Formation CSS

Objectif : réalisation d'un site statique en mobile first

Intro

Définition

Le CSS a été créé en 1996, soit 5 ans après le HTML.
Les sigles « CSS » sont l'abréviation de « **Cascading** StyleSheets » ou « feuilles de styles en cascade » en français.

Le terme **cascade** est très important pour comprendre
la manière dont les CSS se comportent.


CSS ruleset
selector
p {
color: blue;
}
declaration block
p {
color: blue;
}
declaration
p {
color: blue;
}
property
p {
color: blue;
}
value
p {
color: blue;
}
CSS inline style
opening tag
Hello World!
attribute
Hello World!
declaration
Hello World!
property
Hello World!
value
Hello World!

Style inline

On utilise rarement le style inline, c'est une mauvaise pratique.
Cependant on le trouve parfois dans le code HTML.

On l'utilise dans les maquettes HTML d'e-mailing par exemple.

```
<p style='color: red; font-size: 20px;'>j'apprends les CSS !</p>
```

On peut ajouter plusieurs déclarations dans une balise en terminant chaque déclaration avec un **point virgule**.

Le style inline s'applique uniquement à la balise concernée.

La feuille de style interne

```
<head>
  <style>
    p {
      color: red;
      font-size: 20px;
    }
  </style>
</head>

Le style va s'appliquer à tous les éléments <p>
```

La feuille de style externe

La bonne pratique est de créer une feuille de style externe et de la lier dans la page HTML.

Ainsi on sépare le code HTML et le CSS. La maintenance est bien plus facile.

On place ce lien (link) dans la balise **head**

<head>

<title>Formation HTML & CSS</title>

<link href='style.css' rel='stylesheet'>

</head>

href : est le chemin vers le fichier dont l'extension est **.css**

rel : cet attribut décrit la relation entre le fichier HTML et le fichier CSS. Étant donné que vous établissez un lien vers une feuille de style, la valeur doit être fixée à "stylesheet".

Les couleurs

HTML utilise des couleurs hexadecimal précédé d'un #

#FFFFFF => Blanc

#000000 => Noir

#FF0000 => Rouge

#00FF00 => Vert

#0000FF => Bleu

L'opacité se gère avec la propriété RGBa Red | Green | Bleu | alpha

Ex : background-color: rgba(255, 0, 0, 0.5) Red | Green | Bleu | alpha

Attention aux contrastes pour l'accessibilité

Les textes doivent être bien lisibles

https://www.colorhexa.com/color-names

https://color.adobe.com/fr/create/color-wheel

Le sélecteur universel

On l'utilise pour cibler tous les éléments.

Cela permet de mettre à zéro (reset) le style par défaut du navigateur.

```
* {
  box-sizing: border-box;
}
```

Les classes

On peut cibler directement les éléments HTML (h1, p, input, span…)
Mais aussi ajouter l'attribut **class** aux éléments pour les cibler.

```
<h1 class='brand'>Mon entreprise est la meilleure</h1>
```

Les classes peuvent impacter plusieurs éléments contrairement à l'id.

Par exemple, imaginez une page comportant deux titres. L'un doit être gras et bleu, et l'autre gras et vert.

Au lieu d'écrire des règles CSS distinctes pour chaque titre qui répètent le code de l'autre, il est préférable d'écrire une règle CSS .bold, une règle CSS .green et une règle CSS .blue.

Vous pouvez ensuite attribuer à un titre les classes vertes en gras et à l'autre les classes bleues en gras.

```
<div class='heading'>
        <h2 class='blue bold'>Titre bleu et gras</h2>
        <h2 class='green bold'>Titre vert et gras</h2>
</div>
```

```
.brand {
  color: teal;
}
```

Le sélecteur d'une **classe** est précédé d'un point.

Il est possible d'ajouter plusieurs classes à un attribut.

```
<h1 class='brand title'>Mon entreprise est la meilleure</h1>
```

Chaînage de sélecteurs	

`h1.brand {`
`  color: teal;`
`}`
	`.heading .blue {`
	`  font-size: 1.em;`
	`}`

L’id

On peut vouloir cibler un seul élément pour lui appliquer un style unique.

Ils doivent être utilisés avec parcimonie et uniquement sur les éléments qui doivent toujours apparaître de la même façon.

La valeur d’un id est unique,
il ne peut pas y avoir deux id de même valeur dans une page HTML.

<h2 id=‘second-title’>Un seul et unique ID</h2>

```
#second-title {
  font-family: cursive;
  font-size: 24px;
}
```

Le sélecteur d’un **id** est précédé d’un **#**

Chaînage d’éléments

Lorsque vous écrivez des règles CSS, il est possible qu'un élément HTML ait deux sélecteurs CSS ou plus en même temps.

```
<h2 class=‘js'>JavaScript</h2>
<h2 class=‘python’>Python</h2>

h2.js {
   text-transform: uppercase;
}
```

Les combinateurs

Combinateur descendant
sélectionne tous les éléments(li) qui sont des descendants (children) de l'élément spécifié.

```
.main-list li {
  color: red;
}
```

Combinateur enfant (child)
sélectionne tous les éléments(li) qui sont les enfants d'un élément spécifié.

```
.main-list > li {
  color: red;
}
```

https://codepen.io/web-god/pen/KKvpbMR

Combinateur adjacent
sélectionne un élément qui se trouve directement après un autre élément spécifique.

Les éléments frères et sœurs doivent avoir le même élément parent.
Adjacent signifie "immédiatement après".

```
h2 + li {
  color: red;
}
```

https://codepen.io/web-god/pen/RwZPEKG?editors=1100

Combinateur frère (sibling)
sélectionne tous les éléments qui sont les prochains frères et sœurs de l'élément spécifié.

```
.main-list ~ li {
  color: red;
}
```

https://codepen.io/web-god/pen/MWvwZpP

Les attributs

On peut aussi sélectionner les attributs des éléments HTML

< a href='https://www.alsacreations.com' target='_blank' title='Le blog'>Le blog alsacréations

Les éléments **<a>** avec un attribut title

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

Les éléments **<a>** avec un href qui correspond :

```
/* à "https://example.org" */
a[href="https://example.org"] {
  color: green;
}
```

Les éléments **<a>** avec href commençant par "https"

```
a[href^='https']
```

Les éléments **<a>** dont href contient « example"

```
a[href*="example"] {
  font-size: 2em;
}
```

Les éléments **<a>** dont href finit par ".org"

```
a[href$=".org"] {
  font-style: italic;
}
```

Les éléments **<a>** dont l'attribut class contient le mot logo

```
/* comportement identique à a.logo */
a[class~="logo"] {
  padding: 2px;
}
```

Les pseudo-classes

La pseudo-classe **:hover**, par exemple, permet 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).

`:hover :focus, :visited, :disabled, and :active sont également des pseudo-classes`

`On retrouve fréquemment les déclarations suivantes pour formater les liens des pages web.`

```
/* unvisited link */
a:link {
  color: green;
}

/* visited link */
a:visited {
  color: green;
}

/* mouse over link */
a:hover {
  color: red;
}

/* selected link */
a:active {
  color: yellow;
}
```

Il s'agît de pseudo-classes car on pourrait les substituer par une classe.

Il est possible d'ajouter **plusieurs déclarations sur une même ligne** en les séparant par une virgule.

```
a.ex1:hover, a.ex1:active, a.ex1:focus {
  color: red;
}
```

La propriété position

La propriété position vous aide à manipuler l'emplacement d'un élément.
Tous les éléments (balises) HTML peuvent être positionnés, décorés, dimensionnés, ... grâce aux styles CSS

La notion de flux

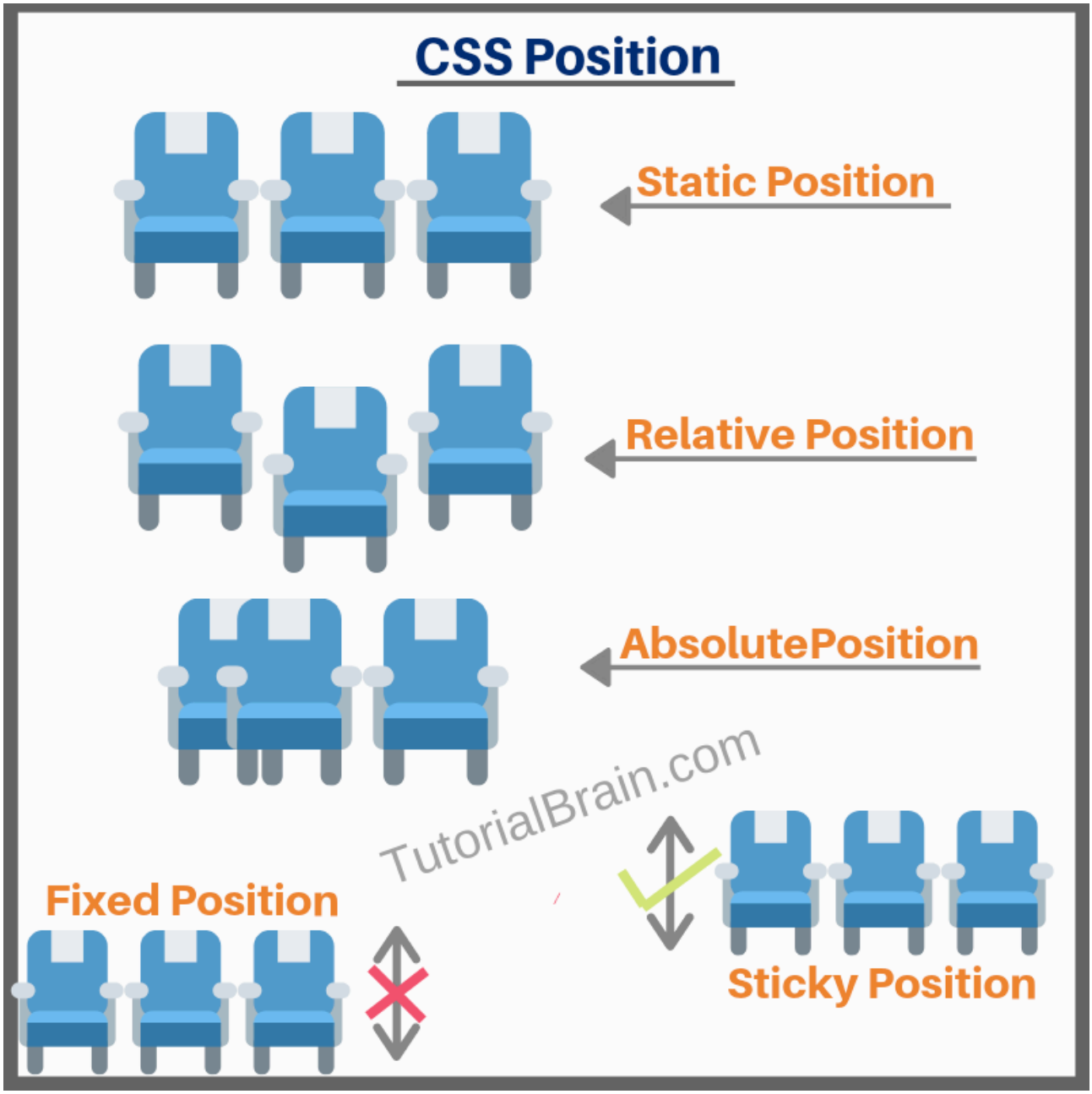
L'ordre dans lequel apparaissent les balises dans le code HTML sera l'ordre dans lequel ils seront affichés et apparaitront dans le document, c'est le Flux.
Cela signifie que, par défaut, chaque élément est dépendant des éléments frères qui l'entourent.

Chaque document HTML est toujours composé de conteneurs. Ceux-ci peuvent être *ancêtres*, *parents*, *enfants* ou *frères*.
Ces différents éléments composent une hiérarchie d'imbrications.

NOTE : les blocs positionnés en "absolute" ou "fixed" sortent du flux naturel et échappent quelque peu à cette règle.

Ils ne sont alors plus dépendant des éléments frères.

Pour se placer, un tel bloc se réfère non pas à son Parent direct, mais au premier Ancêtre *positionné* qu'il rencontre.



Les valeurs de la propriété position

static : reste dans le flux du document
La position des éléments HTML est statique **par défaut**.
L'élément est dans le flux normal de la page.
Les propriétés left / right / top / bottom / z-index n'auront aucun effet.

relative : reste dans le flux du document
Le positionnement relatif positionne l'élément par rapport à l'endroit où il se serait trouvé normalement.
left / right / top / bottom / z-index fonctionneront.

absolute : l'élément est retiré du flux du document
Les autres éléments se comporteront comme s'il n'était pas là.
left / right / top / bottom / z-index fonctionneront.

L'élément sera positionné par rapport à son 1er parent positionné en relatif.

Stuff & Nonsense

a
1 x element selector

Sith power: 0,0,1

p a
2 x element selectors

Sith power: 0,0,2

.foo
1 x class selector *

Sith power: 0,1,0

a.foo
1 x element selector
1 x class selector

Sith power: 0,1,1

p a.foo
2 x element selectors
1 x class selector

Sith power: 0,1,2

.foo .bar
2 x class selectors

Sith power: 0,2,0

p.foo a.bar
2 x element selectors
2 x class selectors

Sith power: 0,2,2

#foo
1 x id selector

Sith power: 1,0,0

a#foo
1 x element selector
1 x id selector

Sith power: 1,0,1

.foo a#bar
1 x element selector
1 x class selector
1 x id selector

Sith power: 1,1,1

.foo .foo #foo
2 x class selectors
1 x id selector

Sith power: 1,2,0

style
1 x style attribute

Sith power: 1,0,0,0

*** Same specificity**
class selector =
attribute attribute =
pseudo-classes

!important

f**ixed :** l'élément est retiré du flux du document
Les éléments positionnés de manière fixe sont toujours relatifs au document, et non à un parent particulier (absolu), et ne sont pas affectés par le scroll.
left / right / top / bottom / z-index fonctionneront.

sticky : Ce positionnement est un hybride entre le positionnement relatif et le positionnement fixe.
L'élément est traité comme un positionnement relatif jusqu'à ce qu'il franchisse un seuil spécifié en scrollant, ensuite il est traité comme un positionnement fixe.
left / right / top / bottom / z-index fonctionneront.

inherit : hérite de la valeur de positionnement de son parent.

z-index : La propriété z-Index est utilisée pour spécifier l'ordre d'empilement des éléments qui se chevauchent.
Le niveau d'empilement fait référence à la position de l'élément sur l'axe Z.

Il fonctionne comme un empilement de calque dans Photoshop.

Spécificité

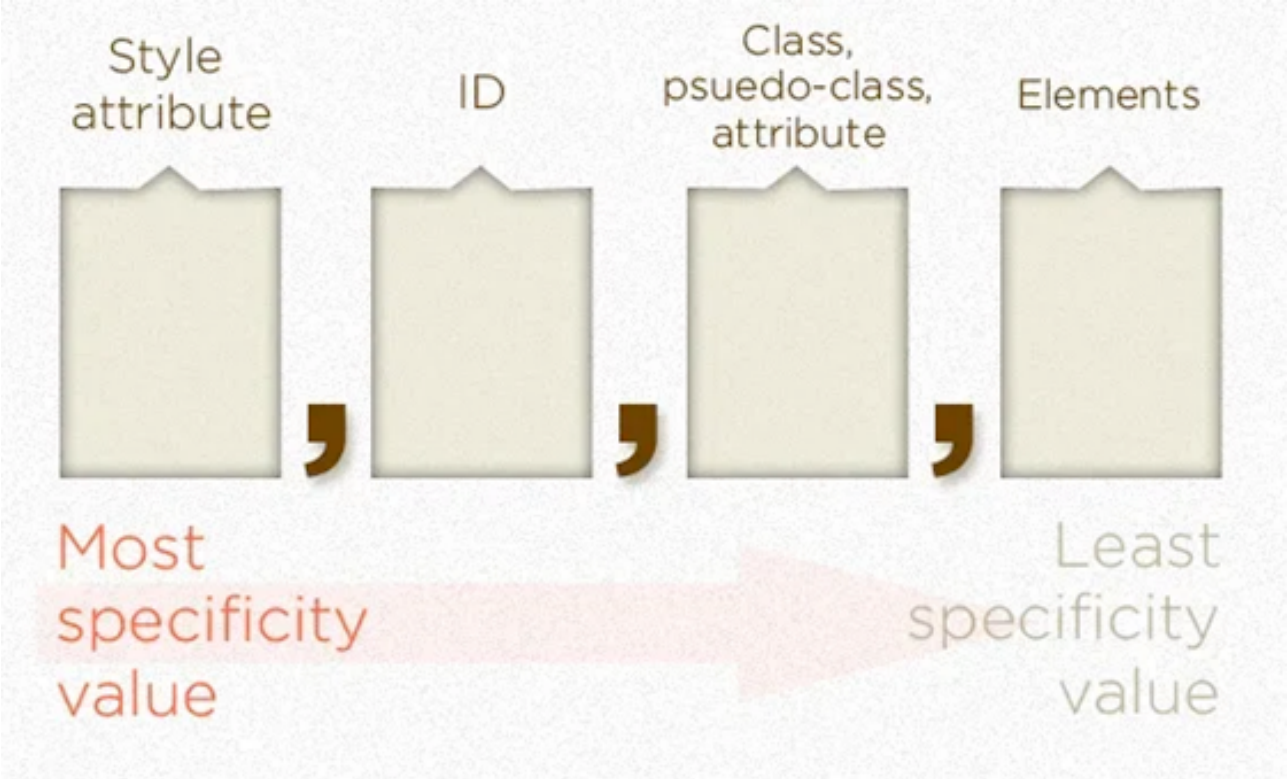
La spécificité est l'ordre dans lequel le navigateur décide des styles CSS à afficher.

Une bonne pratique en CSS consiste à donner un style aux éléments en utilisant le degré de spécificité **le plus faible** possible, de sorte que si un élément a besoin d'un nouveau style, il est facile de le surcharger (overrider). Voir codepen
http://www.standardista.com/wp-content/uploads/2012/01/specifishity1.pdf
https://blog.organicweb.fr/comprendre-le-poids-des-regles-css/
https://www.smashingmagazine.com/2007/07/css-specificity-things-you-should-know/

	!important	Inline	Id	Class	Tag
A	0	0	1	0	3
B	0	0	1	1	3
C	0	0	0	2	4
D	0	0	0	1	1
E	0	0	0	1	1
F	0	0	0	1	4
G	1	1	0	0	0
H	1	0	0	1	1

```
A => ul#menu li > a {
       0,1,0,3
     }

B => ul#menu li.active > a {
       0,1,1,3
     }

C => html body.js h1 + p.lead {
       0,0,2,4
     }

D => p.sample > * + * {
       0,0,1,1
     }

E => a[href^="#"] {
       0,0,1,1
     }

F => ul li ~ li div:not(.foo) {
       0,0,1,4
     }
```

```
G => <span style="color: red;></span>

H => p.info {margin: 1em!important}
```

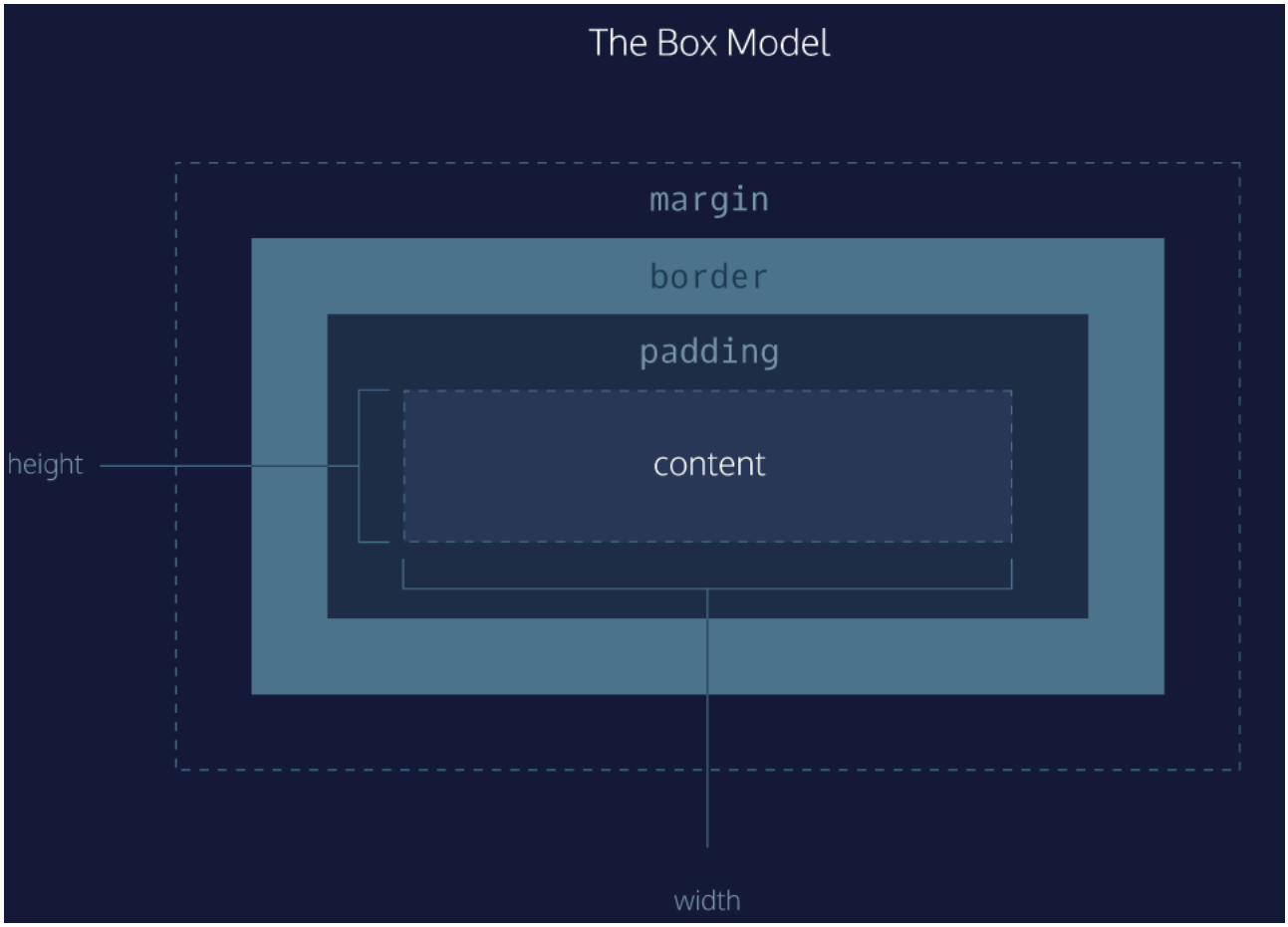


!important remplacera n'importe quel style, mais il ne devrait presque jamais être utilisé, car il est extrêmement difficile de le remplacer.

L'utilisation de !important se justifie notamment lorsque l'on travaille avec plusieurs feuilles de style.
Par exemple, si nous utilisons Bootstrap et que nous voulons remplacer les styles d'un élément HTML spécifique, nous pouvons utiliser la propriété !important.

Modèle de boîte

Un élément (content) HTML typique comporte une largeur (width) et une hauteur (height) avec des :



- padding (marges internes) crée un espace autour du contenu. Cet espace a le même background que la boîte Content
- border (bordures) une bordure entoure le contenu et son padding. Elle peut avoir elle-même son propre background, son épaisseur, sa texture, ses motifs.
- margin (marges externes) la marge crée un espace autour des éléments précédents, cet espace est toujours transparent et n'est jamais affecté par les background de ces éléments.

Quand nous définissons les dimensions d'un élément,
nous ne définissons en fait que les largeur et hauteur du contenu.
Pour calculer la largeur et la hauteur totales de cet élément,
nous devons aussi inclure le padding, la bordure et la marge.

Exemple dimension d'une boîte

```
.box{
    width: 100px;
    padding: 10px;
    border: 5px red solid;
    margin: 10px;
}
```

La largeur (width) du contenu est de 100px. La hauteur (height) est défini par le contenu. L'élément dans son ensemble a une largeur de 150px car nous devons intégrer les dimensions des autres composantes (padding, border, margin).

La largeur totale est égale à :
margin-left: 10px + border-left: 5px + padding-left: 10px + width: 100px + padding-right: 10px + border-right: 5px + margin-right: 10px

10 + 5 + 10 + 100 + 10 + 5 + 10 = 150px

Les propriétés du modèle de boîte

```
.box {box-sizing: content-box|border-box|inherit;}
```

Content-box : valeur par défaut, elle spécifie que les largeur et hauteur définies s'appliquent aux largeur et hauteur du contenu seul, et que le padding, la bordure et les marges sont en dehors de ces dimensions.

Border-box : les largeur et hauteur spécifiées sont celles de l'élément dans son ensemble, en conséquence les dimensions du padding ou des bordures viennent diminuer les dimensions du contenu.

Inherit : comme pour toute valeur héritée, la valeur de box-sizing peut dépendre de celle de son élément parent.

Les mises en pages ne sont pas toujours simples et s'avèrent souvent des casse-tête. Box-sizing vous fera gagner du temps et vous aidera à coder plus efficacement.

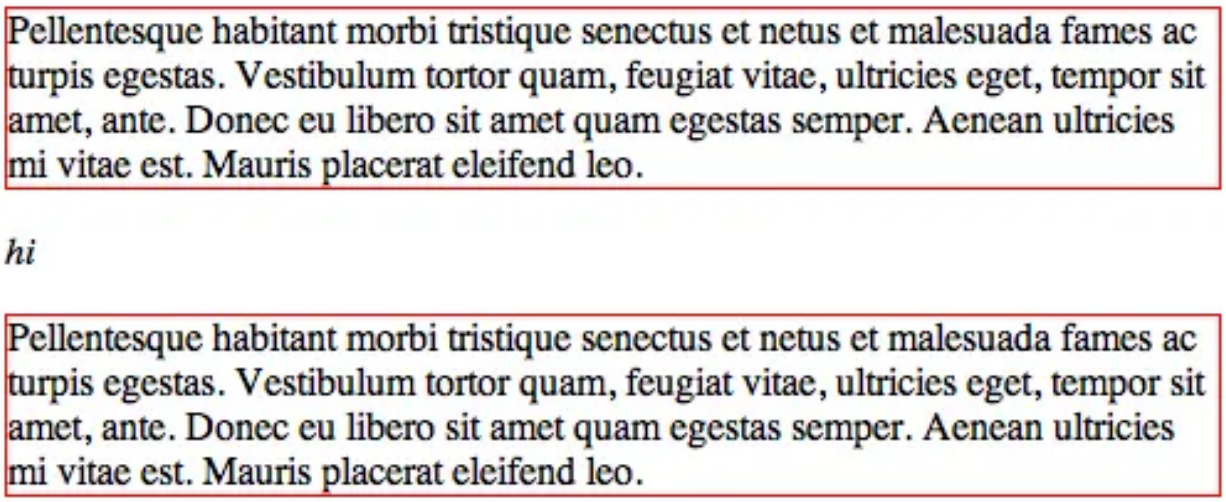
La propriété display

La propriété display de CSS détermine **le comportement** de cette boîte rectangulaire.

display: block;

La valeur par défaut des éléments comme <section> , mais aussi des blocs de texte comme <p> <h1>
Les éléments bloc ne sont pas alignés mais passent à la ligne.

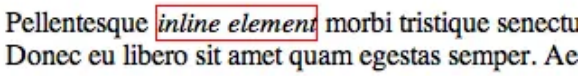
Par défaut (sans définir de largeur), ils occupent tout l'espace horizontal possible.



display: inline;

La valeur par défaut des éléments comme , ou <b>
L'insertion du texte dans ces éléments à l'intérieur d'une chaîne de texte ne casse pas le flux du texte.

Les éléments inline acceptent border, margin et padding en restant dans le flux.



Un élément en ligne n'accepte pas width et height.
Il les ignorera tout simplement.

display: inline-block;

Un élément display en **inline-block** est très similaire à **inline** en ce sens qu'il s'aligne sur le flux naturel du texte (sur la "ligne de base").
La différence est que vous pouvez définir une **largeur** et une **hauteur** qui seront respectées.

display: grid;

Dispose les éléments sous forme de grille.

display: flex;

Est utilisée pour les nouvelles méthodes de mise en page telles que Flexbox.

display: none;

Supprime entièrement l'élément de la page.

Voir codepen Display property
https://www.w3schools.com/cssref/playit.asp?filename=playcss_display&preval=table

Les animations *donnent vie à vos pages web !*

Example : https://codepen.io/rachelcope/pen/YPzZrg

Tutorial complet sur les animations
https://openclassrooms.com/fr/courses/5919246-creez-des-animations-css-modernes/6340912-decouvrez-les-animations-web

Le cerveau humain est conçu pour remarquer les **mouvements** :
En animant des éléments d'une page web, nous pouvons capter l'attention de nos visiteurs et ainsi améliorer son expérience (UX).

Il existe deux moyens de créer des animations en CSS :

1. les **transitions** => animation simple

2. les **keyframes** (images clés) => animation complexe

Les transitions

Pour créer une transition nous avons besoin de :

- une **propriété CSS** à modifier ; background-color
- une **valeur initiale** pour votre propriété CSS ; tomato
- une **valeur finale** pour cette même propriété ; blue
- une **durée** ; 300ms
- un **événement** pour déclencher votre transition. hover

Les déclencheurs d'événement :

`:hover`, qui est déclenché au **survol** de la souris ;

`:active`, activé au **click** de l'utilisateur (le plus souvent pour les liens et boutons) ;

`:focus`, qui se déclenche lorsque son élément reçoit le **focus** (soit il est sélectionné à l'aide du clavier, soit il est activé avec la souris) ;

`:valid`, dont la validation du contenu s'effectue correctement par rapport au type de donnée attendu ;

`:invalid`, qui inversement, correspond à un élément dont la validation du contenu ne s'effectue pas correctement par rapport au type de donnée (email, number, text) attendu ;

`:not()`, qui correspond à la **négation**. Elle prend un sélecteur en argument et permet de cibler les éléments qui ne sont pas représentés par cet argument ;

`:checked`, qui correspond aux **input** de type checkbox, option ou radio qui sont cochés ;

`:enabled`, un élément avec lequel on peut **interagir ;**

:disabled , qui correspond à un élément dont l'interaction a été bloquée.

Les effets

transition-timing-function: linear;

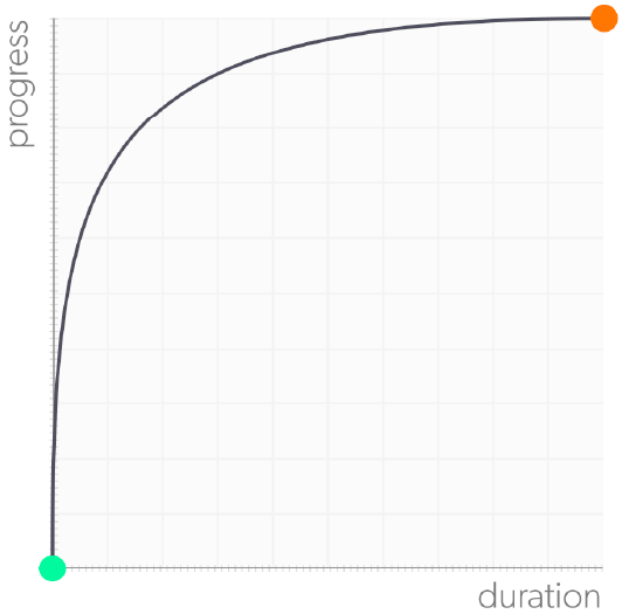
transition-timing-function: ease-in;

transition-timing-function: steps(6, end);

transition-timing-function: cubic-bezier(.29, 1.01, 1, -0.68);
cubic-bezier(x1, y1, x2, y2)

Pour assurer la **fluidité (**60 FPS**)** des animations, il faut se contenter d'animer des propriétés de l'étape composition. Les plus utiles sont `transform` et `opacity`

http://cubic-bezier.com/
https://csstriggers.com/



Les @keyframes

Les @keyframes sont disponibles globalement, donc n'importe quel sélecteur dans notre fichier CSS peut les utiliser.
Nous déclarons les @keyframes au niveau global du fichier CSS.

Les animations avec les @keyframes n'ont pas obligatoirement besoin d'une interaction de l'utilisateur, elles peuvent démarrer au chargement de la page.

```
@keyframes progress-bar {
    from {
        transform: scaleX(0);
    }
    to {
        transform: scaleX(1);
    }
}
```

```
@keyframes progress-bar{
    0% {
        transform: scaleX(0);
    }
    100% {
        transform: scaleX(1);
    }
}
```

Nous appelons l'animation par son nom *progress-bar* dans un sélecteur ainsi :

```
.btn:active {
        animation-name: progress-bar;
}
```

Résumé

- CSS peut sélectionner des éléments HTML par **tag**, **classe**, **ID** et **attribut**.
- Tous les éléments peuvent être sélectionnés à l'aide du sélecteur universel (*****).
- Un élément peut avoir différents états en utilisant le sélecteur de pseudo-classe.
- Plusieurs classes CSS peuvent être appliquées à un élément HTML.
- Les classes peuvent être réutilisées, tandis que les ID ne peuvent être utilisés qu'**une seule fois**.
- Les **ID** sont plus spécifiques que les **classes**, et les **classes** sont plus spécifiques que les **tags**. Cela signifie que les ID remplacent les styles d'une classe et que les classes remplacent les styles d'un sélecteur de tag.
- Plusieurs sélecteurs peuvent être enchaînés pour sélectionner un élément.
  Cela augmente la spécificité mais peut s'avérer nécessaire.
- Les éléments imbriqués peuvent être sélectionnés en séparant les sélecteurs par un espace.
- Plusieurs sélecteurs non liés peuvent recevoir les mêmes styles en séparant les noms des sélecteurs par des virgules.

Vs code sass
https://code.visualstudio.com/Docs/languages/CSS

Grid
https://mozilladevelopers.github.io/playground/css-grid/

Flexbox
https://progressivered.com/fla/?d=3&v=5&h=0&s=0&i=010&a=000
https://the-echoplex.net/flexyboxes/

Sprite
https://www.alsacreations.com/tuto/lire/1068-sprites-css-background-position.html

Flag en CSS
https://pixelastic.github.io/css-flags/

Z-index
https://la-cascade.io/comment-fonctionne-z-index/

http://nicolasgallagher.com/about-normalize-css/

https://necolas.github.io/normalize.css/latest/normalize.css

https://loremflickr.com

Float
Animation