



Welcome





## Objectifs

- ✓ Comprendre le fonctionnement d'un site web
- ✓ Savoir structurer une page web grâce au HTML et différencier les contenus
- ✓ Styliser sa page grâce au CSS
- ✓ Adapter son site en format mobile
- ✓ Mettre son site en ligne grâce à Github pages



00 - Installation



# Vous pensiez tout connaître des Navigateurs web ?

Les navigateurs web ne sont pas uniquement des moteurs de recherche.

Ils sont aussi dotés d'outils indispensables aux développeurs web, pour tester, debugger, et visualiser leur code.

Chaque navigateur à son comportement qui lui est propre.

Certaines fonctionnalités du web ne sont pas forcément accessibles sur tous les navigateurs.

Il faut donc tous les installer pour pouvoir tester et adapter son site afin d'éviter des bugs.



Chrome



Firefox



Opera



Safari



Ne pas utiliser  
Safari pour coder

# Editeur de code

Un éditeur de code est indispensable pour coder efficacement. Des outils intégrés aux éditeurs, et des plugins ajoutés, permettent de faciliter la vie du programmeur en accélérant son workflow et en améliorant son environnement de travail.



[PhpStorm](#)



[Visual Studio  
Code](#)



[Sublime text](#)



[Vim](#)



[Atom](#)



# Dire bonjour au monde

Mon premier site web !

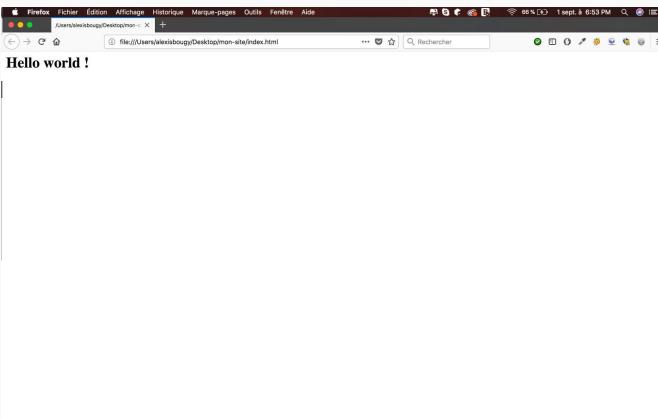
Créer un dossier “mon-site”

Glisser le dossier dans visual studio code

Créer un fichier “index.html” et y insérer le texte ci-contre

L'ouvrir sur un navigateur

<h1>Hello World !</h1>





EXPLORER

...



MON-SITE

index.html



## Conventions de nommage

Par défaut, le premier fichier HTML qui sera lu et ouvert par le navigateur est appelé « index.html », ce sera la porte d'entrée, l'accueil de votre site web.

Dans votre site, les noms des fichiers doivent respecter la convention de nommage « lower\_snake\_case », c'est-à-dire écrit en minuscule, sans espace entre les mots, sans caractères spéciaux (accents...), avec pour séparer les mots des tirets.

L'architecture de votre dossier de site sera affiché sur la gauche dans VScode.



# Culture web

## Comprendre le fonctionnement

## Site web

Mais comment cela fonctionne ?

Lorsque l'on consulte un site web, le navigateur se charge de transformer notre code en un rendu visuel.



## *Ce que l'on voit*

## *Ce que le navigateur voit et interprète pour nous*

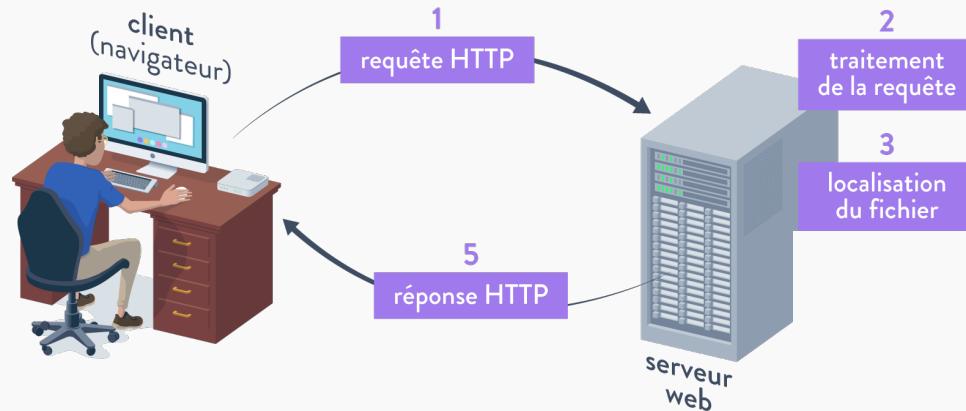
# Site web

Mais il est où ?

Pour être consultable par tous, le contenu d'un site web doit être hébergé sur un **serveur web**.

Lorsque l'on entre notre URL en barre de recherche, le navigateur se charge de faire la passerelle entre l'adresse IP, et le serveur qui lui est associé.

Cette passerelle s'appelle le protocole HTTP.



# Que se passe-t-il au niveau du navigateur ?

Les navigateurs sont en mesure d'interpréter 3 langages de programmation :



## STRUCTURE

Le HTML (HyperText Markup Language) va servir à structurer notre page web (Ajout de textes, d'images, de tableaux...)



## STYLE

Le CSS (Cascading Style Sheet) sert à mettre en forme notre contenu (couleur, taille...)



## ANIMATIONS

Le Javascript va permettre de créer des interactions et animations spécifiques lors d'actions de l'utilisateur. Javascript est le seul langage de programmation compris par les navigateurs!

Ses connaissances sont dès lors essentielles aux développeurs web.

```

$dir) || !is_readable($temp_dir))) {
    ('sys_get_temp_dir')) { // sys_get_temp_dir() is
}; // see https://github.com/Jamesshephard/
// httpdocs/:/tmp/
array('/', '\\"'), DIRECTORY_SEPARATOR);
array('/', '\\"'), DIRECTORY_SEPARATOR);
+= DIRECTORY_SEPARATOR) {
    DIRECTORY_SEPARATOR;
}

SEPARATOR, $open_basedir);
$basedir) {
    != DIRECTORY_SEPARATOR) {
        SEPARATOR;
}

```



01 - Premiers pas



## Le rôle du langage HTML

Sur un site internet il y a plusieurs types de contenus (images, titres, paragraphes, liens, sections ...).

```
<h1>Mon premier titre</h1>  
  
<p>Mon premier paragraphe</p>
```

Le langage HTML permet d'identifier et de classifier ces contenus et construire les fondations de notre site.

Pour construire cette structure on utilise des Balises HTML. Les balises HTML ajoutent du style et des comportements par défaut sur les différents types de contenus qu'elles contiennent.

Vous pouvez ensuite personnaliser ce style au moyen du CSS.

Voici ce à quoi ressemble le site d'AMAZON uniquement avec le style par défaut de l'HTML.

lite  
e on amazon.com. You can also shop on Amazon France for millions of products with fast local delivery. [Click here to go to amazon.fr](#)

ning accessories

ore

**Shop by Category**

Computers & Accessories

Video Games

# Qu'est ce qu'une balise ?

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mon site</title>
  </head>
  <body>
    <h1>Mon premier titre</h1>
    <p>Mon premier paragraphe</p>
  </body>
</html>
```



- L'HTML fonctionne avec des balises. Ce sont des mots clés qui vont indiquer la manière dont doit être interprété notre contenu.

```
<h1>
<p>
```

- On doit les ouvrir, ajouter du contenu, puis les FERMER.

```
<title>Mon site</title>
```

- Le langage HTML fonctionne à la manière d'un emboîtement. Quand on met un élément dans un autre, on fait une tabulation pour le démarquer.

```
<head>
  <title>Mon site</title>
</head>
```

## Structure de base

```
<!DOCTYPE html>  
<html>
```

- Le squelette d'une page HTML est toujours organisé de la même manière.
- On doit tout d'abord spécifier le type de document. Pour utiliser la dernière version du HTML, il suffit de renseigner 'html'

```
<!DOCTYPE html>
```

```
</html>
```

- On ouvre, puis on ferme une balise <html> pour signifier que c'est ici que nous allons commencer à écrire du code

```
<html></html>
```

## Structure de base

```
<!DOCTYPE html>  
<html>
```

<head>

```
</head>
```

<body>

```
</body>
```

</html>

Par emboîtement, nous allons ajouter à l'intérieur 2 balises :

- La balise `<head>`, peut être considérée comme le cerveau du document. Il contiendra toutes les infos nécessaires au bon fonctionnement de ce dernier. Les données du head ne s'affichent donc jamais sur une page web, elles ne sont que des indications pour l'interprétation.

```
<head></head>
```

<body>

```
</body>
```

</html>

- Toujours à l'intérieur des balises <html>, mais pas dans <head>, nous allons ouvrir une balise <body> qui contiendra notre contenu visible sur le site (nos textes, nos images etc...)

```
<body></body>
```

Cette base est universelle à l'ensemble des sites web. C'est la première chose qu'on doit faire lors de la création d'un document html. Les balises <html>, <head>, <body>, sont uniques, il ne peut pas y en avoir plusieurs. IIM



## Les 5 règles du HTML

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

  <body>
    <h1>John Doe</h1>
    <h2>Expériences professionnelles</h2>
    <h3>Entreprise 1</h3>
    <h3>Entreprise 2</h3>
    <h2>Compétences</h2>
    <h2>Formation</h2>
    <h2>Loisir</h2>
  </body>
</html>
```

1 - Toute balise ouvrante doit être fermée



<p>Ouvre et ferme</p>



<p>Ouvre et ferme pas



<p>Ouvre et ferme mal</p>

Certaines exceptions existent. Ce sont des balises qui ne peuvent rien contenir à l'intérieur. On peut citer les balises `<meta>`, `<img>`, `<input>`, `<link>`...



## Les 5 règles du HTML

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

  <body>
    <h1>John Doe</h1>
    <h2>Expériences professionnelles</h2>
    <h3>Entreprise 1</h3>
    <h3>Entreprise 2</h3>
    <h2>Compétences</h2>
    <h2>Formation</h2>
    <h2>Loisir</h2>
  </body>
</html>
```

2 - Les balises et les attributs sont écrits en minuscule



<strong class="ma-classe"></strong>



<STRONG class="ma-classe"></STRONG>



<strong Class="ma-classe"></strong>

## Les 5 règles du HTML

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

  <body>
    <h1>John Doe</h1>
    <h2>Expériences professionnelles</h2>
    <h3>Entreprise 1</h3>
    <h3>Entreprise 2</h3>
    <h2>Compétences</h2>
    <h2>Formation</h2>
    <h2>Loisir</h2>
  </body>
</html>
```



<strong class="ma-classe"></strong>



<strong class=ma-classe></strong>

## Les 5 règles du HTML

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

  <body>
    <h1>John Doe</h1>
    <h2>Expériences professionnelles</h2>
    <h3>Entreprise 1</h3>
    <h3>Entreprise 2</h3>
    <h2>Compétences</h2>
    <h2>Formation</h2>
    <h2>Loisir</h2>
  </body>
</html>
```

4 - Chaque attribut doit contenir une valeur



<strong class="ma-classe"></strong>



<strong class></strong>

Exception : L'attribut “required” que l'on peut trouver dans les balises de formulaires ( <input>, <textarea>, <select> ... )

## Les 5 règles du HTML

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

  <body>
    <h1>John Doe</h1>
    <h2>Expériences professionnelles</h2>
    <h3>Entreprise 1</h3>
    <h3>Entreprise 2</h3>
    <h2>Compétences</h2>
    <h2>Formation</h2>
    <h2>Loisir</h2>
  </body>
</html>
```



<p>Ouvre et <strong> ferme</strong></p>



<p>Ouvre et <strong>ferme mal</p></strong>



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

  <body>
    <h1>John Doe</h1>

    <h2>Expériences professionnelles</h2>
    <h3>Entreprise 1</h3>
    <h3>Entreprise 2</h3>

    <h2>Compétences</h2>

    <h2>Formation</h2>

    <h2>Loisir</h2>
  </body>
</html>
```

## Les balises

### Ajout des premiers titres

Pour découvrir les différentes balises nous allons créer un CV. Sur ce CV, je vais déjà définir mes sections principales.

Pour ce faire, je vais ajouter dans mon `<body>` des titres. En HTML, il existe 6 niveaux de titre.

```
<h1>Titre de niveau 1</h1>
<h2>Titre de niveau 2</h2>
<h3>...</h3>
<h4>...</h4>
<h5>...</h5>
<h6>...</h6>
```

Le titre de niveau 1 correspond au titre de la page le plus important, celui de niveau 6 le moins important. Le niveau 1 (`h1`) ne doit être utilisé qu'une seule fois par page.

Rq : Par cohérence, il ne peut donc pas y avoir de titre de niveau 3 sans que de niveau 2 existe etc. (Cela reste autorisé mais mal vu donc mauvais pour le SEO)

Et là, c'est le drame...

Ajout des premiers titres

Et là, c'est le drame...

Le navigateur ne comprend pas que nous avons mis des accents



## John Doe

### Expériences professionnelles

**Entreprise 1**

**Entreprise 2**

**Compétences**

**Formation**

**Loisir**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>

  <body>
    <h1>John Doe</h1>

    <h2>Expériences professionnelles</h2>
    <h3>Entreprise 1</h3>
    <h3>Entreprise 2</h3>

    <h2>Compétences</h2>

    <h2>Formation</h2>

    <h2>Loisir</h2>
  </body>
</html>
```



## Les Meta tags

### Ajouter les accents sur notre CV

Nous commencerons en ajoutant “l’intelligence” de notre site, grâce aux balises `<meta>`, dans la `<head>`.

Les balises `<meta>` sont des métadonnées, elles serviront à transmettre des informations aux navigateurs sur la façon dont nous voulons que notre site fonctionne.

Ici par exemple nous pouvons résoudre le problème des accents en ajoutant cette balise:

```
<head>
  <meta charset="utf-8">
</head>
```

A l’intérieur nous allons définir un attribut (ici en jaune) pour indiquer que nous allons utiliser le jeu de caractères ‘utf-8’

Rq : La balise `<meta>` est dite auto-fermante. Comme elle ne peut pas avoir de contenu à l’intérieur, pas besoin de faire `<meta></meta>`.

```
<head>
  <meta charset="utf-8">
  <meta name="description" content="Digital,
Création, Web, Développement, Jeu Vidéo,
Animation 3D : depuis 1995, l'IIM Paris,
première école qui forme des spécialistes du
digital ">
</head>
```

## Meta tags

Les informations pour partager votre site sur Google, Facebook, Twitter ...

Dans le `<head>` nous allons également passer des informations qui seront utilisées par les différents réseaux sociaux, moteurs de recherche... pour présenter notre site joliment.

Par exemple ici Google a récupéré le titre et la description passé dans les meta tags du site de l'IIM pour construire ce bloc.

<http://www.iim.fr> · Translate this page ::

**IIM Digital School - Ecole de Jeu vidéo, 3D, Développement ...**

Digital, Création, Web, Développement, Jeu Vidéo, Animation 3D : depuis 1995, l'IIM Paris, première école qui forme des spécialistes du digital.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>

  <body>
    <h1>John Doe</h1>

    <h2>Expériences professionnelles</h2>
    <h3>Entreprise 1</h3>
    <h3>Entreprise 2</h3>

    <h2>Compétences</h2>

    <h2>Formation</h2>

    <h2>Loisir</h2>
  </body>
</html>
```

# Les Attributs

## À propos des attributs html

Nous l'avons vu, nous avons ajouter à la balise `<meta>` un attribut.

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

L'attribut est une donnée qui va venir donner des indications supplémentaires sur le comportement que doit avoir notre balise. Cela sera très utile quand nous verrons le CSS.

Chaque balise HTML peut avoir des attributs. Il existe des attributs communs aux balises, et d'autres spécifiques à certaines balises.

Un attribut se symbolise ainsi :

```
nomDeLattribut="valeur"
```

- Nom de l'attribut
- Le signe “=”
- Une valeur entre guillemet

## Les Blocs de texte

Un texte pour se présenter !

Nous allons ajouter un paragraphe pour nous présenter. Pour faire un texte, nous utiliserons la balise <p>

```
<p>Hello, je m'appelle John et je suis développeur web !</p>
```

A l'intérieur de notre balise <p>, nous pouvons intégrer des éléments classiques de l'édition de texte (gras, italique...).

Nous allons mettre en évidence notre métier avec une balise <strong>, l'équivalent du gras en html.

```
<p>Hello, je m'appelle John et je suis <strong>développeur  
web</strong> !</p>
```

Attention à l'ordre des balises ! Le <strong> se trouve à l'intérieur du <p> et doit le rester, le <strong> ne peut pas être fermé après la fermeture du <p>

```
<body>  
  <h1>John Doe</h1>  
  <p>Hello, je m'appelle John et je suis  
    <strong>développeur web</strong> !</p>  
</body>
```

# Les Images

Une jolie photo pour se présenter

Pour compléter la présentation, nous allons ajouter une image de nous. Pour cela, il faut placer notre photo dans notre dossier de travail.

Puis nous allons utiliser la balise `<img>` pour appeler notre image.

```

```

La balise est auto fermante car nous ne pouvons pas placer de contenu à l'intérieur.

Elle prend 2 attributs obligatoire :

`src` permet de définir le chemin de notre image par rapport au fichier html.

`alt` permet d'écrire un texte de substitution si l'image ne se charge pas. C'est un texte qui est également lu pour les personnes malvoyantes qui utilisent le site en mode audio.

```
<body>
  <h1>John Doe</h1>
  
  <p>Hello, je m'appelle John et je suis
    <strong>développeur web</strong> !</p>
</body>
```

# Les Listes

Des listes pour énumérer

```
<body>
  <h2>Mes compétences</h2>

  <ul>
    <li>Compétence 1</li>
    <li>Compétence 2</li>
  </ul>

  <ol>
    <li>Compétence 1</li>
    <li>Compétence 2</li>
  </ol>
</body>
```

On peut mettre en place des listes pour faire de l' énumération. Il existe des listes ordonnées `<ol>` ou non ordonnées `<ul>`.

Quelque soit la balise, des `<li>` (list items) sont obligatoires à l'intérieur pour chaque élément et non substituables.

# Les liens HTML

## Interne & Externe

Les liens se symbolisent par la balise `<a href=""></a>`. Le href contient le chemin du lien, qui peut amener sur un site externe, où rester à l'intérieur du site pour naviguer vers une autre page html. Il faudra faire attention au dossier dans lequel notre page se trouve pour faire les bons liens.



```
<a href="https://www.google.com/">Lien vers un site externe</a>

<a href="contact.html">Lien vers contact - Se trouve dans le même dossier que la page actuelle</a>
<a href="../../contact.html">Lien vers contact - Se trouve un dossier en arrière que la page actuelle</a>
<a href="info/contact.html">Lien vers contact - Se trouve dans le dossier info</a>

<a href="contact.html" target="_blank">Ouvre dans un nouvel onglet</a>
<a href="contact.html" title="Infobulle">Au survol du lien, une infobulle apparaît</a>

<a href="#ancre">Lien vers une ancre de l'actuelle page</a>
<a href="#">Lien vide, à utiliser de manière temporaire</a>
```

# Les différentes balises

## Liste des différentes balises

Il existe des centaines de balises HTML que le navigateur interprète.

Elles sont toutes documentées ici :

[Balises HTML](#)

<div></div> => division

<p></p> => paragraphe

<a></a> => lien

<h1></h1> => titre

<img> => image

Chacune à son utilisation propre.

Certaines appliquent du style par défaut dans le navigateur.

Certaines sont également beaucoup plus utilisées que d'autres. (div, p, a, h1, img...).

## Les types de balises

Les différents comportements des deux types de balises HTML

Il existe deux types de balises.

**Les balises block :** Elle crée un retour à la ligne avant et après. En CSS on peut personnaliser leur longueur et hauteur.

Exemple : `<p></p> <h1></h1>`

**Les balises inline :** Présente dans des balises block la plupart du temps, elles font l'exacte hauteur et longueur de ce qu'elles contiennent. On ne peut pas modifier cette longueur et hauteur. Il n'y a pas de retour à la ligne avant et après.

Exemple : `<strong></strong> <a></a>`





```
<div>
  <h1>Mise en forme</h1>
  <p>Cette div permet de contenir mon titre
  <span> mon paragraphe que je peux
  personnaliser</span></p>
</div>
```

## Les balises “universelles”.



Pour aider à la mise en forme de notre document, il existe 2 balises “universelles”.

Balise universelle block : `<div></div>`

Balise universelle inline : `<span></span>`

Ce sont des balises vides, qui n'ont pas d'effets particulier par défaut. C'est le CSS qui va permettre de les moduler à notre souhait.

Elles permettent notamment de mettre en forme notre page : taille pour la `<div>`, de la couleur pour la `<span>` par exemple. Nous y reviendrons plus loin dans le cours.

Les spans sont plus rarement utilisées que les div car elles servent uniquement à l'intérieur des balises inline (p, h1, h2 ...)

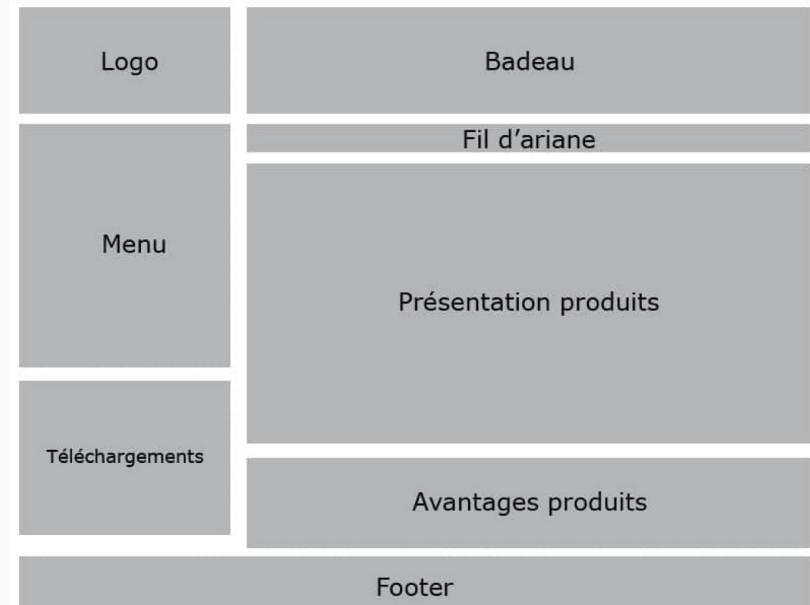
# Représenter son site

## Zoning & Wireframes

Pour se donner une idée de notre apparence de site, nous pouvons découper notre conception.

Le zoning est une étape de conception qui permet de représenter grossièrement les grandes zones de notre site ou application.

Cela peut être réalisé sur ordi ou papier !



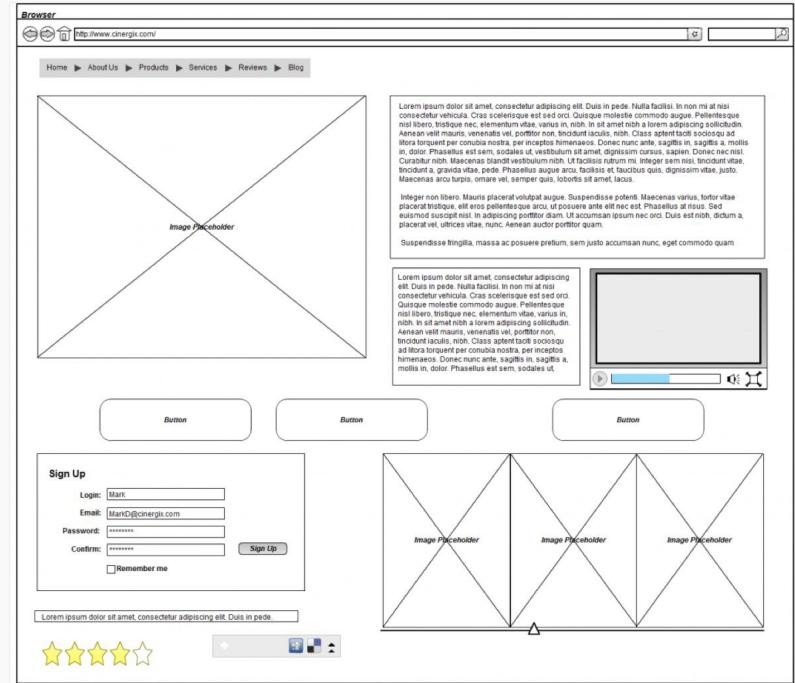
# Représenter son site

## Zoning & Wireframes

Le wireframe est une étape au dessus du zoning !

On structure les éléments, on y met les premières formes, du contenu (souvent faux), le tout sans notion graphique. C'est une étape d'ergonomie et d'expérience d'utilisation avancée.

Cela peut être réalisé sur ordi ou papier !



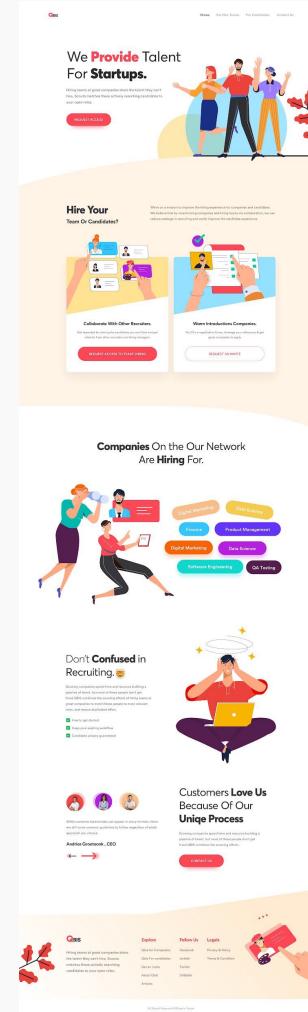
# Représenter son site

## Maquettes

La maquette est la version “finalisé” de notre design représenté sur un outil dédié (Photoshop, XD, Figma...).

Elle offre tous les détails dont à besoin un développeur pour travailler le site en code HTML/CSS. On y voit les couleurs, les typographies, les grandes zones de sections...

Les maquettes doivent être fournies en version ordinateur et mobile pour que le développeur puisse faire les adaptations nécessaires selon le device utilisé par l'internaute.





```
<div>
  <div>
    Contenu du premier bloc
  </div>
  <div>
    Contenu du second bloc
  </div>
  <div>
    Contenu du troisième bloc
  </div>
</div>
```

## L'élément <div>

### La représentation de nos blocs



<div> signifie division de contenu. Elle va permettre de découper notre document html en plein de petits morceaux qui contiendront du contenu. On pourra ensuite modifier leurs positions etc.

1 - La notation est <div>

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

2 - Une <div> peut tout à fait en contenir une autre, pour faire des subdivisions

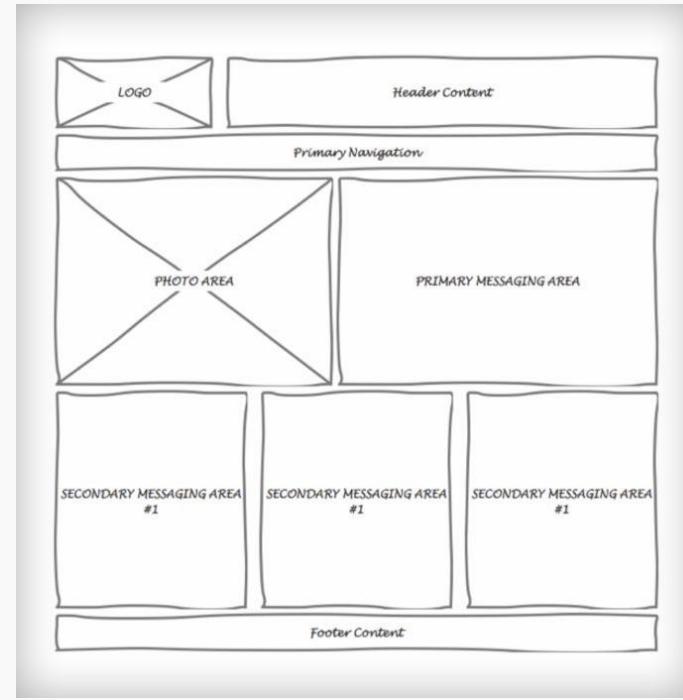
```
<div>
  <div>Elément 1</div>
</div>
```



# Découper son document

Et un bloc devient <div>

Le but du jeu est d'identifier les grandes zones du document qui vont devenir une <div>

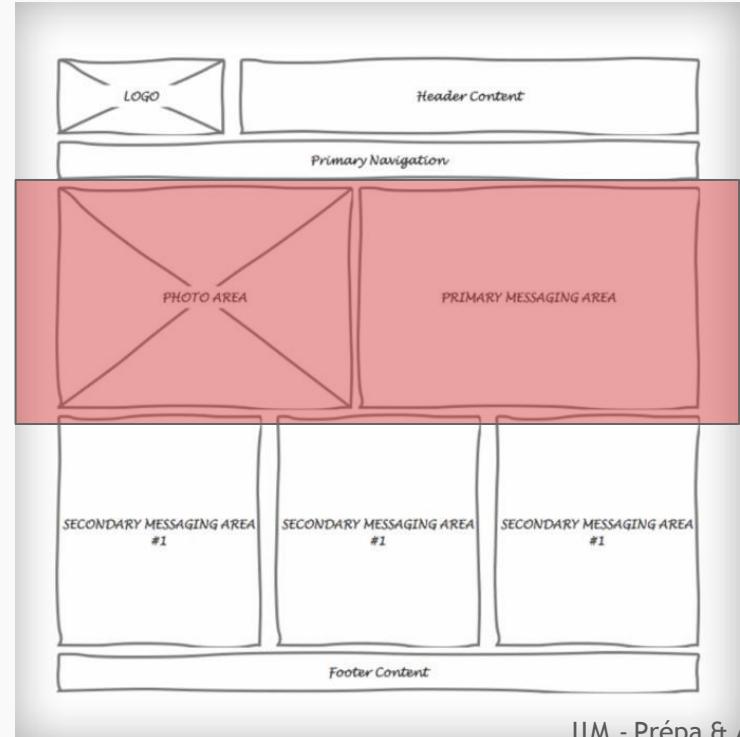


## Découper son document

Et un bloc devient <div>

Le but du jeu est d'identifier les grandes zones du document qui vont devenir une <div>

Je fais tout d'abord un découpage d'une grande zone



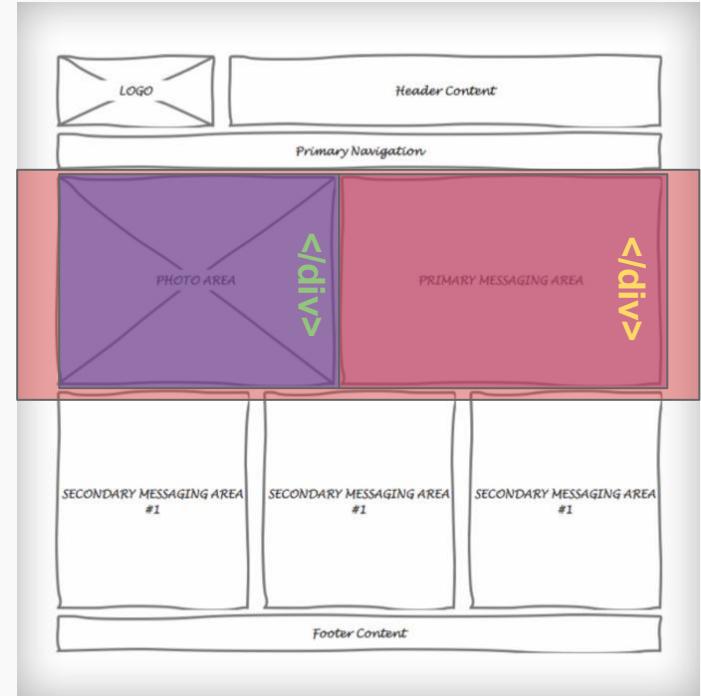
## Découper son document

Et un bloc devient <div>

Le but du jeu est d'identifier les grandes zones du document qui vont devenir une <div>

Je fais tout d'abord un découpage d'une grande zone.

J'identifie ensuite les sous-zones.



## Découper son document

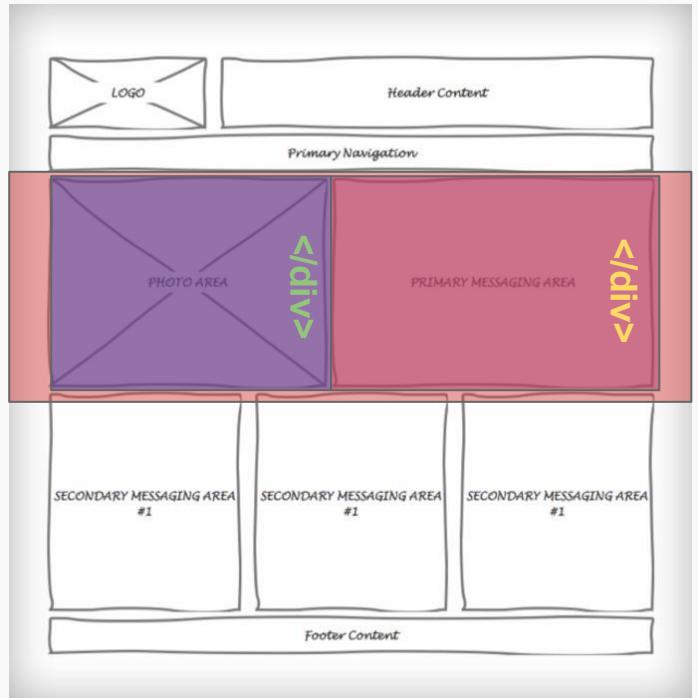
Et un bloc devient <div>

Le but du jeu est d'identifier les grandes zones du document qui vont devenir une <div>

Je fais tout d'abord un découpage d'une grande zone.

J'identifie ensuite les sous-zones.

Enfin, je regarde les éléments html qui seront contenus dans cette zone



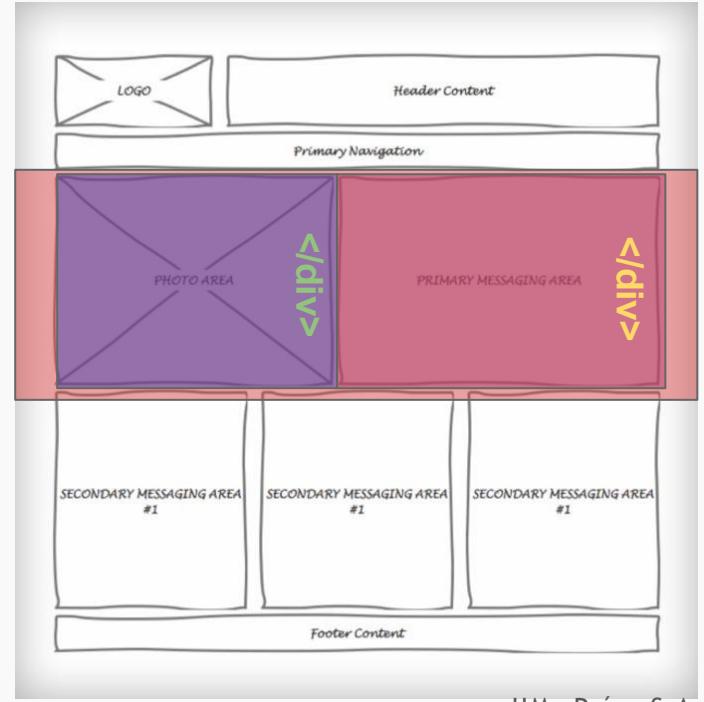
# <div>

## Découper son document

Et un bloc devient <div>



```
<!-- Grande div -->
<div>
    <!-- div de gauche avec l'image -->
    <div>
        
    </div>
    <!-- div de droite avec le texte -->
    <div>
        <p>Mon texte de la partie de droite</p>
    </div>
</div>
```



# <div>

## Découper son document

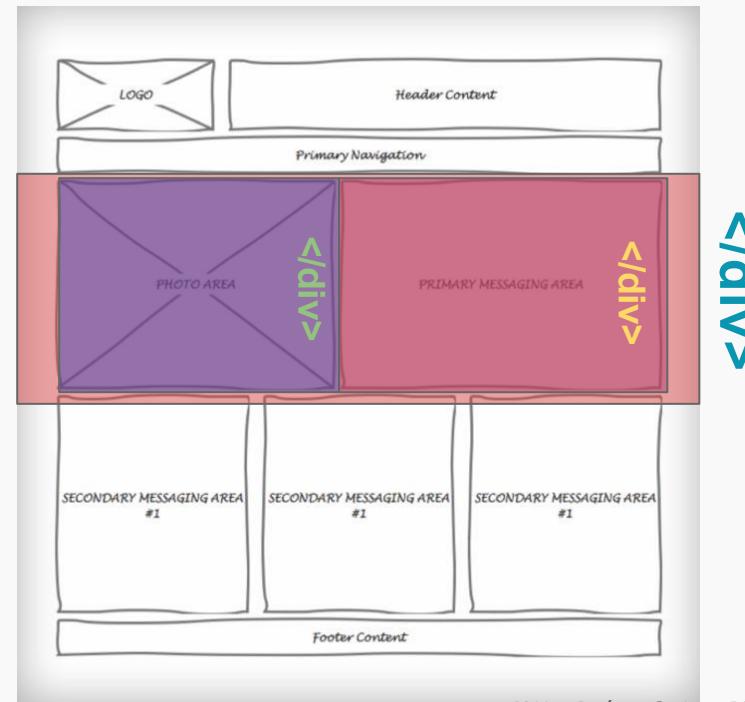
Je peux à partir de ça translater l'ensemble en HTML.

47

Pour pouvoir styliser ces div de manière spécifique nous allons leurs donner des noms (grâce à l'attribut class) pour pouvoir les sélectionner en css puis les styliser.

```
<!-- J'ajoute des classes pour la mise en forme avec css -->
<!-- Grande div -->
<div class="presentation-section">
    <!-- div de gauche avec l'image -->
    <div class="presentation-image">
        
    </div>
    <!-- div de droite avec le texte -->
    <div class="presentation-texte">
        <p>Mon texte de la partie de droite</p>
    </div>
</div>
```

05 - Organiser son contenu



```
<div class="container">
  <div class="bloc-1">
    Contenu du premier bloc
  </div>
  <div class="bloc-2">
    Contenu du second bloc
  </div>
  <div class="bloc-3">
    Contenu du troisième bloc
  </div>
</div>
```

## L'élément <div>

Les limites d'un élément essentiel



Le soucis est que la <div> est tellement utilisée, qu'on peut vite s'y perdre dans la lecture de notre document.

On le voit ici, rien qu'avec un seul niveau de profondeur, on voit déjà des <div> partout et une lecture qui n'est pas forcément fluide.

Pour aider à endiguer le problème, HTML 5 a ajouté plusieurs éléments équivalents à la <div>, avec des noms différents. Le but est de pouvoir ajouter de la variété de noms dans notre document selon le rôle que doit avoir la <div>.

Aussi ce sera important pour le SEO de votre site.

# L'élément <div>

Ses autres noms



**<header>** Le header est un élément que l'on placera pour représenter la grande section haute de notre document. Elle peut contenir un logo, un menu, et/ou la toute première section de notre site.

**<nav>** On utilisera <nav> à la place de <div> pour situer notre bloc de menu. On y retrouve donc les liens pour naviguer sur notre site.

**<section>** Section va permettre de définir des grandes zones de sites.

**<article>** Cet élément comme son nom peut l'indiquer contiendra le contenu d'un article de notre site. D'une manière globale cela va être un emplacement avec une grande zone textuelle.

**<aside>** Aside sert à représenter les sections de type sidebar. C'est une barre de contenu, en générale à côté d'un article sur un blog qui donne des infos globales sur ce dernier.



**<footer>** C'est notre grande section de fin de site. Elle contiendra nos mentions légales, quelques infos textuelles, de contact...

**<main>** Elle définit notre zone principale du site.

```
<section class="container">
  <div class="bloc-1">
    Contenu du premier bloc
  </div>
  <div class="bloc-2">
    Contenu du second bloc
  </div>
  <div class="bloc-3">
    Contenu du troisième bloc
  </div>
</section>
```

## <div> et équivalents

On améliore notre structure



50

En utilisant cette logique, je peux switcher ma première <div> en <section> car elle définit une grande zone de mon site.

Cela nous permet d'améliorer notre confort de lecture.

La qualité de code étant un critère pour un bon positionnement dans les moteurs de recherches, il est conseillé d'y avoir recours au maximum.

```
<section class="container">
<article class="card-blue">
    <div class="bloc-1">
        Contenu du premier bloc
    </div>
    <div class="bloc-2">
        Contenu du second bloc
    </div>
    <div class="bloc-3">
        Contenu du troisième bloc
    </div>
</article>
</section>
```

## <div> et équivalents

On améliore notre structure



**Mon astuce :** En + de la section, je viens ajouter un élément `<div>` ou `<article>` qui entoure également mon contenu. Cela permet certaines manipulations CSS beaucoup plus simples, notamment sur les adaptations mobiles !

Ma grande `<section>` va venir accueillir des éléments CSS tels que le background, tandis qu' `<article>` va venir définir la longueur de mon bloc sur la page. Ainsi ma section peut avoir un background qui fait 100% de la longueur de mon site, et l'article va venir prendre que 80% de la taille et sera centré dans la section.



# INDENTATION

Pourquoi indenter son code ?

Dans ces deux exemples,  
une des `<div>` n'est pas fermée.  
Vous remarquerez qu'il est beaucoup  
plus rapide d'identifier cette erreur sur  
un code bien indenter.

Votre site en s'étoffant deviendra très  
difficile à déchiffrer et debugger si vous  
ne faites pas attention à votre  
organisation et indentation.

```
<section class="container">
  <article class="card-blue">
    <div class="bloc-1">
      Contenu du premier bloc

    <div class="bloc-2">
      Contenu du second bloc
    </div>
    <div class="bloc-3">
      Contenu du troisième bloc
    </div>
  </article>
</section>
```

```
<section class="container"><article
  class="card-blue">
    <div class="bloc-1">
      Contenu du premier bloc
    </div>

    <div class="bloc-2">
      Contenu du second bloc
    </div>
    <div class="bloc-3">
      Contenu du troisième bloc
    </div>
  </article>
</section>.
```





## L'élément <link>

Mettons en style !

```
<head>
  <meta charset="utf-8">
  <title>Mon premier site</title>
  <link rel="stylesheet" href="style.css" />
</head>
```

Faire du HTML c'est bien, mais il ne sert qu'à baliser du contenu. Il va falloir lier l'ensemble à une feuille CSS pour pouvoir apporter des éléments de personnalisation à notre document.

Pour lier notre fichier HTML à notre feuille de style CSS on va utiliser la balise link :

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

<link> est une balise auto-fermante. Elle ne peut rien contenir donc on la ferme directement. Pas besoin de </link>.

Elle prend 2 attributs :

```
rel="stylesheet" href="style.css"
```

rel : Abréviation de relationship. Ici on indique donc stylesheet pour feuille de style.

href : Le chemin de dossier par rapport à notre html pour arriver au css.

```
p {  
    font-size: 24px;  
    text-align: center;  
}  
  
.custom-text {  
    font-size: 24px;  
    text-align: center;  
}  
  
#custom-text {  
    font-size: 24px;  
    text-align: center;  
}
```

## CSS

### Structure

Le CSS a une structure bien définie :

- Le code se met entre accolades { }
- Le nom d'affectation se situe avant les accolades. On l'appelle le sélecteur (on y revient après)
- Le code se structure ainsi :  
*nom-propriété: valeur;*  
Attention au deux-points et au point-virgule

```
p {  
    font-size: 24px;  
    text-align: center;  
}  
  
h1 {  
    font-size: 36px;  
}  
  
h1, p {  
    color: white;  
}
```

## Les sélecteurs

Les sélecteurs d'éléments

Le sélecteur d'élément influe directement sur la balise html correspondante.

Ici, tous les `<p></p>` côté html auront une taille de typographie de 24 pixels et seront alignés au centre.

De la même manière, tous les `<h1></h1>` auront une taille de 36 pixels.

Je peux aussi réunir les sélecteurs par une virgule. De cette manière, l'ensemble des sélecteurs présent seront influencé par le code. Ici `<p>` et `<h1>` seront de couleur blanche.

```
<h1>Titre</h1>
<p class="color-blue">Du
contenu</p>
<p>Encore du contenu</p>
<p class="color-blue">Toujours du
contenu</p>
```

## Les sélecteurs

### Les sélecteurs de classe

Pour personnaliser un élément spécifique, on va privilégier la classe. On va ainsi toucher à un élément plutôt qu'à un ensemble de balises.

Côté html, on va mettre en place notre classe en utilisant l'attribut **class**. Ici notre `<p>` à désormais la classe *color-blue*

```
<p class="color-blue">
```

Côté CSS, on donne le même nom de classe *color-blue* précédé d'un point.

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

*Astuce : Côté html, on peut utiliser ce même nom de classe autant de fois que l'on veut au sein du code. On peut ainsi donner la même apparence à plusieurs endroits.*

```
<h1>Titre</h1>
<p id="text-thin">Du contenu</p>
<p>Encore du contenu</p>
<p class="color-blue">Toujours du
contenu</p>
```

## Les sélecteurs

### Les sélecteurs d'ID

Contrairement à la classe, l'ID ne peut être utilisé qu'à un seul endroit de son html. Il est donc unique.

Côté html, on va mettre en place notre ID en utilisant l'attribut ***id***. Ici notre `<p>` à désormais l'ID *text-thin*

```
<p id="text-thin">
```

Côté CSS, on donne le même nom d'ID *text-thin* précédé d'un dièse #.

*Attention : On va privilégier l'utilisation des ID dans seulement deux cas :*

- 1/ Faire des interactions avec le langage Javascript*
- 2/ Faire des ancrages*

*Dans les autres cas, il convient d'utiliser les classes*

```
#text-thin {
    font-weight: 300;
}
```

```
<p class="color" id="color">Hello  
World</p>
```

```
#color {  
    color: red;  
}  
  
.color {  
    color: blue;  
}
```

Ici « Hello World » apparaîtra en rouge

## Les sélecteurs

Les rapports de force

Que ce passe-t-il lorsque nous utilisons deux sélecteurs différents pour un même élément ?

En CSS, il existe des rapports de force pour savoir quelle règle CSS va être appliquée.

Ces rapports fonctionnent de cette manière :

* = 0
<balise> = 1
.class = 10
#id = 100
<h1 style=""> = 1 000
!important = 10 000

Les deux derniers sont à éviter car ils sont très rarement nécessaires.

On additionne les valeurs de sélecteur de la même règle.

En cas d'égalité, c'est la règle la plus basse du fichier de style qui est prise en compte.

```
<p class="color">Hello World</p>
```

## Les sélecteurs

C'est de quelle couleur alors ?

```
p {  
    color: red;  
}  
  
.color {  
    color: blue;  
}
```

De quelle couleur est le  
Hello World ?

Il est bleu (class plus  
spécifique)

```
<p id="color" style="color: purple;">  
Hello World</p>
```

## Les sélecteurs

C'est de quelle couleur alors ?

```
p {  
    color: red;  
}  
  
#color {  
    color: blue;  
}  
  
.color {  
    color: green !important;  
}
```

Et cette fois, de quelle couleur est le Hello World ?

Il est violet (attribut style)

```
p {  
    font-size: 28px;  
    text-align: center;  
    font-weight: bold;  
}  
  
.texte {  
    font-style: italic;  
    text-decoration: underline;  
}
```

## Les textes

### Quelques styles communs

La `font-size` permet de définir la taille du texte. On l'exprime en pixels (px). Il existe d'autres unités comme le point (pt), l'em et le rem.

Le `text-align` définit l'alignement du texte sur notre document. 4 valeurs possibles : `center` `left` `right` `justify`.

Le `font-weight` spécifie la graisse ("l'épaisseur") du texte : il peut prendre une valeur par mots-clés comme `normal` & `bold` mais aussi des valeurs numériques. Les valeurs communes sont `300` `400` `500` `600` `700` et `900`. Elles sont notamment utilisées par les polices d'écritures hébergées en ligne.

## Les textes

### Quelques styles communs

```
p {  
    font-size: 28px;  
    text-align: center;  
    font-weight: bold;  
}  
  
.texte {  
    font-style: italic;  
    text-decoration: underline;  
}
```

La `font-style` permet notamment de définir un comportement italique.

Le `text-decoration` permet d'ajouter un trait de mise en avant au dessus et/ou en dessous du texte. Il peut avoir des décos spécifiques comme des pointillés et même une couleur !

```
<p>Lorem ipsum dolor sit amet, <span  
class="custom-color">consectetur  
adipisicing elit</span>. Beatae deserunt  
in iusto nam officia totam!</p>
```

## Les textes

Customisation plus avancée

On peut faire un peu de customisation à l'intérieur même d'un texte grâce à l'élément inline vide `<span></span>`. Il suffit de lui ajouter une classe et d'aller customiser le tout côté CSS !

Des éléments HTML existent par défaut pour faire de la customisation. On peut citer `<b></b>` et `<strong></strong>` pour passer du texte en gras ou encore `<i></i>` pour de l'italique.

```
.custom-color {  
    color: lightblue;  
}
```

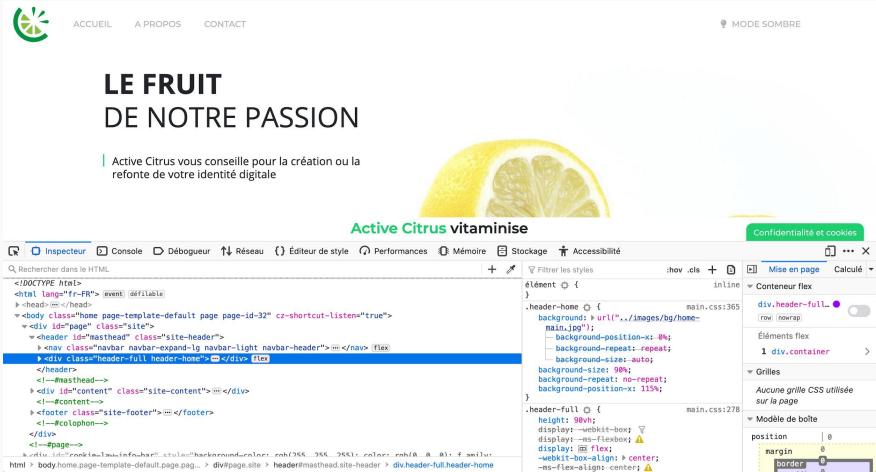
# “Debug de son code”

Faire une analyse à la volée

Sur le navigateur on peut faire un clic droit  
-> Inspecter l'élément.

Une interface s'affiche, permettant de vérifier que notre html et css fonctionne comme on le souhaite. On peut directement modifier dessus pour voir des rendus !

**ATTENTION :** Quand on fait une modification dans l'inspecteur, elle n'est pas prise en compte sur le site définitivement ! Il faut donc reporter ce que l'on a fait sur son éditeur de code.



```
p {  
    /* Défaut */  
    color: lightcoral;  
  
    /* Code Hexa */  
    color: #F08080;  
  
    /* Code RGB */  
    color: rgb(240, 128, 128);  
  
    /* Code RGBA */  
    color: rgba(240, 128, 128, 1);  
  
    /* Code HSL */  
    color: hsl(0, 79%, 72%)  
}
```



## Les couleurs

Différentes typologies

Les couleurs peuvent être appelées de plusieurs manières :

**Code par défaut** : Le CSS intègre un certain nombre de couleurs pré-définies.

**Code RGB** : On définit le niveau de rouge, vert et bleu pour déterminer une couleur

**Code RGBA** : On ajoute un 4ème paramètre qui gère l'opacité. Sa valeur est comprise entre 0 et 1

**Code Hexadécimal** : Raccourci du code RGB où une suite de deux lettres et/ou chiffres représentent un niveau de couleur rouge vert ou bleu

**Code HSL** : Approche plus “humaine”, on définit une teinte, un niveau de saturation et de luminosité.

```
<link
  href="https://fonts.googleapis.com
  /css?family=Roboto&display=swap"
  rel="stylesheet">
```

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

## Les textes

Une typo personnalisée



[Google Font](#) est une bibliothèque en ligne qui réunit tout un ensemble de typographies prêt à l'usage.

Avec les différents paramètres, on obtient une balise `<link>` HTML pour importer notre typographie. On peut ensuite en faire usage via la propriété **font-family**.

**font-family** peut contenir plusieurs typographies (séparé par une virgule). Si le navigateur n'arrive pas à lire la première (pour x et y raisons), il lira la deuxième etc.

```
<link
  href="https://fonts.googleapis.com
  /css?family=Roboto&display=swap"
  rel="stylesheet">
```

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

## Les textes

### Une typo personnalisée



1 Family Selected

Your Selection [Clear All](#)

Robo×

[EMBED](#) [CUSTOMIZE](#) Load Time: Fast

**Embed Font**

To embed your selected fonts into a webpage, copy this code into the <head> of your HTML document.

[STANDARD](#) [@IMPORT](#)

```
<link href="https://fonts.googleapis.com/css?family=Roboto&display=swap"
      rel="stylesheet">
```

---

**Specify in CSS**

Use the following CSS rules to specify these families:

```
font-family: 'Roboto', ;
```

For examples of how fonts can be added to webpages, see the [getting started guide](#).

# Les textes

Une typo personnalisée

Si une police n'existe pas en ligne, nous allons devoir la définir nous même. Pour cela, on va glisser notre font dans notre dossier de projet et utiliser la propriété CSS `@font-face`.

```
/* On indique au css les informations sur notre police */
@font-face {
    font-family: 'OpenSans';
    src: url("fonts/open-sans.ttf"); /* Typologie de base, on peut y mettre beaucoup + de détails */
}

/* On peut utiliser notre font */
h1 {
    font-family: 'OpenSans', Arial, sans-serif;
}
```

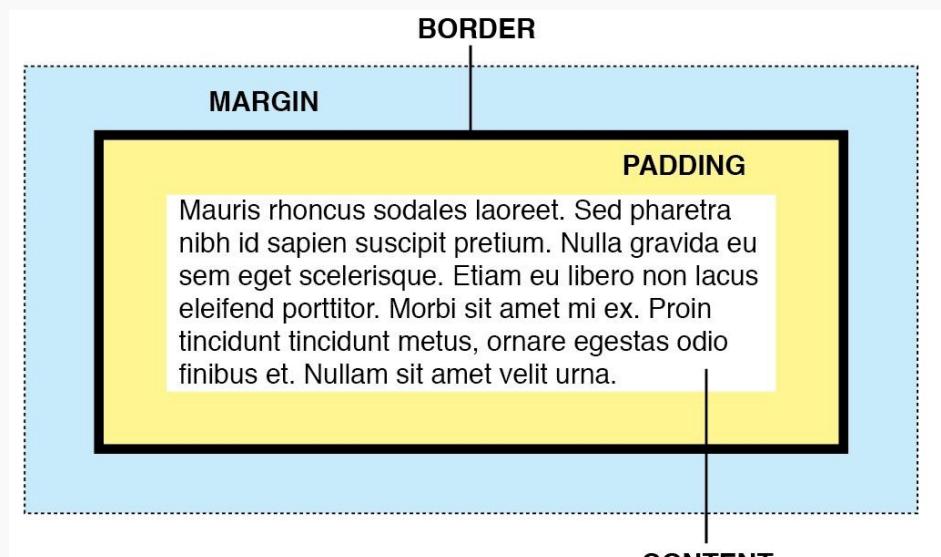


```
.block {  
    /* Au détail */  
    margin-left: 10px;  
    padding-right: 10px;  
  
    /* Global */  
    margin: 10px;  
  
    /* 2 valeurs = Haut + Bas puis Gauche +  
    Droite */  
    margin: 10px 20px;  
  
    /* 4 valeurs : Top - Right - Bottom - Left */  
    margin: 10px 20px 10px 20px;  
}  
  
.block {  
    /* Global */  
    padding: 10px;  
  
    /* 2 valeurs = Haut + Bas puis Gauche +  
    Droite */  
    padding: 10px 20px;  
  
    /* 4 valeurs : Top - Right - Bottom - Left */  
    padding: 10px 20px 10px 20px;  
}
```

# Margin et Padding

## La gestion de l'espacement

La margin définit la marge extérieur au bloc concerné.  
Le padding définit la marge intérieure au bloc concerné.

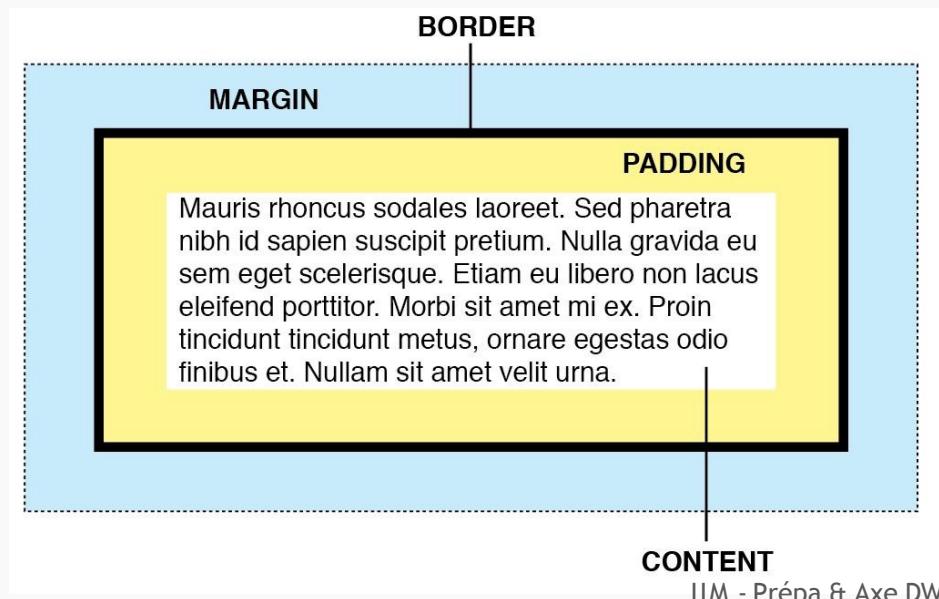


# Les bordures

Le contour d'un bloc

```
.block {  
    border: 3px solid red;  
    border-top: 0;  
    border-bottom: 1px dashed #F08080;  
}
```

La bordure se situe entre la zone de padding et de margin.



```
a {  
    color: lightcoral;  
}  
  
/*  
 * Au survol de la souris, le lien sera  
 * de couleur bleu et ne sera pas surligné  
 */  
a:hover {  
    color: lightblue;  
    text-decoration: none;  
}  
  
/*  
 * Réagit quand <a> est activé  
 * Ex : Quand l'utilisateur clique dessus  
 */  
a:active {}  
  
/*  
 * L'utilisateur à déjà visité cette page  
 */  
a:visited {}
```

## Les pseudo-classes

Modifier le style d'un lien en fonction de son état

Une pseudo-classe est un mot-clé qui peut être ajouté à un sélecteur afin d'indiquer l'état spécifique dans lequel l'élément doit être pour être ciblé par la déclaration.

La pseudo classe vient se greffer au niveau du selecteur, que ce soit un élément, une classe, un id.

Il suffit d'accoler la pseudo class au selecteur à l'aide de deux points ":"

a:hover  
.classe:hover  
#id:hover

La pseudo-classe **:hover**, par exemple, permettra d'appliquer une mise en forme spécifique lorsque l'utilisateur survole l'élément ciblé par le sélecteur (changer la couleur d'un bouton par exemple).

Si ici on l'applique au <a>, elle peut être appliquée à une multitude d'éléments comme la <div>, le <p>, les titres <hn> etc. Le hover est la pseudo-classe la plus répandue.



```
<section class="container">
  <article class="card-blue">
    <div class="bloc-1">
      Bloc-1
    </div>
    <div class="bloc-2">
      Bloc-2
    </div>
    <div class="bloc-3">
      Bloc-3
    </div>
  </article>
</section>
```

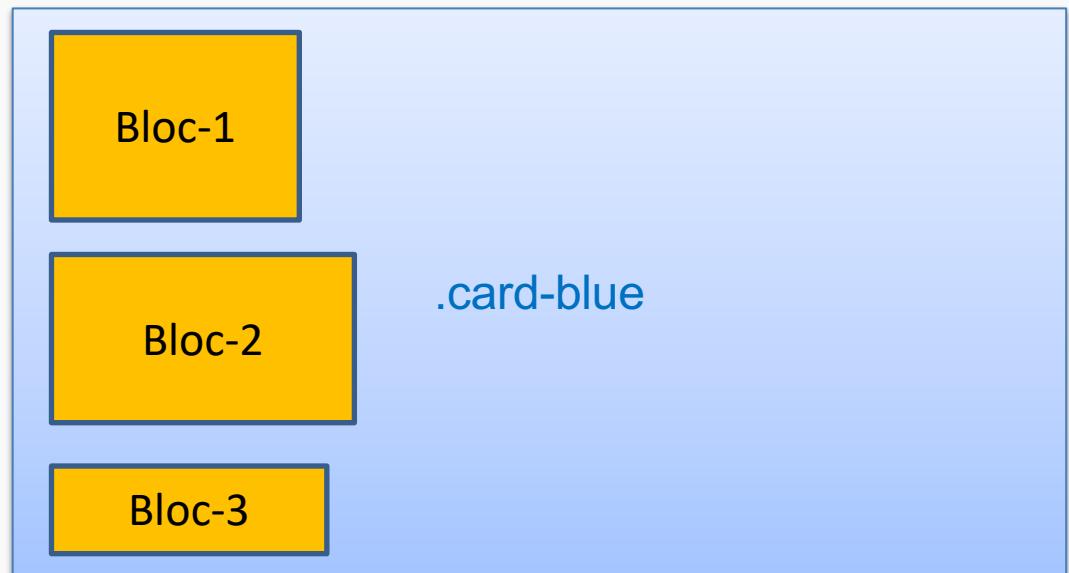
```
.card-blue {
  padding: 20px;
  border: 1px solid blue;
  background-color: blue;
}
```

## Comportement des blocs par défaut

Par défaut, les 3 divs oranges contenues dans la carte bleue vont venir se placer l'une en dessous de l'autre verticalement.

Avant les Flexbox il était difficile et laborieux pour les développeurs web de faire des mises en page responsives rapidement.

Aujourd'hui Flexbox est devenu incontournable car le meilleur moyen d'agencer les éléments d'un site web facilement



```
<section class="container">
<article class="card-blue">
  <div class="bloc-1">
    Bloc-1
  </div>
  <div class="bloc-2">
    Bloc-2
  </div>
  <div class="bloc-3">
    Bloc-3
  </div>
</article>
</section>
```

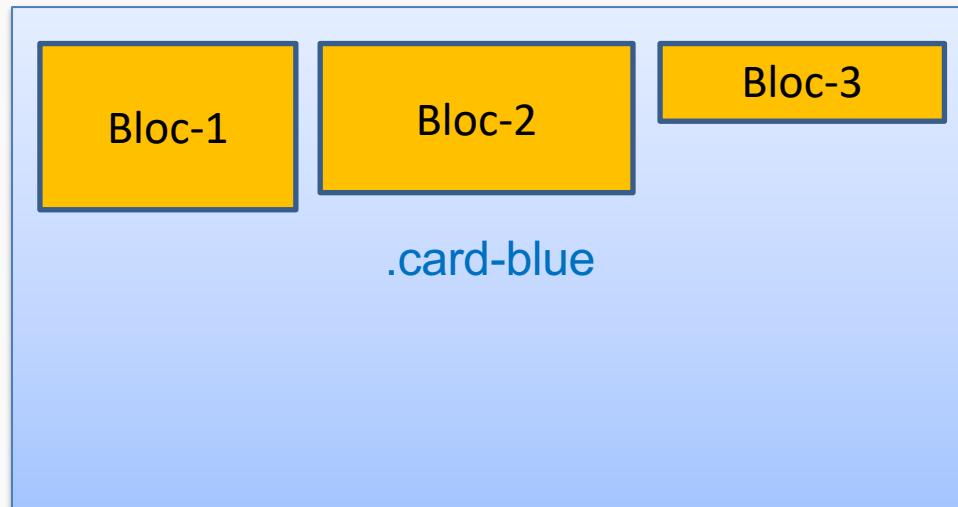
```
.card-blue {
  padding: 20px;
  border: 1px solid blue;
  background-color: blue;
  display: flex;
}
```

## Display: flex;

Pour transformer une div en flexbox il suffit d'ajouter la valeur "display: flex" à son style.

En faisant cela, la carte bleu va bénéficier d'une "intelligence" en plus avec des propriétés css permettant d'agencer facilement les éléments(enfants de premier niveau) qu'elle contient.

Les enfants de premier niveau de cette boite vont automatiquement venir s'aligner horizontalement en se partageant la largeur (width) disponible, c'est le comportement par défaut d'une flexbox.



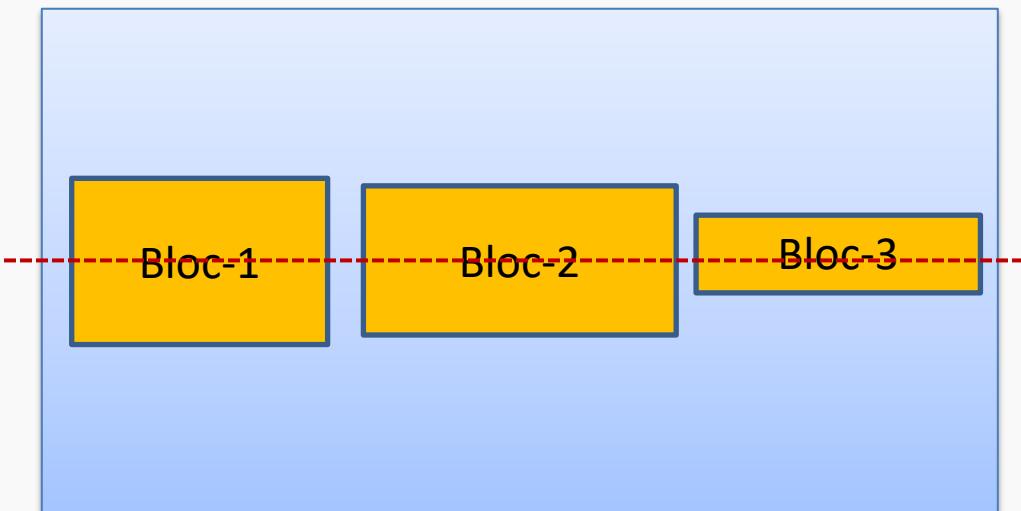
## Align-items

Pour centrer les blocs sur l'axe verticale

La propriété align-items permet dans une flexbox d'aligner verticalement nos différents blocs qui sont sur une même rangée. Cela peut-être pratique quand nos blocs ont une hauteur différente.

On va donc utiliser la propriété align-items avec la valeur "center" pour centrer les éléments verticalement dans la carte bleue.

```
.card-blue {  
    display: flex;  
    align-items: center;  
}
```

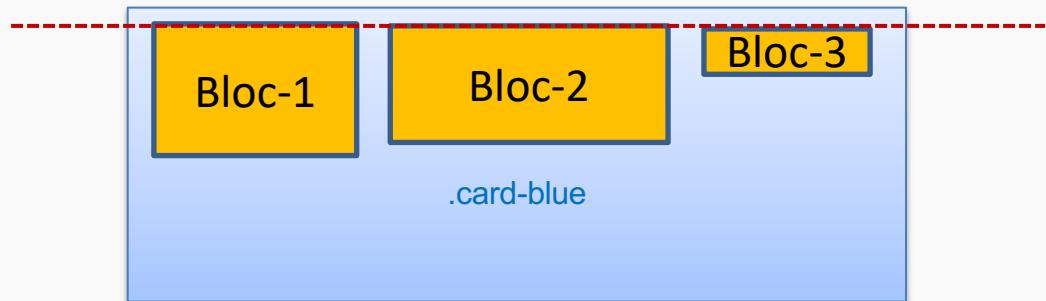


## Align-items

Pour agencer les blocs sur l'axe verticale

Il existe d'autres forme d'alignement :

- align-items: flex-start; (valeur par défaut)



- align-items: flex-end; (aligne le contenu vers le bas)



```
.card-blue {  
  display: flex;  
  align-items: flex-start;  
}
```

```
.card-blue {  
  display: flex;  
  align-items: flex-end;  
}
```

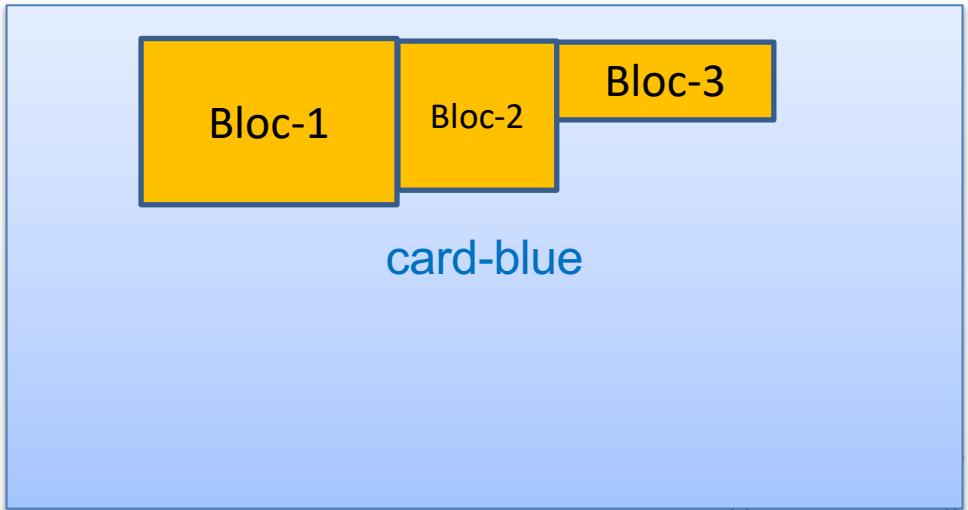
## Justify-content

Pour agencer les blocs sur l'axe horizontale

Lorsque nos blocs sur une rangée n'atteignent pas une valeur totale de 100%, il reste de l'espace disponible. La propriété justify-content permet de définir l'agencement de cet espace.

Une propriété populaire est justify-content: center; qui permet de centrer nos blocs en laissant une marge égale de part et d'autre.

```
.card-blue {  
    display: flex;  
    justify-content: center;  
}
```



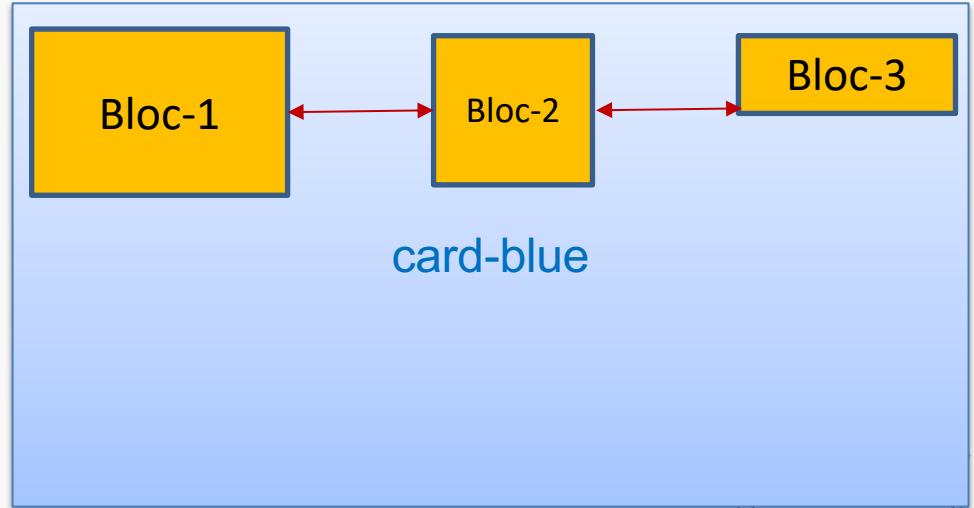
## Justify-content: space-between

Pour agencer les enfants sur l'axe horizontale

On a d'autres propriétés :

justify-content: space-between; Permet d'avoir des marges entre les blocs mais pas sur les bords le plus à gauche et le plus à droite.

```
.card-blue {  
    display: flex;  
    justify-content: space-between;  
}
```

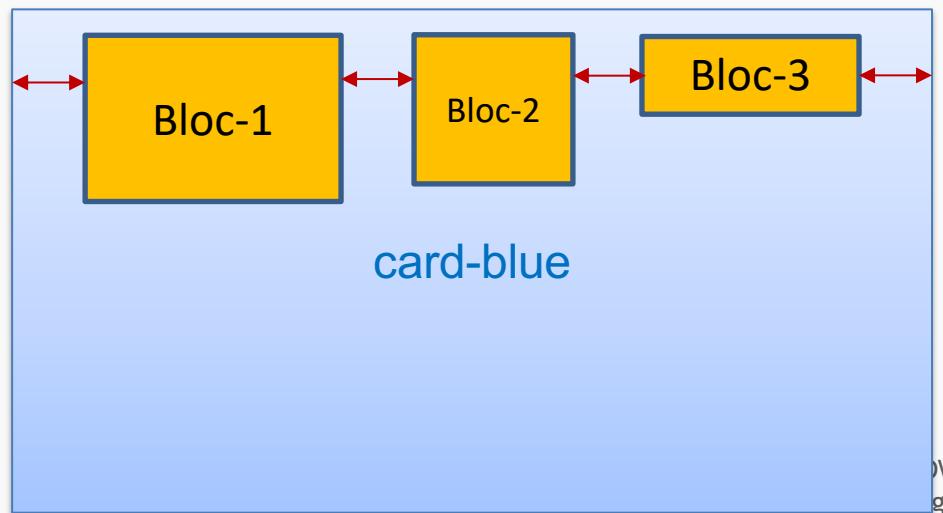


## Justify-content: space-around

Pour agencer les enfants sur l'axe horizontale

Chaque bloc possède une marge de même valeur à gauche et à droite.

```
.card-blue {  
    display: flex;  
    justify-content: space-around;  
}
```

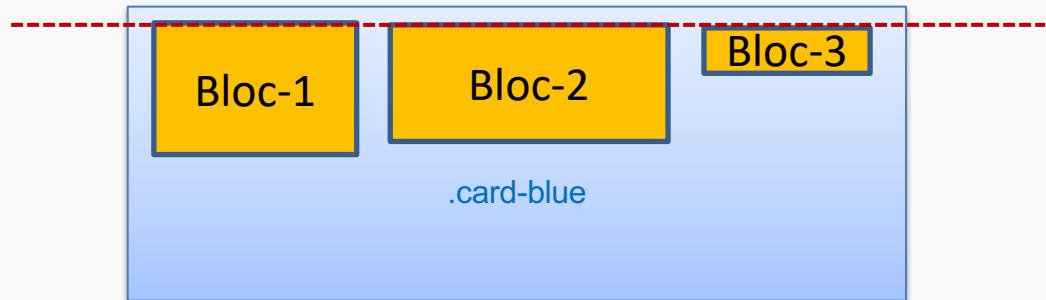


## Align-items

Pour agencer les enfants sur l'axe verticale

Il existe d'autres forme d'alignement :

- align-items: flex-start; (valeur par défaut)



- align-items: flex-end; (aligne le contenu vers le bas)



```
.card-blue {  
  display: flex;  
  align-items: flex-start;  
}
```

```
.card-blue {  
  display: flex;  
  align-items: flex-end;  
}
```

```
.card-blue {  
    display: flex;  
    flex-wrap: wrap;  
}  
  
.bloc-1 {  
    width: 50%;  
}  
  
.bloc-2 {  
    width: 50%;  
}  
  
.bloc-3 {  
    width: 100%;  
}
```

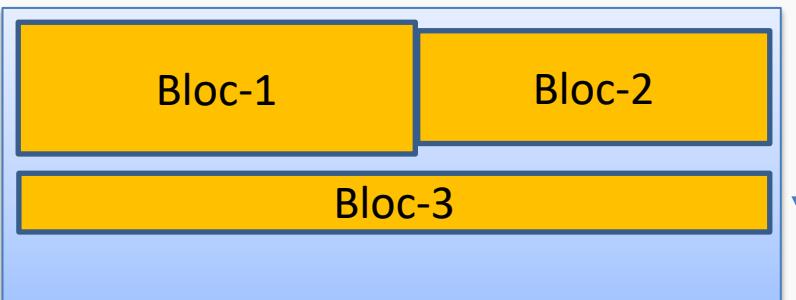
## Be flex

### Agencement avancé

On pourrait avoir envie d'avoir plutôt un enchaînement de deux colonnes sur une première ligne, puis une grande colonne unique sur une ligne en dessous.

Si je mets à jour les width, on va se retrouver dans une situation où le flex ne nous écoute pas, et conserve tout sur une seule ligne. Pour permettre le passage à la ligne, si la width totale des colonnes dépasse 100%, on va devoir appliquer un flex-wrap: wrap; sur le presentation-contenu.

On découvre là les bases du responsive !



## Be flex

### flex-direction

Lorsque nos blocs sont alignés mais que nous les souhaitons en colonne ( ou inversement ), il existe le flex-direction.

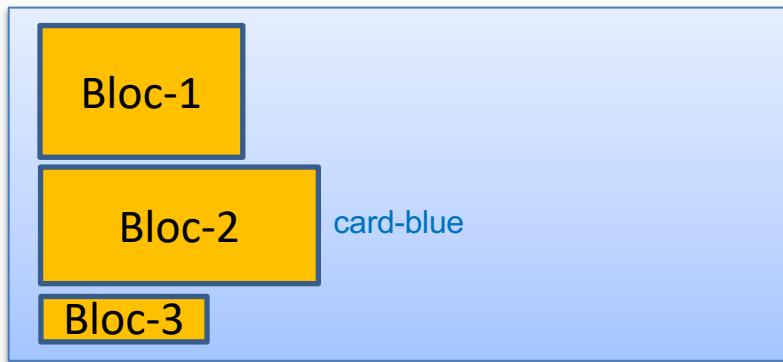
Pour changer cette direction, il suffit de faire flex-direction: column.

On a d'autres propriétés :

flex-direction: row; Permet de passer la colonne en ligne.

flex-direction: row-reverse; / flex-direction: column-reverse;  
Permettent de changer la direction et d'inverser l'ordre des éléments.

```
.card-blue {  
    display: flex;  
    flex-direction: column;  
}
```



# Be flex

## Agencement avancé

```
* {  
  box-sizing: border-box;  
  margin: 0;  
  padding: 0;  
}
```

L'astuce en + : en général, on va avoir tendance à ajouter du padding sur nos colonnes pour faire respirer les textes. Cela peut avoir pour conséquence de dérégler totalement nos tailles de colonnes.

Il existe en CSS une méthode qui permet d'inclure le padding et les bordures dans le calcul global des longueurs. Je l'utilise sur l'ensemble de mes projets. Il s'agit de la propriété box-sizing: border-box. Pour l'appliquer à tous mes éléments, je le mets dans une classe CSS \*. Je mets ce morceau de code en tout premier dans mon CSS.

J'en profite aussi pour supprimer un comportement bizarre des navigateurs, qui appliquent des marges par défaut. J'applique une marge et un padding à 0 au même endroit.

Ps: Pour devenir un pro de flexbox: [Flexbox froggy](#)



# Préparer son HTML

## L'importance du <head>, encore !

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport"
content="width=device-width, initial-scale=1.0">
  </head>

  <body>

  </body>
</html>
```



Lors de l'apparition des premiers smartphones, le responsive n'était pas vraiment celui que l'on connaît aujourd'hui. En effet, un chargement sur téléphone pouvait produire soit un résultat avec un scroll dans tous les sens, soit un dézoom du site pour le mettre à l'échelle pour que tout rentre dessus. L'une ou l'autre des solutions apportent des inconvénients visuels.

Il va donc être introduit une balise meta viewport, qui va permettre de manager la taille de son écran et de préparer son site au responsive.

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

**Balise obligatoire à mettre dans le <head>**, la width permet de définir la taille dans laquelle doit tenir le site (ici device-width = taille du téléphone, et un niveau de zoom initial (Ici de 1, donc une mise à l'échelle de 1).

```
.presentation-contenu {
    display: flex;
    flex-wrap: wrap;
}

.bloc-1 {
    width: 50%;
    height: 100px;
}

@media screen and (max-width: 800px) {

    .bloc-1 {
        width: 100%;
    }
}
```

## Responsive

### Le principe



L'idée du responsive au niveau du CSS, est d'aller poser une condition spéciale, qui vient limiter la taille de l'écran sur laquelle on veut appliquer des classes CSS. Cette condition se nomme la media query.

Pour déclarer une media query, on va écrire ceci :

```
@media screen and (max-width: 800px) {}
```

@media correspond au début de notre condition. On vient préciser sur quoi on veut poser une condition (nous ici screen pour écran ; cela aurait pu être pour une imprimante ou un synthétiseur vocal). Enfin on ajoute le mot clé and et entre parenthèse notre condition d'application. Ici on dit donc pour tous les écrans qui font une taille maximale de 800px. On n'oublie pas d'ouvrir des accolades.

A l'intérieur de cette condition, on peut ajouter de nouvelles classes CSS (et qui fonctionnent que si la condition est respectée), ou venir “écraser” des classes CSS qui existent déjà. La condition devient prioritaire, et donc la propriété CSS se met à jour selon la taille d'écran.

```
.presentation-contenu {  
    display: flex;  
    flex-wrap: wrap;  
}  
  
.bloc-1 {  
    width: 50%;  
    height: 100px;  
}  
  
@media screen and (max-width: 800px) {  
  
    .bloc-1 {  
        width: 100%;  
    }  
}
```

## Responsive

### Le principe



Dans notre exemple ici, le bloc-1 a de base une taille de 50%, à l'exception de tous les écrans dont la taille est inférieur à 800px. Dans cette condition, la taille passe alors à 100%.

Pour rappel, dans le cadre du flex, le flex-wrap: wrap, permet de faire passer les colonnes sur plusieurs lignes.

```
.presentation-contenu {  
    display: flex;  
    flex-wrap: wrap;  
}  
  
.bloc-1 {  
    width: 50%;  
    height: 100px;  
}  
  
@media screen and (max-width: 1100px) {  
  
    .bloc-1 {  
        width: 70%;  
    }  
  
}  
  
@media screen and (max-width: 800px) {  
  
    .bloc-1 {  
        width: 100%;  
    }  
  
}
```

## Responsive

### Le principe



On peut créer autant de condition que l'on souhaite. Ici la taille initiale de notre bloc est de 50%. Puis, pour les écrans en dessous de 1100px il fera 70%. Enfin pour les écrans en dessous de 800px il fera 100%.



```
.bloc-1 {  
    width: 50%;  
    height: 100px;  
}  
  
.bloc-2 {  
    width: 50%;  
    height: 100px;  
}  
  
@media screen and (max-width: 800px) {  
  
    .bloc-1 {  
        width: 70%;  
    }  
  
    .bloc-2 {  
        width: 30%;  
    }  
}
```

## Responsive

### Le principe



A l'intérieur d'une @media, on peut ajouter autant de classes CSS que l'on souhaite influencer.

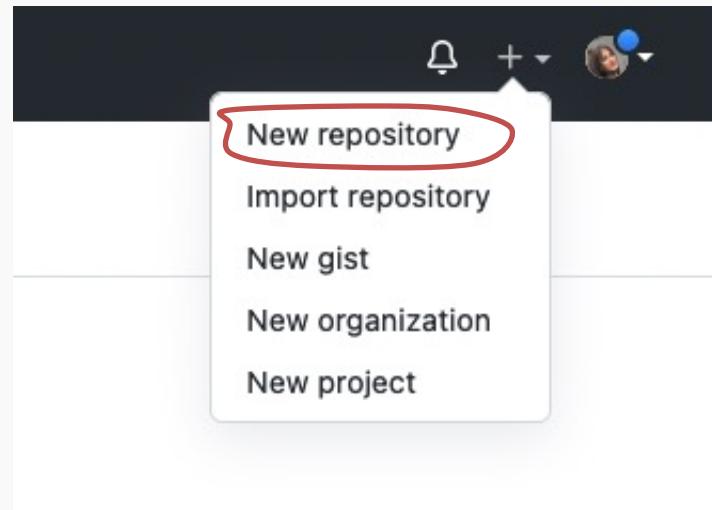


## Rendu

- ✓ Créer une Landing Page pour présenter un objet connecté de votre choix.
- ✓ Styliser la page pour vendre au mieux le produit.
- ✓ Usage des flexbox obligatoire
- ✓ Responsive obligatoire
- ✓ Rendu en zip de tous les fichiers (HTML, CSS, images...) sur DVO

## Mise en ligne sur GITHUB

- Création de votre compte, choisir votre nom puis prénom accolés pour le nom d'utilisateur
- Ajouter un nouveau Repository en cliquant sur le + en haut à droite du tableau de bord



# Mise en ligne sur GITHUB

## Paramétrer le Repository

- Choisir « landing-page » pour votre repository name
- Mettre le repository en public

### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

#### Repository template

Start your repository with a template repository's contents.

No template ▾

#### Owner \*

Select an owner ▾

#### Repository name \*

landing-page

Please select an owner. Names are short and memorable. Need inspiration? How about [fantastic-broccoli?](#)

#### Description (optional)

#### Public

Anyone on the internet can see this repository. You choose who can commit.

#### Private

You choose who can see and commit to this repository.

#### Initialize this repository with:

Skip this step if you're importing an existing repository.

#### Add a README file

This is where you can write a long description for your project. [Learn more.](#)

#### Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: None ▾

#### Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

License: None ▾

ⓘ You are creating a public repository.

# Mise en ligne sur GITHUB

- Choisir l'option pour uploader un fichier existant

**Quick setup — if you've done this kind of thing before**

Set up in Desktop   or    HTTPS    SSH   git@github.com:louisepicot/landing-page.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

**...or create a new repository on the command line**

```
echo "# landing-page" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin git@github.com:louisepicot/landing-page.git  
git push -u origin main
```

**...or push an existing repository from the command line**

```
git remote add origin git@github.com:louisepicot/landing-page.git  
git branch -M main  
git push -u origin main
```

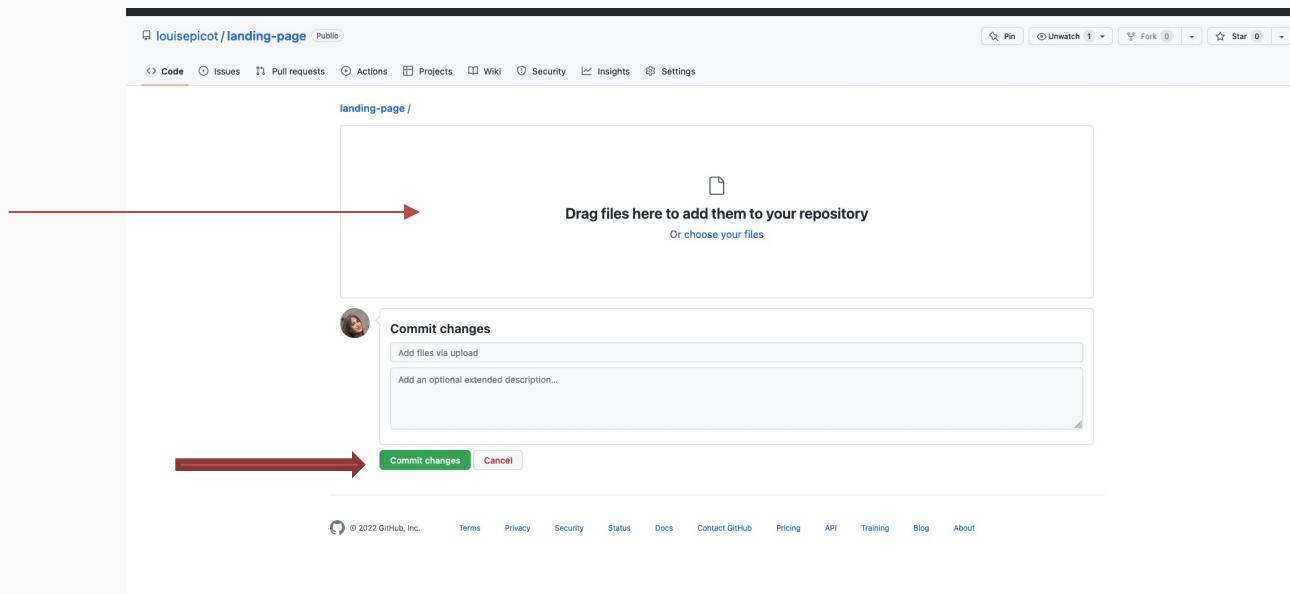
**...or import code from another repository**

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

**ProTip!** Use the URL for this page when adding GitHub as a remote.

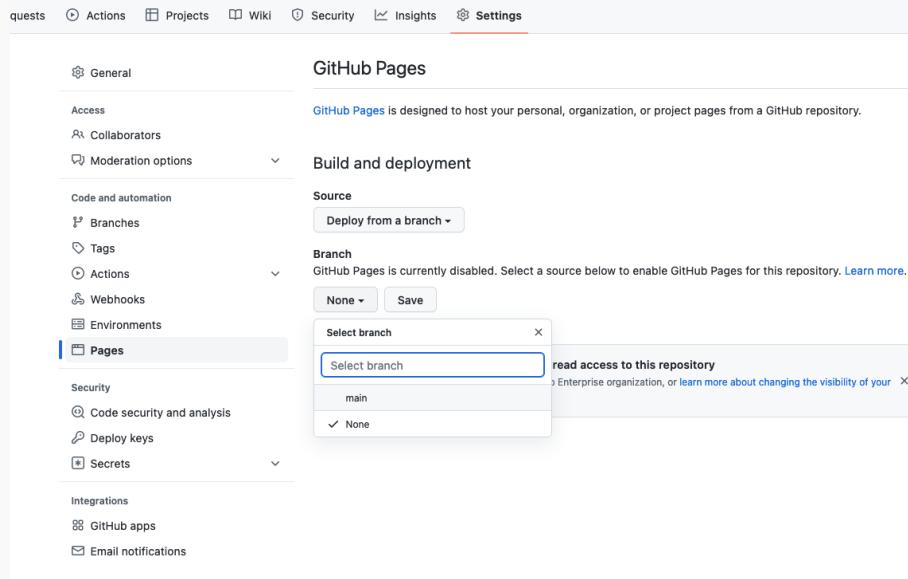
# Mise en ligne sur GITHUB

- Drag and Drop votre dossier
- Quand le dossier est entièrement chargé cliquer sur commit changes



# Mise en ligne sur GITHUB

- Une fois le dossier chargé, accédez aux réglages.
- Dans la section « pages », sélectionnez la branch « main » à la place de « none »
- Vous aurez ensuite accès à votre site sur le lien:  
<https://votrenom.github.io/landing-page/>





Flexbox et Responsive		Structure et syntaxe HTML				CSS			Github mise en ligne	Organisation	TOTAL
Utilisation et compréhension de flexbox	Utilisation de media queries pour une version mobile + desktop	Wireframes et structure cohérente de l'html	Respect de 5 règles du html	Utilisation des balises appropriées	Ajout des meta tags et liens externes	Choix des sélecteurs appropriés	Choix de noms de sélecteurs cohérents	Syntaxe et organisation du css	Ajout du dossier du site sur github et partage du lien	Indentation, organisation du dossier, respect des consignes de nomenclature...	
3	2	2	1	2	1	2	1	2	2	2	20

# Où trouver les réponses à vos questions ?

## MDN

Documentation sur toutes les balises HTML et attributs CSS et autres

<https://developer.mozilla.org/fr/>

## Openclassroom

Formations gratuites sur les langages de programmations

<https://openclassrooms.com/>



## Grafikart

Vidéos tutoriels sur différentes problématiques web et formations

<https://www.grafikart.fr/>

## W3school (**à éviter**)

Contenu sur les bases des langages de programmation

<https://www.w3schools.com/>

# Formations payantes

## Elephorm

Formations complètes en français multi-domaines (Web, graphisme, 3D...)

<https://www.elephorm.com/>

## Udemy

Formations sur les langages de programmations (en Anglais)

<https://www.udemy.com/>



# Demander de l'aide

## StackOverflow

Forum d'aide à destination des développeurs (en Anglais)

<https://stackoverflow.com/>