

Formation CSS avancé et Responsive Design

M2I - Formation - 2023

Donjon Audrey

Module CSS 3 et Responsive Design



Introduction

Points principaux qui seront abordés :

- Rappel sur les CSS
- Les écrans
- HTML
- Les modes de positionnement
- Responsive Design
- Les transitions et animations
- Cas particuliers

Outils utilisés:

- VSC (Visual Studio Code) : Éditeur de texte
- Navigateurs : Firefox, Chrome, Edge...
- Recherche internet / Support de cours

Rappel sur les CSS



[1 - Intro](#)

[2 - Rappel](#)

[3 - Sélecteurs](#)

[4 - Combinateurs](#)

[5 - Disposition des boîtes](#)

[6 - Propriétés des boîtes](#)

[7 - Dimensions des boîtes](#)

[8 - Flux et positionnement](#)

[9 - Unités de mesures](#)

Introduction

- ▶ Naissance du **CSS** pour accompagner le HTML fin 1996.
- ▶ Le **CSS** (Cascading Style Sheets : feuille de style en cascade) est un langage de mise en forme.
- ▶ **HTML** affiche et structure les données, **CSS** les met en forme (positionnement, couleurs, polices, etc...).
- ▶ Il est possible d'utiliser le CSS de trois manières différentes :
 - ▶ Dans un fichier externe avec l'extension **.css** avec un lien dans le **<head>** du fichier **.html** (méthode la plus courante):

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

- ▶ Directement dans le code HTML (partie **<head>**) avec la balise **<style>** **</style>** (méthode à éviter autant que possible).
- ▶ Directement dans le code HTML sur l'élément à styliser (méthode à éviter autant que possible) :

```
> <h1 color="red">Titre de niveau 1</h1>
```

```
selecteur{  
  propriété : valeur ;  
}
```

Ici, le **sélecteur**, c'est l'**élément HTML ciblé**, exemple :

CSS

```
p{/*Sélecteur*/  
  color:red;  
  /*Propriété : valeur;*/  
}
```

Sélecteurs CSS

(sert à appliquer des propriétés CSS sur les éléments HTML que l'on sélectionne)

<p>Sélecteur d'élément ou balise HTML</p> <p>Ici : Touchera toutes les balises <code><p></p></code> de la page web</p>	<div>CSS</div> <pre>p{ color: red; }</pre>
<p>Sélecteur de class</p> <p>Ici : Touchera la/les balises ayant la class <code>"text-red"</code></p>	<div>CSS</div> <pre>.text_red{ color: red; }</pre>
<p>Sélecteur d'ID</p> <p>Ici : Touchera la balise ayant l'id <code>"color_main_title"</code></p>	<div>CSS</div> <pre>#color_main_title{ color: brown; }</pre>
<p>Sélecteurs universels</p> <p>Ici : Touchera tous les éléments HTML de la page</p>	<div>CSS</div> <pre>*{ color: red; }</pre>
<p>Sélecteur d'attribut</p> <p>Ici : Touchera toutes les balises <code><a></code> ayant comme attribut : <code>href="https://google.com"</code></p>	<div>CSS</div> <pre>a[href="https://google.com"]{ color: green; }</pre>

<p>Sélection en liste</p> <p>Ici : Touchera toutes les balises <code><p></p></code> et la/les balises ayant la class <code>"text-red"</code>.</p>	<p>CSS</p> <pre>p, .text_red { color: ■ red; }</pre>
<p>Sélection des enfants directs</p> <p>Ici : Touchera toutes les balises <code><p></p></code> qui sont enfants directs d'une balise <code><div></div></code>.</p>	<p>CSS</p> <pre>div > p{ color: ■ green; }</pre>
<p>Sélection des enfants à n'importe quel niveau</p> <p>Ici : Touchera toutes les balises <code><p></p></code> qui sont descendant d'une balise <code><div></div>..</code></p>	<p>CSS</p> <pre>div p{ color: ■ blue; }</pre>
<p>Selection du voisin direct</p> <p>Ici : Touchera toutes les balises <code><p></p></code> qui sont voisines d'une balise <code><div></div>..</code></p>	<p>CSS</p> <pre>div + p{ color: ■ brown; }</pre>
<p>Autres...</p> <p>Entraînement</p>	<p>https://www.w3.org/TR/selectors-3/#combinators</p> <p>https://flukeout.github.io/</p>

Disposition des boîtes

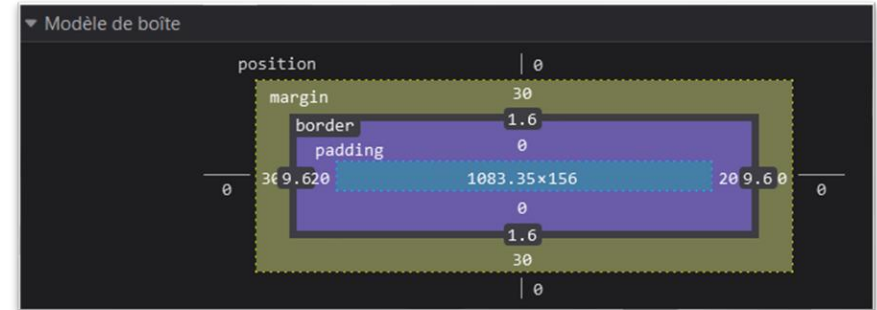
Modèle de boîte : tout élément est inclus dans une boîte, cela aide à comprendre la mise en page CSS et le positionnement des éléments d'une page HTML. En HTML tous les éléments ont un de ces 3 types : **block**, **inline** ou **inline-block**. Le type peut être changé avec la propriété css **display**.

- Élément HTML de type **block**
 - **Occupe toute la largeur** disponible de son parent
 - **L'élément suivant ou précédent** l'élément block se positionnera **au dessus ou en dessous de celui-ci** (retour à la ligne)
 - Width & Height modifiables

Exemple de Type block générique: `<h1>...<h6>` / `<p>` / `<div>`

- Élément HTML de type **inline**
 - **Occupe la place de leurs contenu**
 - L'élément inline suivant se **positionne à la suite**
 - **Width & Height non modifiables**
- et pas de **padding et margin (top et bottom)**

Exemple de Type inline générique: `<a>` / ``



- Élément HTML de type **inline-block**
 - Se comporte **en partie comme un inline** :
Occupe la place de leurs contenu / L'élément inline suivant se **positionne à la suite**
 - Se comporte **en partie comme un block** :
Width & Height modifiables et padding et margin (top et bottom) possible

Exemple de Type inline-block générique: `<button>`



Propriétés des Boîtes

Le modèle de boîte consiste à considérer qu'en HTML tous élément est inclus dans une boîte (dans une div par exemple).
Chaque boîte possède **des propriétés qui peuvent être changées** :

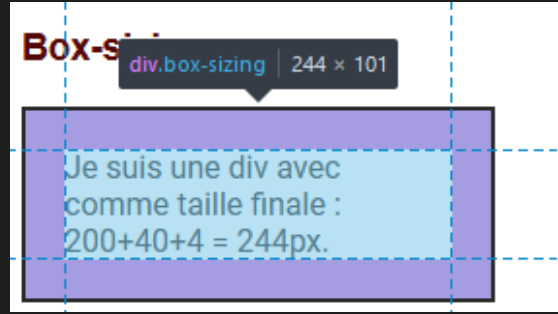
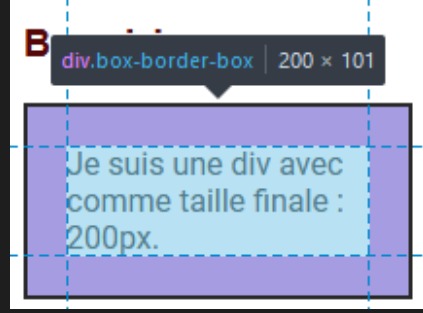
Propriété	Syntaxe	Exemples
Une dimension (une largeur et une hauteur)	width height	<code>width: 250px; height: 300px;</code>
Une marge extérieure (utilisé pour créer de l'espacement entre éléments)	margin	<code>margin: 5px 5px 5px 5px; margin: 5px 5px 5px; margin: 5px 5px; margin: 5px; margin-top: 10px; margin-bottom: 30px; margin-left: 10px; margin-right: 30px;</code>
Une marge intérieure (utilisé pour créer de l'espacement au sein même de l'élément)	padding	<code>padding: 5px 5px 5px 5px; padding: 5px 5px 5px; padding: 5px 5px; padding: 5px; padding-top: 10px; padding-bottom: 30px; padding-left: 10px; padding-right: 30px;</code>
Une bordure	border border-radius	<code>border: #267c27 5px solid; border-radius: 25px;</code>
Une ombre	box-shadow	<code>box-shadow: 0px 0px 15px 2px #000000;</code>
Une taille minimum et maximum	min-width min-height max-width max-height	<code>min-width: 150px; min-height: 400px max-width: 600px; max-height: 400px;</code>

Propriétés des Boîtes

Propriété	Syntaxe	Exemples
Centrage d'élément de type block	margin:auto;	<pre>/* Cet élément block sera centré dans son parent */ .box-center-auto{ width: 200px; margin: auto; border: #0A1A23 solid 2px; }</pre>
Faire disparaître un élément	display:none;	<pre>/* Cet élément n'apparaîtra pas à l'écran */ .box-none{ display: none; width: 200px; padding:10px; border: #0A1A23 solid 2px; }</pre>

Dimensionnement des Boîtes

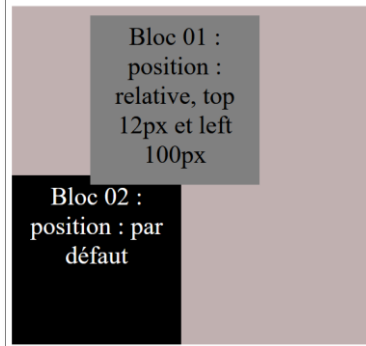
Par défaut tous les éléments ont le type de modèle de boîte « **content-box** », ce qui signifie que la **taille finale** d'un élément sera égale à **sa taille initiale** + la **largeur de ses bordures** + la **largeur de ses marges intérieures**.

Propriété	Exemples
<p>Box-sizing : content-box; Taille finale de l'élément = taille initiale + largeurs bordures + marges intérieures</p>	<pre>/* Dans cet exemple, la largeur définitive totale de l'élément sera 200 + 2x20 + 2x2 = 244 pixels de large */ .box-sizing{ width: 200px; padding: 20px; border: 2px solid black; }</pre> 
<p>Box-sizing : border-box; Taille finale de l'élément = taille initiale (les marges et bordures sont comptés dans la taille définie)</p>	<pre>/* Dans cet autre exemple, la largeur définitive totale de l'élément sera 200 pixels de large */ /* (les marges et bordures sont comptées dans la taille définie avec le modèle border-box) */ .box-border-box{ box-sizing: border-box; width: 200px; padding: 20px; border: 2px solid black; }</pre> 

Flux et Positionnement

- ▶ Le **flux HTML** correspond à l'écoulement des informations (ou données) que le navigateur va interpréter et afficher. **Un navigateur commence par le haut de la page**, place les éléments HTML qu'il rencontre les uns à la suite des autres, **de gauche à droite** puis **de haut en bas**, à ceci près que cela dépend si ce sont des balises bloc ou en-ligne.
- ▶ La propriété CSS **position** permet de **gérer le positionnement** des éléments.
- ▶ On distingue **4 types principaux** de positionnement :
 - ▶ **(static)**: Valeur par défaut - Positionnement **statique** des éléments **dans le flux**
 - ▶ **relative** : Positionnement ajustée par rapport à la **position initiale** de l'élément **dans le flux**
 - ▶ **absolute** : Positionnement ajustée par rapport au **parent relative** le plus proche **hors flux** (chevauchement d'éléments alors possible)
 - ▶ **fixed** : **semblable a absolute**, mais en s'appuyant sur la **fenêtre visible** du navigateur et **hors flux**.
- ▶ **Positionnement ajustée avec les propriétés top, left, bottom et right**. Les valeurs négatives sont possibles.
- ▶ La propriété **z-index** permet de prioriser l'élément à afficher (**en cas de chevauchement**), si la valeur est la plus élevée par rapport aux autres, l'élément sera au dessus des autres.

Positionnement relatif



Positionnement absolue



Unités de mesures **absolues** et **relatives**

Pour beaucoup de **propriétés CSS**, il est possible de **définir des tailles**. Plusieurs unités de mesures sont à notre disposition mais toutes ne sont pas forcément les plus adaptées à certaines situations.

Il existe **2 catégories d'unités de mesures** :

Unités de mesures absolues (il n'y a pas tout) :

Les valeurs absolues sont des **valeurs fixes**. Le plus souvent on s'en sert pour **fixer des marges/bordures** afin que ces éléments restent les mêmes tout le temps.

- **px** = pixels
- **cm** = centimètres
- **mm** = millimètres

Unités de mesures relatives (il n'y a pas tout) :

Les valeurs relatives sont **relatives** à un autre élément ou à la taille de la fenêtre. Le plus souvent on s'en sert **pour définir la taille d'un texte**, l'unité de mesure recommandée est "rem" : ainsi la taille des textes est directement modifiable par les utilisateurs via leurs préférences enregistrées dans le navigateur.

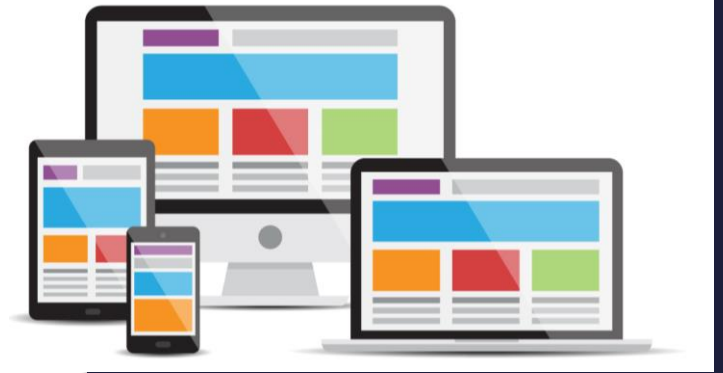
- **%** = pourcentage
- **rem** = unité relative à la taille des caractères définie dans la page sur la balise "html" ou sur ":root" (4rem = 4 fois plus gros que la taille des caractères)
- **em** = unité relative à la taille des caractères de l'élément parent (3em = 3 fois plus gros que la taille des caractères du parents)
- **vw** = unité relative (en pourcentage) à la largeur de la fenêtre du navigateur. (50vw = la moitié de la largeur de la fenêtre du navigateur)
- **vh** = unité relative (en pourcentage) à la hauteur de la fenêtre du navigateur. (200vh = le double de la hauteur de la fenêtre du navigateur)

Pour définir les dimensions des divs de **structuration d'une page**, il est **conseillé** d'utiliser l'**unité de mesure** "%", ainsi **les boîtes pourront s'adapter en fonction de la taille de la fenêtre du navigateur**. (dans certains cas les valeurs absolues en "px" peuvent être aussi utiles).

Attention : Une page web n'a pas de hauteur fixe contrairement à sa largeur. On ne peut donc pas fixer une hauteur à un élément en pourcentage, sauf si son parent possède une hauteur fixe absolues (en pixels par exemple).

Les écrans

- 1 - Taille, résolution et valeur du pixel
- 2 - Pixel et sous Pixel
- 3 - CSS Pixel, Device Pixel, Pixel Ratio et HiDPI



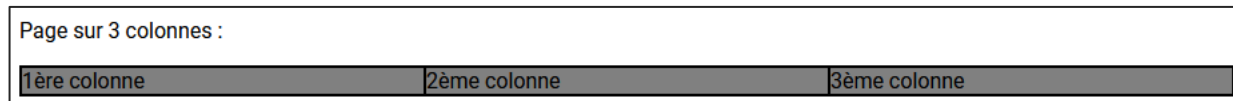
Taille, résolution et valeur du pixel

Les **informations sur les écrans**, telles que la **taille**, la **résolution**, la **valeur du pixel**, etc., sont très utiles pour déterminer comment un site ou une application web sera affichée sur différents appareils.

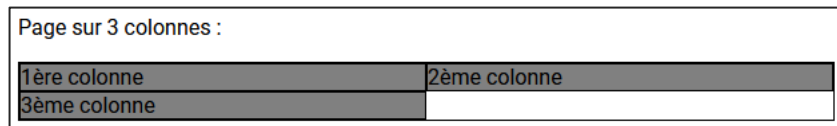
1 - Taille de l'écran : En connaissant les différentes tailles d'écran que l'on va devoir concevoir, on peut utiliser le CSS pour ajuster la disposition et les éléments de d'un site web pour qu'ils s'affichent correctement sur chaque appareil. Par exemple, on peut choisir de rendre certains éléments plus petits sur un écran de téléphone mobile, tout en les affichant plus grand sur un écran de bureau.

Exemple :

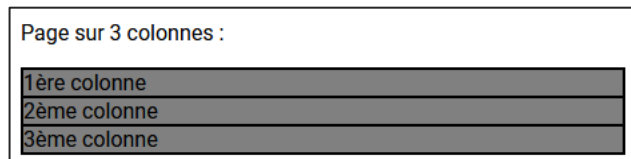
Page de largeur supérieur ou égal à 900px



Page de largeur se situant entre 600px et 900px



Page de largeur inférieur à 600px



Taille, résolution et valeur du pixel

2 - Résolution de l'écran : Savoir comment les différentes résolutions affectent l'affichage d'un site web peut aider à choisir la bonne taille pour les images et autres éléments graphiques. On peut utiliser le CSS pour servir des images de différentes résolutions en fonction de l'appareil de l'utilisateur.

Exemple :

Sur un écran avec une haute résolution, vous pouvez vouloir servir une image de haute qualité pour profiter de la résolution supplémentaire. Vous pouvez le faire en utilisant l'attribut **srcset** dans une balise ****.

CSS

```

```

low-res.jpg est une **version basse résolution** de l'image, qui sera chargée par défaut. C'est l'image qui est chargée si le navigateur ne prend pas en charge **srcset**, ou si aucune des options de srcset n'est pas appropriée.

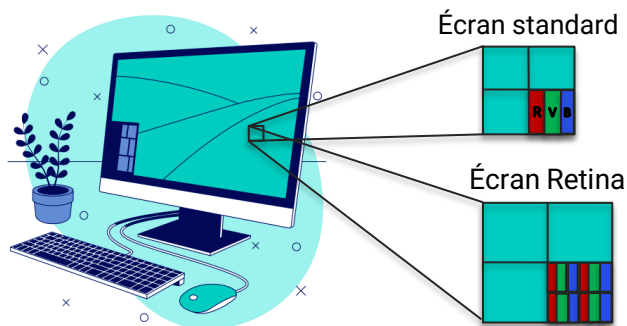
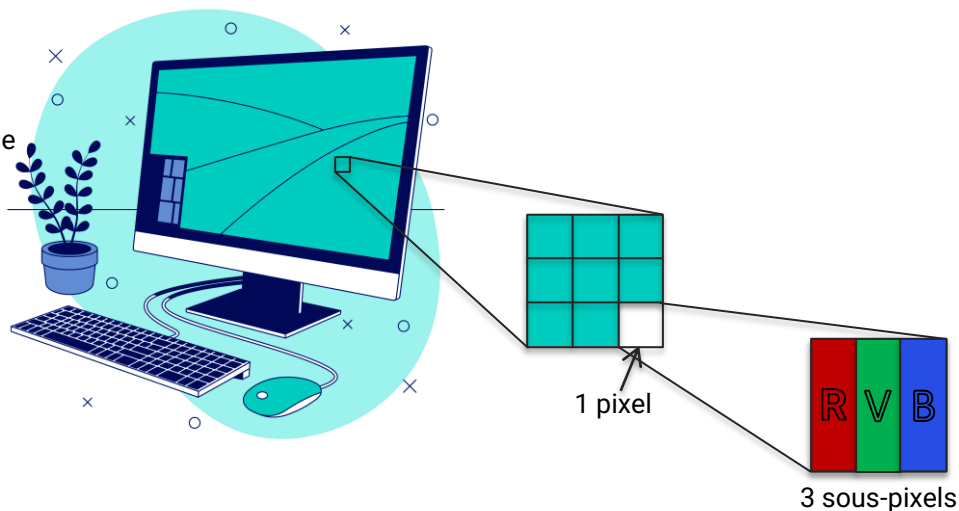
L'**attribut 'srcset'** fournit au navigateur un ensemble d'images à choisir en fonction des conditions du périphérique.

high-res.jpg est une **version haute résolution** de l'image, qui sera **utilisée lorsque le navigateur détecte que l'écran a une densité de pixels de 2x ou plus** (typique des écrans "Retina" ou "HiDPI"). Cela permet **d'afficher une image plus nette sur ces écrans à haute résolution**.

Pixel et sous Pixel

3 – Valeur du pixel : Lorsque l'on regarde un écran, on voit une image composée de tout petits points lumineux appelés pixels. Chaque pixel est en fait constitué de trois petites lumières de couleurs différentes : rouge, vert et bleu. En changeant la luminosité de ces trois lumières, on peut faire toutes les couleurs.

Quand vous créez une page web et que vous dites, par exemple, "Je veux une image qui fait 100 pixels de large", le navigateur va utiliser 100 de ces petits points lumineux pour afficher votre image.



Cependant, tous les écrans ne sont pas pareils. Certains écrans, comme ceux des **iPhones** ou des **MacBooks d'Apple**, ont beaucoup plus de ces petits points lumineux dans le même espace. Ils sont appelés écrans à **haute densité de pixels** ou écrans **Retina**. Sur ces écrans, un pixel sur votre page web peut en fait être plusieurs petits points lumineux sur l'écran. Cela permet à ces écrans d'afficher des images plus nettes et plus détaillées.

C'est là qu'interviennent le "CSS pixel", le "Device pixel", le "Pixel Ratio" et le "HiDPI". Ils décrivent comment les navigateurs et les écrans travaillent ensemble pour afficher une page web de la meilleure façon possible, peu importe le type d'écran.

CSS Pixel, Device Pixel, Pixel Ratio et HiDPI

3 - a) CSS Pixel / Device Pixel : Un pixel CSS n'est pas nécessairement un pixel physique sur l'écran. En fait, un pixel CSS est une unité de mesure abstraite utilisée dans les feuilles de style CSS pour le rendu et le formatage de la page Web. Un pixel de périphérique, en revanche, est un pixel physique sur un écran. La relation entre les pixels CSS et les pixels de périphérique peut varier en fonction de la densité de pixels de l'écran et du niveau de zoom du navigateur.

CSS Pixel : Sur un écran à faible densité de pixels, un pixel CSS peut correspondre à un pixel de l'appareil. Mais sur un écran à haute densité de pixels (comme les écrans Retina), un pixel CSS peut être représenté par plusieurs pixels de l'appareil.

Device Pixel : Les pixels de l'appareil, en revanche, sont les vrais points lumineux physiques sur votre écran. Chaque pixel de l'appareil est composé de trois sous-pixels (rouge, vert et bleu) qui peuvent produire différentes couleurs.

La relation entre ces deux types de pixels est définie par le ratio de pixels de l'appareil, qui est simplement le nombre de pixels de l'appareil qui correspondent à un pixel CSS. Sur un écran standard.

3 - b) Pixel Ratio : Le ratio de pixels est le nombre de pixels de périphérique par pixel CSS. Les écrans à haute densité, tels que les écrans Retina d'Apple, ont un ratio de pixels supérieur à 1. Par exemple, un ratio de pixels de 2 signifie qu'il y a $2 \times 2 = 4$ pixels de périphérique pour chaque pixel CSS. Les termes hdpi, xhdpi, xxhdpi, etc., sont utilisés dans Android pour indiquer différentes densités de pixels : hdpi est 1.5, xhdpi est 2.0, xxhdpi est 3.0, et ainsi de suite.

Lorsque vous définissez une taille absolue exemple : width : 50px sur un élément, celui-ci fera toujours 50px de large peu importe la densité de pixels de l'écran.

3 - c) HiDPI : HiDPI (High Dots Per Inch) est un **terme utilisé pour décrire les écrans qui ont une densité de pixels plus élevée que l'habituel**. Un écran **HiDPI** peut afficher des graphiques et du texte **plus nets** car il a plus de pixels dans la même zone physique. **Pour rendre correctement une page Web sur un écran HiDPI**, on peut utiliser des **images à résolution plus élevée**, ou utiliser des **techniques CSS** pour s'assurer que le texte et les autres éléments de la page sont affichés à la bonne taille (c'est-à-dire l'utilisation **d'unités relatives**, de **media queries sur les résolutions**, de la **meta viewport** et des **images vectorielles** comme SVG).

Pratique :

Énoncé :

un dossier contenant le **fichier html**, le **dossier et fichier css** ainsi que le **dossier et fichiers images** vous **seront fournis**, vous devrez **ouvrir le fichier index.html sur le navigateur** pour prendre connaissance de ce qui sera demandé lors de cet exercice. Vous devrez **ajouter le code nécessaire** dans le fichier **styles.css** et pour le **bonus** il sera nécessaire de rajouter du code dans le **fichier index.html**.



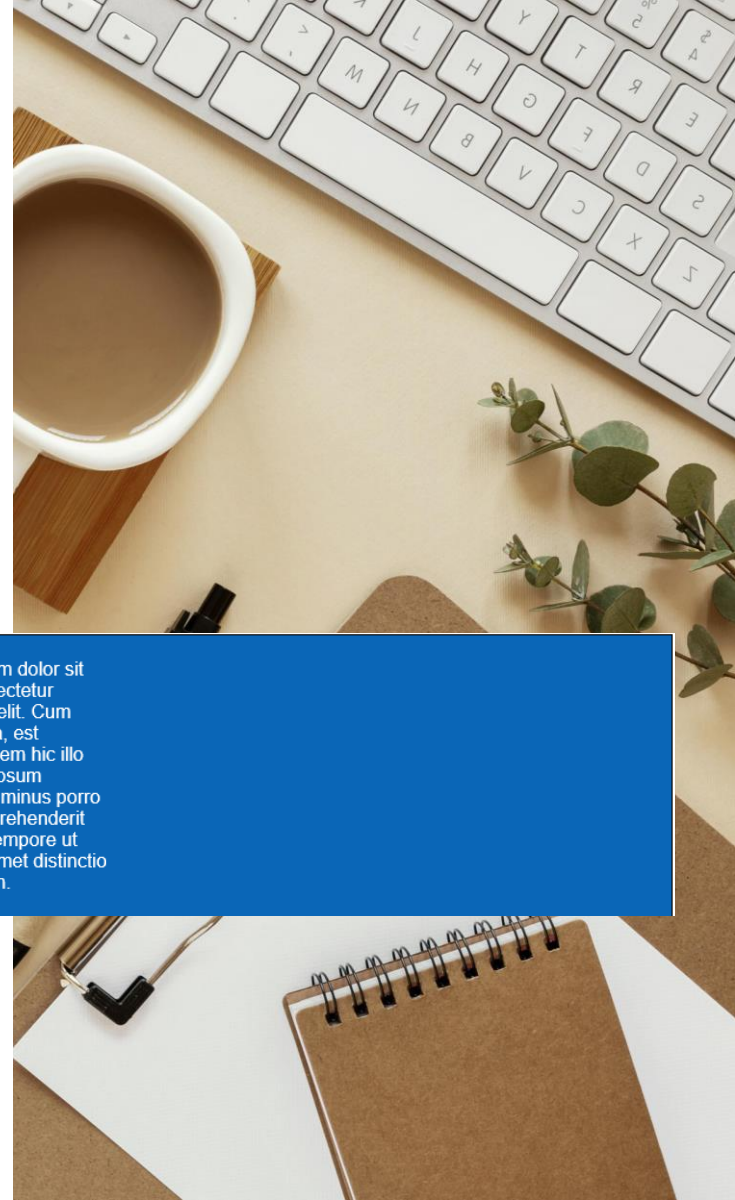
Pratique :

1 - Alignement et dimensionnement d'éléments block :

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cum debitis dicta, est exercitationem hic illo inventore, ipsum laudantium minus porro quaerat reprehenderit similique, tempore ut voluptas. Amet distinctio incidunt rem.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cum debitis dicta, est exercitationem hic illo inventore, ipsum laudantium minus porro quaerat reprehenderit similique, tempore ut voluptas. Amet distinctio incidunt rem.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cum debitis dicta, est exercitationem hic illo inventore, ipsum laudantium minus porro quaerat reprehenderit similique, tempore ut voluptas. Amet distinctio incidunt rem.



Pratique :

2 - Alignement et dimensionnement d'éléments inline :

[Lien 01](#)

[Lien 02](#)

[Lien 03](#)

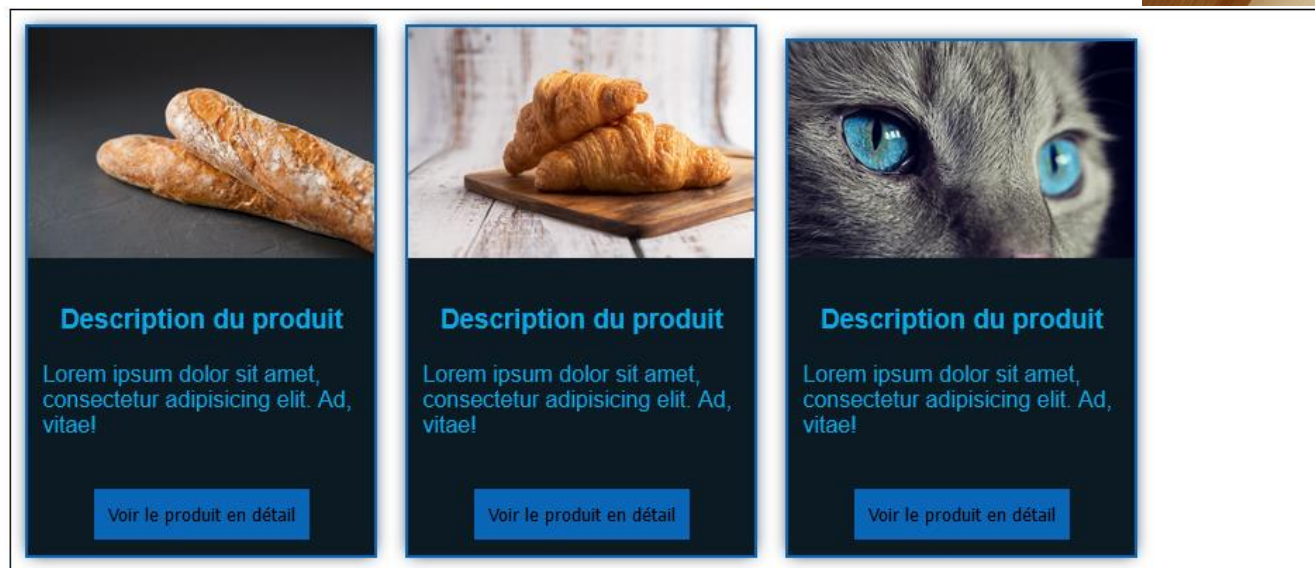
[Lien 04](#)

[Lien 05](#)



Pratique :

3 - Alignement et dimensionnement d'éléments :



HTML : Rappel



- 1 - Structure de la page
- 2 - Gestion du viewport
- 3 - @import

Structure de la page

L'ossature HTML de la page est essentielle car elle donne un sens à au contenu et permet aux moteurs de recherche et aux technologies d'assistance de comprendre notre contenu. Une structure de page HTML typique pourrait ressembler à cela :

html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Titre de la page</title>
  </head>
  <body>
    <header>
      <!-- Navigation et titre du site -->
    </header>
    <main>
      <!-- Contenu principal de la page -->
    </main>
    <footer>
      <!-- Pied de page, liens supplémentaires -->
    </footer>
  </body>
</html>
```

Structure **sémantique** de la page

Pour construire visuellement une page web, on utilise massivement les boîtes comme la `<div></div>`.

Depuis HTML5, un certain nombre de nouvelles balises à été créé pour apporter de la sémantique à certaines boîtes importantes. **Elles ont le même comportement et la même utilité que la div mais avec de la sémantique en plus.**

<code><header></header></code>	Conteneur des éléments d'en-tête des contenus (titre par exemple)
<code><nav></nav></code>	Conteneur des menus de navigation avec des liens
<code><main></main></code>	Conteneur du contenu majoritaire de la page
<code><section></section></code>	Conteneur d'une section générique de la page (si le contenu est indépendant du contexte, plutôt utiliser <code><article></code>)
<code><article></article></code>	Conteneur ayant du contenu indépendant du reste de la page
<code><aside></aside></code>	Conteneur des blocs un peu à part sur la page dont le contenu n'a qu'un rapport indirect avec le contenu principal de la page (comme les sidebars sur le côté)
<code><footer></footer></code>	Conteneur des pieds de page des éléments (page, article, etc...)

Gestion du **viewport**

Le **viewport** est la **zone de la fenêtre du navigateur** où votre page web s'affiche. Sur les appareils mobiles et les ordinateurs de bureau, la taille du viewport peut varier considérablement. Pour **vous assurer que votre design est responsive et s'affiche correctement sur tous les appareils**, vous pouvez utiliser la balise **meta viewport** :

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

Cette balise indique au navigateur de définir la **largeur du viewport à la largeur de l'écran** de l'appareil et **d'établir le niveau de zoom initial à 1**.

@import est une **règle css** utilisée pour **importer une feuille de style dans une autre**. Elle est souvent utilisée pour **structurer et organiser de grands projets CSS** en **séparant le code CSS en plusieurs fichiers**.

CSS

```
@import url('style2.css');
```

Dans mon exemple ceci importe le contenu de style2.css dans la feuille de style actuelle.

Attention : l'utilisation de **@import** a ses **inconvénients**. En particulier, **l'utilisation excessive de @import** peut **ralentir le chargement d'une page Web**, car chaque **@import** introduit une **requête HTTP supplémentaire**. Par conséquent, le **navigateur doit attendre que chaque fichier CSS soit téléchargé et analysé** avant de pouvoir continuer à construire l'arbre de rendu.

Pour cette raison, il est plutôt recommandé **d'éviter l'utilisation de @import** autant que possible et d'utiliser **d'autres méthodes pour organiser et structurer le code CSS**, comme les **préprocesseurs CSS (Sass, Less)** qui permettent de **diviser le code en plusieurs fichiers** lors du développement, mais qui sont ensuite **compilés en un seul fichier CSS pour la production**.

CSS : Les modes de positionnement



[1 - Flottement](#)

[2 - Boîtes Flexibles](#)

[3 - Grille](#)

Les éléments flottants

La propriété CSS **float** permet de **sortir un élément du flux** normal du HTML pour le placer à **gauche ou à droite** de son conteneur. Les éléments **inlines** présents dans le même conteneur (comme les **textes**) entoureront cet élément.

html

```
<h1>Dev front & dev back end</h1>
<section>
  <h2>Le développeur Front-End</h2>
  <p>Lorsque l'on parle de «Front-End», il s'agit finalement des
éléments du site que l'on voit à l'écran et avec lesquels on peut
interagir.....</p>
</section>
<section>
  <h2>Le développeur Back-End</h2>
  <p> Le
Back-End, c'est un peu comme la partie immergée de l'iceberg.
Elle est invisible pour les visiteurs, mais représente une grande
partie du développement d'un projet web. Sans elle, le site web
reste une coquille vide.....</p>
</section>
```

Css

```
.img-front{
  float:right;
}
.img-back{
  float:left;
}
```

DEV FRONT END & DEV BACK END

Le développeur Front-End

Lorsque l'on parle de «Front-End», il s'agit finalement des éléments du site que l'on voit à l'écran et avec lesquels on peut interagir. Ces éléments sont composés de HTML, CSS et de Javascript contrôlés par le navigateur web de l'utilisateur. Les champs de compétence du Front-End peuvent être séparé en deux : - Le design - Le développement HTML, CSS, Javascript. Le design est traditionnellement réalisé par un web designer qui produit des maquettes graphiques à l'aide de Photoshop ou Fireworks. Cependant, de plus en plus de web designers ont franchi la barrière et savent coder en HTML et CSS. Dans certains cas ils sont aussi capables de produire du Javascript.



Le développeur Back-End



Le Back-End, c'est un peu comme la partie immergée de l'iceberg. Elle est invisible pour les visiteurs, mais représente une grande partie du développement d'un projet web. Sans elle, le site web reste une coquille vide. On peut décomposer le Back-End en trois parties essentielles : - Un serveur (ou hébergement web) - Une application (en l'occurrence le site web) - Une base de données (ou l'on stocke les données de l'application). Le serveur est comme un disque dur accessible 24 heures sur 24, sur lequel les pages du site web sont enregistrées.

Flexbox est un **modèle de boîte flexible** permettant de gérer le positionnement des éléments dans un conteneur d'une manière plus fluide qu'avec les modèles de boîte classique.

Pour qu'un conteneur devienne un conteneur flex, il faut changer son type.

Il y a 4 directions possibles pour **distribuer les enfants** d'un élément flex :

row : horizontale, c'est la valeur par défaut

column : verticale

row-reverse : horizontale inversé

column-reverse : verticale inversé

Il est possible de **gérer l'alignement sur l'axe principal** avec la propriété "**justify-content**" et sur **l'axe secondaire** avec la propriété "**align-items**" :

L'axe **principal** et **secondaire** varient en fonction du **paramétrage** de la **direction** de votre flexbox :

- Si "flex-direction" vaut "**row**" ou "**row-reverse**", alors l'axe **principal** sera **l'axe horizontal** et **l'axe secondaire** sera **l'axe vertical**.

- Si "flex-direction" vaut "**column**" ou "**column-reverse**", alors l'axe **principal** sera **l'axe vertical** et **l'axe secondaire** sera **l'axe horizontal**.

Les **valeurs possibles** pour les **alignements** sont les suivantes :

- **flex-start** : éléments positionnés au début de l'axe, c'est la valeur par défaut.

- **flex-end** : éléments positionnés à la fin de l'axe.

- **center** : éléments centrés au milieu de l'axe.

- **space-between** : les éléments sont espacés à égale distance sur tout l'axe (et sont collés aux "bords" de l'axe).

- **space-around** : les éléments sont espacés à égale distance sur tout l'axe (et ne sont pas collés aux "bords" de l'axe).

CSS

```
div{
  display: flex;
}
```

Css

```
div{
  display: flex;
  /* Tous les éléments enfants seront disposés en ligne
  côte à côte */
  flex-direction: row;
}
```

Css

```
div{
  display: flex;

  /* Tous les éléments enfants seront disposés en
  ligne côte à côte */
  flex-direction: row;

  /* Si il y a un manque de place, les éléments
  passeront à la ligne du dessous */
  flex-wrap: wrap;

  /* Les éléments seront centrés sur l'axe
  principal (sur l'axe horizontal dans cet exemple)
  */
  justify-content: center;

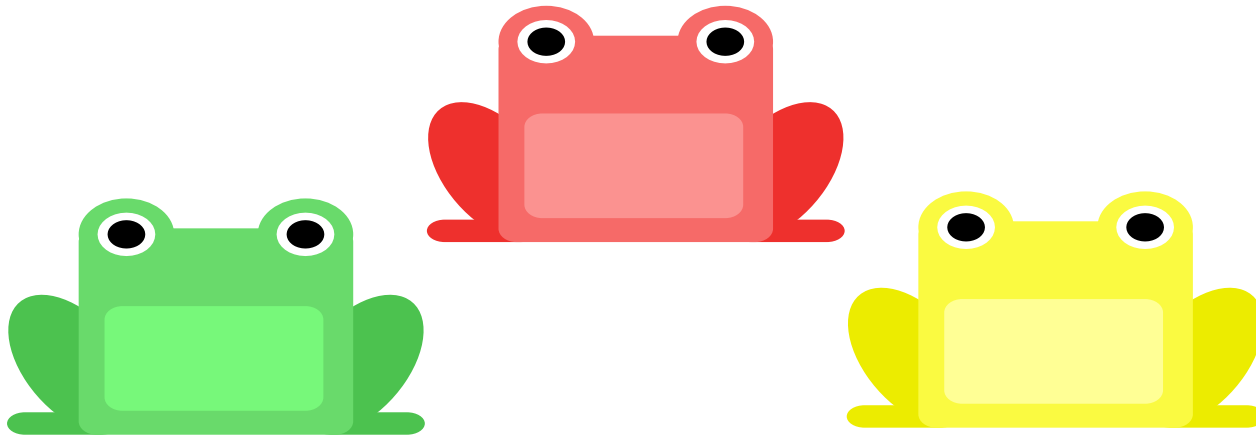
  /* Les éléments seront centrés sur l'axe
  secondaire (sur l'axe vertical dans cet exemple) */
  align-items: center;
}
```

Entraînement **Flexbox** : Jeux **Flexbox Froggy**

Outil pour tester les propriétés css qui existent :

<https://codepen.io/enxaneta/full/adLPwv/>

Lien vers le jeu : <https://flexboxfroggy.com/#fr>



Le **système de grille CSS**, également connu sous le nom de **CSS Grid Layout**, est une technique de **mise en page bidimensionnelle** qui nous permet de concevoir des mises en page complexes avec différentes sections.

Pour utiliser CSS Grid, on doit appliquer la propriété css **display: grid;** à **un conteneur**, ce qui fait de lui un "**grid container**".

Les éléments **enfants directs** de ce **conteneur** deviennent des "**grid items**" ce qui implique qu'ils pourront être placés sous un certain nombre de lignes et de colonnes.

Une fois la propriété grid en place, on peut alors utiliser d'autres propriétés rattachées au container pour définir différentes choses :

- Le **nombre de colonnes** de notre grille
- Le **nombre de rangées** de notre grille
- L'**espacement** entre colonnes et rangées de notre grille
- Les **zones** de notre grille avec leurs noms

Ensuite nous pourrions également utiliser d'autres propriétés qui seront elles rattachées aux enfants de notre grille pour pouvoir définir au besoin:

- Où **chacun de mes éléments commence et termine** horizontalement et verticalement
- Un **nom** à un élément
- Son **centrage verticale** au sein de sa cellule dans ma grille
- Son **centrage horizontal** au sein de sa cellule dans ma grille
- Un **ordre** différent de celui défini en html

CSS

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

CSS

```
.container{
  display: grid;
  grid-template-rows: 1fr 1fr 1fr ;
  grid-template-columns: 1fr 1fr 1fr ;
  grid-gap: 30px;
  grid-template-areas:
    'i1 i2 i3'
    'i4 i5'
  ;
}
```

CSS

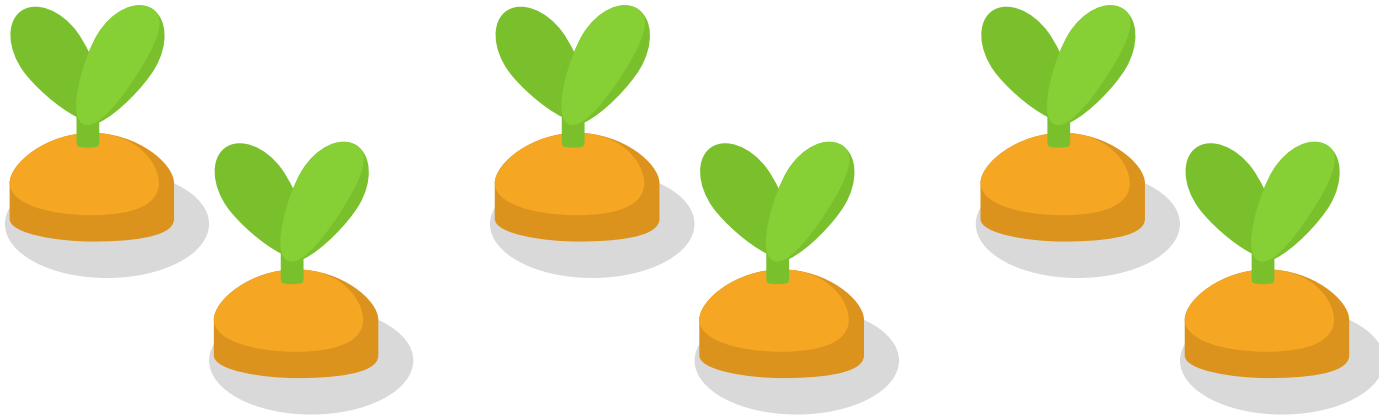
```
.item01{
  grid-area:i1;
  grid-row: 1/3;
  grid-column:1/2;
  align-self:center;
  justify-self:center;
  order:2;
}
.item02{
  order:1;
}
```

Entraînement Grille css : Jeux Grid Garden

Outil pour tester les propriétés de grille css:

https://grid.layoutit.com/?source=codeur-com-blog&utm_source=codeur-com-blog

Lien vers le jeu : <https://codepip.com/games/grid-garden/#fr>



Flexible Box Layout et Grid Layout

Flexbox et **CSS Grid** sont deux **systèmes de mise en page modernes** en CSS, mais ils servent des **objectifs différents** et ont des différences clés :

➤ **Flexbox** (Flexible Box Layout) :

- ❑ **Unidimensionnel** : Flexbox est principalement conçu pour la **mise en page** dans **une seule dimension**, soit **une ligne (horizontale)** soit **une colonne (verticale)**.
- ❑ **Flexibilité** : Il est idéal pour **aligner des éléments et distribuer l'espace le long d'une ligne ou d'une colonne**, même lorsque les tailles des éléments sont inconnues ou dynamiques.
- ❑ **Utilisation** : Très utile pour **créer des barres de navigation**, des **en-têtes**, des **pieds de page**, etc.

➤ **CSS Grid** (Grid Layout) :

- ❑ **Bidimensionnel** : CSS Grid est conçu pour la **mise en page bidimensionnelle**, où vous pouvez **contrôler à la fois les colonnes et les lignes**.
- ❑ **Contrôle complet** : Il offre un **contrôle plus précis** sur la **disposition** et permet des **conceptions plus complexes avec des zones** et des **alignements définis**.
- ❑ **Utilisation** : Idéal pour **créer des mises en page complexes** avec **différentes sections**, comme un **système de grille pour une page entière** ou une **mise en page de carte**.

En résumé, utilisez **Flexbox** lorsque vous travaillez **avec une seule dimension** (ligne ou colonne) et que vous avez **besoin de flexibilité**. Utilisez **CSS Grid** lorsque vous avez besoin de **contrôler à la fois les lignes et les colonnes** pour une **mise en page plus complexe**. Les deux **peuvent être utilisés ensemble**, et choisir l'un sur l'autre **dépend souvent des besoins spécifiques du projet**.

Pratique :

Énoncé :

Créer une page avec **une en-tête, une section principale** avec **trois articles contenant d texte en Flexbox** et un **pied de page** composer de 3 divs en **CSS Grid**.

Une base vous sera fournit et vous devrez rajouterle html et le css nécessaire pour être fidèle à la maquette.



Pratique : screen avec code couleur



Mon Site

Code couleur : #e9e9e9



Article 1 Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum.

Code couleur : #fff
code couleur bordure : ccc

Article 2 : Lorem ipsum dolor sit amet, consectetur adipisicing elit. Blanditiis cum delectus dolore illum impedit nesciunt non porro provident sapiente ullam? Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Blanditiis cum delectus dolore illum impedit nesciunt non porro provident sapiente ullam? Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum..

Article 3 : Lorem ipsum dolor sit amet, consectetur adipisicing elit. Assumenda dolorum esse facere fuga fugiat inventore minus, natus quibusdam, quis similique soluta sunt unde voluptatem? Aspernatur dolor dolore in odio possimus? Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Blanditiis cum delectus dolore illum impedit nesciunt non porro provident sapiente ullam? Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequatur, earum.

Contact

Code couleur : #333 et #fff

À propos

Mentions légales

Le responsive design & Les medias queries

- [1 - Le Responsive Design Web](#)
- [2 - Adaptation](#)
- [3 - Le Mobile First](#)
- [4 - Les media queries](#)
- [5 - Les breakpoints](#)
- [6 - Les différents types de medias queries](#)
- [7 - Cas particuliers](#)
- [8 - Combinaison de medias queries](#)



Le Responsive Web Design

Responsive Web Design désigne le fait qu'un site web est **construit de manière à le rendre compatible, lisible et utilisable sur différentes tailles d'écrans**.

Il existe **plusieurs techniques** permettant de faire du responsive. Une technique qui se faisait beaucoup était de **créer deux site distincts** : un pour desktop et un pour mobile. Cette technique a été très utilisée dans le début des années 2010.

Exemple avec Wikipédia :

Version Desktop : <https://fr.wikipedia.org/wiki/France>

Version mobile : <https://fr.m.wikipedia.org/wiki/France>

La **responsive web design** est une approche qui permet de créer des pages web qui sont toujours bien rendu quelque soit la taille du périphérique d'affichage.

1. Des contenus :

Adapter le contenu signifie **s'assurer que le texte, les images et d'autres éléments** s'affichent correctement sur différents types d'écrans.

Exemple :

Sur un site d'actualités, on pourrait choisir d'afficher seulement les titres des articles sur un petit écran, tandis que sur un écran plus grand, on montrerait le titre, l'image et un résumé.

2. Du positionnement :

Le positionnement concerne **la façon dont les éléments sont organisés sur la page**.

Exemple :

Sur un écran plus large, on pourrait avoir une disposition à trois colonnes avec une barre latérale, tandis que sur un écran plus petit, ces éléments se réorganisent en une seule colonne pour un meilleur affichage.

3. De la navigation :

Cela implique de repenser la façon dont les utilisateurs **navignent sur le site sur différents appareils**.

Exemple :

Sur un écran d'ordinateur, on pourrait avoir un menu de navigation horizontal complet, tandis que sur un téléphone portable, ce menu pourrait être remplacé par un menu déroulant ou un menu hamburger pour économiser de l'espace.

Mobile First est une stratégie visant à concevoir un site web **d'abord pour mobile**, puis ensuite à **utiliser les media queries** pour le **rendre bien sur les tailles d'écrans plus élevées**.

Cette stratégie **est très importante** car il est beaucoup plus facile de transformer un site web mobile avec des media queries que l'inverse !

Cette stratégie repose également sur **les concepts suivants** :

- ☐ Rester sur l'essentiel
- ☐ Ne rien ajouter de superflue
- ☐ Pouvoir accéder le plus rapidement et facilement possible aux infos
- ☐ Cibler les performances

html

```
<nav class="nav-main">
  <ul>
    <li><a href="">Bouton 1</a></li>
    <li><a href="">Bouton 2</a></li>
    <li><a href="">Bouton 3</a></li>
  </ul>
</nav>
```

Bouton 1

Bouton 2

Bouton 3

css

```
/*Mobile first*/
nav.nav-main{
    background-color: #030336;
}

nav.nav-main ul{
    padding: 0;
    margin: 0;
}

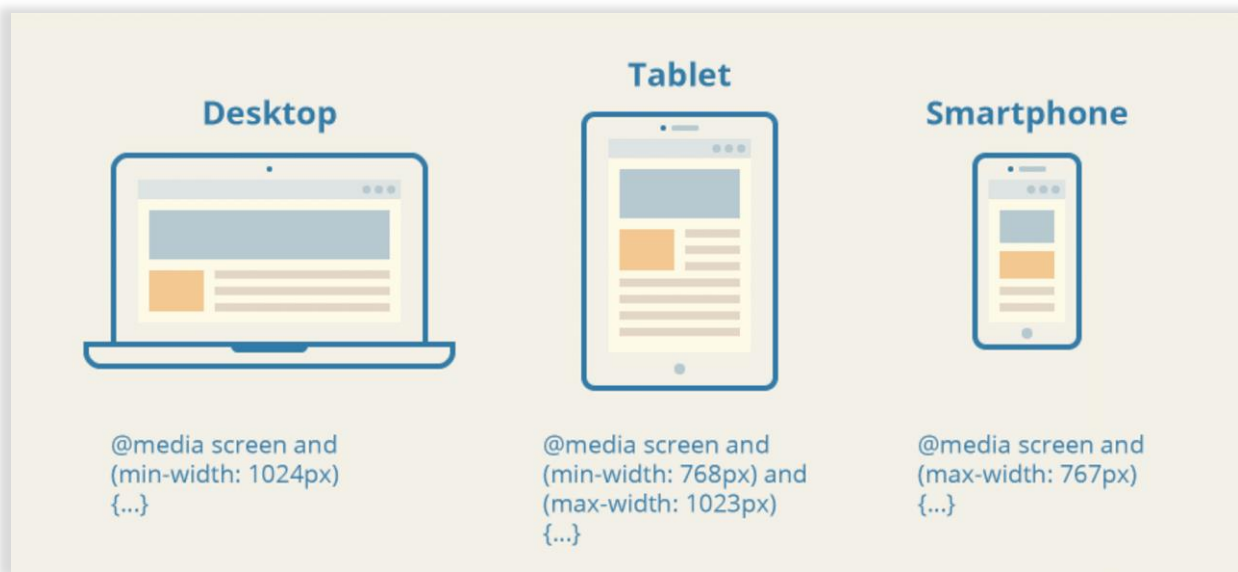
nav.nav-main ul li{
    list-style : none;
    border : solid 1px #0a78be;
    width: 100%;
    box-sizing: border-box;
    display: flex;
    flex-direction: column;
    align-items: center;
}

nav.nav-main ul li a{
    text-decoration: none;
    color: #0a78be;
    display: block;
    padding: 10px;
}
```

Les media queries

En CSS nous allons mettre en place des sites responsives grâce aux **média queries**.

Les **media queries** sont des **conditions CSS** permettant d'appliquer des styles CSS seulement si ces conditions sont remplies. Ces conditions portent toujours sur les caractéristiques de l'agent utilisateur utilisé pour rendre la page web (navigateur, imprimante, lecteur d'écran, etc...)



Les breakpoints

- ▶ **Les points de rupture, ou breakpoints**, sont **spécifiques aux projets** et **dépendent souvent du contenu et de la conception**. Cependant, il existe des points de rupture communs qui correspondent généralement aux tailles d'écran de différents appareils.
- ▶ Voici quelques breakpoints couramment utilisés :
 - **Mobile Portrait** : Par défaut, aucun media query nécessaire.
 - **Mobile Landscape** : min-width: 321px
 - **Tablette Portrait** : min-width: 481px
 - **Tablette Landscape** : min-width: 769px
 - **Petit écran d'ordinateur / ordinateur portable** : min-width: 1025px
 - **Grand écran d'ordinateur / ordinateur portable** : min-width: 1281px ou même min-width: 1441px

Attention :

- ▶ Il est important de comprendre que ces valeurs peuvent varier, et qu'il peut **être plus efficace de définir vos propres points de rupture** en fonction du **contenu et de la conception de votre site** spécifique.
- ▶ L'idée principale est donc **d'ajuster nos breakpoints là où le contenu commence à "casser"** ou à **perdre son aspect souhaité**, plutôt que de se fier uniquement à des tailles d'écran spécifiques pour les appareils.



Les différents type de media queries

Type	Syntaxe	Exemples
Media Type : Le type de média détermine le type d'appareil sur lequel le style sera appliqué, comme un écran ou une imprimante.	<pre>@media screen { /* styles pour écran */ } @media print { /* styles pour impression */ }</pre>	<pre>@media print { body { font-size: 12pt; } }</pre>
Width et Height : Ces media queries permettent de cibler les appareils en fonction de la largeur et de la hauteur de la zone d'affichage.	<pre>@media (min-width: 600px) { /* styles */ } @media (max-height: 800px) { /* styles */ }</pre>	<pre>@media (max-width: 600px) { body { font-size: 2rem; } }</pre>
Orientation : L'orientation permet de cibler les appareils en fonction de l'orientation de l'écran, que ce soit en mode portrait ou paysage.	<pre>@media (orientation: portrait) { /* styles */ } @media (orientation: landscape) { /* styles */ }</pre>	<pre>@media (orientation: landscape) { body { font-size: 1.5rem; } }</pre>
Resolution : La résolution cible les appareils en fonction de la densité de pixels de l'écran.	<pre>@media (min-resolution: 300dpi) { /* styles */ }</pre>	<pre>@media (min-resolution: 2dppx) { body { background-image: url('high-res-bg.jpg'); } }</pre>
Aspect Ratio : Le ratio d'aspect permet de cibler les appareils en fonction du ratio entre la largeur et la hauteur de l'écran.	<pre>@media (aspect-ratio: 16/9) { /* styles */ }</pre>	<pre>@media (aspect-ratio: 4/3) { body { margin: 0 auto; } }</pre>

D'autres type pour des cas particuliers

Type	Syntaxe	Exemples
Color : Cible les dispositifs en fonction du nombre de bits par composante de couleur.	<pre>@media (color: 8) { /* styles */ }</pre>	<pre>@media (color: 8) { body { background-color: #fafafa; } }</pre>
Monochrome: Cible les appareils monochromes, comme les liseuses.	<pre>@media (monochrome) { /* styles */ }</pre>	<pre>@media (monochrome) { body { color: black; background-color: white; } }</pre>
Pointer : Cible les appareils en fonction de la précision du pointeur, comme un doigt (coarse) ou un stylet (fine).	<pre>@media (pointer: coarse) { /* styles */ } @media (pointer: fine) { /* styles */ }</pre>	<pre>@media (pointer: coarse) { button { font-size: 2rem; } }</pre>
Et bien d'autres : https://developer.mozilla.org/fr/docs/Web/CSS/CSS_media_queries/Using_media_queries#caract%C3%A9ristiques_m%C3%A9dia_media_features		

La combinaison de plusieurs types de media

On peut **combiner plusieurs media queries** pour **cibler des appareils ou des utilisateurs** qui **répondent à plusieurs critères spécifiques**. La combinaison de media queries peut être réalisée en utilisant des **opérateurs logiques** tels que **and, not, et , (virgule, qui fonctionne comme un or logique)**.

Type	Syntaxe / Exemples
and	<pre>@media (min-width: 600px) and (orientation: landscape) { /* styles pour les écrans d'au moins 600px de large en mode paysage */ }</pre>
, (virgule)	<pre>@media screen and (max-width: 600px), print and (max-width: 800px) { /*Si Le type de média est screen ET que la largeur de la fenêtre d'affichage est de 600 pixels ou moins, OU Le type de média est print ET que la largeur de la page imprimée est de 800 pixels ou moins.*/ body { font-size: 14px; } }</pre>
not:	<pre>@media not all and (monochrome) { /* styles pour les écrans qui ne sont pas monochromes */ }</pre>

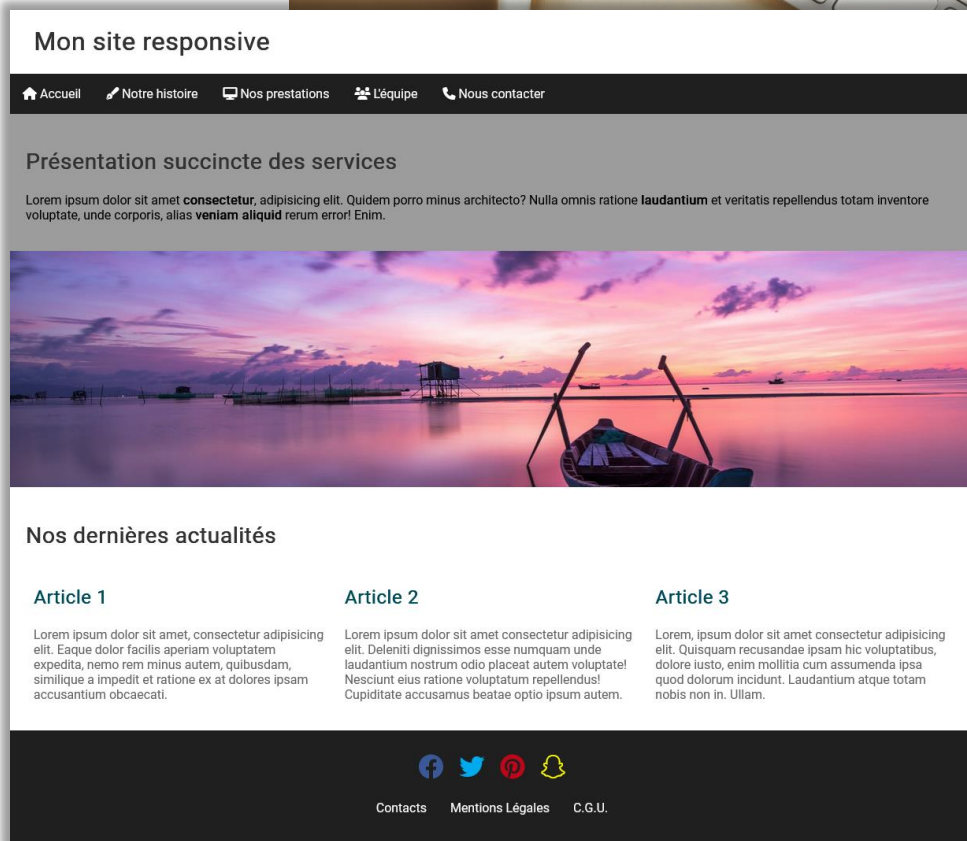
À noter :

En général, On peut combiner n'importe quelles media queries, tant **que la combinaison a un sens pour le cas d'utilisation que l'on cible**. Cependant, il faut être attentif à **ne pas créer des combinaisons tellement spécifiques qu'elles n'atteignent aucun appareil ou utilisateur réel**, ou à ne pas créer de **contradictions** dans les requêtes.

L'utilisation de combinaisons peut permettre un **contrôle très fin du style de notre site** ou application pour différents appareils, orientations, préférences utilisateur, etc. Mais cela **peut également complexifier rapidement le CSS**, alors il est généralement recommandé de **garder les choses aussi simples que possible** et de **tester soigneusement les combinaisons** pour s'assurer qu'elles fonctionnent comme prévu.

Pratique :

Énoncé : le but de cet exercice sera de **reproduire** les **maquettes** qui vous seront **fournis**, vous aurez également un **pdf** avec les **détails** qui seront à respecter.



Les transitions et animations

1 - Courbe d'accélération d'une transition

2 - Mettre en place une transition

3 - Créer et utiliser une animation CSS

Passe ta souris



Mettre en place une **transition**

Une **transition CSS** permet de **faire durer un changement de style sur une durée** afin de **créer une animation**. Dans cet exemple, le fait de **survoler la div avec la souris** va **changer la couleur du texte** avec une **transition de 1s** :

CSS

```
.transition{
  color: #0a78be;
  border: 2px solid #0a78be;
  /* il faut préciser quel style est impacté par la
  transition, la durée de cette dernière ainsi que sa courbe
  d'accélération */
  transition: color 1s ease;
}
.transition:hover{
  color: #030336;
}
```

Passe ta souris

Attention : toutes les propriétés CSS **ne peuvent pas être animées grâce à une transition**.
(liste complète ici : https://developer.mozilla.org/fr/docs/Web/CSS/CSS_animated_properties)

Il est possible de mettre une transition sur plusieurs styles en même temps avec "all" :

CSS

```
.transition02{
  color: #0a78be;
  background-color: #030336;;
  transition: all 1s ease;
}
.transition02:hover{
  color: #030336;;
  background-color: #0a78be;
}
```

Passe ta souris

Courbe d'accélération d'une transition

On peut choisir parmi plusieurs courbes d'accélération pour l'effet de la transition :

ease : Rapide sur le début et ralenti sur la fin.

linear : La vitesse est constante sur toute la durée de l'animation.

ease-in : Lent sur le début et accélère de plus en plus vers la fin.

ease-out : Rapide sur le début et décélère sur la fin.

ease-in-out : Le départ et la fin sont lents.

Il est même possible de créer soi-même ses propres courbes d'accélération (appelées courbes de Bézier) grâce à un générateur comme celui-ci :

<https://cubic-bezier.com/>

Ease 1s aller



Linear 1s aller



Ease-in 1s aller



Ease-out 1s aller



Ease-in-out 1s aller



CSS

```
.transition03{
  color: #0a78be;
  background-color: #030336;
  /* Transition utilisant une courbe personnalisée */
  transition: all 1s cubic-bezier(.14,1.43,.99,-0.71);
}
.transition03:hover{
  color: #030336;;
  background-color: #0a78be;
}
```

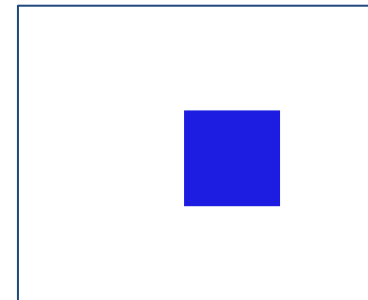
Passe ta souris

Créer et utiliser une animation CSS

Il est possible en CSS de créer des **animations** qui se dérouleront en **plusieurs étapes**.

Dans chaque étape, on peut **lister toutes les propriétés qui changeront de valeurs** par rapport à l'état initial de l'élément.

Exemple ci dessous d'une animation sur un carré, permettant de le faire **changer de couleur et d'emplacement en 5 étapes** (matérialisées par des pourcentages):



html `<div class="animation"></div>`

CSS

```
.animation {
  /* L'élément sera un carré violet par défaut au
  chargement de la page */
  width: 100px;
  height: 100px;
  background-color: purple;
  position: relative;

  /* Cet élément sera animé par l'animation "exemple"
  (créée en dessous) */
  animation-name: exemple;
  /* Durée de l'animation */
  animation-duration: 4s;
  /* Temps avant le premier lancement de l'animation */
  animation-delay: 1s;
  /* Nombre de fois que l'animation se fera d'affilé */
  animation-iteration-count: infinite;
  /* Dans quel sens les étapes de l'animation se
  dérouleront */
  animation-direction: normal;
  /* Courbe d'accélération de l'animation */
  animation-timing-function: linear;
}
```

CSS

```
/* Création de l'animation */
@keyframes exemple {

  /* État de l'élément au début de l'animation */
  0% {
    background-color: purple;
    left: 0;
    top: 0;
  }

  /* État de l'élément à 25% de l'animation */
  25% {
    background-color: yellow;
    left: 100px;
    top: 0;
  }

  /* État de l'élément à 50% de l'animation */
  50% {
    background-color: blue;
    left: 100px;
    top: 100px;
  }

  /* État de l'élément à 75% de l'animation */
  75% {
    background-color: green;
    left: 0;
    top: 100px;
  }

  /* État de l'élément à la fin de l'animation */
  100% {
    background-color: purple;
    left: 0;
    top: 0;
  }
}
```

Créer et utiliser une **animation CSS**

Il est possible de regrouper les caractéristiques d'une animation en une seule propriété :

CSS

```
@keyframes slide{
  from { left: 0; }
  to { left: 100px; }
}

div{
  position: relative;
  animation: slide 2s linear 1s 3 normal;
  /* animation : [nom de l'animation]
  [durée de l'animation] [courbe d'accélération
  de l'animation] [délais avant que l'animation
  ne commence] [le nombre de fois que se répète
  l'animation] [le sens dans lequel les étapes
  de l'animation se dérouleront ] */
}
```

1. **animation-name**: Le nom de l'animation définie avec @keyframes.
2. **animation-duration**: La durée de l'animation. Par exemple, 2s pour 2 secondes.
3. **animation-timing-function**: Comment l'animation est accélérée au cours de sa durée. Par exemple, linear, ease-in, ease-out, etc.
4. **animation-delay**: Le délai avant que l'animation ne commence.
5. **animation-iteration-count**: Le nombre de fois que l'animation doit être jouée. Par exemple, 1, infinite, etc.
6. **animation-direction**: La direction de l'animation. Par exemple, normal, reverse, alternate, alternate-reverse.
7. **animation-fill-mode**: Définit comment les valeurs sont appliquées avant et après que l'animation est exécutée. Par exemple, none, forwards, backwards, both.
8. **animation-play-state**: Contrôle la lecture de l'animation. Par exemple, paused, running.

Si votre animation commence à un état et se termine à un autre sans étapes intermédiaires spécifiques, on peut utiliser **from** pour l'état de départ et **to** pour l'état final. On peut également **combinaison des pourcentages** avec **from** et **to**. **from** est équivalent à **0%**, et **to** est équivalent à **100%**.

Pratique :

Énoncé : une base vous sera fournie où vous aurez à coder la partie CSS qui nous permettra d'effectuer les animations et transitions demandées.

[Accueil](#)[À propos](#)[Contact](#)

Exercice : Animations de sous-menu et animation de bouton

Énoncé : En premier lieu, vous devrez poursuivre le CSS pour faire les animations suivantes sur les sous-menus de nos 3 liens, accueil, à

Transition de couleur de fond :

Cliquez-moi

Transition de rotation :

Cliquez-moi

Transition d'échelle :

Cliquez-moi

Transition multiple (couleur de fond, rotation et zoom) :

Cliquez-moi



Cas particuliers

1 – Textes et images

2 – Balises et vérifications



I. Points sur les textes :

Taille de police:

Sur les petits écrans (comme les mobiles), une taille de police trop petite sera difficile à lire. Un bon point de départ est d'avoir une taille de base de 16px pour le corps du texte.

Débordements:

Veillez à ce que votre texte ne déborde pas de son conteneur, surtout pour les titres ou les étiquettes dans les petits espaces. Utilisez des propriétés comme `overflow-wrap: break-word;` ou `word-wrap: break-word;` pour gérer les longs mots ou URLs.

II. Point sur les images :

Balise `img` avec `srcset`:

Permet d'indiquer plusieurs sources pour une image, le navigateur choisira la plus adaptée à la résolution de l'écran.

III. Points sur les images HiDPI :

Résolution : Vos images doivent avoir au moins le double de la résolution nécessaire pour les écrans Retina.

Par exemple, si votre espace pour une image est de 300x300 pixels, fournissez une image de 600x600 pixels pour les écrans HiDPI.

Utilisez **`srcset`** ou **`picture`** pour servir la bonne image en fonction de la densité de pixels de l'écran.

I. Balises <audio> et <video>:

Balises <audio>:

Attention à faire en sorte que mon élément soit réactifs en utilisant width: 100%; et height: auto;.

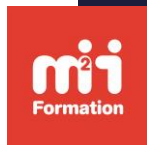
Balises <video>::

Attention également pour la vidéo à faire en sorte que mon élément soit réactifs en utilisant width: 100%; et height: auto;.
Préférez utiliser des attributs comme controls pour offrir une meilleure expérience utilisateur.

II. Test sous Firefox:

Bien que les principaux navigateurs modernes soient assez uniformes dans l'interprétation du CSS, des différences peuvent subsister, il est possible de vérifier les compatibilités via le site : <https://caniuse.com/> .

N'hésitez pas également à tester votre site sous différents navigateurs.



Fin du module



m2iinformation.fr