

# HTML & CSS

Créer le visuel de son site internet

# Table des matières

- Introduction
- Les bases du langage HTML
  - Types de balises, attributs, code de base, métas, commentaires
- Les balises HTML
  - Texte
  - Multimédias
  - Structurantes
- Les bases du langage CSS
  - Écrire son code, structure
- Les sélecteurs CSS
  - Les sélecteurs de base
  - Les sélecteurs avancés
  - Les pseudo-classes
  - Les pseudo-éléments
  - Le CSS nesting
- Les propriétés CSS
  - Les unités
  - Les couleurs
  - Rendu de base : Flow Layout
  - Formatage du texte

# Table des matières

- Les propriétés CSS (suite)
  - Les fonds et images de fond
  - Modifier la taille
  - Les espaces
  - Les bordures
  - Calcul de la taille des éléments
  - Ombres
  - Positions
  - Scrollbar
  - ResetCSS
  - Les variables CSS
- Les flexbox
  - Les deux axes
  - Dépassement des éléments
  - Espacement entre les éléments
  - Dimension dynamique des éléments
  - Ordre d'affichage des éléments
- Le Responsive Design
  - Le viewport
  - Les media queries
- Les formulaires
  - Les champs
  - Les boutons
  - Les attributs

# Table des matières

- Les tableaux
  - Base
  - Structure
  - Style
- Transformations et Animations
  - Transformations
  - Transitions
  - Animations
- Accessibilité

# Introduction

HTML



HTML+CSS



# HTML

HyperText Markup Language :

- Créé en 1991 par Tim Berners
- Aussi appelé langage de balisage
- Apporte une structure **visuelle** et **sémantique** à la page Web
- Permet la création de formulaires
- Permet l'intégration de contenus multimédias

# HTML



[En savoir plus](#)

# CSS

**CSS**



[En savoir plus](#)

**C**ascading **S**tyle **S**heets :

- Créé en 1996 par Hakon Wium Lie, Bert Bos et la W3C.
- Son rôle est de gérer l'**apparence** de la page web.

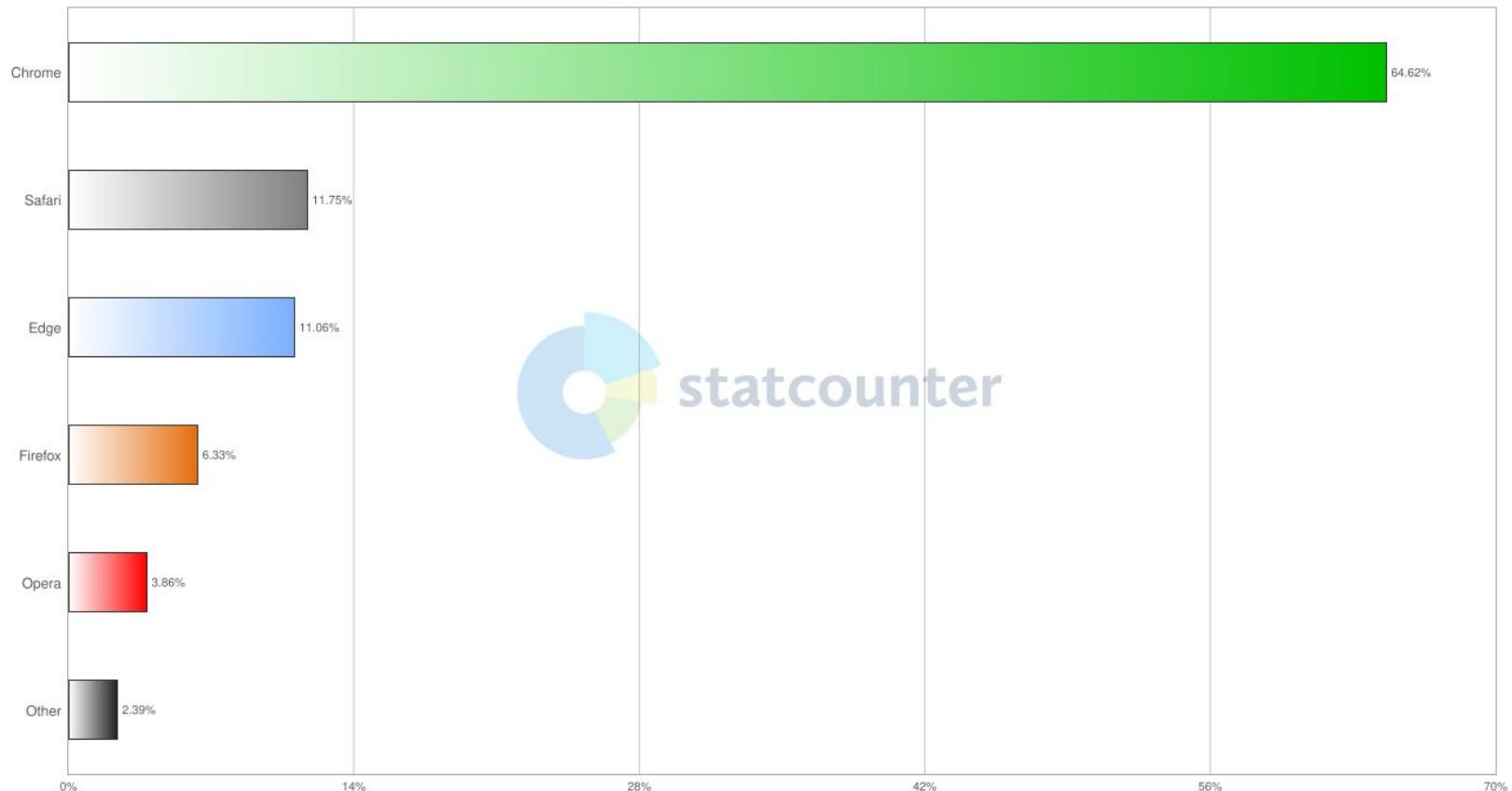
# Interprétés par les navigateurs

Ces deux langages sont interprétés par les navigateurs. Attention, chaque navigateur n'offre pas le même résultat à l'écran, pensez à tester votre page sur plusieurs navigateurs.

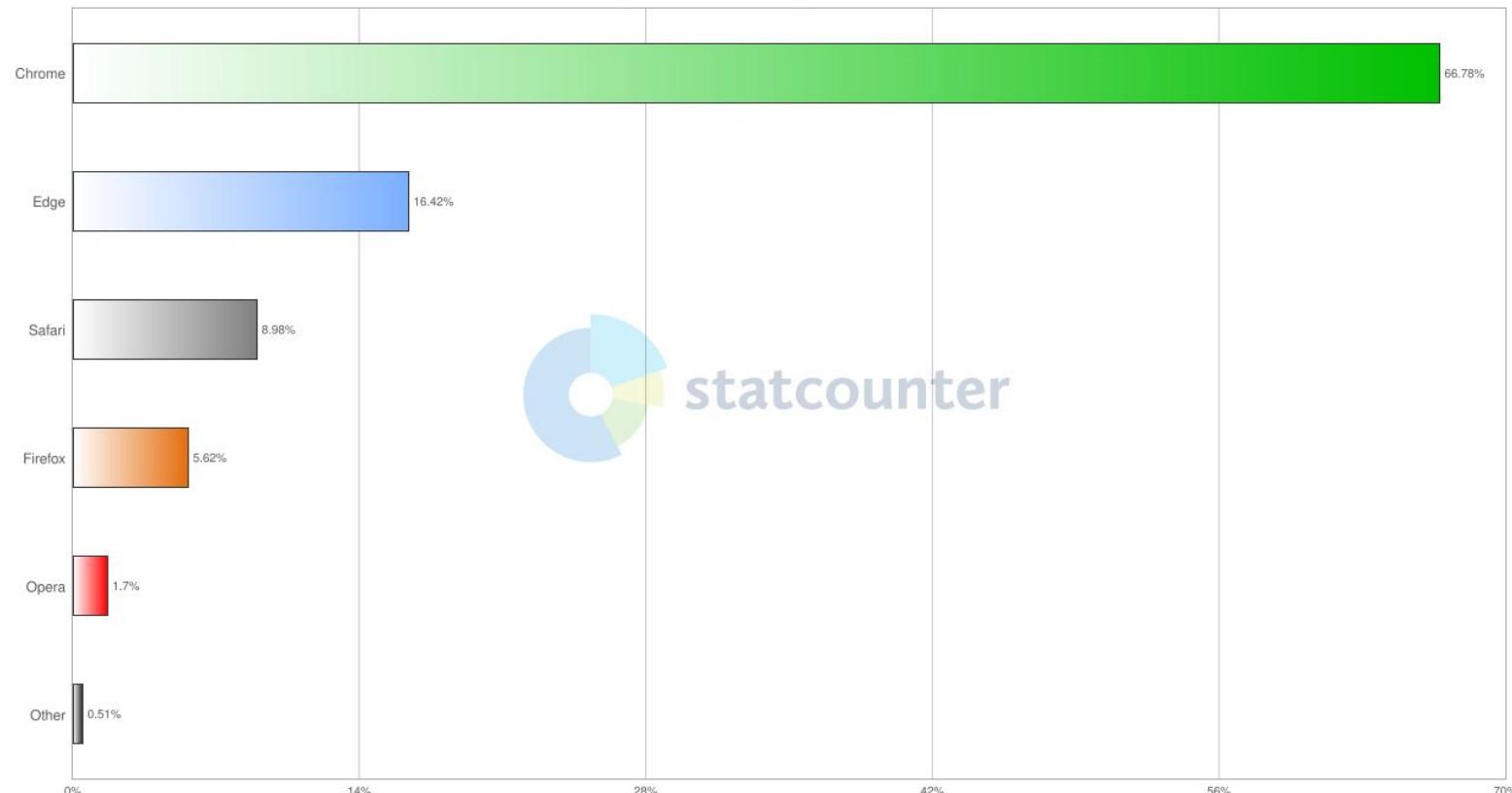
Pour savoir si une fonctionnalité est disponible sur tous les navigateurs qui vous intéressent, n'hésitez pas à consulter le site [CanIUse](#)

Vous trouverez sur les slides suivantes quelques statistiques sur le taux d'utilisation des navigateurs sur desktop et mobile pour l'année 2023. Source : [StatCounter](#)

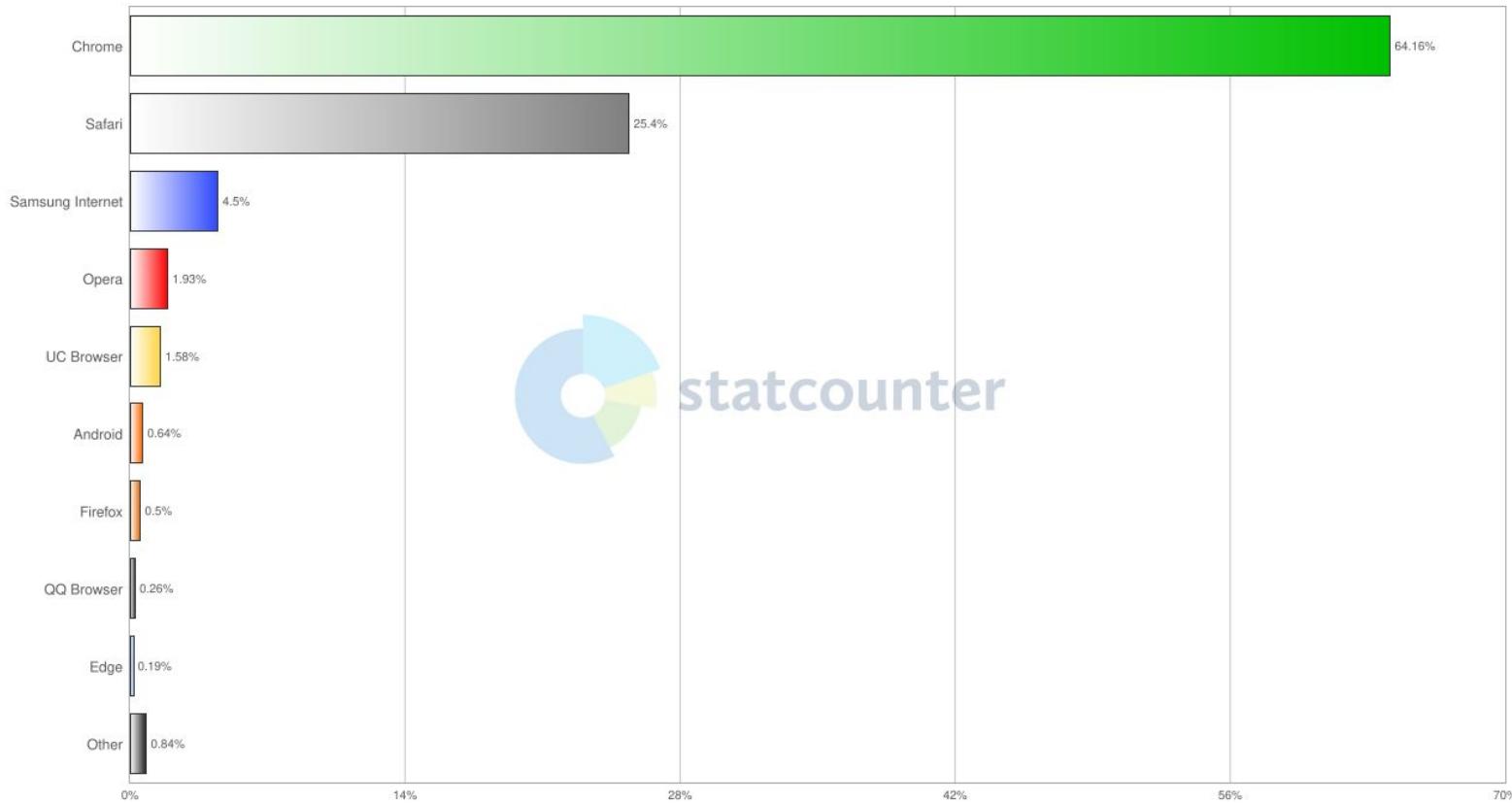
# Utilisation des navigateurs Desktop 2023 (mondial)



# Utilisation des navigateurs Desktop 2023 (Belgique)

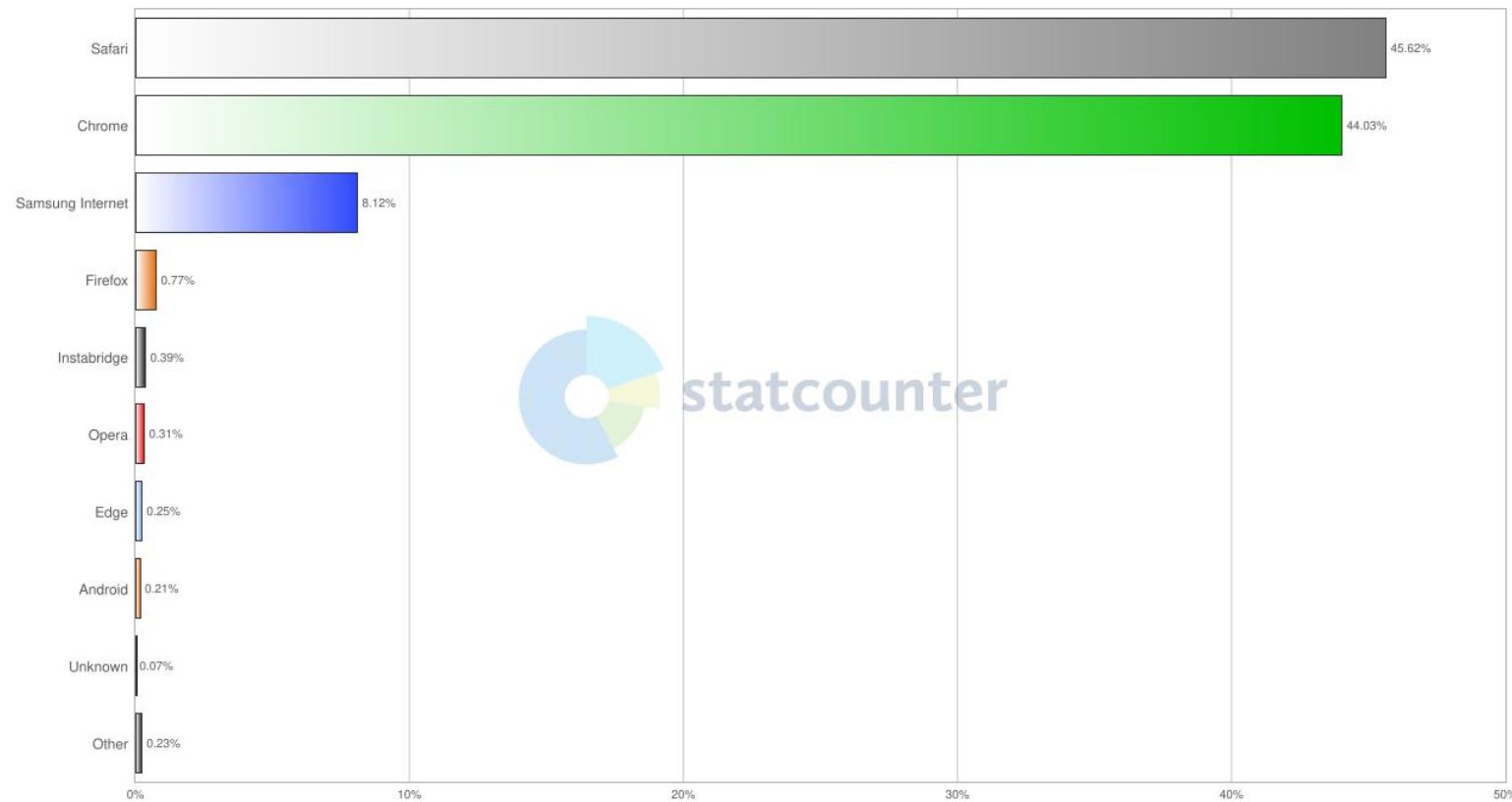


# Utilisation des navigateurs Mobile 2023 (mondial)



statcounter

# Utilisation des navigateurs Mobile 2023 (Belgique)



# Écrire son code :

Pour écrire son code, un simple **éditeur** de texte suffit.

Vous pourriez créer vos fichiers HTML et CSS avec le bloc note mais nous vous conseillons ces éditeurs de texte :

- Visual Studio Code (le plus connu, gratuit)
- WebStorm (appartient à la suite JetBrains, version Community gratuite mais réservée à un usage non commercial)



Visual Studio Code

[Installer Visual Studio Code](#)

# Les bases du langage HTML

# Les balises

Le langage HTML est composé de **balises** qui se situent entre <>.

Il y a deux types de balises :

- **Orphelines** : composées d'une seule balise <nomBalise>, on passera les informations via des attributs (ici src).

```

```

- **En paires** : composées d'une balise ouvrante et fermante <nomBalise></nomBalise>

```
<p>Un paragraphe</p>
```

# Les attributs

Les attributs permettent de rajouter des **informations** en plus sur les balises. Il en existe plusieurs types, certains communs à toutes les balises, d'autres propres à certaines balises. Nous en verrons plusieurs plus tard, selon leur utilité.

Il s'écrivent toujours **nomAttribut**=“valeur”, voici quelques exemples :

- id : permet d'identifier de manière unique un élément dans la page

```
<p id="p1">Un paragraphe</p>
```

- class : permet d'ajouter du style css via un nom de class (cumulables)

```
<p class="text-blue text-bold">Un paragraphe</p>
```

# Le code de base

Le code minimal d'une page web est :

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>

    </body>
</html>
```

- **DOCTYPE** : n'est pas une balise HTML, d'où la présence du « ! ». Il indique au navigateur que notre code est du langage HTML.
- **html** : C'est entre ces balises que se situe le code HTML
- **head** : Contient toutes les informations sur la page Web comme le titre, les métas, les liens vers des fichiers, etc...
- **body** : C'est entre ces balises que se situe le contenu de notre page

# Les métas

Ce sont des **infos** sur la page. Elles ne sont pas visibles mais sont interprétées par le navigateur, utilisées par les **moteurs de recherche**, utiles pour le **référencement...**

Quelques exemples de méta :

- La meta charset permet de définir le type d'encodage à utiliser

```
<meta charset="UTF-8">
```

- La meta description est utile au référencement. C'est le texte affiché quand votre site apparaît en résultat de recherche via un moteur de recherche

```
<meta name="description" content="Cours d'HTML & CSS par Bstorm">
```

# Les commentaires

Une balise spéciale existe pour nous permettre d'ajouter des commentaires.

Celle-ci commence par `<!--` et termine par `-->`.

Gardez toujours en tête qu'un commentaire reste visible dans la page, attention donc à ce que vous y mettez !

```
<!-- Ceci est un commentaire -->
```

Il existe des raccourcis pour mettre du code en commentaire :

- Sur VSC : **Ctrl + :**
- Sur WebStorm : **Ctrl + /**

# Style de base des balises

Certaines balises que nous allons voir ont un style par défaut.

Par exemple, un titre se met en gras, avec une police plus ou moins grande selon l'importance du titre.

**Attention à ne pas les utiliser pour styliser votre page, on les utilisera pour leur intérêt sémantique ou structurel uniquement.**

Pas de panique, nous pourrons modifier le style grâce au CSS ensuite !

D'ailleurs, on utilise souvent un fichier CSS, appelé resetCss, qui remet à 0 tous les styles par défaut des éléments. Nous en reparlerons au chapitre CSS.

# Les balises HTML

# Les balises de texte

# Les titres

Au nombre de 6, ils permettent d'afficher des titres du plus ou moins importants dans la page. Ils ont un intérêt niveau référencement et une mauvaise utilisation pourra vous punir à ce niveau.

2 règles d'or sont à respecter :

- Un seul **h1** par page
- Ne pas employer un **hX+1** si un **hX** n'est pas présent plus haut dans la page

# Les titres

```
<h1>Titre principal de la page (unique)</h1>
<h2>Titres secondaires</h2>
<h3>Sous-titres ou titres moins importants</h3>
<h4>Titres moins importants</h4>
<h5>Titres peu importants</h5>
<h6>Titres très peu importants</h6>
```



**Titre principal de la page (unique)**

**Titres secondaires**

**Sous-titres ou titres moins importants**

**Titres moins importants**

**Titres peu importants**

**Titres très peu importants**

# Les paragraphes

La balise `<p>` représente un paragraphe dans un texte :

```
<p>Un paragraphe</p>
```

# Les sauts de ligne

La balise `<br>` sert à faire un saut de ligne simple et `<hr>` un saut de ligne avec une ligne

```
<p>Un premier <br> paragraphe ici</p>
<hr>
<p>Un deuxième paragraphe ici</p>
```

Un premier  
paragraphe ici

---

Un deuxième paragraphe ici

# Informations sur du texte

Attention, les balises suivantes sont à utiliser pour leur but sémantique et non pour gérer l'apparence visuelle.

```
<strong>Ce texte est important !</strong>  
  
<p>Tu <em>dois</em> accentuer !</p>  
  
<p>Sert à <mark>surligner</mark> </p>  
  
<blockquote>Une très grosse citation d'une source externe</blockquote>  
<q>Une courte citation</q>  
  
<p>J'adore <abbr title="HyperText Markup Language">HTML</abbr> !</p>
```

**Ce texte est important !**

Tu *dois* accentuer !

Sert à **surligner**

Une très grosse citation d'une source externe

«Une courte citation»

J'adore HTML !

HyperText Markup Language

# Les listes

Non ordonnée (la + utilisée)

ul : début et fin de la liste

li : les éléments

```
<ul>
    <li>HTML</li>
    <li>CSS</li>
</ul>
```

- HTML
- CSS

Ordonnée

ol : début et fin de la liste

li : les éléments

```
<ol>
    <li>HTML</li>
    <li>CSS</li>
</ol>
```

1. HTML
2. CSS

Définition

dl : début et fin de la liste

dt : terme (titre)

dd : description

```
<dl>
    <dt>HTML</dt>
    <dd>Squelette</dd>
    <dt>CSS</dt>
    <dd>Esthétique</dd>
</dl>
```

HTML  
Squelette  
CSS  
Esthétique

# Les liens

Pour insérer des liens dans la page HTML, il faudra passer par la balise `<a> </a>`. L'attribut minimum requis est **href**, qui vous permet de renseigner la cible de votre lien.

Vous pouvez rajouter les attributs suivants :

- **title** : permet de mettre un texte dans une info-bulle au survol
- **download** : si c'est un lien vers un fichier, il va être téléchargé par l'utilisateur au click
- **referrerpolicy** : définit quelles informations sont envoyées au site sur lequel on va
  - no-referrer : pas d'informations envoyées
  - origin : envoie l'origin (scheme, host, port) du site
- **target** : définit où le lien sera ouvert (même onglet, nouvel onglet, nouvelle fenêtre...)
  - \_self : page courante (valeur par défaut)
  - \_blank : nouvel onglet

# Les liens

Lien vers une page externe :

```
<a href="https://google.be">Google est ton ami</a>
```

Lien vers une autre page du site (un fichier html dans votre dossier courant) :

```
<a href="./contact.html">Contactez-moi</a>
```

Lien vers un endroit spécifique dans la page (ancre via un id) :

```
<h1 id="top">Le titre tout en haut de ma page</h1>

<a href="#">#top">Revenir en haut</a>
```

# Les liens

Lien vers un fichier :

```
<a href="../monSuperCv.pdf">Voir mon CV</a>
<a href="../monSuperCv.pdf" download>Télécharger mon CV</a>
```

Lien pour envoyer un email :

```
<a href="mailto:brain.storm@bstorm.be">Envoyer un mail</a>
```

Lien pour appeler un numéro de téléphone sur mobile :

```
<a href="tel:+33422521010">Allo Papa Noël ?</a>
```

# Les balises multimédia

# Les images

Pour intégrer une image dans votre page, il faudra utiliser la balise orpheline `<img>`.

Celle-ci possède 2 attributs **OBLIGATOIRES** :

- **src** : La source de l'image (en ligne ou en local)
- **alt** : Le texte alternatif qui apparaîtra si l'image n'a pas pu être chargée. Elle est aussi très utile pour le référencement et pour l'accessibilité.

Vous pouvez rajouter, en plus de ces 2 attributs :

- **title** : permet de mettre un texte dans une info-bulle au survol
- **loading** : définit la façon dont l'image va être chargée
  - eager : par défaut, charge l'image au chargement de la page
  - lazy : charge l'image au moment où elle devrait apparaître à l'écran

# Les images

La balise **img** :

```

```

La balise **figure** :

Elle permet de rajouter une légende à notre image, pour illustrer un article par exemple ou ajouter une référence.

```
<figure>
  
  <figcaption>Attention au coup de pied du kangourou !</figcaption>
</figure>
```

# Les images (img vs figure)

Balise img



Balise figure



Attention au coup de pied du kangourou !

# Les images

La balise **<picture> </picture>** :

Elle permet d'afficher des images différentes en changeant la source de l'image selon la taille de l'écran. On affichera, par exemple, une image en format paysage avec beaucoup de détails sur desktop et une image au format carré qui met l'accent sur un élément en particulier sur mobile.

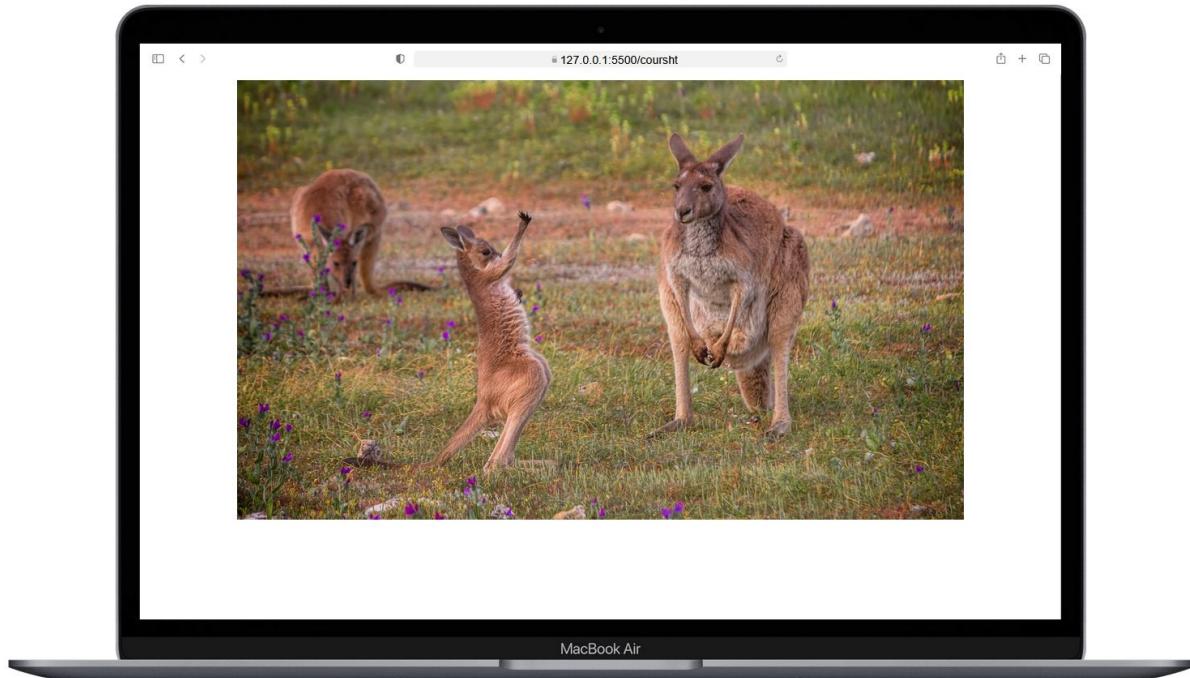
```
<picture>
  <source media="(max-width:650px)" srcset="./images/kangourouMobile.jpg">
    
</picture>
```

# Les images (picture)

Sur un écran < 650 px



Sur un écran > 650 px



# Les iframes

La balise **<iframe> </iframe>** permet d'intégrer du contenu venant d'un autre site web dans votre site. Il est souvent utilisé pour intégrer un aperçu d'une autre page, une vidéo youtube, une map etc...

```
<iframe src="https://fr.wikipedia.org/wiki/Kangourou"  
title="Les Kangourous"></iframe>
```

```
<iframe src="https://www.youtube.com/embed/7p41rWD3s-c?si=cmjNpihuWdKR4ZPv"  
title="YouTube video player" frameborder="0" allow="accelerometer; autoplay;  
clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share"  
referrerpolicy="strict-origin-when-cross-origin" allowfullscreen></iframe>
```

# Les iframes

The screenshot shows a Wikipedia article page for "Kangourou". At the top, the Wikipedia logo and the text "WIKIPÉDIA L'encyclopédie libre" are visible. Below the header, there's a search bar and a menu icon. The main title "Kangourou" is displayed, along with a link to "112 langues". Below the title, there are links for "Article" and "Discussion". A sidebar on the right contains a note about homonyms and a red box with a question mark icon containing a note about improving sources.

Pour les articles homonymes, voir [Kangourou \(homonymie\)](#).

Cet article ou cette section fait référence à des sources qui ne semblent pas présenter la fiabilité et/ou l'indépendance requises.

? Vous pouvez aider, soit en recherchant des sources de meilleure qualité pour étayer les informations concernées, soit en attribuant clairement ces informations aux sources qui semblent insuffisantes, ce qui permet de mettre en garde le lecteur sur l'origine de l'information. Voir la [page de discussion](#) pour plus de détails.



# Les canvas

La balise `<canvas> </canvas>` permet de définir une zone dans laquelle on peut dessiner à l'aide de JavaScript.

```
<canvas id="votreDessin">
|   Votre navigateur ne supporte pas les canvas.
</canvas>
```

Quelques exemples ici :

Mini Paint : <https://codepen.io/KamyarLajani/pen/xxKZzXv>

Vache qui vole : <https://codepen.io/mimikim/pen/x0ygpJ>

# Les audios

La balise **<audio></audio>** permet d'intégrer un fichier audio à votre page. Sont disponibles plusieurs attributs dont :

- **src** : permet de définir votre fichier audio
- **controls** : permet d'activer les contrôles de l'audio (play/pause/volume). Le design de ceux-ci dépend du navigateur utilisé et, tant qu'ils ne sont pas activés, l'audio n'est pas visible dans la page.
- **autoplay** : permet de lire automatiquement l'audio. Depuis quelques années, sur tous les navigateurs Chromium, cette fonctionnalité est désactivée.

```
<audio src="./audios/duck.mp3" controls></audio>
```

# Les vidéos

La balise **<video></video>** permet d'intégrer un fichier vidéo à votre page. Sont disponibles plusieurs attributs dont :

- **src** : permet de définir votre fichier vidéo
- **controls** : permet d'activer les contrôles de la vidéo (play/pause/volume...). Le design de ceux-ci dépend du navigateur utilisé et, tant qu'ils ne sont pas activés, la vidéo n'est pas visible dans la page.
- **autoplay** : permet de lire automatiquement la vidéo. Depuis quelques années, sur tous les navigateurs Chromium, cette fonctionnalité est désactivée.

```
<video src="./videos/ducks.mp3" controls></video>
```

# Les balises structurantes

# Les balises “container”

La balise `<div> </div>` :

- Balise de type **block**.
- Utilisée pour regrouper du contenu pour de la mise en page CSS, elle n'a aucun intérêt sémantique.
- À n'utiliser que si aucune autre balise ne convient pour représenter le contenu regroupé. (ex: article, nav, etc)

La balise `<span> </span>` :

- Balise de type **inline**.
- Utilisée pour styliser les contenus phrasés avec le CSS. Elle n'a aucun intérêt sémantique.
- À n'utiliser que si aucune autre balise ne convient pour représenter le contenu. (ex : mark, strong, etc)

```
<div class="product-card">
    <h3>Nom Produit</h3>
    <p>Description du produit</p>
    <p>Prix : <span class="price">12.50 €</span></p>
</div>
```

# Balise d'en-tête

La balise **<header> </header>** permet de créer un en-tête. On peut en retrouver plusieurs dans une page.

- Si mis tout en haut de la page, il reflète l'en-tête de votre page Web et on y retrouve, par exemple, le logo du site, une bannière, la navigation principale, etc.
- Mis dans une autre structure, c'est l'en-tête de cette structure. Par exemple, l'en-tête d'une section, d'un article, d'une card, etc...

```
<header>
    
    <nav> ←— navigation site —> </nav>
</header>

<article>
    <header>
        | ←— En tête de l'article avec titre, date, auteur etc ... —>
    </header>
    | ←— ... contenu —>
</article>
```

# Balise de pied de page

La balise **<footer> </footer>** permet de créer un pied de page. On peut en retrouver plusieurs dans une page.

- Si mis tout en bas de la page, il reflète le pied de page de votre site et on y retrouve, par exemple, les conditions générales, les FAQ, les liens vers les réseaux etc...
- Mis dans une autre structure, c'est le pied de page de cette structure. Par exemple, celui d'une section, d'un article, d'une card, etc...

```
←— Directement dans la page →  
<footer>  
| ←— liens conditions, FAQ, etc ... →  
</footer>  
  
<article>  
| <footer> ←— infos sur l'auteur·ice, la date etc ... → </footer>  
</article>
```

# Balise de contenu principal

La balise `<main> </main>` représente le contenu général de la page. Elle doit se trouver dans le body et il ne peut y en avoir qu'une seule par page.

```
<header>
|   ← En-tête de la page →
</header>

<main>
|   ← Contenu de toute la page →
</main>

<footer>
|   ← Pied de page →
</footer>
```

# Balise de navigation

La balise **<nav> </nav>** permet d'indiquer qu'il y a une navigation. En général, on y retrouve une liste de liens. Il peut y avoir plusieurs navigations différentes dans une page (principale, secondaire etc)

```
<nav>
    <ul>
        <li> <a href="">Accueil</a> </li>
        <li> <a href="./about.html">À propos</a> </li>
        <li> <a href="./contact.html">Contact</a> </li>
    </ul>
</nav>
```

# Balise section

La balise **<section> </section>** permet de regrouper du contenu appartenant à un même thème. On y trouve la plupart du temps un titre.

```
<h1>Développement Front-End</h1>

<section>
    <h2>HTML :</h2>
    <p>Squelette de la page [ ... ]</p>
</section>

<section>
    <h2>CSS :</h2>
    <p>Design de la page [ ... ]</p>
</section>
```

# Balise article

La balise **<article> </article>** représente un élément indépendant d'une page (un article de presse, un post sur un forum). De nos jours, il est de moins en moins utilisé SAUF justement dans le cadre d'articles de journaux, blog, forum etc...

```
<article>
    <h2>Titre de l'article</h2>
    <p>Contenu de l'article</p>
</article>
```

# Balise aside

La balise **<aside> </aside>** permet de représenter du contenu qui n'a pas de rapport direct avec le contenu principal de votre élément (un "aparté"). De plus en plus oublié, il a pourtant toujours un intérêt niveau sémantique et accessibilité.

```
<p>
    HTML, HyperText Markup Language, sert à
    créer la structure de votre page web.
</p>

<aside>
    <q>J'aime trop penser à la structure idéale
    de ma page</q> <span> - John Doe</span>
</aside>

<p>
    Chaque balise a un intérêt sémantique et
    structurel. Une bonne gestion des balises
    permet aussi de rendre
    votre site plus accessible.
</p>
```

HTML, HyperText Markup Language,  
sert à créer la structure de votre page  
web.

Chaque balise a un  
intérêt sémantique et  
structurel. Une bonne  
gestion des balises  
permet aussi de  
rendre votre site plus  
accessible.

«J'aime trop  
penser à la  
structure  
idéale de ma  
page» - John  
Doe

# Balise de boîte de dialog

La balise **<dialog> </dialog>** permet d'afficher une modal ou boîte de dialogue. Elle offre tout un tas de fonctions pour faciliter l'interaction avec le JavaScript pour ouvrir, fermer, récupérer une valeur de retour par exemple.

```
<dialog open>
  <p>Voulez-vous vraiment supprimer ?</p>
  <button>Supprimer</button>
  <button>Annuler</button>
</dialog>
```

Voulez-vous vraiment supprimer ?

Supprimer Annuler

<header>

<nav>

<main>

<section>

<article>

<aside>

<article>

<footer>

<section>

<header>

<footer>

Résumé

# Les bases du langage CSS

# Écrire son code CSS

Il y a 3 endroits où vous pouvez écrire du CSS.

- directement dans la balise HTML (**déconseillé**). Cela s'appelle du style inline et aura toujours priorité.

```
<p style="color : chartreuse">Du texte</p>
```

- dans une balise style, dans le head de votre page. Peu pratique, doit être répété sur toutes vos pages, rallonge le fichier etc...

```
<style>
  p {
    color : tomato;
  }
</style>
```

# Écrire son code CSS

- dans un fichier séparé avec l'extension de fichier .css qu'on va ensuite relier à notre page html grâce à la balise link. (**fortement conseillé**)

monStyle.css > ...

```
1  p {  
2      color : chartreuse  
3 }
```

```
<link rel="stylesheet" href="./monStyle.css">
```

Sur certains tuto, on retrouve un attribut supplémentaire type="text/css" qui n'est plus obligatoire puisque celui par défaut

# Structure du Css

Le langage CSS est constitué de **sélecteurs** et de **propriétés** qui possède une (ou des) valeur(s).

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

# Les sélecteurs Css

# Les sélecteurs de base

# Les sélecteurs de base

- Le sélecteur universel  
→ le symbole \*  
→ signifie "tout"
- Le sélecteur par tag (balise)  
→ le nom de la balise
- Le sélecteur par nom de classe  
→ le symbole .
- Le sélecteur par id  
→ le symbole #

```
* {  
|   color : tomato;  
}
```

```
p {  
|   color : chartreuse  
}
```

```
.text-blue {  
|   color : blue;  
}
```

```
#mon-id {  
|   color : orange;  
}
```

```
<p class="text-blue">  
|   Du texte  
</p>
```

```
<p id="mon-id">  
|   Du texte  
</p>
```

# Le poids des sélecteurs

Ces trois sélecteurs de base ont un poids défini, aussi appelé la spécificité. C'est elle qui va déterminer, si deux sélecteurs ciblent le même élément, lequel aura la priorité. Ce poids est en trois niveaux différents et est représenté par 3 nombres entre parenthèses (X, X, X).

- Le sélecteur par **id** possède le niveau 1

```
<element id="mon-id">
```

Selector Specificity: (1, 0, 0)

- Le sélecteur par **class** possède le niveau 2

```
<element class="text-blue">
```

Selector Specificity: (0, 1, 0)

- Le sélecteur par **tag** possède le niveau 3

```
<p>
```

Selector Specificity: (0, 0, 1)

# Les sélecteurs avancés

# Les sélecteurs par attributs

Le sélecteur **attribut** : [attribut]

Syntaxe : A[B]

Sélectionne tous les éléments A qui possèdent l'attribut B

ex : Sélectionne les <a> qui possèdent un attribut title

```
a[title] {  
    color : chartreuse;  
}
```

```
<a href="" title="lien1"> Lien selectionné ✓ </a>  
<a href=""> Lien pas sélectionné ✗ </a>  
<a href="" title="lien2"> Lien selectionné ✓ </a>  
<a href="" title=""> Lien selectionné ✓ </a>
```

# Les sélecteurs par attributs

Le sélecteur **attribut+valeur** : [attribut="valeur"]

Syntaxe : A[B="C"]

Sélectionne tous les éléments A qui possèdent l'attribut B dont la valeur est C

ex : Sélectionne les <a> qui possèdent un attribut title donc la valeur est "lien1"

```
a[title="lien1"] {  
    color : chartreuse;  
}
```

```
<a href="" title="lien1"> Lien selectionné ✓ </a>  
<a href=""> Lien pas sélectionné ✗ </a>  
<a href="" title="lien2"> Lien pas sélectionné ✗ </a>  
<a href="" title=""> Lien pas sélectionné ✗ </a>
```

# Les sélecteurs combinateurs

Le sélecteur **descendant** : ""(espace)

Syntaxe : A B

Sélectionne tous les éléments B se trouvant dans un élément A

ex : Sélectionne les <p> qui se trouvent dans une <div>

```
div p {  
    color : chartreuse;  
}
```

```
<section>  
    <p> pas sélectionné ✗ </p>  
    <div>  
        <p> sélectionné ✓ </p>  
        <article>  
            <p> sélectionné ✓ </p>  
        </article>  
    </div>  
</section>
```

# Les sélecteurs combinateurs

Le sélecteur **enfant direct** : >

Syntaxe : A > B

Sélectionne les éléments B qui sont enfants directs de A.

ex : sélectionne les <p> qui sont enfants directs de la <section>

```
section > p {  
    color : chartreuse;  
}
```

```
<section>  
    <p> selectionné ✓ </p>  
    <div>  
        <p> pas sélectionné ✗ </p>  
    </div>  
</section>
```

# Les sélecteurs combinateurs

Le sélecteur **voisin direct** : +

Syntaxe : A + B

Sélectionne l'élément B qui suit un élément A.

ex : sélectionne le <p> qui est voisin direct par la droite d'un <span>

```
span + p {  
    color : chartreuse;  
}
```

<p> pas sélectionné ✗ </p>
<span>Pouet</span>
<p> sélectionné ✓ </p>
<p> pas sélectionné ✗ </p>
<a href="">Lien</a>
<p> pas sélectionné ✗ </p>

# Les sélecteurs combinateurs

Le sélecteur **voisins** : ~

Syntaxe : A ~ B

Sélectionne tous les éléments B qui suivent un élément A.

ex : sélectionne les <p> qui sont voisins directs par la droite d'un <span>

```
span ~ p {  
    color : chartreuse;  
}
```

<p> pas sélectionné	<input type="checkbox"/>	</p>
<span>Pouet</span>		
<p> sélectionné	<input checked="" type="checkbox"/>	</p>
<p> sélectionné	<input checked="" type="checkbox"/>	</p>
<a href="">Lien</a>		
<p> sélectionné	<input checked="" type="checkbox"/>	</p>

# Les sélecteurs combinateurs

Le sélecteur **et** :

Syntaxe : A, B

Sélectionne tous les éléments A ET tous les éléments B

ex : sélectionne les <p> et les <q>

```
p, q {  
    color : chartreuse;  
}
```

ex : sélectionne les <a> qui ont en attribut title les valeurs lien1 et lien2

```
a[title="lien1"], a[title="lien2"] {  
    color : chartreuse;  
}
```

# Les pseudo-classes

# Les pseudo-classes

Les pseudo-classes, ajoutées à un sélecteur, permettent de cibler un état spécifique de l'élément pour qu'il soit sélectionné.

Elles sont précédées du symbole ":" comme suit :

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

# Pseudo-classes de page

# Les pseudo-classes : La page

- `:root`  
→ sélectionne la racine de la page, c'est à dire, la balise html mais sa spécificité (poids) est plus forte que celle de la balise html.
- `:target`  
→ sélectionne l'élément dont l'id est actuellement ciblé par l'ancre présente dans l'url de la page

# Pseudo-classes d'événements

# Les pseudo-classes : Événements

- A:active  
→ sélectionne un élément A s'il est activé par l'utilisateur. Un élément est considéré actif quand on est en train de cliquer mais qu'on n'a pas encore relâché.
- A:hover  
→ sélectionne un élément A s'il est survolé par la souris de l'utilisateur.
- A:focus  
→ sélectionne un élément A quand il reçoit le focus de l'utilisateur (sélectionné par la tabulation, click dans un champ).
- A:focus-within  
→ sélectionne un élément A si lui ou un de ses descendants reçoit le focus.

# Les pseudo-classes : Événements

- A:link  
→ sélectionne un lien pas encore visité (doit être placé avant :visited, :hover et :active).
- A:visited  
→ sélectionne un lien qui a déjà été visité.

# Pseudo-classes pour préciser les éléments

# Les pseudo-classes : Préciser les éléments

- A:first-child  
→ sélectionne n'importe quel élément A uniquement s'il est le premier enfant de son parent.
- A:first-of-type  
→ sélectionne le premier élément de type A présent dans son parent.
- A:last-child  
→ sélectionne n'importe quel élément A uniquement s'il est le dernier enfant de son parent.
- A:last-of-type  
→ sélectionne le dernier élément de type A présent dans son parent.

# Les pseudo-classes : Préciser les éléments

- A:nth-child(odd)  
→ sélectionne les éléments A s'ils sont impairs dans l'élément parent.
- A:nth-of-type(odd)  
→ sélectionne les éléments A impairs parmi tous les éléments de type A dans l'élément parent.
- A:nth-child(even)  
→ sélectionne les éléments A s'ils sont pairs dans l'élément parent.
- A:nth-of-type(even)  
→ sélectionne les éléments A pairs parmi tous les éléments de type A dans l'élément parent.

# Les pseudo-classes : Préciser les éléments

- A:nth-child(Xn+Y)  
→ sélectionne l'élément A s'il répond au motif Xn+Y dans son parent.  
X détermine l'intervalle de répétition et Y le début de la séquence.  
ex : 2n+4 : à partir du 4ème élément et tous les 2 éléments
- A:nth-of-type(Xn+Y)  
→ sélectionne, parmi tous les éléments A présents dans le parent, les éléments A qui correspondent au motif Xn+Y.
- A:nth-last-child()  
→ comme nth-child() mais en partant de la fin.
- A:nth-last-of-type()  
→ comme nth-of-type() mais en partant de la fin.

# Les pseudo-classes : Préciser les éléments

- A:only-child
  - sélectionne l'élément A s'il est enfant unique de son parent.
- A:only-of-type
  - sélectionne l'élément A s'il est le seul de son type dans son parent.
- A:empty
  - sélectionne l'élément A s'il ne possède aucun contenu (seuls les commentaires sont autorisés à condition qu'il n'y ai pas d'espaces avant ni après)
- A:not(sélecteur(s))
  - sélectionne l'élément A s'il ne correspond pas au(x) sélecteur(s) spécifiés entre ()

# Les pseudo-classes : Préciser les éléments

- A:has(sélecteur(s))  
→ sélectionne l'élément A s'il possède le(s) sélecteur(s) renseigné(s)
- A:modal  
→ sélectionne l'élément A qui est dans un état qui interdit tout interaction avec la page (utilisé avec la balise dialog)

# Pseudo-classes des formulaires

# Les pseudo-classes : Formulaires

- A:indeterminate  
→ sélectionne un élément A dont l'état est indéterminé (ex: des radios reliés entre eux où aucun élément n'est encore coché).
- A:in-range  
→ sélectionne un élément A dont l'état se trouve entre le min et le max spécifié.
- A:out-of-range  
→ sélectionne un élément A dont l'état se trouve dehors du min et du max spécifié.
- A:read-only  
→ sélectionne un élément A qui est en lecture seule.

# Les pseudo-classes : Formulaires

- A:invalid  
→ sélectionne un élément A dont la validité a échoué.
- A:valid  
→ sélectionne un élément A dont la validité s'est effectuée correctement.
- A:required  
→ sélectionne un élément A qui possède l'attribut required actif.
- A:optional  
→ sélectionne un élément A qui ne possède pas l'attribut required actif.

# Les pseudo-classes : Formulaires

- A:checked  
→ sélectionne l'élément A s'il est coché ou activé (radio, checkbox, option)
- A:disabled  
→ sélectionne l'élément A qui possède l'attribut disabled (désactivé) actif
- A:enabled  
→ sélectionne l'élément A qui ne possède pas l'attribut disabled actif
- A:placeholder-shown  
→ sélectionne l'élément A si son placeholder est visible

# Les pseudo-éléments

# Les pseudo-éléments

Les pseudo-éléments, ajoutés à un sélecteur, permettent de cibler une partie spécifique de l'élément afin de le styliser.

Ils sont précédés du symbole “::” comme suit :

```
sélecteur :: pseudo-element {  
    |   propriété : valeur;  
    | }  
}
```

# Les pseudo-éléments

- A::before
  - crée un pseudo-élément html qui sera premier enfant de A. Principalement utilisé pour ajouter du contenu esthétique à l'élément sélectionné.
- A::after
  - crée un pseudo-élément html qui sera dernier enfant de A. Même but que before.
- A::first-letter
  - sélectionne la première lettre de l'élément A.
- A::first-line
  - sélectionne la première ligne de l'élément A.

# Les pseudo-éléments

- A::placeholder
  - permet de sélectionner le placeholder d'un champ.
- A::selection
  - permet d'appliquer du style à la sélection textuelle de l'utilisateur.
- A::cue
  - permet la mise en forme CSS des sous-titre WebVTT des vidéos.

# Le CSS Nesting

# CSS Nesting

Le CSS Nesting, disponible depuis 2023, est une nouvelle façon d'écrire votre CSS, inspirée du langage [Sass](#).

Elle permet une meilleure lecture et maintenabilité du code, ainsi qu'une réduction de la taille de vos fichiers CSS.

Deux concepts seulement sont à connaître :

- L'imbrication des blocs
- Le sélecteur &

# CSS Nesting : L'imbrication des blocs

L'imbrication de blocs est l'équivalent du sélecteur descendant. Ainsi :

version nestée :

```
section {  
    /* style de la section */  
    div {  
        /* style des div d'une section */  
        p {  
            /* style des p qui sont dans  
               les div d'une section */  
        }  
    }  
}
```



version classique :

```
section {  
    /* style de la section */  
}  
section div {  
    /* style des div d'une section */  
}  
section div p {  
    /* style des p qui sont dans  
       les div d'une section */  
}
```

# CSS Nesting : Le sélecteur &

Le **&** permet d'ajouter une règle à l'élément courant plutôt qu'à son descendant.

```
section {  
    /* style de la section */  
}  
  
section:hover {  
    /* style de la section si elle  
     * est survolée par l'utilisateur */  
}  
  
section div {  
    /* style des div d'une section */  
}  
  
section div p {  
    /* style des p qui sont dans  
     * les div d'une section */  
}  
  
section div p.pouet {  
    /* style des p qui sont  
     * dans les div d'une section  
     * s'ils ont la class pouet */  
}
```



```
section {  
    /* style de la section */  
}  
  
&:hover {  
    /* style de la section si elle  
     * est survolée par l'utilisateur */  
}  
  
div {  
    /* style des div d'une section */  
}  
  
p {  
    /* style des p qui sont dans  
     * les div d'une section */  
}  
  
&.pouet {  
    /* style des p qui sont  
     * dans les div d'une section  
     * s'ils ont la class pouet */  
}
```

# Les propriétés CSS

# Les unités

# Les unités

Pour de nombreuses propriétés, nous allons devoir préciser une valeur avec une unité. Il existe plusieurs types d'unités. Nous allons voir les plus utilisées mais si vous souhaitez voir la liste complète, [c'est par ici.](#)

- Les unités de distance (ex : gérer les tailles, les espacements)
- Les unités angulaires (ex : gérer les rotations)
- Les unités temporelles (ex : les durées des animations)
- Les pourcentages
- Les fonctions

# Les unités de distance

Utilisées pour gérer les tailles et espacement des éléments, voici une liste des plus utilisées :

- **rem** → relatif à la font-size de l'élément root (définie par votre navigateur, la plupart sont à 16px).
- **em** → relatif à la font-size de l'élément actuel ou son parent le plus proche.
- **vw** → pourcentage de la largeur du viewport
- **vh** → pourcentage de la hauteur du viewport
- **cm & mm** → centimètres et millimètres (utilisé en impression)
- **px** → nombre de pixels

## Les unités angulaires

Utilisées pour définir des angles, notamment pour les rotations.

- **deg** → degrés ° (0 à 360°, peuvent être positifs ou négatifs pour inverser le sens)
- **grad** → grades (un cercle = 400 grades)
- **rad** → radians (un cercle =  $2\pi$  radians)
- **turn** → nombre de tour (un cercle = 1 tour)

## Les unités temporelles

Utilisées pour définir des temps, notamment pour les animations et transitions.

- **ms** → millisecondes
- **s** → secondes

# Les unités en pourcentages

Une valeur exprimée en pourcentage sera relative à une autre quantité, la plus souvent, celle de son parent. Par exemple :

- Un `<p>` dont la largeur serait de 50% occuperait la moitié de la largeur de son parent.
- Un `<p>` dont la taille de la police ferait 80% aurait une police dont la taille est à 80% la taille de la police de son parent.

# Les fonctions

- **calc** → permet de calculer une valeur avec n'importe quelles unités  
ex : calc( 100% - 15px)
- **min** → permet de sélectionner la valeur minimale parmi toutes les valeurs données  
ex : min(4rem, 5%, 2vw)
- **max** → permet de sélectionner la valeur maximale parmi celles données  
ex : min(25rem, 80%, 75vw)
- **clamp(valMin, val, valMax)** → permet d'écrire une valeur entre une borne max et min.  
C'est en gros l'équivalent d'un `max(valMin, min(val, valMax))`  
ex : clamp(10px, 64px, 80px) = max(10px, min(64px, 80px)) = max(10px, 64px) = 64px

# Les couleurs

# Types de valeurs des couleurs

Il existe plusieurs façons de renseigner une couleur.

- Couleur prédéfinie → il existe 140 couleurs déjà prédéfinies en CSS

 chartreuse

- Couleur en rgb(red, green, blue)

 rgb(150, 0, 150)

- Couleur en hsl(hue, saturation, lightness)

 hsl(150deg 30% 60%)

- Couleur en hexadécimal(r,g,b mais avec les valeurs en hexa)

 #26b48f

# Gérer l'opacité des couleurs

Pour gérer l'opacité d'un élément, il existe la propriété **opacity** mais **attention**, celle ci change l'opacité de tout le bloc, contenu inclus.

Pour ne changer que l'opacité de la couleur, il faudra rajouter une 3ème valeur à nos couleurs en rgb, hsl et hexa, qui représentera le % d'opacité.

```
p {  
    opacity: 0.5;  
    /* Met l'opacité de tout l'élément à 50% */  
    color : purplergb(150, 0, 150, 0.5);  
    color : #5beef188;  
    /* Met l'opacité de ces couleurs à 50% */  
}
```

# Rendu de base

# Le Flow Layout

Les éléments ont un rendu de base prédéfini. Nous pouvons cependant le changer avec la propriété **display**.

- block : apparaît sous l'élément précédent et au-dessus du suivant. (ex : p, div, hX)
- inline : se place sur la même ligne dans le rendu. (ex : img, a)
- none : n'est pas présent dans le rendu (sert à cacher des éléments)

```
ul li {  
    /* fera apparaître tous les  
     éléments de la liste en ligne */  
    display: inline;  
}
```

# Éléments flottants

# Les éléments flottants

Avant que les flexbox et les grid n'existent, pour positionner les éléments où on le souhaitait, on utilisait beaucoup les positions et les flottants.

Pour rendre les éléments flottants, on utilise la propriété **float** suivie de la valeur : left ou right. Pour arrêter cet habillage, on utilise la propriété **clear**.

```
p {  
    float: right; /* left */  
    clear: both; /* left | right */  
}
```

Nous verrons une autre solution pour placer nos éléments, beaucoup plus agréable à utiliser.

# Formatage du texte

# Changer la couleur du text

**color** → permet de changer la couleur du texte que ce soit en rgb, hsl, hexa ou une couleur prédéfinie.

```
p {  
    color : purple; //rgb(150, 0, 150);  
    color : #5beef1;  
}
```

# Changer la taille du texte

**font-size** → permet de changer la taille de la police avec une taille définie ou prédéfinie

```
p {  
    /* Avec unité */  
    font-size: 1rem;  
    /* Avec valeur prédéfinie */  
    font-size : xx-small;  
    /* ou x-small | smaller |  
     small | medium | large |  
     larger | x-large | xx-large */  
}
```

# Mettre en gras

## font-weight

```
p {  
    font-weight: bold;  
    /* lighter | normal | bolder */  
    /* centaines de 100 à 900 */  
}
```

# Mettre en italique

## font-style

```
p {  
    font-style: italic;  
}
```

# Alignement

**text-align** → alignement horizontal du text dans sa box

```
p {  
    text-align: left;  
    /* right | center | justify */  
}
```

**text-align-last** → alignement horizontal de la dernière ligne d'un texte

```
p {  
    text-align-last: left;  
    /* right | center | justify */  
}
```

# Alignement

**direction** → change la direction du texte

```
p {  
    direction: ltr; /* left-to-right */  
    direction: rtl; /* right-to-left */  
}
```

**vertical-align** → alignement vertical dans sa box

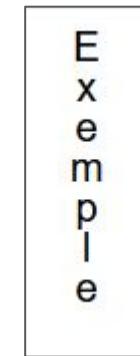
```
p {  
    vertical-align: baseline;  
    /* bottom | middle | sub | super | top */  
}
```

# Alignement

**writing-mode** → permet d'écrire horizontalement ou verticalement

**text-orientation** → permet de changer l'orientation du texte pour écrire de haut en bas par exemple

```
p {  
    writing-mode: vertical-lr;  
    text-orientation: upright;  
}
```



# Décoration

**text-decoration-line** → choisir le type de ligne de décoration

- underline : souligné
- overline : ligne au dessus
- line-through : barré
- overline underline : ligne au dessus et en dessous

**text-decoration-color** → choisir la couleur de la décoration

**text-decoration-thickness** → choisir l'épaisseur du trait

**text-decoration-style** → choisir le style de la ligne

(solid – , double = , dotted . . . , dashed - - -, wavy ~)

# Décoration

**text-decoration** → écriture raccourcie qui reprend toutes les propriétés précédentes

- text-decoration : line color style thickness

```
p {  
    text-decoration: underline chartreuse solid 2px ;  
}
```

Pour désactiver une décoration qui serait déjà mise (ex : les liens) :

```
p {  
    text-decoration: none;  
}
```

# Transformation

**text-transform** → permet de modifier la casse du texte

- uppercase : majuscules
- lowercase : minuscules
- capitalize : majuscule à chaque mot

```
p {  
    text-transform: lowercase;  
    /* uppercase | capitalize */  
}
```

# Espacement du texte

**text-indent** → espacement (indentation) de la première ligne du texte

**letter-spacing** → espacement entre chaque caractère

**line-height** → espacement entre les lignes

**word-spacing** → espacement entre les mots

**white-space** → permet d'activer ou non le wrapping du texte dans son élément

```
p {  
    text-indent: 1rem;  
    letter-spacing: 0.75rem;  
    line-height: 1.5rem;  
    word-spacing: 1.5rem;  
    white-space: nowrap;  
}
```

# Ombre du texte

**text-shadow** → permet de définir une ombre au texte

- **text-shadow : Xpx Ypx**
  - X : décalage horizontal
  - Y : décalage vertical
- **text-shadow : Xpx Ypx color**
- **text-shadow : Xpx Ypx Zpx color**
  - Z : blur

```
p {  
    text-shadow: 2px 2px;  
    text-shadow: 2px 2px chartreuse;  
    text-shadow: 2px 2px 2px chartreuse;  
}
```

# Dépassemment

Si le texte dépasse de son conteneur, nous pouvons utiliser overflow pour indiquer comment celui-ci doit se comporter

**overflow** : visible → le contenu est affiché en dehors de l'élément

**overflow** : hidden → le contenu est rogné si besoin

**overflow** : scroll : une scrollbar verticale et horizontale sont ajoutées

Sont aussi disponibles overflow-x (axe horizontal) et overflow-y (axe vertical).

# Dépassemment

Une fois l'overflow activé en hidden, nous pouvons préciser comment le texte se coupe quand il est caché grâce à

## **text-overflow :**

- clip → simplement découpé (par défaut)
- ellipsis → affiche ... où le texte est découpé
- "symbol" → affiche le symbol de votre choix où le texte est découpé

```
p {  
    overflow: hidden;  
    text-overflow: ellipsis;  
    text-overflow: "[ ... ]";  
}
```

# Dépassemment

**overflow-wrap** → permet de préciser si un mot peut être découpé ou doit rester entier (normal ou break-word)

**hyphens** → permet de préciser si le texte est coupé avec ou sans le symbole “-” (manual ou none)

```
p {  
    overflow-wrap: break-word;  
    hyphens: manual;  
}
```

# Changement la police

**font-family** → permet de spécifier la police souhaitée sur l'élément. Si plusieurs polices renseignées, il va prendre la première disponible.

```
p {  
    font-family: Arial;  
    font-family: Arial, Helvetica, sans-serif;  
}
```

Il est possible d'installer d'autres polices que celles de base. Pour cela, il existe 2 méthodes.

# Installer des polices customs

- 1) **En utilisant un lien** vers la police souhaitée : (ex: Google Font)

Si nous souhaitons, par exemple, installer la police Roboto, nous devrons placer ce code dans le HTML

Embed code in the <head> of your html

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400;1,500;1,700;1,900&display=swap" rel="stylesheet">
```

 Copy code

Ensuite, pour utiliser la police dans le CSS, c'est très simple

```
p {
  font-family: "Roboto";
}
```

# Installer des polices customs

2) **En téléchargeant** la police dans le projet :

Il faudra télécharger la police dans votre projet.

Ensuite, en utilisant la règle CSS **@font-face**, on indique au CSS où aller chercher la police et le nom qu'elle devra avoir :

```
@font-face {  
    font-family : "MaPoliceCustom";  
    src : url("../fonts/maPoliceCustom.ttf")  
}
```

Pour l'utiliser, on utilise le nom qu'on a indiqué dans la font-family de la règle :

```
p {  
    | font-family: "MaPoliceCustom";  
}
```

# Les fonds et images de fond

# Fond coloré et gradients

**background-color** → permet de modifier la couleur de fond

```
p {  
|   background-color: #6db96d;  
|}  
}
```

avec l'utilisation de la fonction linear-gradient, on peut créer un dégradé selon un angle précis :

```
p {  
|   background: rgb(227,189,118);  
|   background: linear-gradient(0deg, rgba(227,189,118,1) 0%, rgba(128,23,130,1) 65%, rgba(28,22,118,1) 100%);  
|}  
}
```

# Effet sur le fond

**backdrop-filter** → permet de rajouter un effet (ex: de flou, sepia, inversion de couleurs) sur le fond de l'élément

```
p {  
    background-color: #6db96d;  
    -webkit-backdrop-filter: blur(5px);  
    backdrop-filter: blur(5px);  
}
```

# Image de fond

Il y a deux méthodes pour mettre une image en fond d'un élément HTML.

1. Via le **CSS**, en utilisant background-image
2. Via le **HTML**, en utilisant une balise img et la propriété object-fit

Pour savoir quelle méthode vous devez utiliser, c'est simple.

Votre image a-t-elle un intérêt purement décoratif et est ignorée pour les personnes malvoyantes ? **1**

Est-elle utile pour le référencement et doit-elle être décrite pour les personnes malvoyantes ? **2**

# 1) Image en fond dans le CSS

```
p {  
    /* Lien vers l'image */  
    background-image: url("./images/ducks.jpg");  
    /* L'image va être déformée pour contenir l'élément */  
    background-size: contain;  
    /* L'image va être crop pour rentrer dans l'élément */  
    background-size: cover;  
    /* On peut ensuite se déplacer dans l'image pour la  
    placer où on le souhaite */  
    background-position: 10% 25%; /* horizontal vertical */  
}
```

## 2) Image en fond via le HTML + CSS

Dans le HTML :

```
<div>
  
</div>
```

Dans le CSS :

```
div {
  /* l'élément parent doit
  avoir une taille*/
  width: 500px;
  height: 200px;
}

div img {
  /* l'image doit remplir 100%
  de la taille du parent */
  width: 100%;
  height: 100%;
  object-fit: cover;
  object-position: 25% 10%;
}
```

# Modifier la taille

# Largeur

**width** → permet d'indiquer la largeur de l'élément.

- width : taille fixe
- max-width : la taille maximum que l'élément a le droit d'atteindre
- min-width : la taille minimum que l'élément doit faire

```
p {  
    /* Fera jusqu'à max 952px de large */  
    max-width: 952px;  
    /* Fera minimum 4rem de large mais  
     * peut faire plus */  
    min-width: 4rem;  
    /* Fera 42% de son parent */  
    width : 42%;  
}
```

# Hauteur

**height** → permet d'indiquer la hauteur de l'élément.

- height : taille fixe
- max-height : la taille maximum que l'élément a le droit d'atteindre
- min-height : la taille minimum que l'élément doit faire

```
p {  
    /* Fera jusqu'à max 200px de haut */  
    max-height: 200px;  
    /* Fera minimum 4rem de haut mais  
     * peut faire plus */  
    min-height: 4rem;  
    /* Fera 42% de son parent */  
    height : 42%;  
}
```

# Aspect-ratio

**aspect-ratio** → permet, peu importe la taille de l'élément, de garder le même format

```
p {  
    /* Carré */  
    aspect-ratio: 1/1;  
    /* 16/9 paysage */  
    aspect-ratio: 16/9;  
    /* 16/9 portrait */  
    aspect-ratio: 9/16;  
    /* format 4/3 paysage */  
    aspect-ratio: 4/3;  
}
```

# Redimensionner un élément

Pour rendre un élément redimensionnable (ou non) par l'utilisateur, on utilise la propriété **resize** suivie de la valeur :

- `none` (pas redimensionnable)
- `both` (redimensionnable sur les deux côtés)
- `horizontal` (redimensionnable horizontalement)
- `vertical` (redimensionnable verticalement)

Par exemple, cette propriété permet de "bloquer" la taille des éléments de type textarea que nous verrons plus tard.

# Les espaces

# Les margins

Ils servent à définir l'espacement à **l'extérieur** de l'élément

- margin-top : espacement du dessus
- margin-bottom : espacement du dessous
- margin-left : espacement de gauche
- margin-right : espacement de droite

Écriture raccourcie :

```
p {  
    /* même espace à gauche et à droite */  
    /* centre l'élément horizontalement */  
    margin : auto;  
    /* 2 valeurs : haut+bas gauche+droite */  
    margin : 2rem 4rem;  
    /* 4 valeurs : haut droite bas gauche */  
    margin : 4px 2% 1rem 6px;  
}
```

# Les paddings

Ils servent à définir l'espacement à **l'intérieur** de l'élément

- padding-top : espacement entre bord haut et contenu
- padding-bottom : espacement en bord bas et contenu
- padding-left : espacement entre bord gauche et contenu
- padding-right : espacement entre bord droit et contenu

Écriture raccourcie :

```
p {  
    /* 2 valeurs : haut+bas gauche+droite */  
    padding : 2rem 4rem;  
    /* 4 valeurs : haut droite bas gauche */  
    padding : 4px 2% 1rem 6px;  
}
```

# Les bordures

# Les bordures

Nous pouvons changer la bordure d'un côté seulement grâce à :

- **border-top** : bordure en haut
- **border-bottom** : bordure en bas
- **border-left** : bordure à gauche
- **border-right** : bordure à droite

Cependant, si nous souhaitons modifier la bordure de tous les côtés, il existe un raccourci :

- **border**

# Les bordures

Nous pouvons ensuite modifier les éléments suivants :

- **border-style** : style de la bordure (dotted, dashed, solid, double, groove etc...)
- **border-width** : taille de la bordure
- **border-color** : couleur de la bordure

Ou utiliser l'écriture raccourcie :

```
p {  
    /* width style color */  
    border : 2px solid tomato;  
}
```

# Les bordures autour des bordures

**outline** → permet de créer une bordure autour de la bordure. Elle s'écrit exactement comme les borders.

```
p {  
    /* width style color */  
    border : 2px solid tomato;  
    /* width style color */  
    outline: 2px dotted chartreuse;  
}
```

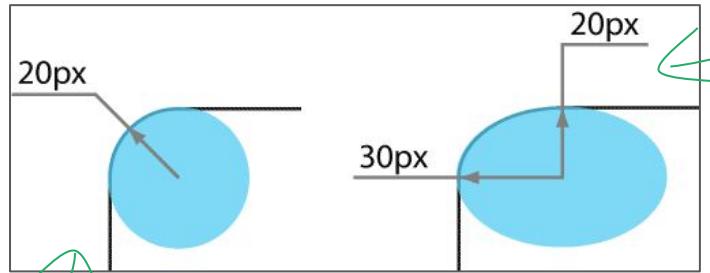
# Les bordures

Qu'il y ai une bordure ou non à notre élément, nous pouvons l'arrondir sur les bords grâce à **border-radius**

```
p {  
    /* Arrondir aux 4 coins */  
    border-radius: 2%;  
    /* Si l'élément est carré, fait un rond parfait */  
    border-radius: 50%;  
    /* 2 valeurs : diagonale.  
     valeur 1 : haut gauche + bas droite  
     valeur 2 : haut droite + bas gauche */  
    border-radius: 25% 10%;  
    /* 4 valeurs :  
     valeur 1 : haut gauche  
     valeur 2 : haut droite  
     valeur 3 : bas droite  
     valeur 4 : bas gauche */  
    border-radius: 10% 30% 50% 70%;  
}
```

# Les bordures : calcul

Par défaut, les bordures sont calculées comme sur le dessin de gauche. Nous pouvons mettre une fraction comme valeur pour avoir le calcul comme sur le dessin de droite.



A green curved arrow points from the top-right corner of the right-hand box in the previous diagram to a code block. The code block contains the following CSS rule:

```
p {  
    border-radius : 20px;  
    border-radius : 20px / 30px;  
}
```

# Calcul de la taille des éléments

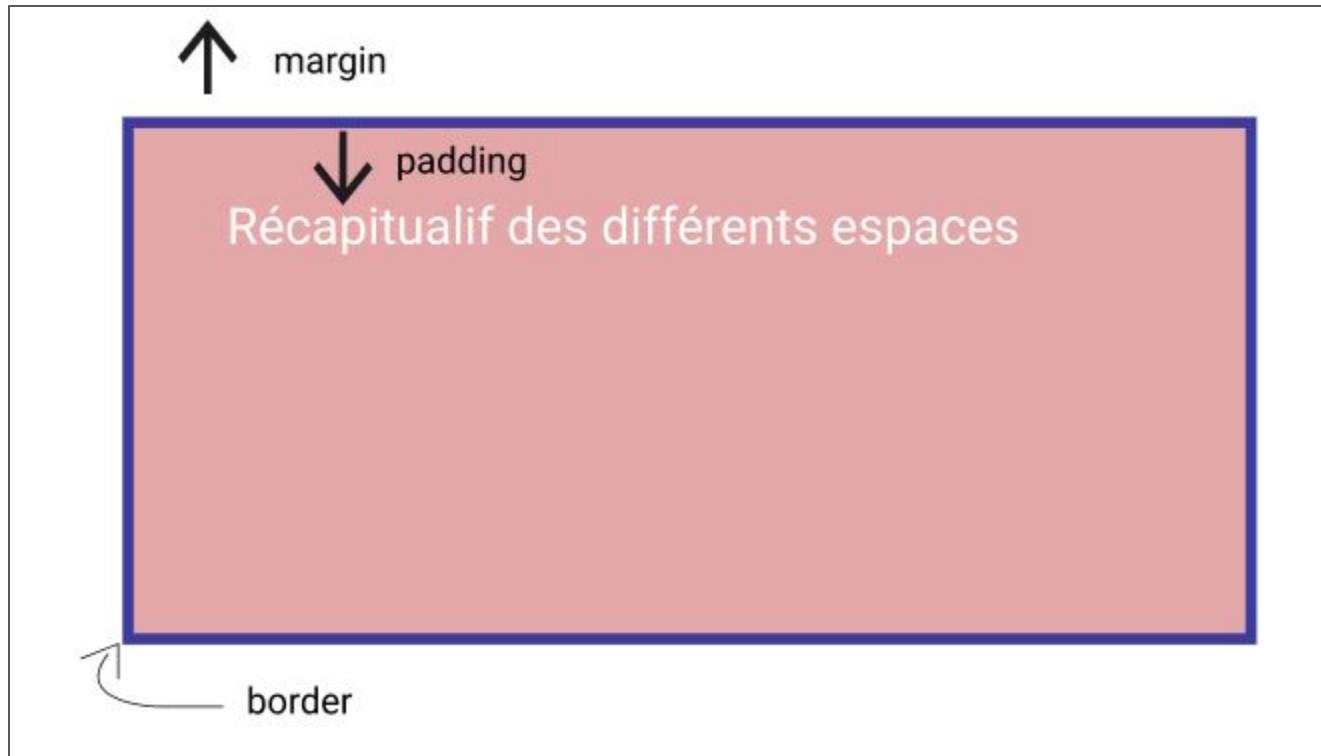
# Calcul de la taille des éléments

Attention, de base les bordures et les paddings ne sont pas pris en compte dans la taille de vos éléments. Si vous souhaitez qu'ils le soient, il faudra changer le **box-sizing**.

```
p {  
    width: 525px;  
    padding: 2px;  
    border : 3px solid red;  
    /* Valeur par défaut */  
    box-sizing: content-box;  
    /* Ici, l'élément fera au total :  
    525px  
    + 2 (padding left) + 2 (padding right)  
    + 3 (border left) + 3 (border right)  
    soit 535px */  
}
```

```
p {  
    width: 525px;  
    padding: 2px;  
    border : 3px solid red;  
  
    box-sizing: border-box;  
    /* Ici, l'élément fera au total 525px  
    | padding et bordures comprises :) */  
}
```

# Récap espacements



# Ombres sur les éléments

# Ombre externe

**box-shadow** → permet de définir une ombre

```
p {  
    /* décalageHorizontal décalageVertical color */  
    box-shadow: 10px 5px teal;  
    /* décalageHorizontal décalageVertical blur color */  
    box-shadow: 10px 5px 5px chartreuse;  
    /* décalageHorizontal décalageVertical blur spread color */  
    box-shadow: 10px 5px 5px 4px chartreuse;  
}
```

# Ombre interne

En ajoutant le mot-clef inset, l'ombre se met à l'intérieur de l'élément

```
p {  
|   box-shadow: inset 10px 5px 5px 4px □chartreuse;  
| }  
| }
```

# Cas particulier, les png et les svg

La box-shadow crée une ombre autour de l'élément html.

Dans le cas d'une image png par exemple, elle fera une ombre autour du rectangle qui contient l'image.

Si on veut créer une ombre à notre image et pas au cadre de notre image, il faudra utiliser **filter** (que nous verrons plus en détail plus tard) et **drop-shadow**.

```
p {  
    filter : drop-shadow(2px 2px 2px grey);  
}
```

# Les positions

# Les positions

**position** → permet de changer le positionnement d'un élément

- static : valeur par défaut
- relative : ne change pas le positionnement de l'élément mais le rend relatif pour d'autres
- absolute : se positionne par rapport au premier parent absolu qu'il rencontre (si aucun, c'est la page)
- fixed : se positionne à une position fixe déterminée
- sticky : quand l'élément aura rencontré la position déterminée, elle y restera collée

# Les positions

La propriété **position** n'est jamais utilisée seule. Il faudra déterminer les coordonnées du positionnement. Pour cela, nous avons accès à :

- top : positionnement par rapport au haut
- bottom : positionnement par rapport au bas
- left : positionnement par rapport à la gauche
- right : positionnement par rapport à la droite

# Les positions

```
.parent {  
    position: relative;  
}  
  
.enfant {  
    position: absolute;  
    top : 15px;  
    left : 25px;  
}
```

```
<div class="parent">  
    <div class="enfant">  
        </div>  
</div>
```

parent

15 px



25px

enfant

# Style de la scrollbar

# Style de la scrollbar

**scrollbar-width** → permet de définir la largeur de notre scrollbar soit avec la valeur prédéfinie `thin` pour la rendre fine soit une valeur qu'on aura définie nous même.

**scrollbar-color** → permet de changer la couleur de la scrollbar, prend deux valeurs

- la première "thumb" permet de changer la couleur de l'élément mobile
- la deuxième "track" permet de changer la couleur de fond

**scroll-behavior** → permet de gérer le rend du scroll via les ancre

```
html {  
    scrollbar-width: thin;  
    scrollbar-color: chartreuse darkblue;  
    scroll-behavior: smooth;  
}
```

# Les resetCSS

# Les fichiers de resetCSS

Comme nous l'avons vu dans l'introduction, les balises ont un style par défaut (les titres en gras, le margin présent sur certains éléments).

Pour remettre tout à 0, certaines personnes utilisent ce qu'on appelle des fichiers resetCSS qui enlèvent tout style CSS de base.

Quelques exemples ici :

- <https://meyerweb.com/eric/tools/css/reset/>
- <https://www.joshwcomeau.com/css/custom-css-reset/>

Une autre solution est d'utiliser Normalize <https://necolas.github.io/normalize.css/>

# Les variables CSS

# Les variables CSS

Dans un but de réutilisabilité et maintenabilité du code CSS, les variables ont été introduites. Elles vous permettent de stocker n'importe quelle valeur dans une "boîte" portant un nom. Pour l'utiliser, il suffira juste d'écrire le nom de votre variable.

Afin que vos variables soient disponibles dans tout votre projet, on les déclare généralement dans le root.

```
:root {  
    --couleur1 : #828dd4;  
    --couleur2 : #546e5a;  
    --cardSmall : 250px;  
    --cardMedium : 500px;  
}
```

```
p {  
    background-color: var(--couleur2);  
    color : var(--couleur1);  
}  
  
.card {  
    width: var(--cardMedium);  
}
```

# Les FlexBox

# Les Flexbox

Arrivées en 2009, elles ont révolutionné la mise en page des sites. Pendant des années, les seuls propriétés de mises en pages utilisées étaient les positions et les flottants.

Elles permettent de positionner des éléments selon deux axes et fonctionnent avec un système de "boîtes" (box) : un conteneur flexible et des éléments à l'intérieur.

Les éléments à l'intérieur seront placés selon les axes du conteneur et pourront se dilater ou se rétracter pour occuper l'espace octroyé.

# Les Flexbox

Afin de rendre un conteneur flexible, il faudra modifier la propriété **display** de l'élément pour celui soit en **flex**.

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



1 2 3

```
.conteneur {
    display: flex;
    background-color: #23ac67;
}

.element {
    background-color: #1c0d6d;
    color: whitesmoke;
    margin : 2px;
    padding: 2px;
}
```

# Les deux axes

# L'axe principal

Pour changer la direction de l'axe principal afin que nos éléments s'affichent en ligne ou en colonne, nous devrons utiliser la propriété **flex-direction** (sur le conteneur) qui peut prendre 4 valeurs différentes :

- row (valeur par défaut)
- row-reverse
- column
- column-reverse

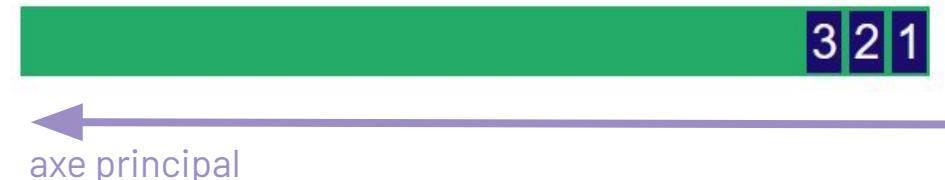
- row

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: row;  
}
```



- row-reverse

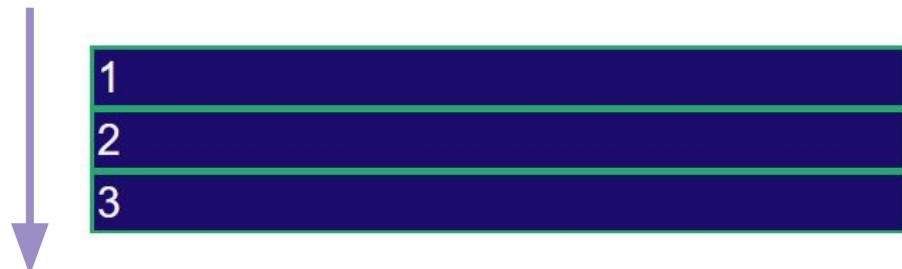
```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: row-reverse;  
}
```



- column

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: column;  
}
```

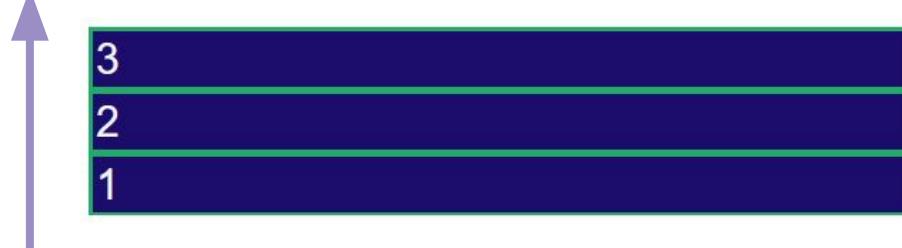
axe principal



- column-reverse

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: column-reverse;  
}
```

axe principal



# Alignement sur l'axe principal

Pour changer l'alignement des éléments sur l'axe principal, nous utiliserons la propriété **justify-content**.

Si l'axe principal est horizontal, on modifiera alors la position horizontale des éléments.  
Si l'axe principal est vertical, on modifiera alors la position verticale des éléments.

Les valeurs disponibles sont :

- (flex-)start (valeur par défaut)
- (flex-)end
- center
- space-between
- space-around
- space-evenly

# (flex-)start

row

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: row;  
    justify-content: start;|
```

1 2 3

column

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: column;  
    justify-content: start;|
```

1  
2  
3

# (flex-)end

row

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: row;  
    justify-content: end;|  
}
```



column

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: column;  
    justify-content: end;|  
}
```



# center

row

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: row;  
    justify-content: center;  
}
```



column

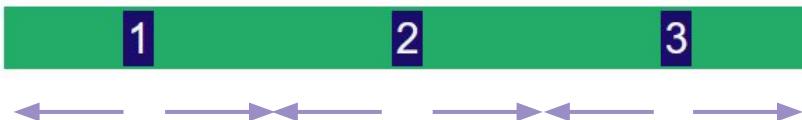
```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: column;  
    justify-content: center;  
}
```



# space-around

row

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: row;  
    justify-content: space-around;  
}
```



column

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: column;  
    justify-content: space-around;  
}
```



même espacement **autour** de chaque élément

# space-between

row

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: row;  
    justify-content: space-between;  
}
```



même espacement **entre** chaque élément

column

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: column;  
    justify-content: space-between;  
}
```



# space-evenly

row

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: row;  
    justify-content: space-evenly;  
}
```



même espacement de **chaque côté** de l'élément

column

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    flex-direction: column;  
    justify-content: space-evenly;  
}
```



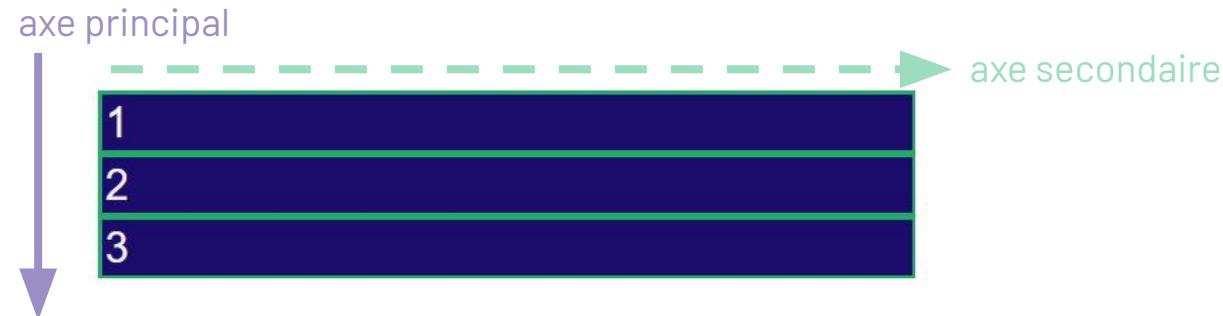
# L'axe secondaire

L'axe secondaire est, par défaut, l'inverse de l'axe principal. Si nous sommes en row, alors l'axe secondaire sera en colum et inversement.

row :



column :



# Alignement sur l'axe secondaire

Pour modifier l'alignement des éléments sur l'axe secondaire, il faudra utiliser la propriété **align-items**. Celui ci peut prendre les valeurs suivantes :

- (flex-)start
- (flex-)end
- stretch(valeur par défaut)
- center
- baseline

## (flex-)start

Pour ces exemples, le conteneur doit avoir une hauteur prédéfinie (pour column, elles devront avoir une largeur prédéfinie)

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    height: 100px; ←  
    flex-direction: row;  
    justify-content: start;  
    align-items: start;|
```



```
1 2 3
```

## (flex-)end

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    height: 100px;  
    flex-direction: row;  
    justify-content: start;  
    align-items: end;  
}
```



# center

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    height: 100px;  
    flex-direction: row;  
    justify-content: start;  
    align-items: center;  
}
```



# stretch

Les éléments, s'ils le peuvent, s'étirent pour remplir tout l'espace.

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    height: 100px;  
    flex-direction: row;  
    justify-content: start;  
    align-items: stretch;|
```



# baseline

Celui ci, ne sera effectif que si vos éléments n'ont pas la même taille. Nous avons augmenté la taille de police du 2ème élément pour l'exemple.

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    height: 100px;  
    flex-direction: row;  
    justify-content: start;  
    align-items: baseline;  
}
```



# Exception d'alignement pour un élément

Uniquement sur l'alignement secondaire, nous pouvons spécifier qu'un élément à une règle d'alignement différente par rapport aux autres. Ceci se fera avec la propriété **align-self** et s'utilise directement sur l'élément. Les valeurs sont identiques aux précédentes.

```
.conteneur {  
    display: flex;  
    background-color: #23ac67;  
    height: 100px;  
    flex-direction: row;  
    justify-content: start;  
    align-items: start;  
}  
.element:nth-child(2){  
    align-self: end;  
}
```

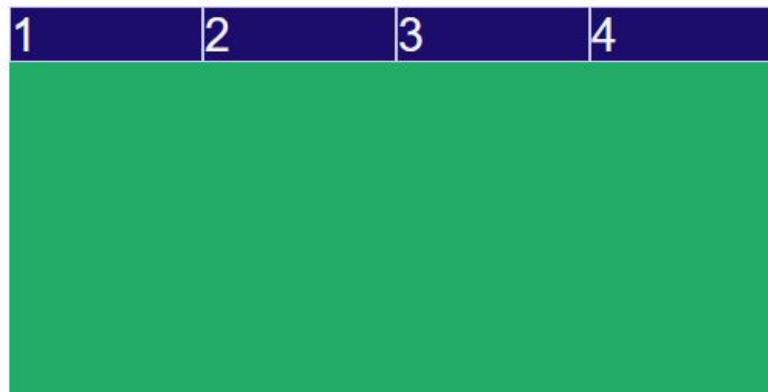


# Dépassemement des éléments

# Passage à la ligne/colonne

Par défaut, s'ils n'ont pas une taille fixe et s'ils peuvent être rétrécis (on verra par la suite comment changer ceci), les éléments vont se "tasser" pour essayer de rentrer dans la boîte.

```
.conteneur {  
    height: 200px;  
    width: 400px;  
    background-color: #23ac67;  
    display: flex;  
    justify-content: start;  
    align-items: start;  
}  
  
.element {  
    background-color: #1c0d6d;  
    color: whitesmoke;  
    border: 1px solid #cbc6eb ;  
    width: 110px;  
}
```



Comme on peut le voir, le conteneur fait 400px de large. Chaque élément fait 110px donc il n'y a pas la place pour les accueillir ( $4 \times 110\text{px} = 440\text{px}$ )  
Les éléments ont rétrécis pour entrer dans la boîte.

# Passage à la ligne/colonne

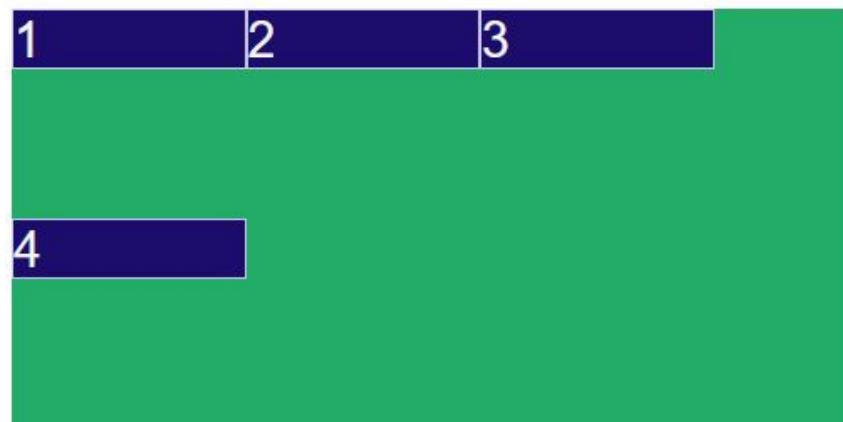
Pour que les éléments puissent passer à la ligne/colonne s'ils ne rentrent pas dans la boîte, nous devrons utiliser la propriété **flex-wrap** qui peut prendre les valeurs suivantes :

- wrap  
passage à la ligne/colonne
- wrap-reverse  
passage à la ligne/colonne en partant dans le sens inverse de l'axe secondaire
- nowrap (valeur par défaut)  
pas de passage à la ligne/colonne

# Passage à la ligne/colonne

```
.conteneur {  
    height: 200px;  
    width: 400px;  
    background-color: #23ac67;  
    display: flex;  
    justify-content: start;  
    align-items: start;  
    flex-wrap: wrap;  
}  
  
.element {  
    background-color: #1c0d6d;  
    color: whitesmoke;  
    border: 1px solid #cbc6eb ;  
    width: 110px;  
}
```

Et hop ! Le 4ème élément retourne à la ligne !



# Alignement des lignes/colonnes

Une fois le wrapping activé, la propriété align-items, qui nous servait à aligner les éléments sur l'axe secondaire ne sert plus.

En effet, nous n'avons plus d'éléments à aligner mais des lignes/colonnes. Nous allons alors devoir utiliser **align-content** qui prend les valeurs suivantes :

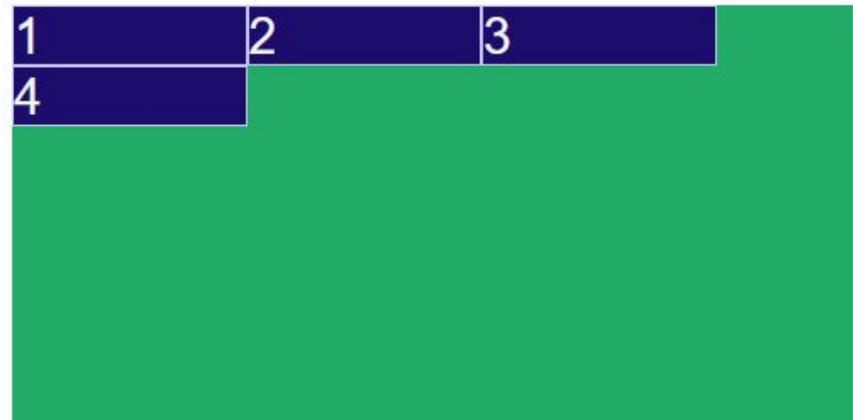
- (flex-)start
- (flex-)end
- center
- stretch
- space-around
- space-between
- space-evenly

Attention, pour que cette propriété fonctionne, votre conteneur doit être plus grand que la taille totale des lignes/colonnes générées par le wrap.

# (flex-)start

Aligne toutes les lignes/colonnes au début

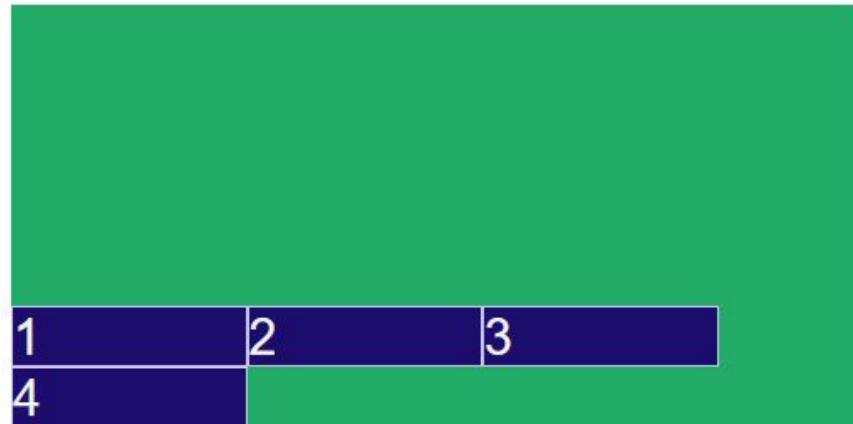
```
.conteneur {  
    height: 200px;  
    width: 400px;  
    background-color: #23ac67;  
    display: flex;  
    justify-content: start;  
    flex-wrap: wrap;  
    align-content: start;  
}
```



## (flex-)end

Aligne toutes les lignes/colonnes à la fin

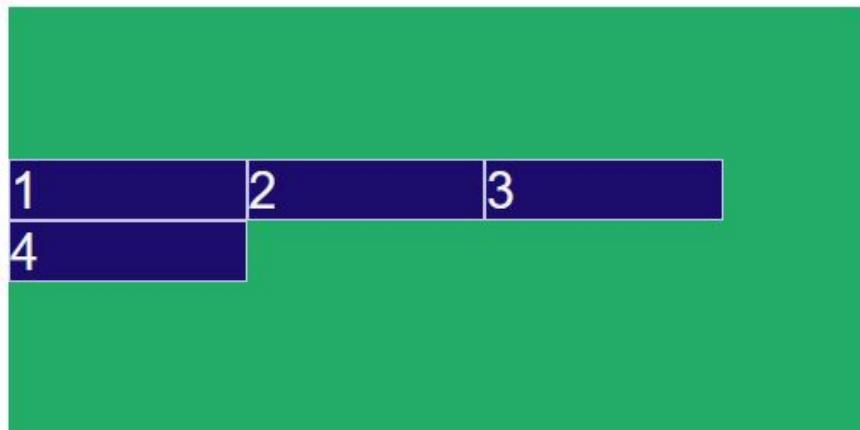
```
.conteneur {  
    height: 200px;  
    width: 400px;  
    background-color: #23ac67;  
    display: flex;  
    justify-content: start;  
    flex-wrap: wrap;  
    align-content: end;  
}
```



# center

Aligne toutes les lignes/colonnes au milieu

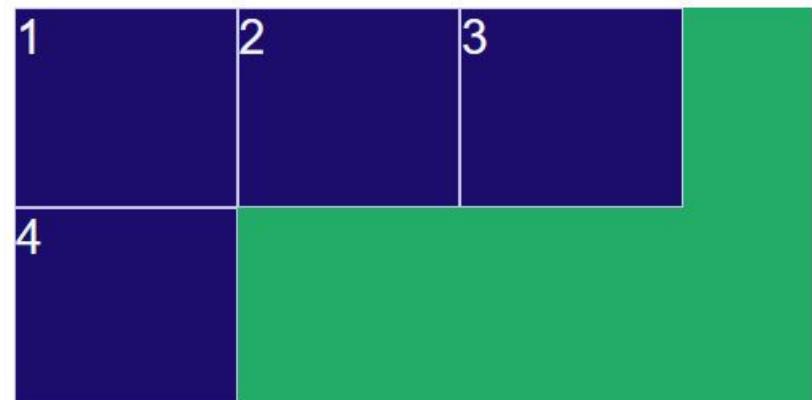
```
.conteneur {  
    height: 200px;  
    width: 400px;  
    background-color: #23ac67;  
    display: flex;  
    justify-content: start;  
    flex-wrap: wrap;  
    align-content: center;  
}
```



# stretch

Occupe tout l'espace disponible

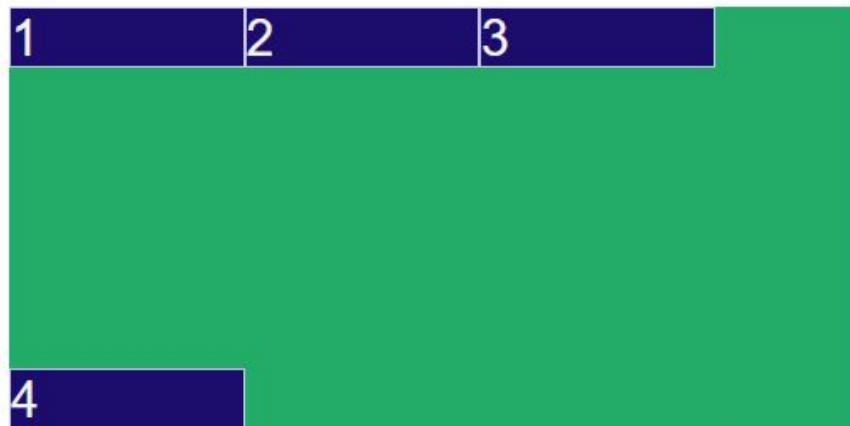
```
.conteneur {  
    height: 200px;  
    width: 400px;  
    background-color: #23ac67;  
    display: flex;  
    justify-content: start;  
    flex-wrap: wrap;  
    align-content: stretch;  
}
```



# space-between

Mettra le même espace entre chaque lignes/colonnes

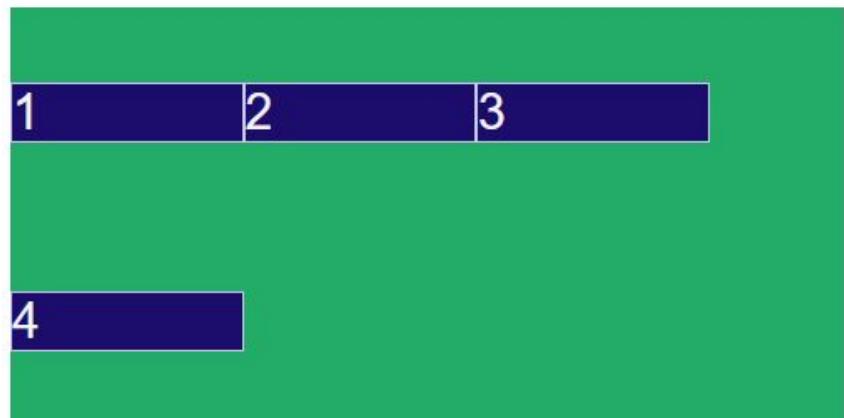
```
.conteneur {  
    height: 200px;  
    width: 400px;  
    background-color: #23ac67;  
    display: flex;  
    justify-content: start;  
    flex-wrap: wrap;  
    align-content: space-between;  
}
```



# space-around

Mettra le même espace autour de chaque lignes/colonnes

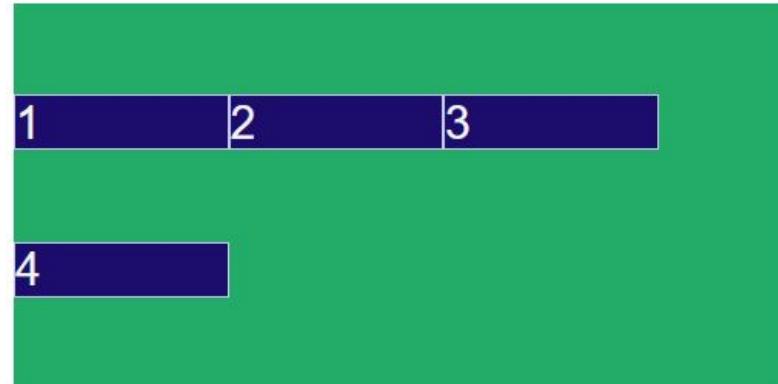
```
.conteneur {  
    height: 200px;  
    width: 400px;  
    background-color: #23ac67;  
    display: flex;  
    justify-content: start;  
    flex-wrap: wrap;  
    align-content: space-around;}
```



## space-evenly

Mettra le même espace de chaque côté des lignes/colonnes

```
.conteneur {  
    height: 200px;  
    width: 400px;  
    background-color: #23ac67;  
    display: flex;  
    justify-content: start;  
    flex-wrap: wrap;  
    align-content: space-evenly;  
}
```

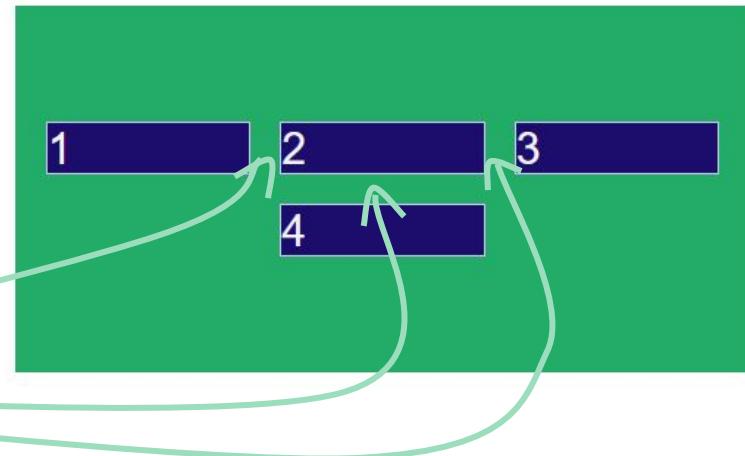


Espacement défini entre les  
éléments

# Espace défini entre les éléments

Quand on travaille en flex, on va éviter de jouer avec les margin pour espacer nos éléments, lignes et colonnes. Nous avons à notre disposition la propriété **gap** qui nous permet de définir un espace entre nos éléments sur l'axe principal et nos lignes/colonnes sur l'axe secondaire quand le wrapping est actif.

```
.conteneur {  
    height: 200px;  
    width: 400px;  
    background-color: #23ac67;  
    display: flex;  
    justify-content: center;  
    flex-wrap: wrap;  
    align-content: center;  
    gap: 1rem; |}
```



# Dimension dynamique des éléments

# Contrôler la taille des éléments

Pour contrôler la taille des éléments sur l'axe principal au sein d'un conteneur, nous avons la possibilité de définir trois propriétés :

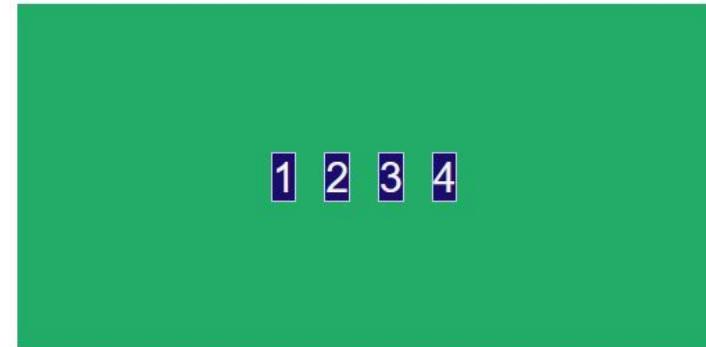
- **flex-basis** → permet de définir la taille occupée par l'élément
- **flex-grow** → permet de définir si un élément peut être agrandi
- **flex-shrink** → permet de définir si un élément peut être rétréci

## flex-basis

Cette propriété permet de définir la taille occupée par un élément. Elle peut prendre comme valeurs :

- une taille(px, em, % etc...)
- auto (valeur par défaut, les éléments occupent la place disponible)
- content (calcul auto par rapport au contenu de l'élément)
- ...

```
.element {  
    background-color: #1c0d6d;  
    color: whitesmoke;  
    border: 1px solid #cbc6eb ;  
    /* width: 110px; */  
    flex-basis: content;  
}
```



## flex-grow

Cette propriété permet de définir si un élément peut être agrandi et de quelle façon. Elle attend comme valeur un nombre entier positif.

- 0 → l'élément ne peut pas être agrandi (valeur par défaut)
- > 0 → l'élément peut être agrandi et le nombre définit le facteur d'expansion de l'élément, c'est à dire la quantité d'espace restant qu'elle peut occuper.

```
.element {  
    background-color: #1c0d6d;  
    color: whitesmoke;  
    border: 1px solid #cbc6eb ;  
    flex-grow: 1;  
}
```

## flex-grow (exemples)

Tous les éléments ont un flex-grow identique :



Le deuxième a un flex-grow de 2 et les autres de 1 :



Le deuxième a un flex-grow de 2, le troisième de 3 et les autres de 1 :



# flex-shrink

Cette propriété permet de définir si un élément peut être rétréci et de quelle façon. Elle attend comme valeur un nombre entier positif.

- 0 → l'élément ne peut pas être rétréci
- > 0 → l'élément peut être rétréci et le nombre définit le facteur de rétraction de l'élément, c'est à dire la quantité de compression pour remplir l'espace. (par défaut la valeur est 1)

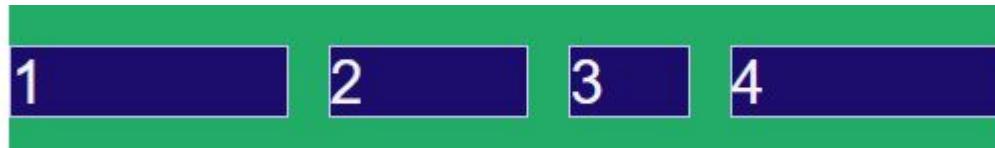
```
.element {  
    background-color: #1c0d6d;  
    color: whitesmoke;  
    border: 1px solid #cbc6eb ;  
    flex-shrink: 1;  
}
```

## flex-shrink (exemples)

Tous les éléments ont un flex-shrink de 0, non redimensionnables :



Tous les éléments ont un flex-shrink de 0 sauf le deuxième qui est à 1 et le troisième qui est à 2 :



## Écriture raccourcie :

Il est possible de combiner ces trois propriétés en une seule s'appelant **flex**. Celle-ci fonctionne avec deux types de valeurs:

- 3 valeurs, dans l'ordre : flex-grow, flex-shrink, flex-basis

```
flex : 1 2 110px;
```

- Une valeur prédéfinie :
  - initial : 0 1 auto;
  - auto : 1 1 auto;
  - none : 0 0 auto;

# Ordre d'affichage des éléments

# Ordre d'affichage des éléments

La propriété **order** permet de définir l'ordre dans lequel vont être affichés les éléments. Celle-ci attend un nombre entier. Les éléments sont ensuite classés dans l'ordre croissant des order définis.

Par défaut, order vaut 0.

```
.element:nth-child(1) {  
    order : 5;  
}  
.element:nth-child(2) {  
    order : 1;  
}  
.element:nth-child(3) {  
    order : 3;  
}  
/* élément 4 intouché donc 0 */
```



# Le Responsive Design

# Le Responsive Design

C'est une approche visant à ce que le visuel de votre site s'adapte en fonction de la taille de l'écran. En effet, si vous prenez attention, la plupart des sites ont un rendu différent et adaptatif selon qu'on soit sur un ordinateur ou un téléphone mobile par exemple.

Pour tester votre site rapidement en mobile, vous pouvez activer la vue responsive de l'outil de développement du navigateur. Attention, ce n'est qu'une simulation, vous devrez tout de même tester vos sites sur de vrais devices.



ex: Firefox

# Le viewport

# Le viewport

Une meta viewport est disponible pour nous aider dans la gestion du responsive. Elle permet de définir plusieurs choses sur ce dernier.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Par défaut, elle contient les infos suivantes :

- width=device-width → permet d'indiquer que la largeur du viewport est celle du device
- initial-scale=1 → permet de définir le niveau de zoom lors du premier chargement de la page

# Le viewport

Vous pouvez rajouter les options suivantes :

- user-scalable : permet d'autoriser l'utilisateur à zoomer ou dézoomer sur votre site (attention à l'accessibilité cependant)
- minimum-scale & maximum-scale : permettent de définir un zoom minimum et maximum

# Les media queries

# Les media queries

Elles permettent de modifier l'apparence du site en fonction du type d'appareil, de sa taille, son orientation, son thème etc...

Pour préciser le **type**, il faudra l'écrire après la règle **@media** :

- @media all → tous les types
- @media screen → les écrans
- @media print → l'impression
- @media speech → outils de synthèse vocale

# Les media queries

Pour modifier une caractéristique, il faudra l'ajouter entre parenthèses @media (carac). Il en existe énormément, en voici quelques unes :

- width → largeur de la zone d'affichage
- orientation → écran en mode paysage ou portrait
- prefers-color-scheme → thème du navigateur (light ou dark)

[voir la liste complète](#)

Vous pouvez rajouter des opérateurs logiques pour combiner des types et des caractéristiques. (and - not - only - , )

# Les media queries (exemple)

```
<nav class="nav-desktop">  
    ←— navbar sur desktop avec  
    | des liens vers les pages →  
</nav>  
<nav class="nav-mobile">  
    ←— navbar sur mobile avec  
    | le fameux menu burger →  
</nav>
```



```
/* style par défaut : mobile */  
.nav-mobile {  
    display: flex; /* visible */  
}  
.nav-desktop {  
    display: none; /* caché */  
}  
  
/* style à partir de 640px */  
@media screen and (min-width:640px) {  
.nav-mobile {  
    display: none; /* caché */  
}  
.nav-desktop {  
    display: flex; /* visible */  
}
```

# Les formulaires

# Les formulaires

Très utilisés, ils sont un moyen d'interagir avec l'utilisateur et de récupérer des informations encodées par ce dernier. La plupart du temps, ces données seront envoyées à un serveur, qui va les traiter.

Souvent, du JavaScript sera ajouté pour vérifier la validité des données et afficher des messages d'erreur en conséquence.

Un formulaire est composé de différents éléments comme des champs, des labels, des boutons.

# Créer son formulaire

Pour créer un formulaire, on devra utiliser la balise **<form>** et placer à l'intérieur tous les champs, labels et boutons du formulaire.

Voici quelques uns des attributs disponibles :

- **name** : nous permettra de nommer le formulaire pour le récupérer en JS
- **method** : indique la manière dont les données seront envoyées
- **action** : indique l'adresse du serveur qui va traiter le formulaire
- **enctype** : détermine sous quel format sont envoyées les données

```
<form method="post" action="monScript.php">  
    ←— Contenu du formulaire —→  
</form>
```

# Méthode d'envoi des données

Dans l'attribut **method**, deux principales valeurs sont possibles :

- **get** → envoie les données via l'url du site
  - Limité à 255 caractères maximum
  - Les données sont visibles dans l'adresse
  - C'est la valeur par défaut si on ne met pas l'attribut ou qu'on n'envoie pas les données nous même avec du JS.
- **post** → un objet est créé avec les données
  - Pas de limite de caractères
  - Invisible dans le navigateur
  - Plus simple à manipuler

# Les champs

# Les types de champs

Tout ce qui permet à l'utilisateur d'entrer une donnée est appelé un champ (ou control). Il en existe plusieurs sortes :

- **l'input** → de plusieurs types (text, date, case à cocher etc), c'est le plus commun
- le **textarea** → il permet de définir une grosse zone de texte (un message par ex)
- le **select** → permet de sélectionner une (ou des) donnée(s) parmi plusieurs dans un menu déroulant

Pour tous, il sera important de définir leur nom avec l'attribut **name**. C'est ce qui permettra d'associer la valeur envoyée par le formulaire avec un nom.

# Relier un champ à son label

Souvent, un champ sera associé à un texte, qui permet d'indiquer ce qui est attendu. Ce texte s'appelle un label. Au clic sur celui ci, nous aurons le focus sur le champ associé.



Pour lier une balise label à un champ (peu importe son type), il faudra ajouter l'attribut for et sa valeur devra être l'id du champ qu'on veut lui associer.

```
<label for="nom">Nom</label>
<input id="nom" name="lastname" type="text">
```

# Les inputs

Le champ de type input permet d'encoder une (ou des) donnée(s). Il existe énormément de types d'input, nous allons voir les plus utilisés. Vous trouverez la liste complète [ici](#).

- **text** → champ de saisie textuelle sur une seule ligne

```
<input id="nom" name="lastname" type="text">
```

Hello World

# Les inputs

- **number** → champ permettant de saisir un nombre

```
<input id="note" name="result" type="number">
```



A screenshot of a web browser showing a number input field. The field contains the value "42" and has up and down arrow buttons for incrementing or decrementing the value.

- **email** → champ textuel qui permet de saisir une adresse email. Une validation sera alors effectuée pour savoir si le format correspond bien à un email.

```
<input id="email" name="email" type="email">
```

# Les inputs

- **password** → champ de saisie textuelle dans lequel chaque caractère va être visuellement remplacé par un symbole.

```
<input id="pwd" name="password" type="password">
```



.....

- **tel** → champ de saisie textuelle pour entrer un numéro de téléphone. Son intérêt est d'ouvrir le clavier numérique sur mobile.

```
<input id="tel" name="phone" type="tel">
```

# Les inputs

- **file** → permet d'envoyer un ou plusieurs fichiers depuis son appareil.

```
<input id="id-card" name="idCard" type="file" multiple accept=".pdf">
```

Si on utilise ce type d'input dans notre formulaire, celui ci devra alors être envoyé sous une autre forme. Pour cela, nous devrons mettre la valeur **multipart/form-data** dans l'attribut **enctype** de notre formulaire.

Avec l'attribut **accept**, nous pouvons préciser quels types de fichiers sont acceptés.

# Les inputs

- **date** → champ de saisie permettant d'encoder une date

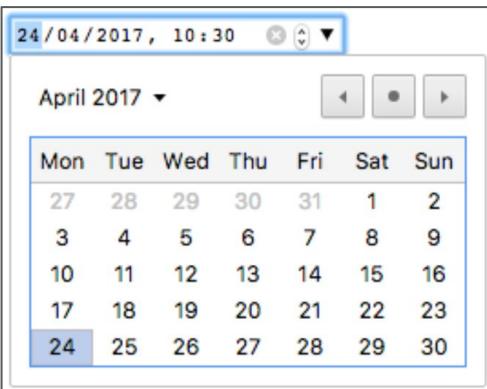
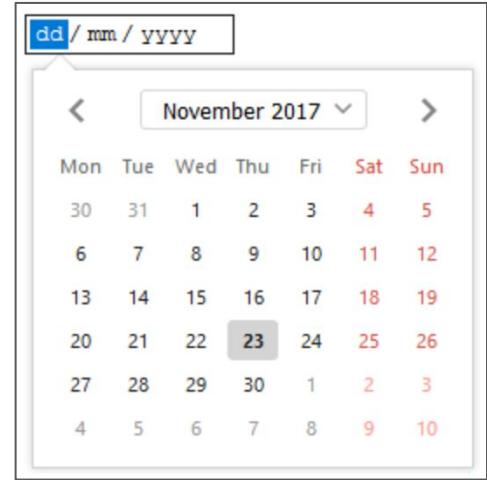
```
<input id="naissance" name="birthdate" type="date">
```

- **time** → champ de saisie permettant d'encoder une heure

```
<input id="rdv" name="rdv" type="time">
```

- **datetime-local** → combinaison des deux précédents

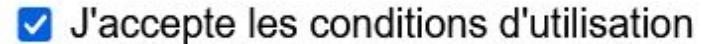
```
<input id="rdv" name="rdv" type="datetime-local">
```



# Les inputs

- **checkbox** → champ permettant de saisir un état (vrai/faux).

```
<input id="conditions" type="checkbox" name="conditions">
<label for="conditios">J'accepte les conditions d'utilisation</label>
```



- **radio** → champ permettant de faire un choix unique parmi plusieurs éléments.  
Tous les radios reliés devront porter la même valeur pour l'attribut name et posséder une value.

Homme  Femme  Autre

```
<input id="male" name="gender" value="h" type="radio">
<label for="male">Homme</label>
<input id="female" name="gender" value="f" type="radio">
<label for="female">Femme</label>
<input id="other" name="gender" value="o" type="radio">
<label for="other">Autre</label>
```

# Le textarea

Pour insérer un champ qui permet de saisir un long texte sur plusieurs lignes, on utilisera le **textarea**. Les attributs cols et rows serviront à définir le nombre de lignes qui apparaissent dans le champs sans devoir scroller.

```
<textarea name="message" id="msg" rows="5" cols="40">  
</textarea>
```

Par défaut, cet élément est redimensionnable par l'utilisateur. Nous pourrons le désactiver avec la propriété resize en CSS.

Je vous écris ce message parce que

# Le select

Pour insérer une liste déroulante, on utilisera **select**. Chaque choix possible sera mis dans une balise **option**.

```
<select name="color" id="color">
    <option selected hidden value="">
        Choisissez votre couleur
    </option>
    <option value="blue">Bleu</option>
    <option value="red">Rouge</option>
    <option value="green">Vert</option>
    <option value="purple">Violet</option>
    <option value="orange">Orange</option>
</select>
```

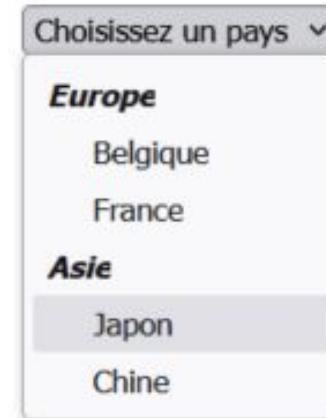


L'attribut `hidden` permet de ne pas afficher une option dans la liste déroulante, le placeholder n'étant pas disponible pour ce champ.

# Le select

On peut ranger nos options dans des groupements grâce à **optgroup**.

```
<select name="country" id="country">
    <option selected hidden value="">
        Choisissez un pays
    </option>
    <optgroup label="Europe">
        <option value="BE">Belgique</option>
        <option value="FR">France</option>
    </optgroup>
    <optgroup label="Asie">
        <option value="JP">Japon</option>
        <option value="CN">Chine</option>
    </optgroup>
</select>
```



# Regrouper des champs

Nous pouvons regrouper plusieurs champs avec la même thématique dans une balise appelée **fieldset**. Si l'on souhaite y ajouter une légende/titre, on ajoutera la balise **legend** dans notre fieldset.

```
<fieldset>
    <legend>Adresse :</legend>
    <label for="street">Rue</label>
    <input id="street" name="street" type="text">
    <label for="number">Numéro/Boite</label>
    <input id="number" name="number" type="text">
    ← ... →
</fieldset>
```

# Les boutons

# Les boutons

Il y a deux façons d'ajouter des boutons dans notre formulaire et 3 types de boutons.

- Soit avec la balise **input** et nous préciserons dans type, les trois suivants
  - **submit** → un bouton qui envoie le formulaire
  - **button** → un bouton simple, sans déclenchement de l'envoi du formulaire
  - **reset** → un bouton qui permet de remettre à 0 les données du formulaire

```
<input type="submit" value="Envoyer">
<input type="button" value="Version anglaise">
<input type="reset" value="Effacer">
```

# Les boutons

- Soit avec la balise **button** mais attention, si on ne précise pas le type, ce sera un submit par défaut !

```
<button type="submit"> Envoyer </button>
<button type="button"> Version anglaise </button>
<button type="reset"> Effacer </button>
```

# Les attributs

## Quelques attributs souvent utilisés

- placeholder → Permet d'ajouter une indication à l'utilisateur sur ce qu'il doit entrer comme donnée. Disparaît dès qu'on écrit dans le champ.
- required → Permet d'indiquer qu'un champ doit être complété pour que l'utilisateur puisse envoyer le formulaire

```
<input name="lastname" type="text" placeholder="ex:Dupont" required>
```

## Quelques attributs souvent utilisés

- `readonly` → L'utilisateur peut avoir le focus sur le champ mais ne pourra pas modifier la valeur du champ. La donnée est toutefois soumise.
- `disabled` → L'utilisateur ne peut pas interagir avec le champ. La donnée n'est pas soumise.
- `pattern` → Permet de définir une "RegEx" qui permettra de vérifier si la valeur saisie respecte un certain format.
- `autocomplete` → Permet d'indiquer si ce champ peut être complété automatiquement par le navigateur.

## Quelques attributs souvent utilisés

- minlength / maxlength → Permet de définir la taille minimum et maximum que la chaîne de caractères saisie peut avoir.
- min/max → Permet de définir la valeur minimum et maximum que l'utilisateur peut encoder (number, date, ...)
- checked → Permet de déjà cocher une checkbox ou un radio au chargement de la page
- selected → Permet de sélectionner une des options au chargement de la page

## Quelques attributs souvent utilisés

- list → Combiné à la balise datalist, permet d'activer l'autocomplétion avec une liste fournie
- multiple → Permet de sélectionner plusieurs valeurs (file, email, select)
- step → Permet de définir un pas d'incrémentation (number, date, ...)
- spellcheck → Permet d'activer la vérification orthographique et grammaticale du contenu

# Quelques attributs souvent utilisés

- autofocus → Permet d'indiquer quel l'élément a le focus au chargement de la page
- tabindex → Permet de redéfinir l'ordre de navigation au clavier (avec la touche tabulation). Par défaut, c'est l'ordre dans lequel ils sont dans l'HTML (haut vers bas)
  - Valeur positive → classés par ordre croissant
  - 0 → permet d'activer le tabindex sur des éléments qui ne l'ont pas naturellement (ordre HTML)
  - Valeur négative → rend le champ inaccessible à la tabulation

**Remarque :** Même si quelques balises et attributs permettent de vérifier les données utilisateurs, on fait souvent une vérification en JS et il faudra **TOUJOURS** effectuer une vérification côté serveur.

# Les tableaux

# Les tableaux

Très utilisés dans une très lointaine époque (les débuts de HTML/CSS), les tableaux avaient été détournés pour créer de la mise en page.

Heureusement, avec l'arrivée des FlexBox et des Grid, les tableaux ont repris leur rôle d'origine : afficher des données.

Ils permettent de structurer une liste de données par exemple ou un panier sur un e-shop.

# Base des tableaux

# Base des tableaux

Pour créer un tableau nous aurons besoin de définir 3 choses :

- le tableau grâce à la balise **table**
- les lignes grâce à la balise **tr** (table row)
- les cellules grâce à la balise **td** (table datacell)

The diagram shows a 2x3 table structure. It consists of two rows, each containing three cells. The entire table is enclosed in a blue border. The first cell of the first row is highlighted with a red box. The second cell of the first row is highlighted with a red box. The third cell of the first row is highlighted with a red box. A red vertical line points from the label "ligne" to the second cell of the first row. A black vertical line points from the label "cellule" to the third cell of the first row.

Ligne 1 - Cellule 1	Ligne 1 - Cellule 2	Ligne 1 - Cellule 3
Ligne 2 - Cellule 1	Ligne 2 - Cellule 2	Ligne 2 - Cellule 3

tableau

ligne

cellule

# Base des tableaux

```
<table>
    <tr>
        <td>Ligne 1 - Cellule 1</td>
        <td>Ligne 1 - Cellule 2</td>
        <td>Ligne 1 - Cellule 3</td>
    </tr>
    <tr>
        <td>Ligne 2 - Cellule 1</td>
        <td>Ligne 2 - Cellule 2</td>
        <td>Ligne 2 - Cellule 3</td>
    </tr>
</table>
```

# Structure des tableaux

# Structure des tableaux

On peut structurer nos tableaux en 3 parties :

- **thead** → En-tête du tableau, là où on met le descriptif des colonnes par exemple (optionnel, se met au dessus du tableau par défaut)
- **tbody** → Corps du tableau, là où on met les données (requis, le navigateur en crée un si oubli mais peut causer des soucis)
- **tfoot** → Pied du tableau, là où on met une somme totale pour un panier par exemple (optionnel, se met au dessous du tableau par défaut)

# Structure des tableaux

Nous pouvons également rajouter :

- **th** → cellule (td) spéciale qui indique que c'est une cellule d'en-tête
- **caption** → permet d'ajouter une légende à notre tableau

# Structure des tableaux

Votre panier :

Article	Quantité	Prix
Cheese Burger	1	6€
Prix total :	6€	

```
<table>
  <caption>Votre panier :</caption>
  <thead>
    <tr>
      <th>Article</th>
      <th>Quantité</th>
      <th>Prix</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Cheese Burger</td>
      <td>1</td>
      <td>6€</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td>Prix total :</td>
      <td>6€</td>
    </tr>
  </tfoot>
</table>
```

# Fusion de cellules

Pour fusionner des cellules **horizontalement**, nous devrons utiliser l'attribut **colspan** en précisant le nombres de cellules à fusionner.

Pour fusionner des cellules **verticalement**, nous devrons utiliser l'attribut **rowspan** en précisant le nombres de cellules à fusionner.

```
<table>
  <tbody>
    <tr>
      <td rowspan="2">Fusion verticale</td>
      <td>Pas fusion</td>
      <td>Pas fusion</td>
    </tr>
    <tr>
      <!-- première td occupée par la fusion verticale -->
      <td colspan="2">Fusion horizontale</td>
    </tr>
  </tbody>
</table>
```

Fusion verticale	Pas fusion	Pas fusion
	Fusion horizontale	

# Style des tableaux

# Bordures

Par défaut, les tableaux n'ont pas de bordure. Il faudra en ajouter une avec la propriété que l'on connaît déjà : **border**.

Par contre, on obtiendra le résultat suivant :

Pour que les bordures se rejoignent pour ne former qu'une seule bordure, il faudra utiliser la propriété **border-collapse : collapse**

Résumé	
Nom complet	Abréviation
HyperText Markup Language	HTML
Cascading Style Sheets	CSS

# Caption

On peut modifier la position de la légende pour qu'elle se place en dessous ou au dessus du tableau grâce à **caption-side : top (ou bottom)**

## Largeur des colonnes

Par défaut, la largeur des colonnes est calculée automatiquement en fonction du contenu à l'intérieur, ce qui peut entraîner des colonnes plus larges que d'autres. Si vous souhaitez mettre une taille définie à vos colonnes, vous pourrez rajouter **table-layout : fixed** (par défaut auto) sur la table et définir une taille.

Remarque : Pour des raisons évidentes, le margin ne fonctionne pas à l'intérieur d'un tableau.

# Transformations et animations

# Transformations

# Transform

La propriété **transform** de CSS permet d'appliquer une transformation à l'élément ciblé.  
(attention les éléments inline et les colonnes de tableaux ne peuvent pas être transformés)

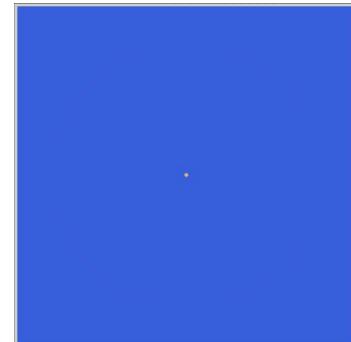
Sont disponibles plusieurs types de transformations : des rotations, des redimensionnements, des déplacements etc...

# Les rotations

- `rotate(angle)` → rotation d'un élément selon un angle donné (sens des aiguilles d'une montre) en 2d
- `rotate3d(x, y, z, angle)` → rotation d'un élément selon un angle donné sur un axe 3d
- `rotateX(angle)` → rotation selon un axe horizontal
- `rotateY(angle)` → rotation selon un axe vertical
- `rotateZ(angle)` → rotation selon un axe z

# Les redimensionnements

- `scale(x)`, `scale(x, y)` → redimensionne un élément en X (et Y)
- `scale3d(x, y, z)` → redimensionne un élément en X, Y et Z
- `scaleX(x)` → redimensionne un élément en X
- `scaleY(x)` → redimensionne un élément en Y
- `scaleZ(x)` → redimensionne un élément en Z



# Les distorsions

- `skew(x)`, `skew(x,y)` → déforme sur l'axe X (et Y) selon un angle donné
- `skewX(x)` → déforme sur l'axe X selon un angle
- `skewY(y)` → déforme sur l'axe Y selon un angle



# Les translations

- `translate(x)`, `translate(x,y)` → déplace l'élément sur l'axe des X (et Y)
- `translate3d(x, y, z)` → déplace l'élément en 3d
- `translateX(x)` → déplace l'élément sur l'axe X
- `translateY(y)` → déplace l'élément sur l'axe Y
- `translateZ(z)` → déplace l'élément sur l'axe Z

Translate



# Perspective et matrices

- `perspective(distance)` → permet de définir la distance entre l'axe Z et l'oeil de l'utilisateur pour ajouter plus ou moins de profondeur à un élément
- `matrix(a, b, c, d, tx, ty)` → permet de définir une transformation 2D selon une matrice définie
- `matrix3d(a1, b1, c1, d1, a2, b2, c2, d2, a3, b3, c3, d3, a4, b4, c4, d4)` → permet de définir une transformation 3D selon une matrice définie

# Transitions

# Transitions

Pour contrôler la vitesse et la façon dont le passage d'un état à un autre (que ce soit une transformation, un changement de couleur ou autre) va se faire, nous pouvons ajouter une transition.

Nous pouvons préciser 4 propriétés :

- **transition-property** → définit sur quelle propriété va se faire la transition (transform, color, background-color etc)
- **transition-duration** → définit la durée de la transition en ms ou s
- **transition-timing-function** → définit une courbe d'accélération/décélération pour la façon dont la transition va se faire
- **transition-delay** → définit le délai après lequel se fait la transition

# Timing functions

Voici les valeurs possibles pour les timing functions :

- **ease** → la vitesse de la transition augmente à partir du milieu puis ralentit à la fin
- **linear** → la vitesse de la transition est la même du début à la fin
- **ease-in** → la transition commence doucement et sa vitesse augmente jusqu'à la fin
- **ease-out** → la transition commence rapidement puis diminue jusqu'à la fin
- **ease-in-out** → la transition commence lentement, accélère puis décélère jusqu'à la fin
- **cubic-bezier** → la transition suit la courbe de bézier renseignée
- **steps** → la transition s'effectue en plusieurs étapes

# Transitions : raccourci

Une écriture raccourcie est disponible. Il suffit d'écrire la propriété **transition** et de donner dans l'ordre :

- property
- duration
- timing-function
- delay

```
.rond {  
  transition : color 1.5s linear 0;  
}
```

# Animations

Merci pour votre attention.

