sur une seule ligne) :

- flex-start : les éléments sont placés au début
- flex-end : les éléments sont placés à la fin
- center : les éléments sont placés au centre
- space-between : les éléments sont séparés avec de l'espace entre eux
- space-around : idem, mais il y a aussi de l'espace au début et à la fin
- stretch (par défaut) : les éléments s'étirent pour occuper tout l'espace

Rappel : lorsqu'il y a trop d'éléments pour une seule ligne, penser à utiliser la propriété flex-wrap: wrap

5) Ordonner les éléments

order : permet de modifier l'ordre des éléments dans CSS sans changer le code HTML.

Exemple : placer le premier élément en 3ème position, le second en 1ère position et le troisième en 2nde position.

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

6) Faire grossir ou maigrir les éléments

flex : permet de grossir un élément pour occuper tout l'espace restant.

Rq : flex est en fait une super-propriété qui combine flex-grow (capacité à grossir), flex-shrink (capacité à maigrir) et flex-basis (taille par défaut).

Exemple :

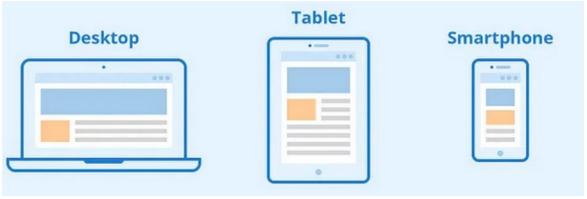
```
.element:nth-child(2)
{
   flex: 1; /* Le second élément s'étire pour prendre tout l'espace*/
}
```

Le nombre de la propriété flex indique dans quelle mesure il peut grossir par rapport aux autres.

```
.element:nth-child(1)
{
   flex: 2;
}
.element:nth-child(2)
{
   flex: 1; /* le premier élément peut grossir 2 fois plus que le second élément */
}
```

IV) Responsive design

Le responsive design sert à adapter un site à toutes les tailles d'écran (téléphone, tablette, ordinateur).



La stratégie **mobile first** consiste à faire le site d'abord pour téléphone, puis à l'adapter aux écrans plus grands.

On commence par le style pour téléphone, sans media query :

```
body {
 font-size: 14px;
}
```

```
.container {
 display: block;
}
```

Puis on utilise les media queries pour faire des ajustement pour les tablettes :

```
@media (min-width: 769px) {
  body {
    font-size: 16px;
  }
  .container {
    display: flex;
  }
}
```

Pour finir, on fait la mêm chose pour les ordinateurs :

```
@media (min-width: 1025px) {
  body {
    font-size: 18px;
  }
}
```

Bonnes pratiques :

- Largeurs en % ou vw.
- Images avec `max-width: 100%`.
- Utiliser Flexbox ou Grid

V) La mise en page avec des techniques plus anciennes

Aujourd'hui, Flexbox est un nouvel outil de mise en page recommandé. Mais pour comprendre les vieux codes, il peut être utile de connaitre les techniques plus anciennes qui ont l'avantage d'être reconnues par les vieilles versions des navigateurs.

1) Le positionnement flottant

En associant la propriété float au bloc <nav> et la propriété margin-left au bloc <section>, on peut obtenir le menu de navigation sur le coté et la section sur le reste de la page.

```
nav
{
   float: left;
}

section
{
   margin-left: 170px;
}
```

Rq : si on souhaite que la section se place obligatoirement sous le menu, on utilisera `clear: both`, qui oblige la suite du texte à se positionner sous l'élément flottant.

2) Transformation des éléments avec display

Display : permet de transformer n'importe quel élément de votre page d'un type vers un autre.

- inline : éléments d'une ligne - se placent les uns à côté des autres (<a>,,, ...)
- block : éléments en forme de blocs - se placent les uns en-dessous des autres et peuvent être redimensionnés (<p>, ,<section>…)
- inline-block : éléments positionnés les uns à côté des autres (comme les inlines) mais qui peuvent être redimensionnés (comme les blocs). <select>,<input>
- none : éléments non affichés

Exemple : imposer aux liens (originellement de type inline) d'apparaître sous forme de blocs. Les liens vont se positionner les uns en-dessous des autres (comme des blocs) et il devient possible de modifier leurs dimensions.

```
a
{
   display: block;
}
```

vertical-align : permet de modifier l'alignement vertical des éléments.

- baseline: aligne de la base de l'élément avec celle de l'élément parent (par défaut) ;
- top: aligne en haut ;
- middle: centre verticalement ;
- bottom: aligne en bas ;
- (valeur en px ou %) : aligne à une certaine distance de la ligne de base (baseline).

```
nav
{
   display: inline-block;
   vertical-align: top;
}

section
{
   display: inline-block;
   vertical-align: top;
}
```

Rq : vous noterez que le corps <section> ne prend pas toute la largeur ? Ce n'est plus un bloc, il occupe seulement la place dont il a besoin. Si cela ne convient pas on utilise width.

3) Le positionnement absolu, fixe et relatif

position : permet de déterminer le positionnement d'un élément

- absolute: positionnement absolu permet de placer un élément n'importe où sur la page (en haut à gauche, en bas à droite, tout au centre, etc.).
- fixed: positionnement fixe, identique au positionnement absolu mais, l'élément reste toujours visible, même si on descend plus bas dans la page
- relative: positionnement relatif permet de décaler l'élément par rapport à sa position normale

Rq : ces différents positionnements s'utilisent le plus souvent sur les balises block, mais ils fonctionnent aussi sur des balises de type inline.

position: absolute;

- left: position par rapport à la gauche de la page (en px ou en %)
- right: position par rapport à la droite de la page (en px ou en %)
- top: position par rapport au haut de la page (en px ou en %)
- bottom: position par rapport au bas de la page (en px ou en %)

Rq : pour utiliser position: absolute, il faut préciser au moins une des quatre propriétés : top, left, right ou bottom.

Exemple : le bloc est positionné tout en bas à droite

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

Attention : si vous placez deux éléments en absolu vers le même endroit, ils risquent de se chevaucher. Dans ce cas, utilisez la propriété z-index (en lui assignant un nombre). L'élément ayant la valeur de z-index la plus élevée sera au-dessus des autres.

Exemple : l'élément 2 sera au-dessus de l'élément 1

```
element
{
   position: absolute;
   right: 0px;
   bottom: 0px;
   z-index: 1;
}
element2
{
   position: absolute;
   right: 30px;
   bottom: 30px;
   z-index: 2;
}
```

Remarque : le positionnement absolu ne se fait pas forcément par rapport au coin en haut à gauche de la fenêtre. Si un bloc A, positionné en absolu, se trouve dans un autre bloc B, lui-même positionné en absolu (ou fixe ou relatif), alors votre bloc A se positionnera par rapport au coin supérieur gauche du bloc B.

position: relative;

Exemple :

```
strong
{
  background-color: red; /* Fond rouge */
  color: yellow; /* Texte de couleur jaune
*/
}
```

En appliquant la position: relative

```
strong
{
  background-color: red;
  color: yellow;

  position: relative;
  left: 55px;
  top: 10px;
}
```