Responsive Web Design

Romain GONÇALVES

epsi

wis

l'école [tech]
de l'expertise digitale

Plan de cours

I. Rappels de HTML/CSS
II. Conception d'IHM
III. Media Queries
IV. Grid(s) & Flexbox
V. Frameworks

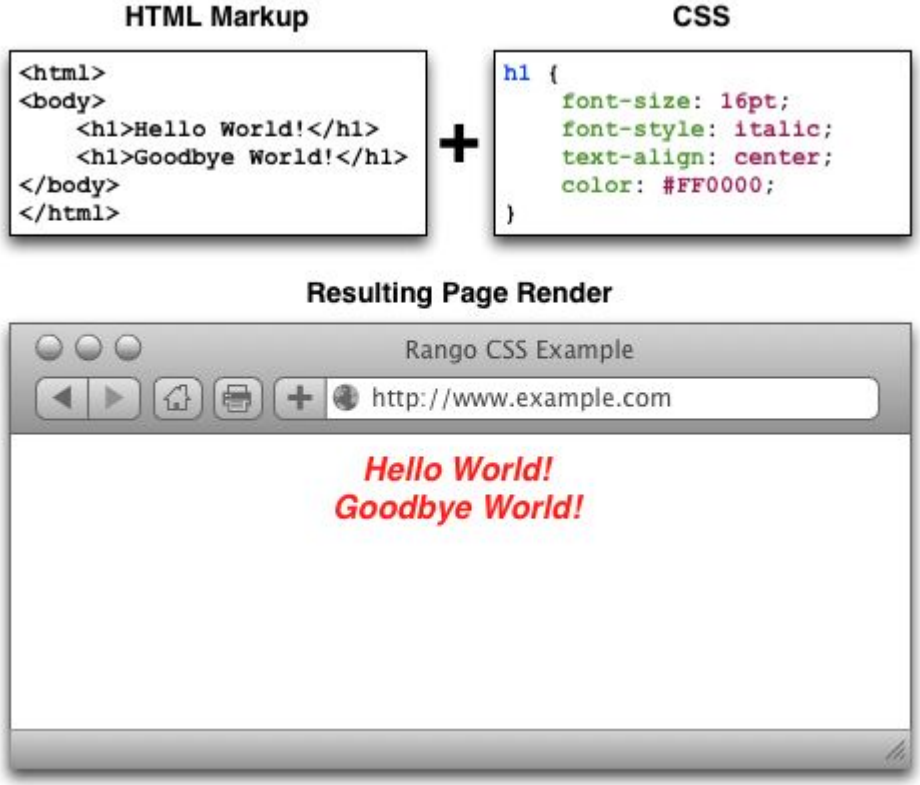
I. Rappels de HTML/CSS

Anatomie d'une page web

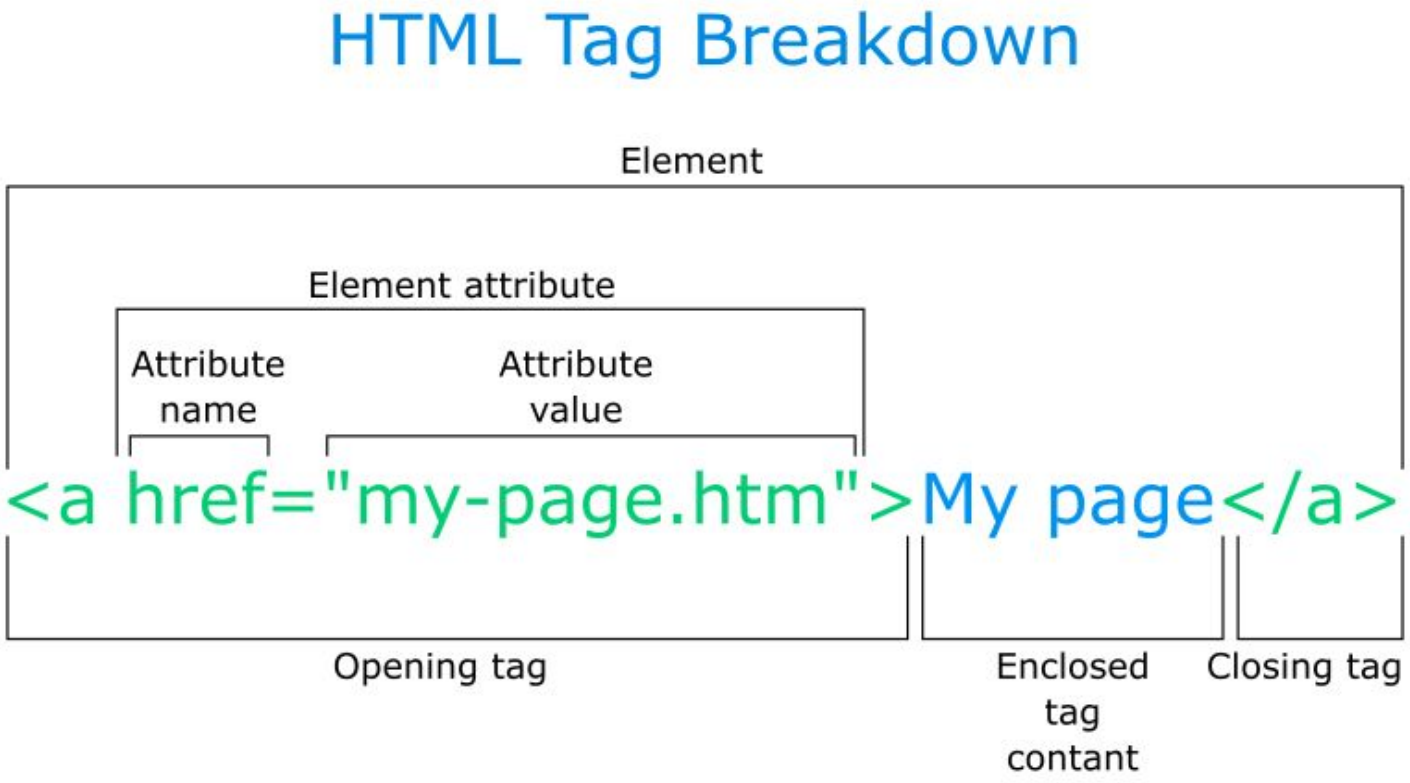
Les sites web que l'on consulte se composent de 3 éléments :

- **HTML** (HyperText Markup Language, Langage de Balisage d'HyperTexte) : Langage à balises permettant de décrire la structure d'une page
- **CSS** (*Cascading Style Sheets*, Feuilles de Style en Cascade) : Langage permettant de décrire la présentation d'un document HTML
- **JavaScript** (facultatif) : Langage de programmation permettant de mettre en place des interactions dynamiques sur la page

<u>N.B. :</u> HTML et CSS ne sont pas des langages de programmation, ils ne possèdent pas de logique.



Structure d'une balise



Structure d'une page

```html
<!-- La balise DOCTYPE indique que le document est du HTML5 -->
<!DOCTYPE html>
<!-- La balise <html> délimite le document (et indique éventuellement la langue) -->
<html lang="en">
    <!-- La balise <head> contient l'entête de page (informations de caractérisation) -->
    <head>
        <!-- La balise <meta> contient des informations de caractérisation -->
        <!-- Le paramètre charset permet de spécifier l'encodage de caractères utilisé par le fichier HTML -->
        <meta charset="UTF-8">
        <!-- Le paramètre X-UA-Compatible sert à définir la stratégie de compatibilité avec Internet Explorer -->
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <!-- Le paramètre viewport sert à définir la mise en page sur mobile -->
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <!-- La balise title contient le titre de la page tel qu'affiché par le navigateur -->
        <title>Titre de la page</title>
    </head>
    <!-- La balise <body> contient le corps de page (contenu affiché à l'utilisateur) -->
    <body>

    </body>
</html>
```

Structure d'une page : inclusion de CSS

Pour inclure du CSS, il y a deux possibilités :

- L'ajout de code CSS directement dans une balise **`style`**
- L'ajout d'une balise **`link`** permettant de référencer une feuille de style externe

Ces 2 méthodes sont combinables et répétables.

A moins de n'avoir que très peu de règles CSS (rare), on préfèrera utiliser un ou plusieurs fichiers CSS externes afin de mieux structurer le code.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Titre de la page</title>
    <!-- CSS inline : directement dans une balise style -->
    <style>
        body {
            font-weight: bold;
            color: red;
        }
    </style>
    <!-- CSS externe : via une balise link -->
    <link rel="stylesheet" href="style.css">
</head>
<body>
    Corps de texte
</body>
</html>
```

Structure d'une page : inclusion de JS

Pour inclure du JS, il y a deux possibilités :

- L'ajout de code JS directement dans une balise **`script`**
- L'ajout d'une balise **`script`** possédant l'attribut src permettant de référencer un fichier de script externe

Ces 2 méthodes sont combinables et répétables. Comme pour le CSS, on préfèrera utiliser un ou plusieurs fichiers JS externes afin de mieux structurer le code.

Les bonnes pratiques indiquent de charger le CSS après le corps de page (juste avant la balise fermante **`body`**) pour optimiser la vitesse de chargement de page

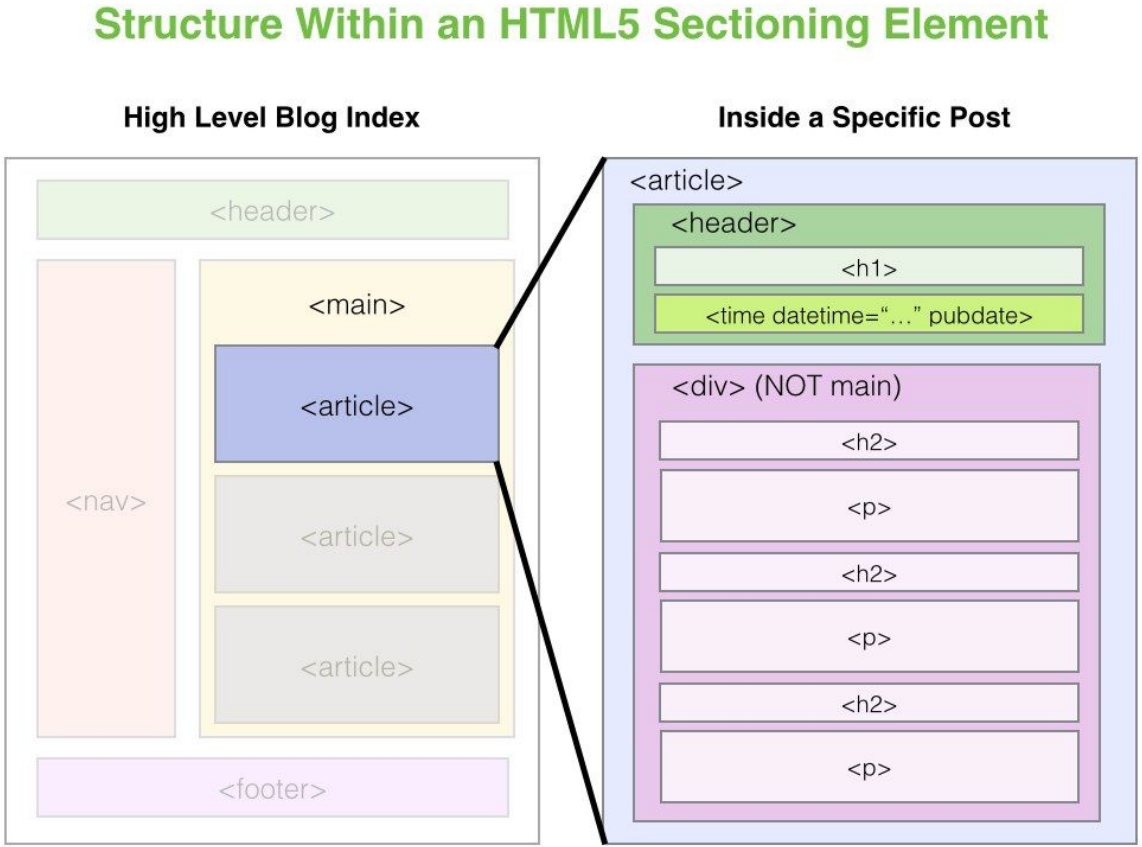
```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Titre de la page</title>
    </head>

    <body>
        Corps de texte
        <!-- JS externe : via une balise script possédant l'attribut src -->
        <script src="script.js" type="text/javascript"></script>
        <!-- JS inline : directement dans une balise script -->
        <script type="text/javascript">
            alert("Bonjour !");
        </script>
    </body>
</html>
```

HTML : Balises sémantiques

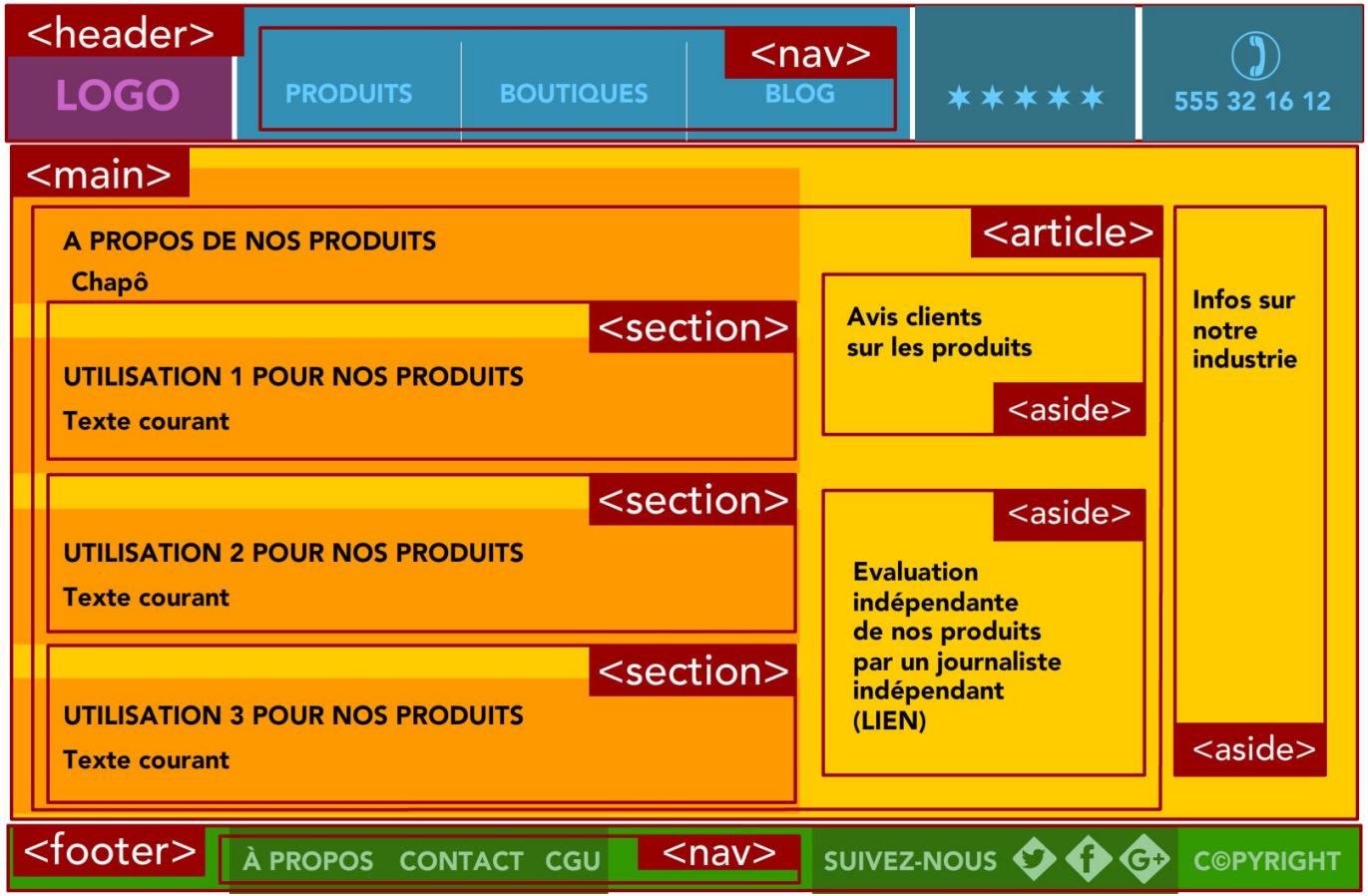
Ces balises permettent de structurer le contenu de la page et aux robots de mieux l'indexer. Il y a deux catégories : les balises de structure et les balises de contenu

- Balises de structure *(n'appliquent aucune mise en forme)* :
  - **`header`** : Entête de page (bandeau supérieur, logo, ...)
  - **`nav`** : Menu de navigation
  - **`main`** : Contenu principal
  - **`footer`** : Pied de page
  - **`aside`** : Contenu annexe (barre latérale, éléments complémentaires au contenu principal)
  - **`section`** : Identifications de contenus
  - **`article`** : Identification de contenus
- Balises de contenu *(peuvent appliquer de la mise en forme)* :
  - **`h1`**, **`h2`**, **`h3`**, **`h4`**, **`h5`**, **`h6`** : Titres de niveau 1 à 6
  - **`p`** : Paragraphe de texte
  - **`blockquote`** et **`cite`** : Bloc de citation et source
  - **`address`** : Adresse
  - **`time`** : Temporalité
  - **`data`** : Donnée générique
  - ...

HTML : Balises sémantiques

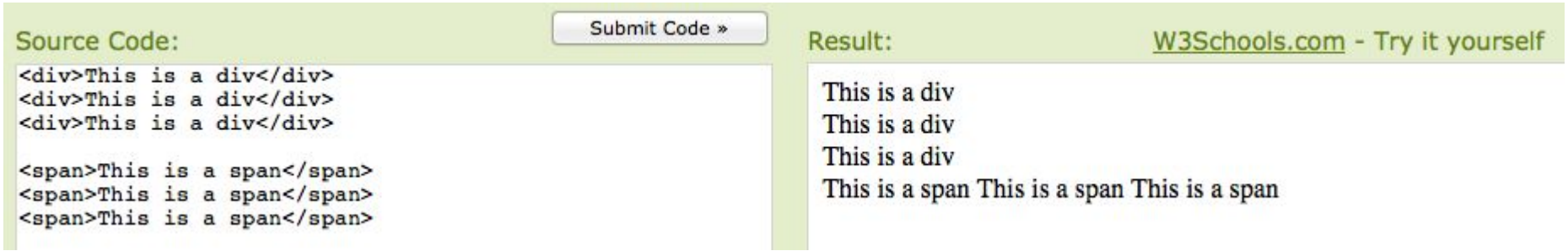


HTML : Balises sémantiques



HTML : Blocs et inline

- Les balises **`div`** et **`span`** sont des conteneurs génériques (ils n'appliquent aucune mise en forme)
- La différence entre les deux est que **`div`** est un conteneur de type **`block`** :
  - Il est destiné à lui même contenir des éléments HTML de bloc (paragraphe, listes, ...)
  - Il va automatiquement forcer un retour à la ligne après
- Au contraire, **`span`** est un conteneur **`inline`** :
  - Il est destiné à être utilisé à l'intérieur d'un enfant, et ne contiendra pas (ou peu) d'autres éléments à l'intérieur
  - Il est intégré au flux de la page et ne force pas de retour à la ligne
- Ces valeurs peuvent être forcées à l'aide de la propriété CSS **`display`**



HTML : Formulaires

- La balise **`form`** permet de définir un formulaire contenant des champs :
  - **`input`** : Élément de formulaire caractérisé par l'attribut **`type`** :
    - **`text`** : texte (sans retour à la ligne)
    - **`password`** : mot de passe
    - **`checkbox`** : case à cocher
    - **`radio`** : bouton de sélection
    - **`file`** : fichier à téléverser
    - **`hidden`** : champ caché
  - **`select`** et **`option`** : Liste déroulante de sélection et élément de la liste
  - **`textarea`** : Zone de texte (retour à la ligne possible)
  - **`button`** : Bouton
- D'autres balises permettent d'améliorer l'ergonomie du formulaire :
  - **`label`** : Label de champ
  - **`fieldset`** : Regroupement de champs
- La balise **`form`** contient généralement 2 attributs :
  - **`action`** : Détermine l'adresse à laquelle le contenu du formulaire doit être envoyé
  - **`method`** : Méthode HTTP (**`GET`** ou **`POST`**) utilisée pour l'envoi (généralement **`POST`**)

HTML : Tableaux

La balise **`table`** permet de définir un tableau. Cette balise contient des enfants sur plusieurs niveaux, avec par exemple la hiérarchie suivante :

- **`title`** : Titre du tableau
- **`thead`** : Entête du tableau
  - **`tr`** : Ligne
    - **`th`** : Entête de colonne
- **`tbody`** : Corps du tableau
  - **`tr`** : Ligne
    - **`td`** : Cellule
- **`tfoot`** : Pied du tableau
  - **`tr`** : Ligne
    - **`td`** : Cellule

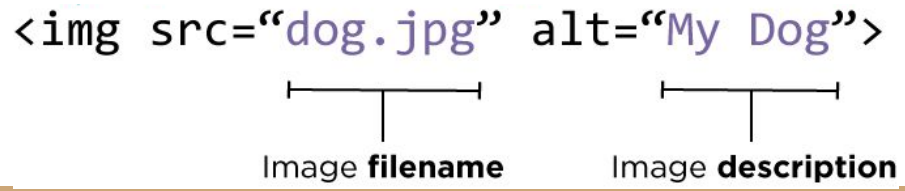
Grâce aux attributs **`rowspan`** et **`colspan`**, il est possible de fusionner des cellules verticalement et horizontalement

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Titre de la page</title>
</head>
<body>
    <table>
        <title>Notes</title>
        <thead>
            <tr>
                <th>Nom</th>
                <th>Prénom</th>
                <th>Note sur 20</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>T</td>
                <td>Alice</td>
                <td>13</td>
            </tr>
            <tr>
                <td>N</td>
                <td>Bob</td>
                <td>14</td>
            </tr>
            <tr>
                <td>M</td>
                <td>Eve</td>
                <td>18</td>
            </tr>
            <tr>
                <td>W</td>
                <td>Oscar</td>
                <td>15</td>
            </tr>
            <tr>
                <td>S</td>
                <td>Julie</td>
                <td>16</td>
            </tr>
        </tbody>
        <tfoot>
            <tr>
                <td colspan="2">Moyenne</td>
                <td>15,2 / 20</td>
            </tr>
        </tfoot>
    </table>
</body>
</html>
```

Nom	Prénom	Note sur 20
T	Alice	13
N	Bob	14
M	Eve	18
W	Oscar	15
S	Julie	16
Moyenne		15,2 / 20

HTML : Images

- En HTML, l'affichage d'une image dans la page se fait à l'aide de la balise **`img`**.
- Il s'agit d'une balise auto-fermante, ce qui signifie qu'on ne place rien à l'intérieur de la balise, elle se suffit à elle même
- On peut ajouter un slash à la fin de la balise mais ce n'est pas obligatoire
- Toutes les informations de caractérisation sont fournies via les attributs :
  - **`src`** : Chemin vers l'image
  - **`alt`** : Texte de remplacement, utilisé également comme description courte de l'image
  - **`role`** : Rôle de l'image (par exemple **`presentation`** pour les images purement décoratives)
  - **`longdesc`** : Description longue (lien vers une ancre ou une autre page)
  - **`height`** : Hauteur en pixels
  - **`width`** : Largeur en pixels
- L'attribut **`src`** est obligatoire pour charger l'image, et **`alt`** est très fortement recommandé pour l'accessibilité du site (liseuse d'écran)



HTML : Autres balises

Parmi les autres balises importantes :

- **`a`** : Lien vers une section de la page (lien d'ancre) ou vers une autre page (lien hypertexte) avec l'attribut **`href`**
- **`strong`** : Mise en gras du texte
- **`em`** : Mise en italique du texte
- **`u`** : Soulignement du texte
- **`s`** : Texte barré
- **`sup`** et **`sub`** : Exposant et indice
- **`ul`** et **`li`** : Liste à puces et élément de liste
- **`ol`** et **`li`** : Liste numérotée et élément de liste
- **`dl`**, **`dt`** et **`dd`** : Liste de définitions, terme défini et définition
- **`br`** : Retour à la ligne *(balise auto-fermante)*
- **`hr`** : Séparateur (ligne) horizontal *(balise auto-fermante)*
- **`audio`** : Contenu audio
- **`video`** : Contenu vidéo
- ...

HTML : ID et classes

Afin de se repérer dans le HTML pour lui appliquer du style en CSS, on dispose de 2 attributs permettant de caractériser une ou plusieurs balises : **`id`** et **`class`**

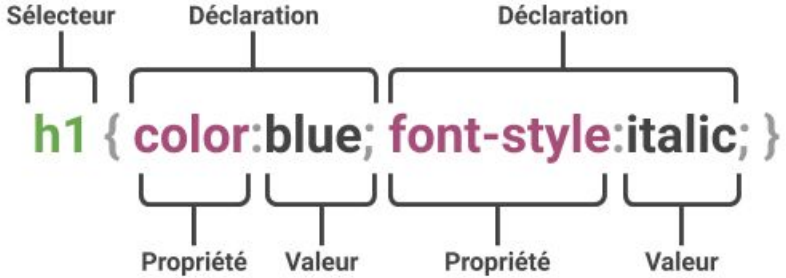
	`id`	**`class`**
Type de valeur	texte (alphanumérique, tirets et tirets-bas)	texte (alphanumérique, tirets et tirets-bas)
Nombre de valeurs	1	Infinies (séparées par un espace
Répétition dans la page	1	Infinies

Ces attributs sont combinables : une balise peut posséder un id, une classe, les deux, ou aucun des deux.

Attention : il est possible d'utiliser un **`id`** plusieurs fois dans une même page, mais cela est fortement déconseillé car source d'erreurs. Aucun message d'erreur ne sera affiché si c'est le cas (seul un avertissement dans la console développeur)

CSS : Syntaxe

- Le CSS se compose d'un ensemble de "règles", portant toutes la même structure :
  - Un sélecteur permettant d'identifier les éléments HTML concernés
  - Les déclarations de style à appliquer
- Les déclarations de style d'un sélecteur sont encadrées par des accolades (`{` et `}`)
- Chaque déclaration de style contient une propriété et une valeur séparées par un deux-points
- Chaque déclaration de style se termine par un point virgule
- On préfère mettre une seule déclaration de style par ligne, mais ce n'est pas obligatoire

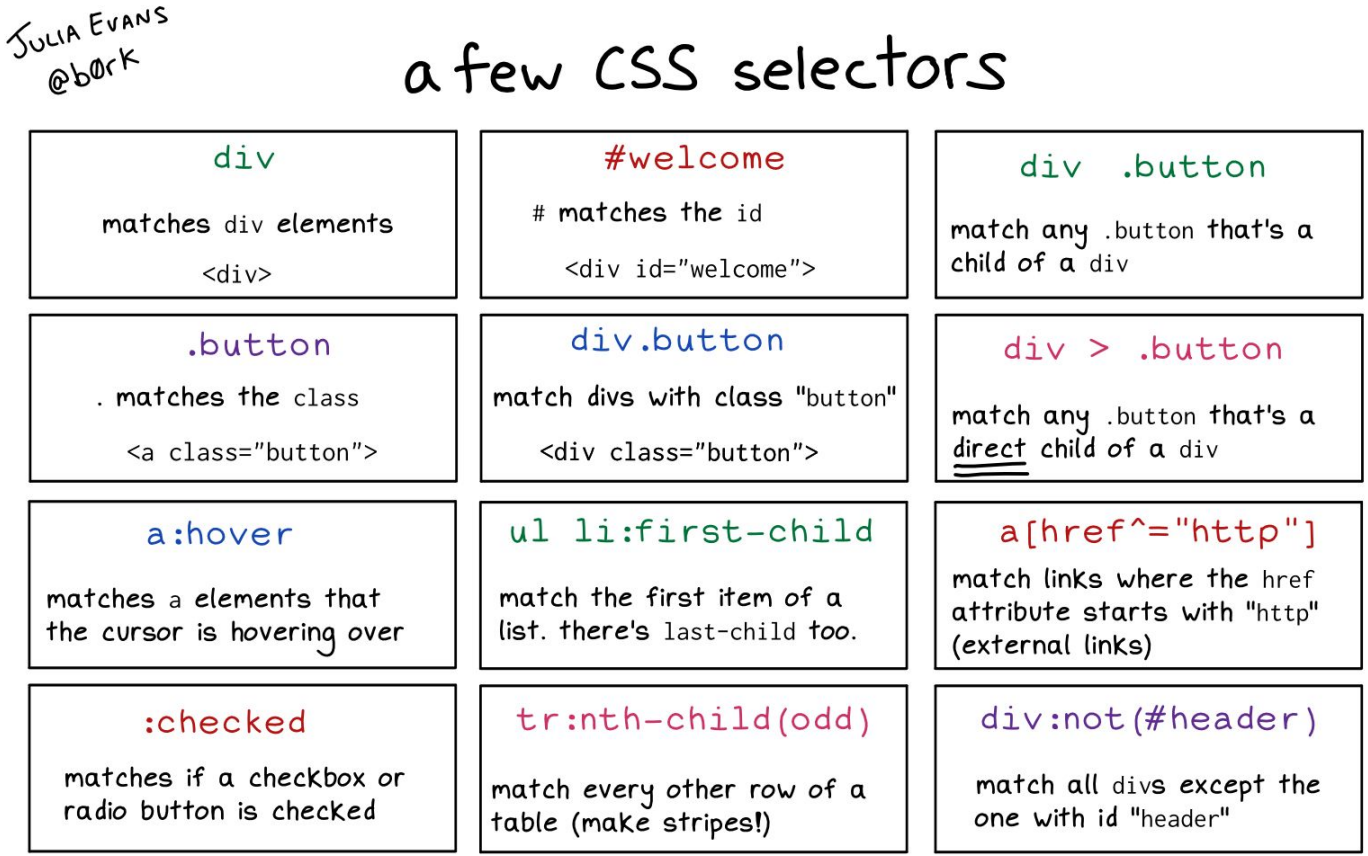


- Si une balise correspond à plusieurs règles, alors le sélecteur le plus précis dispose de la priorité.
- Si deux sélecteurs sont du même niveau de précision, alors la dernière règle déclarée est conservée.

CSS : Sélecteurs

Appellation	Exemple	Description de l'exemple
Balise	`img`	Les balises HTML `img`
ID	`#titre1`	L'élément HTML portant l'ID `titre1`
Classe	`.bleu`	Les éléments portant la classe `bleu`
Combinaison	`div.titre`	Les élément `div` portant la classe `titre`
Descendant	`div.corps .lien`	Les éléments avec la classe `lien` enfant (direct ou non) d'un élément div portant la classe `corps`
Enfant	`.corps > a.lien`	Les éléments `a` avec la classe `lien` enfant direct d'un élément portant la classe `corps`
Successeur immédiat	`.titre + .corps`	Les éléments avec la classe `corps` placés directement après un élément portant la classe `titre` (même parent)
Successeur	`p ~ ul`	Les éléments `ul` placés (directement ou non) après une balise `p` (même parent)

CSS : Sélecteurs



II. Conception d'IHM

Responsive Web Design

- En 2011, Ethan Marcotte, designer indépendant, imagine un système d'interface appelé "Responsive Web Design" (ou RWD), qui utilise les possibilités offertes par la version 3 du standard CSS pour adapter l'affichage de la page.
- 3 fonctionnalités sont pour cela mise en oeuvre :
  - Les "Media Queries" qui permettent d'utiliser des règles CSS différentes en fonction de la largeur d'écran du terminal de consultation. Ces largeurs (relativement standards), sont appelées breakpoints (points de rupture) et sont détaillées dans la partie 3 de ce cours
  - Le concept de "grille fluide" qui permet de dimensionner les éléments de blocs par rapport aux autres. On utilise pour cela des unités relatives (pourcentages, EM, REM) plutôt que des unités absolues (pixels, points)
  - Les images nécessitant de s'adapter au terminal sont également dimensionnées en unités relatives
- Il existe à l'origine deux approches au RWD :
  - **Responsive degradation :** En partant de la version ordinateur, on réduit progressivement la largeur d'écran et on redimensionne ou on retire les éléments de la page.
  - **Mobile first**

Mobile First

- L’approche “Mobile First” est aujourd’hui de plus en plus plébiscitée par les développeurs, designers et ergonomes.
- En concevant d’abord l’interface mobile, on épure au maximum la page de ses éléments inutiles
- Puis au fur et à mesure que la largeur de page augmente, on redimensionne et on réorganise les éléments pour faciliter la lecture
- Au final le gain au niveau de l’expérience utilisateur (UX) est non négligeable
- Cette approche est motivée par l’augmentation constante de la proportion d’utilisateurs mobiles, et la priorisation du référencement naturel des sites proposant une version mobile adaptée par les robots des moteurs de recherche
- Seule exception à cette règle sont les applications web, qui vont généralement proposer des interfaces différentes selon le terminal de consultation

Impact du mobile

- L’utilisation de terminaux mobiles (smartphones, tablettes) a plusieurs impacts à prendre en compte sur la conception d’interface, car l’utilisation qui en sera faite sera forcément différente de celle sur ordinateur :
  - La transition portrait/paysage (peut être théoriquement bloquée mais c’est de la “bidouille”)
  - La gestion tactile du site :
    - Boutons et liens plus gros
    - Pas de drag&drop (glisser/déplacer)
  - L’espace limité sur la page :
    - Contenu essentiel uniquement
    - Navigation masquable (burger menu)
- On utilisera généralement sur mobile une disposition (un “layout”) dit “mono-colonne” (“single column”) par opposition au bureau (et éventuellement une tablette en paysage) qui va comporter plusieurs colonnes.

III. MediaQueries

Navigateurs mobiles

Afin de pouvoir adapter la navigation aux appareils mobiles, plusieurs fonctionnalités ont été mises à disposition des développeurs.
Au démarrage, lorsque le site devait être accessible depuis le mobile, on utilisait le JavaScript. L'objet **`navigator`** permet d'obtenir des informations sur le navigateur de l'utilisateur, et ainsi de détecter un accès via le mobile (par exemple en analysant le **User Agent**). Les utilisateurs mobiles peuvent ainsi être redirigés sur un site mobile adapté à la visualisation sur des petits écrans.
Cette solution a vite été oubliée en raison du coût associé au maintien de 2 sites distincts.

En CSS2, il était déjà possible de détecter le type d'écran (**`screen`**, **`print`**, **`projection`**, ...) pour choisir les styles appliqués (propriété **`media`** sur les balises **`link`**, ou directive **`@media`** dans les feuilles de style). Depuis le CSS 3, il est possible via les **media queries** de combiner cette propriété avec des propriétés CSS, afin de détecter entre autres la largeur et la hauteur de l'affichage, ou le ratio d'affichage.

Aujourd'hui un site responsive doit être navigable verticalement sur mobile, afin d'être cohérent avec le reste des applications. **La page ne doit pas pouvoir être défilée horizontalement**, car ce serait perçu comme une incompatibilité (partielle) par les moteurs de recherche, qui pénaliserait le référencement du site.

Breakpoints

Lorsqu'on développe un site **responsive**, on utilise généralement 5 intervalles de largeur d'écran afin de pouvoir s'adapter à la majorité des situations :

Breakpoint	**XS**	**SM**	**MD**	**LG**	**XL**
Largeur d' écran *(librairie Bootstrap)*	< 576 px	≥ 576 px < 768 px	≥ 768 px < 992 px	≥ 992 px < 1200 px	> 1200 px
Utilisation *(non exhaustif)*	Smartphone (portrait)	Smartphone (paysage) Tablette (portrait)	Tablette (paysage)	PC (petit écran)	PC (grand écran) TV

Attention à la résolution des smartphones/tablettes : par exemple, l'écran d'un iPhone 12 Pro a une résolution physique de 1170x2532 pixels (portrait), soit un écran dit "2K" (supérieur au Full HD 1920x1080). Cependant, l'appareil possède un DPR (device pixel ratio) de 3, et donc une résolution logique 3 fois inférieure pour la majorité des usages (web, application), soit 390x844 (toujours en portrait) !
Chaque fabricant de smartphone possède ses propres valeurs DPR.

Règles conditionnelles

Grâce à la directive **`@media`** des feuilles de style, il est possible de créer des conditions permettant d'activer ou non des déclarations de style en fonction du type de média (**`screen`**, **`print`**, **`speech`**, **`all`**) et de ses caractéristiques, parmi lesquelles :

- **`width`**, **`min-width`**, **`max-width`** : La largeur (en pixels)
- **`height`**, **`min-height`**, **`max-height`** : La hauteur (en pixels)
- **`aspect-ratio`**, **`min-aspect-ratio`**, **`max-aspect-ratio`** : Le ratio d'affichage
- **`orientation`** : Orientation (**`portrait`** ou **`landscape`**)
- ...

Ces règles peuvent être combinées avec des opérateur logiques (et, ou, négation), et peuvent également être appliquées via la propriété **`media`** de la balise **`link`**, ce qui permet de charger différents fichiers CSS en fonction des media queries.

Règles conditionnelles

```
/* normal style */
#header-image {
    background-repeat: no-repeat;
    background-image:url('image.gif');
}

/* show a larger image when you're on a big screen */
@media screen and (min-width: 1200px) {
    #header-image {
        background-image:url('large-image.gif');
    }
}

/* remove header image when printing. */
@media print {
    #header-image {
        display: none;
    }
}
```

Framework breakpoints

Pour éviter d'avoir à définir soi-même toutes les media queries et tous les styles d'éléments d'un site, on utilise en général des frameworks CSS, qui pré-définissent des styles (et potentiellement des composants) sous forme de classes. Ces classes peuvent souvent être appliquées conditionnellement à un breakpoint minimum.

Par exemple, Bootstrap dispose d'un ensemble de classe "display", permettant de définir le type d'affichage d'un élément (par exemple **`display: block;`**), via des classes suivant le format **`.d-{value}`** (ou value vaut **`block`**, **`inline`**, ...). Elle définit également des classes sous le format **`.d-{breakpoint}-{value}`** (ou breakpoint vaut **`sm`**, **`md`**, **`lg`**, **`xl`** ou **`xxl`**) qui s'appliqueront à partir du breakpoint donné et aux breakpoints de largeur plus grande (le breakpoint XS n'est pas présent car il correspond à l'absence de breakpoint).
Dans ce cas, si on souhaite afficher un élément spécifique pour les breakpoint XS et SM, et un autre à partir du breakpoint MD, on pourra faire comme ceci :

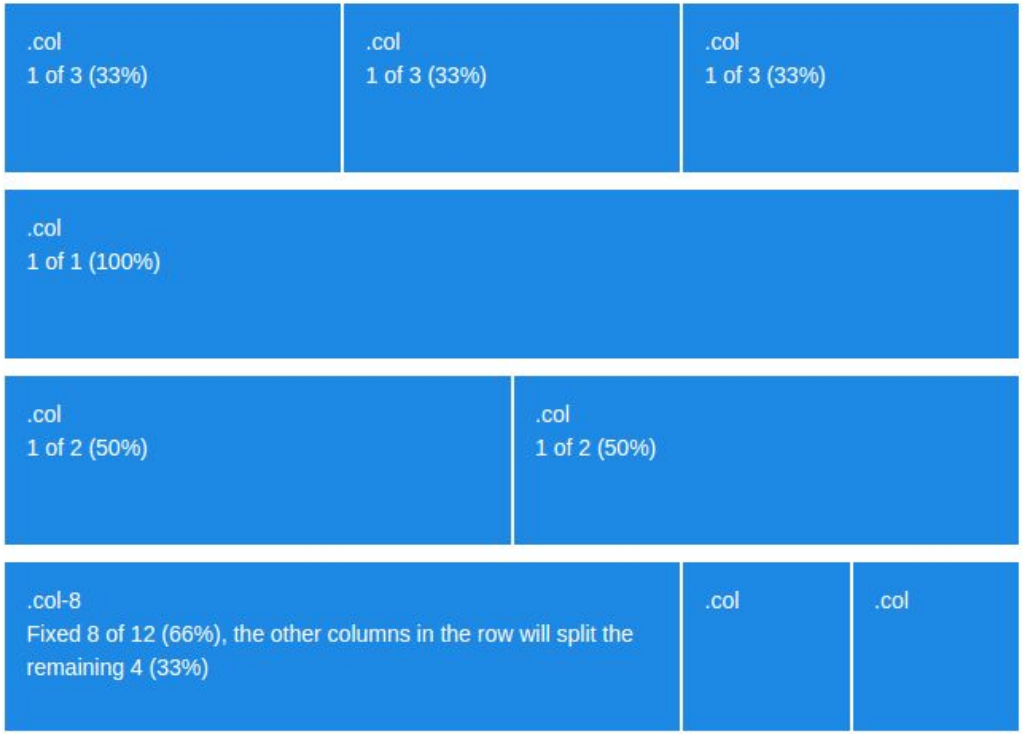
```html
<div id="affichage_mobile" class="d-block d-md-none">
    Contenu mobile
</div>
<div id="affichage_pc" class="d-none d-md-block">
    Contenu PC
</div>
```

IV. Grid(s) et Flexbox

Bootstrap grid

- Afin de faciliter la disposition d'éléments sur une page web, Bootstrap a introduit le concept de grid
- Cette grid doit être placée dans un **`container`** c'est à dire une "boîte" de largeur 100% ou prédéterminée selon le breakpoint (https://getbootstrap.com/docs/5.1/layout/containers/)
- Le principe est simple :
  - Chaque ligne est identifiée par une classe row et contient 12 "unités" de largeur
  - Chaque ligne contient un nombre de colonnes
  - Les colonnes sont identifiées soit par une classe
    - **`col`** : largeur automatiquement répartie
    - **`col-X`** : largeur X comprise entre 1 et 12
    - **`col-auto`** : largeur adaptée au contenu
  - Si le total des largeurs de colonnes dépasse 12, alors la grid ira automatiquement à la ligne
  - Ces largeurs peuvent être définies par breakpoint, par exemple : **`col-6 col-md-3`**
- Ce système est très pratique pour répartir le contenu horizontalement, et notamment pour faire un système 3 colonnes qui était difficile à implémenter auparavant
- Documentation complète : https://getbootstrap.com/docs/5.1/layout/grid/
- Ce système existe dans de nombreux frameworks (Simple Grid, Bulma, Muuri, 960 grid system, ...)

Bootstrap grid



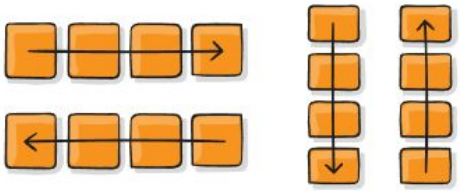
Flexbox

- En 2009 apparaît dans le CSS 3 le concept de Flexbox (Flexible Box Layout Module)
- Le principe est de permettre à un conteneur de disposer, aligner et distribuer ses enfants dans l'espace qui lui est alloué
- Pour cela, 2 nouvelles valeurs de la propriété display sont disponibles : **`flex`** et **`inline-flex`**
- Un conteneur flex fonctionne avec une direction : **`row`** ou **`column`**, qui détermine le placement relatif des enfants. De cette direction découlent 2 axes (main-axis et cross-axis)
- Il est ensuite possible de disposer les enfants sur le main-axis avec **`justify-content`** et sur le cross-axis avec **`align-items`**
- Chaque enfant peut outrepasser les consignes du parent via les propriétés
  - **`justify-self`** (qui n'agit que sur l'espace alloué par le parent)
  - **`align-self`**
  - **`order`** (qui permet de se réordonner sans tenir compte de l'ordre dans le code HTML)
  - **`flex-grow`** (s'entendre sur l'ensemble de l'espace disponible dans le parent)
- La propriété **`flex-wrap`** permet enfin d'aller automatiquement à la ligne (plutôt que de dépasser) si le conteneur n'a pas assez d'espace pour afficher tous les enfants (dans le cas où les enfants ont des tailles fixes)

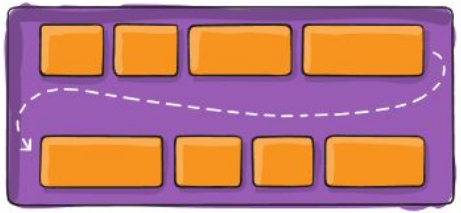
Flexbox

https://css-tricks.com/snippets/css/a-guide-to-flexbox/

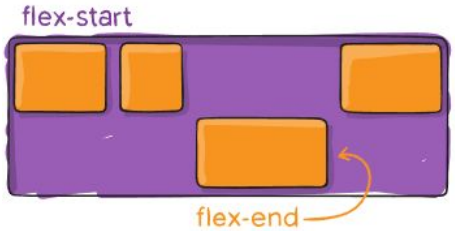
flex-direction



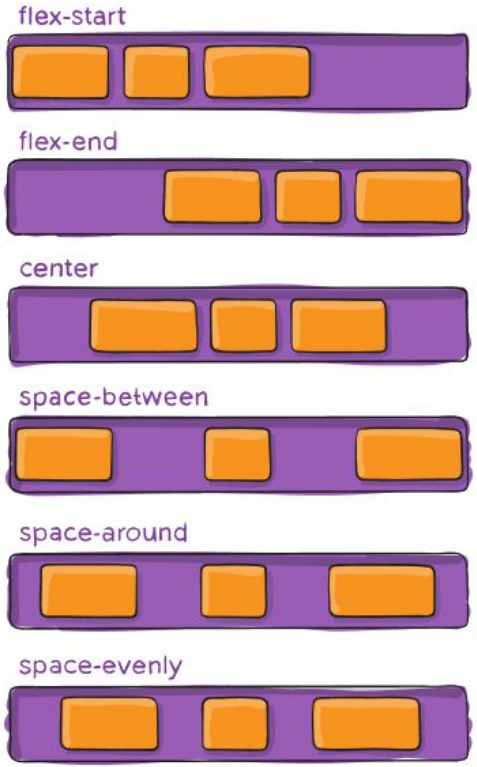
flex-wrap



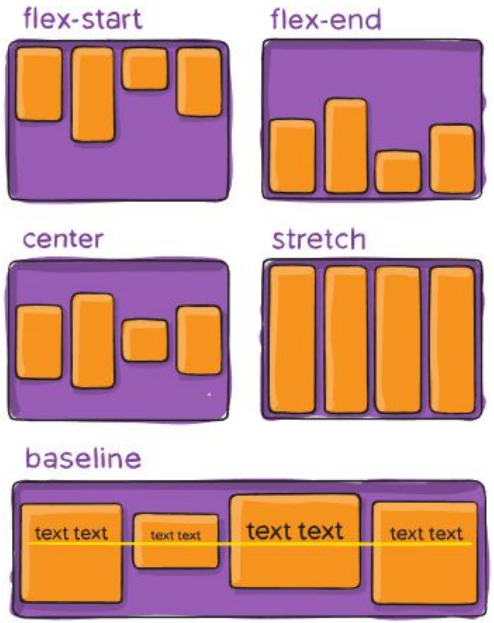
align-self



justify-content



align-items



CSS Grids

- Depuis 2017, les navigateurs modernes implémentent une fonctionnalité de grid, similaire à Bootstrap, mais native
- Pour cela, 2 nouvelles valeurs de la propriété display sont disponibles : **`grid`** et **`inline-grid`**
- Un conteneur grid fonctionne avec un dimensionnement absolu ou relatif de ses lignes et colonnes via 2 propriétés CSS : **`grid-template-columns`** et **`grid-template-rows`**.
- Ces propriétés partagent la même syntaxe, c'est à dire qu'on donne autant de dimensions que de lignes/colonnes. Ces dimensions peuvent être de plusieurs types (combinables) : pixels, pourcentages, fractions (via la nouvelle unité **`fr`**), auto
- Il est également possible de nommer les lignes/colonnes pour référencer individuellement leurs dimensions (par exemple pour le RWD)
- Les éléments sont ensuite disposés dans les cellules, et peuvent potentiellement s'étendre sur plusieurs lignes et/ou colonnes via les propriétés **`grid-row-start`**, **`grid-row-end`**, **`grid-column-start`** et **`grid-column-end`**
- La syntaxe est très puissante, mais également très lourde. On préfèrera pour cela laisser faire les frameworks CSS qui implémentent cette fonctionnalité (Bootstrap, TailwindCSS, ...)
- https://css-tricks.com/snippets/css/complete-guide-grid/

V. Frameworks

Bootstrap

- Créé en 2011 par les équipes de Twitter (open-source depuis)
- Responsive depuis la version 2, version 5 disponible depuis septembre 2021
- Ensemble de feuilles de styles permettant de prédéfinir de nombreux composants
-

Avantages	Inconvénients
Beaucoup de composants prédéfinis	Composants inutiles
	Difficile de créer de nouveaux composants
Style harmonisé	Style trop répandu depuis, et pas très sexy
Icônes intégrées Grand choix de thèmes	Lourd (160Ko de CSS + 77Ko de JS optionnel) *Grosse amélioration depuis la v5 (suppression de la dépendance à jQuery)*
Grosse communauté	

https://getbootstrap.com/

Materialize

- Créé en 2014
- Basé sur le Material Design de Google (sans affiliation initiale avec la société)
- Ensemble de feuilles de styles permettant de prédéfinir de nombreux composants

Avantages	Inconvénients
Beaucoups de composants prédéfinis	Composants inutiles
	Difficile de créer de nouveaux composants
Style harmonisé, sexy Cohérent avec l'écosystème Google Ressenti d'une application	Très dogmatique, ne plaira pas à tout le monde
Léger (22Ko de CSS + 42Ko de JS)	Pas d'utilitaire flexbox

https://materializecss.com/

TailwindCSS

- Créé en 2017
- Ensemble de classes CSS permettant de définir soi-même les composants

Avantages	Inconvénients
Aucun composant prédéfini	Aucun composant prédéfini
Style entièrement personnalisable	Aucune orientation graphique
Beaucoup de classes CSS disponibles	Syntaxe lourde
Plus d'options que la plupart des autres frameworks (flexbox, CSS Grids, gradients, …)	
Très léger (10Ko de CSS + 35Ko de JS optionnel) *Le JS (AlpineJS) est utilisable indépendamment et propose une excellent alternative à jQuery*	

https://tailwindcss.com/

Bulma

- Créé en 2016
- Ensemble de feuilles de styles permettant de prédéfinir de nombreux composants

Avantages	Inconvénients
Beaucoup de composants prédéfinis	Difficile de créer de nouveaux composants
Style harmonisé, minimaliste	
Modulaire	Lourd (102Ko de CSS), pas de JS
Syntaxe très légère	Outil de niche comparé à Bootstrap

https://bulma.io/