{ Conception d’interfaces }

(Ré) Introduction

Introduction au CSS

CSS a été introduit initialement pour formatter des documents texte : gras, italique, alignement du texte etc.

Ce n’est qu’à partir de la version 3 que les spécifications se concentrent sur la conception d’interfaces.

CSS décrit comment les éléments doivent être restitués à l’écran et sur d’autres supports de présentations.

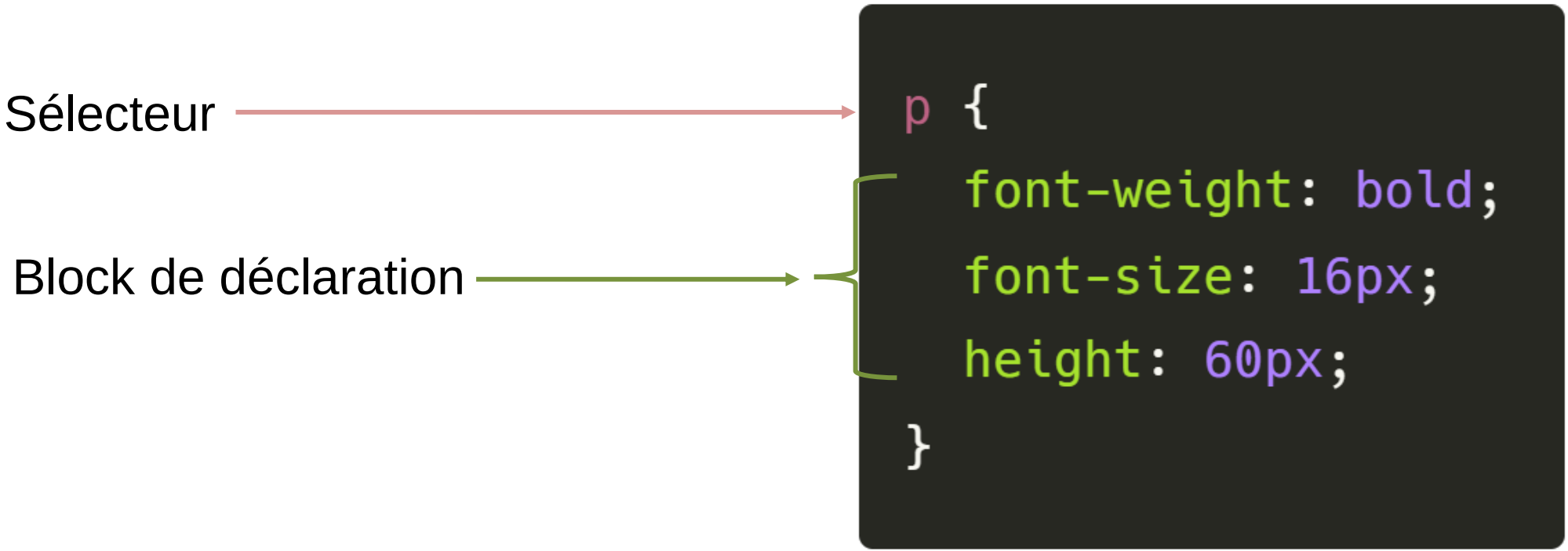
Syntaxe

Techniquement, CSS se traduit par une liste de règles à appliquer au document HTML.

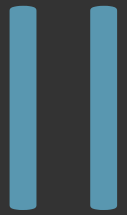
La syntaxe d'une règle CSS se compose d'un ou plusieurs **sélecteurs** suivi d'un **block de déclaration**.

Le **sélecteur** permet de spécifier l'élément sur lequel le style définit dans le **block de déclaration** doit s'appliquer.

Syntaxe



Tester les sélecteurs CSS :
http://www.w3schools.com/cssref/trysel.asp



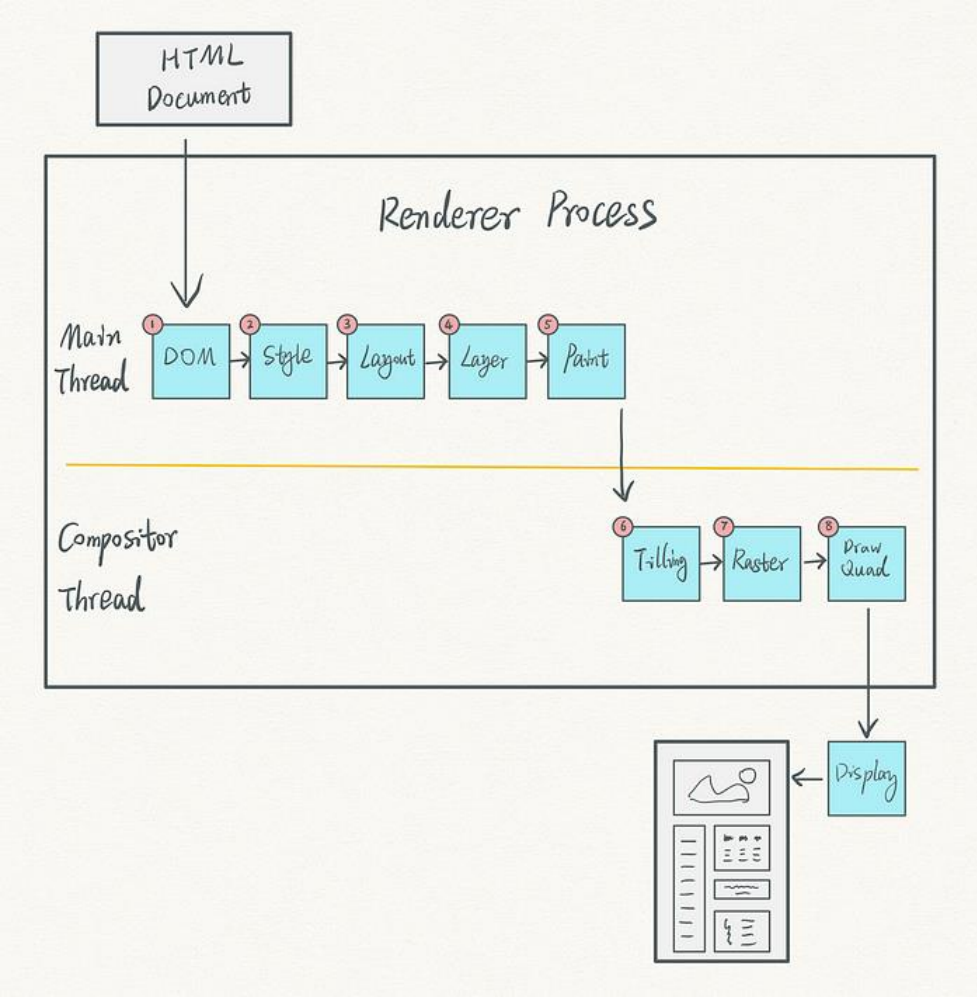
CSS Rendering

Rendering Process

Le navigateur passe par plusieurs phases pour afficher un document HTML (Rendering Process).

Pour se faire, le navigateur va solliciter 2 threads:

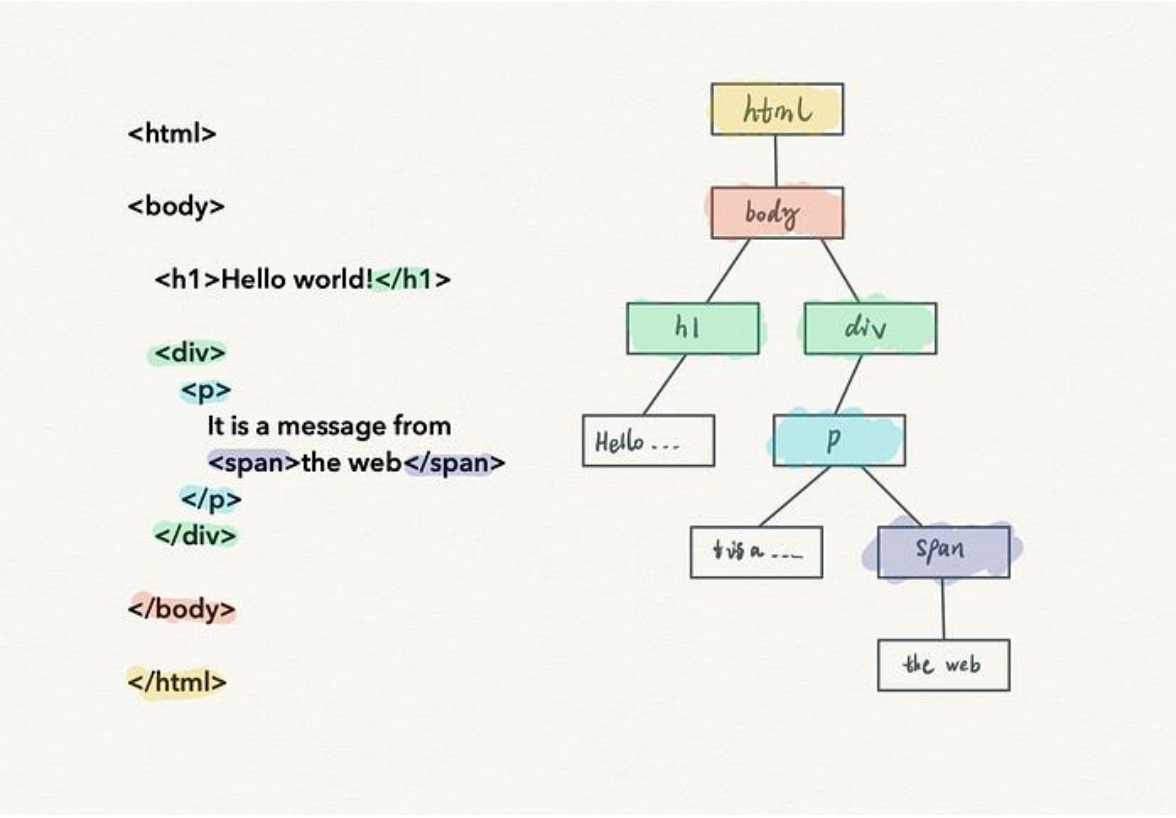
- Le main qui va gérer les 6 premières étapes
- Le composite pour gérer les 3 dernières



DOM

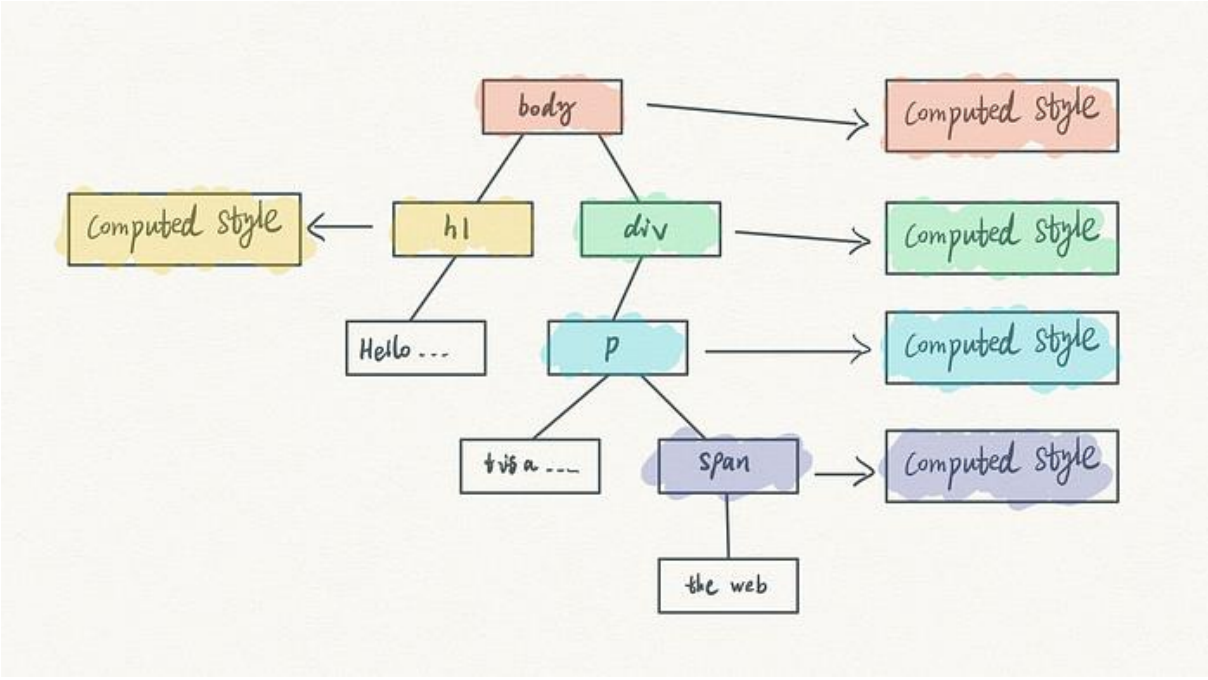
La première étape est la construction du **DOM** :

Le passage du HTML à une représentation en mémoire dés différents éléments du document.



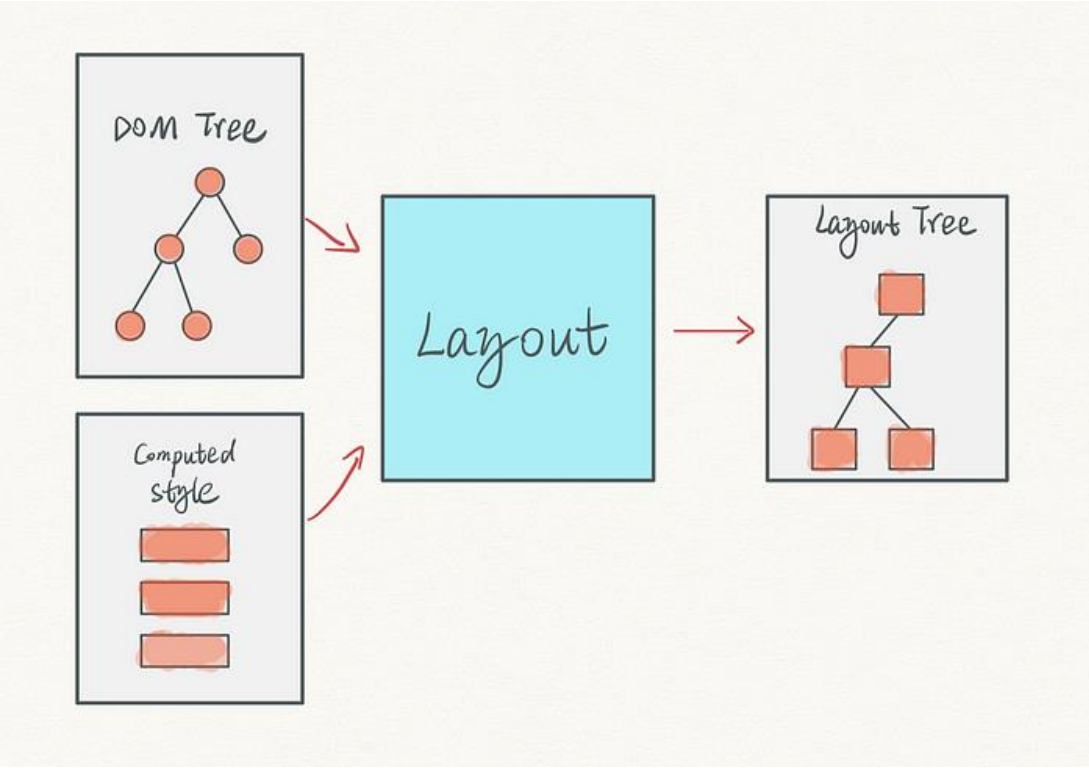
Style

L’étape de **style** sert à calculer les différentes règles présentes dans la CSS et les associer aux éléments du DOM.



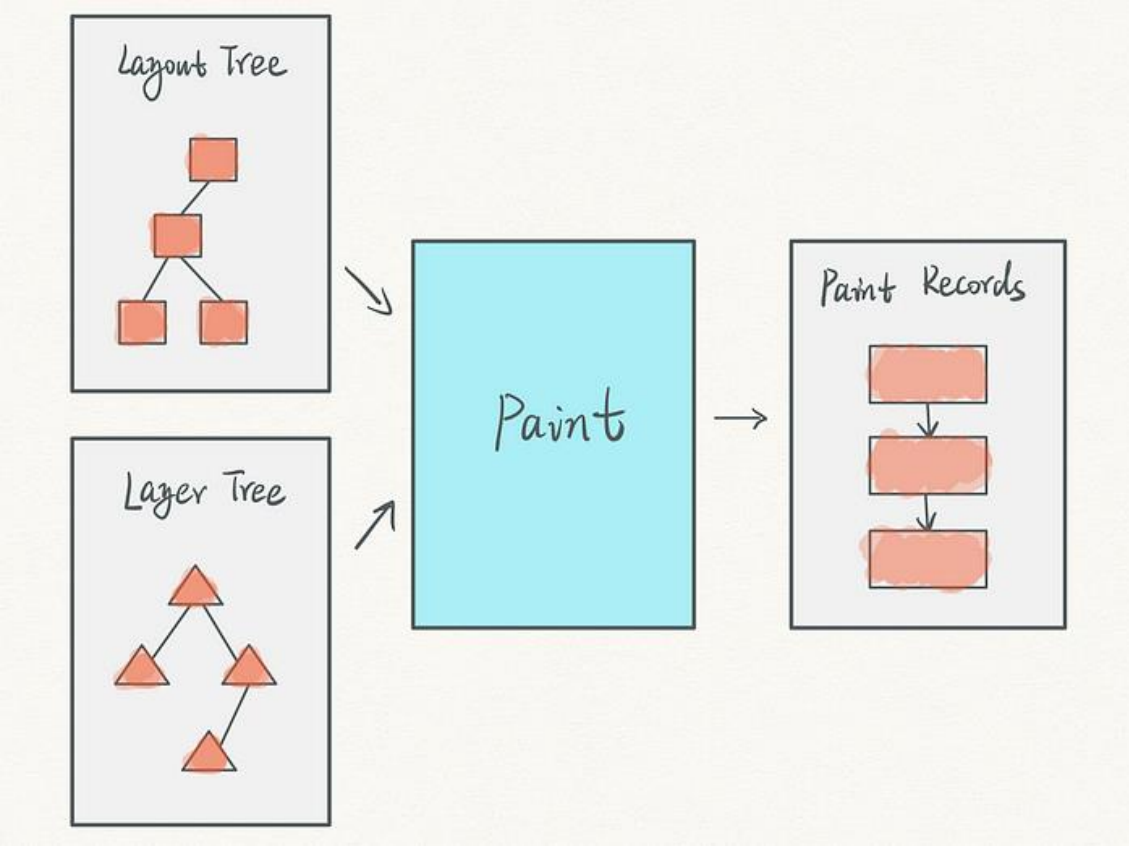
Layout

L’étape de **layout** définie quels éléments devront être affichés, de quelle taille, à quel endroit dans la page et dans quel ordre.



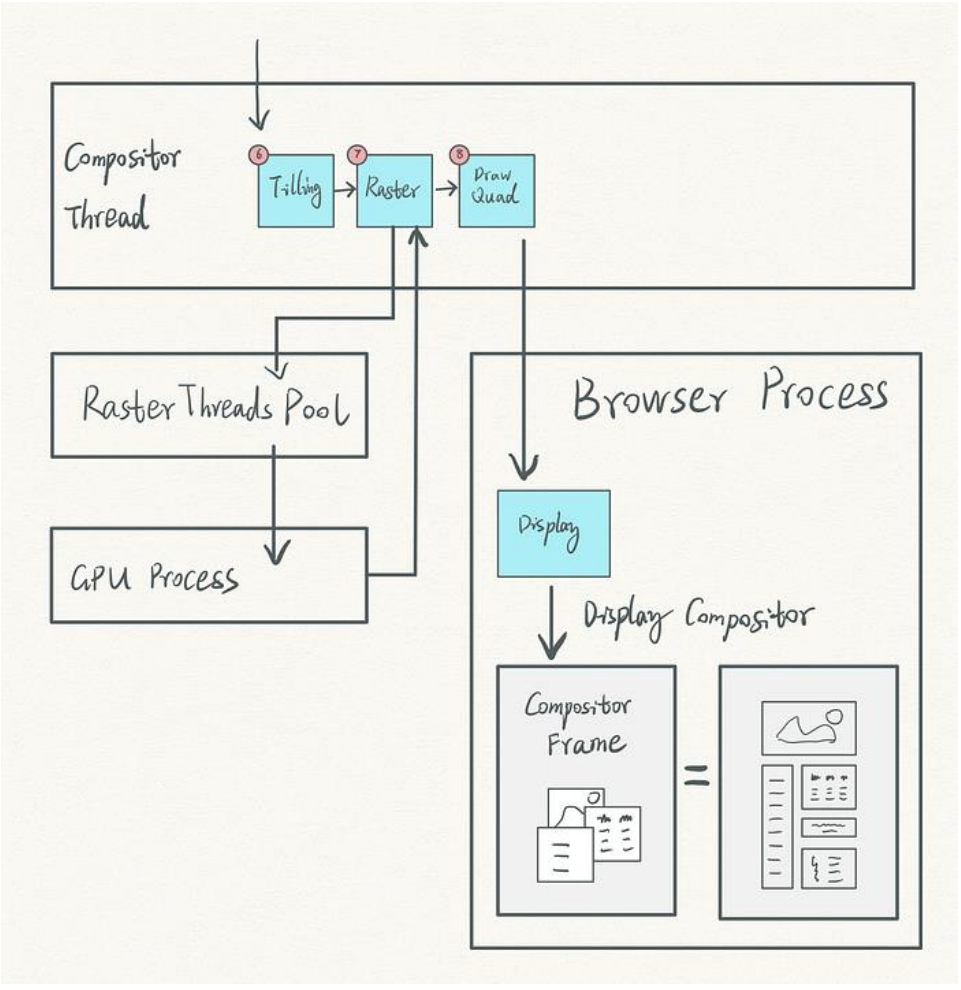
Paint

L'étape de **paint** liste les opérations à effectuer pour dessiner les éléments à l'écran : couleurs, formes et bordures vont être définies.



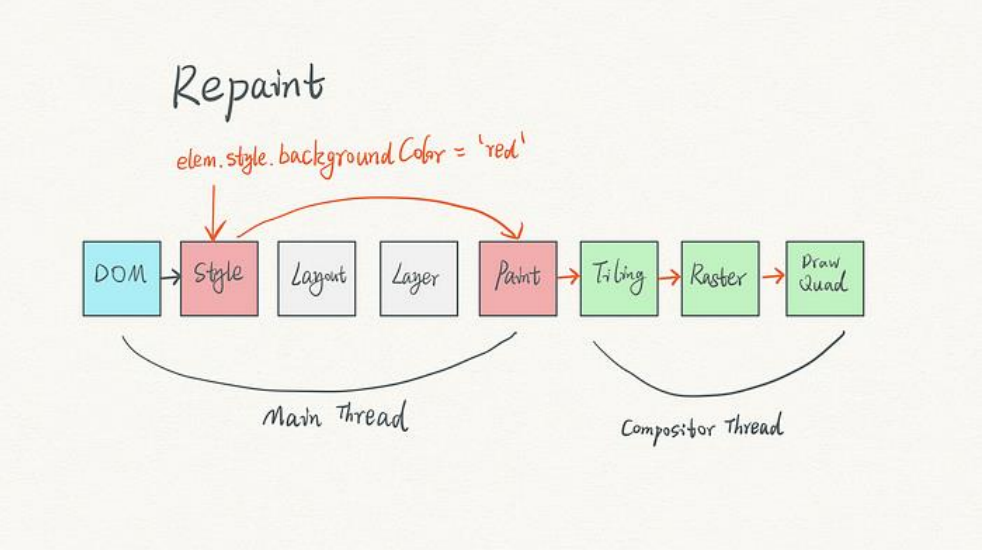
Composition

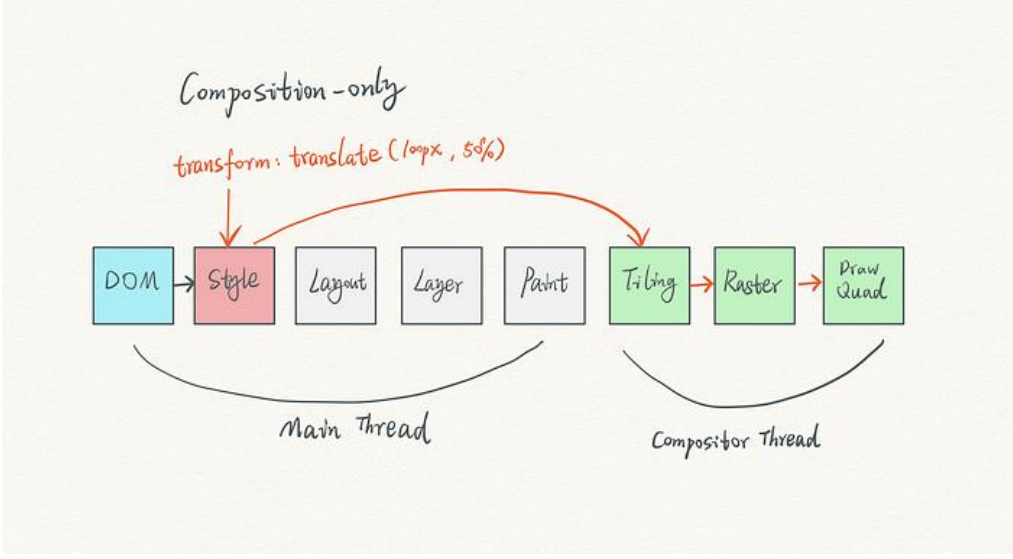
Les 3 dernières étapes dites de **composition** permettent de transformer le résultat des précédentes étapes en instructions permettant au GPU de calculer l'image finale à restituer sur le support.

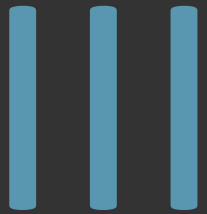


Rendering lors du changement

Lorsqu'une propriété CSS change, seules les étapes impactées sont recalculées.







Transitions

Les transitions

Les interfaces modernes restituent beaucoup d'informations et sont très interactives.

Beaucoup d'éléments sont amenés à apparaitre et à disparaitre à l'écran.

Pour améliorer l'expérience utilisateur et pour accompagné l'évolution des éléments à l'écran, CSS dispose de la propriété **transition.**

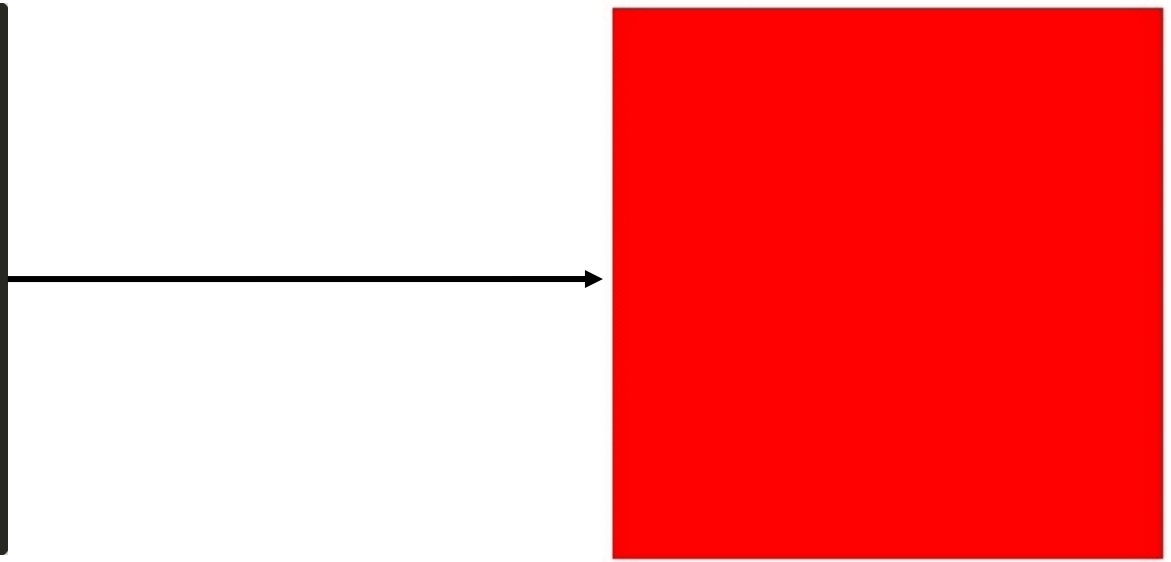
Une **transition** permet d'animer le changement d'état d'un élément : changement de couleur de fond, de taille, de position etc.

Transition en CSS

La transition s'utilise sur n'importe quel élément. Elle a besoin :

La transition attend : le nom de la propriété à animer ou **all**, le temps avant de démarrer la transition, sa durée et une **fonction d'easing**

```
.element {
  background: red;
  transition: background 0.5s
}s;

.element:hover {
  background: blue;
}
```



Limites de la transition

Toute les propriétés ne sont pas « animables » à l'image de la propriété display.

Passer de display: none à block ne donnera aucune transition particulière.

Liste des propriétés « animables » :
http://www.w3schools.com/cssref/css_animatable.asp

L’easing

L’easing est une fonction mathématique qui permet de décrire la vitesse de l’animation dans le temps.

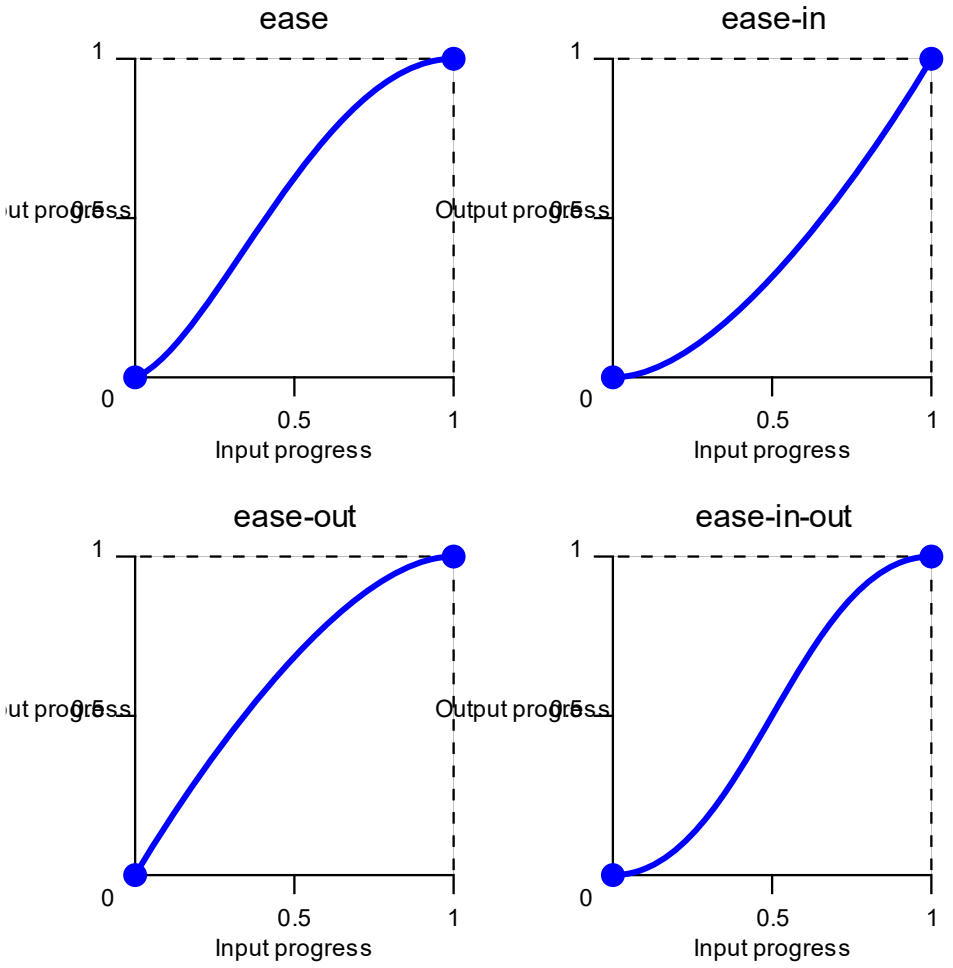
Par défaut, la fonction d’easing utilisée est ease : lent au début, rapide au milieu, lent à la fin.

Différentes fonctions d'easing

Liste des fonctions d'easing prédéfinies :

Vous pouvez également personnaliser votre easing avec la fonction **cubic-bezier(…)**

Démos sur http://easings.net/



IV

Transformations

La transformation

La **transformation** en CSS permet d'appliquer une déformation sur un élément.

Il peut s'agir par exemple d'une translation, d'une rotation ou d'un grossissement/rétrécissement.

La transformation est l'une des rares propriétés à n'impacter que l'étape de **composition**, autrement dit elle ne sollicite majoritairement que le GPU.

La translation

Quelque soit la transformation, la propriété **transform** va utiliser des fonctions de transformation.

Une translation permet de déplacer la représentation visuelle d'un élément en fonction des coordonnées passée en paramètre.

La translation va utiliser la fonction **translate(x, y) :**

```
div {
  transform: translate(30px, 20px);
}
```

La rotation

La rotation s'applique avec la fonction **rotate(angle)** et prend en paramètre l'angle de rotation qui peut être exprimée en radian rad ou en degrées deg :

```
div {
  transform: rotate(45deg);
  transform: rotate(0.78rad);
}
```

Le scale

L'agrandissement, mise à l'échelle ou *scale* permet d'appliquer un effet de « zoom » sur l'élément via la fonction **scale(ratio).**

La fonction prend en paramètre un ratio qui correspond au niveau de zoom ou **1** sera la valeur initiale, **2** un grossissement x2, **3** un de x3 etc.

Toute valeur inférieure à 1 fera un rétrécissement : **0,5** sera **½** de la taille**, 0.25** sera 1/4 etc.

```
div {
  transform: scale(1.5);
}
```

La 3e dimension

Les précédentes transformations s'appliquent toutes en fonction d'un ou plusieurs axes : x et y.

Certaines transformations peuvent également s'appliquer sur 3 axes y, y et z.

En réalité, la rotation s'applique par défaut sur l'axe z.

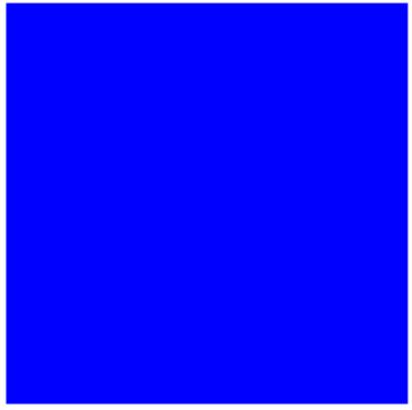
Pour les 3 fonctions présentés, vous avez également une fonction disponible par axe : translatex(), translatey(), translatez(), rotatex() etc.

Rotation 3D

Utiliser une transformation dans un repère en 3 dimensions ne donne pas toujours le résultat attendu.

Example avec une rotation sur l’axe Y :

Original



transform: rotateY(45deg)

La Perspective

La rotation n’est pas perçue par notre cerveau pour 2 choses :

- L’absence de perspective
- L’absence d’éclairage et d’ombrage

Il est possible de rajouter de la perspective en CSS avec la propriété **perspective: distance.**

La distance est la profondeur (px, %, em…) de votre scène et doit s’appliquer sur l’élément parent.

Rotation mise en perspective

Une fois mis en perspective, l'élément subissant une rotation va également être déformé en fonction de la profondeur de la scène et donc de la distance du point de fuite.

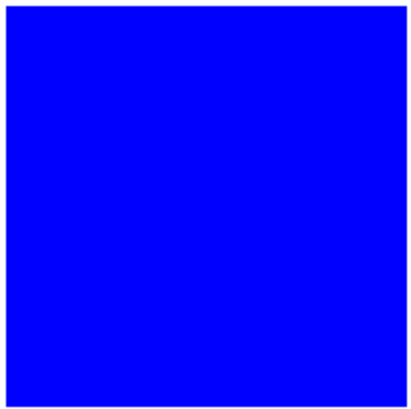
```css
body {
  perspective: 500px;
}

div {
  transform: rotateY(45deg);
}
```

Translation 3D et perspective

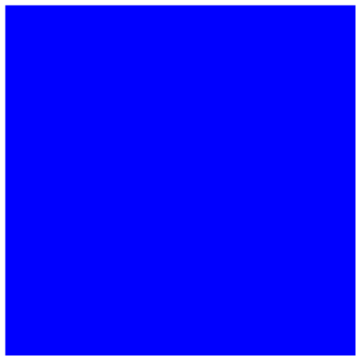
Pour la translation sur l'axe z avec translatez(z), l'élément va se « rapprocher » ou « s'éloigner ».

L'élément va donc se rétrécir ou s'agrandir en fonction de la perspective définie :

Original



translatez(-50px), perspective: 800px



V

Animation

L’animation

Les animations en CSS permettent d’animer une ou plusieurs propriétés d’un élément comme peut le faire une transition.

La différence avec la transition est qu’entre le début et la fin d’une animation, l’élément va pouvoir passer pas une succession d’étapes.

L’élément animé va pouvoir changer plusieurs fois d’état ou cours de l’animation où un changement d’état correspond à une étape.

Les Keyframes

L’animation repose donc une séquence d’étapes : les **keyframes**.

La règle CSS **@keyframes** va permettre de définir les différentes étapes d’une animation :

```
@keyframes round {
    0%   {background-color:red; left:0px; top:0px;}
    25%  {background-color:yellow; left:200px; top:0px;}
    50%  {background-color:blue; left:200px; top:200px;}
    75%  {background-color:green; left:0px; top:200px;}
    100% {background-color:red; left:0px; top:0px;}
}
```

Une fois les étapes de l'animation prête, la propriété CSS **animation** permet de la déclencher sur l'élément.

La propriété animation va prendre en premier le nom de l'animation à exécuter, des propriétés de l'animation : délais avant exécution, durée …

```
div {
    animation: round 5s;
}
```

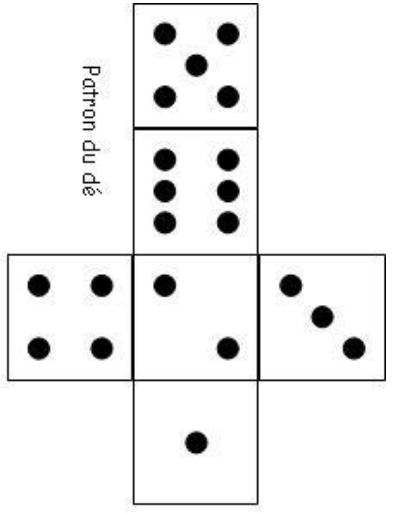
Propriétés d'une animation

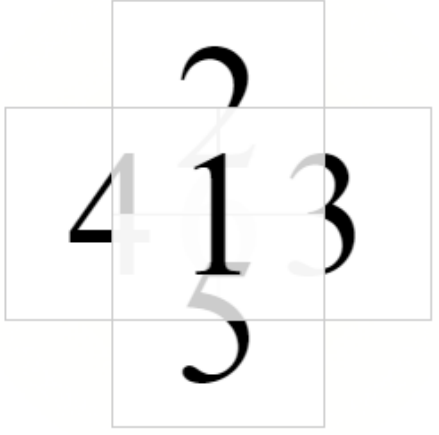
L'animation possède un certain nombre de paramètre :

- animation-direction : spécifie le sens d'exécution
- animation-iteration-count : spécifie le nombre de fois qu'elle doit être jouée. Peut être mis à infinite
- animation-fill-mode : permet d'indiquer dans quel état doit être l'élément après l'animation
- animation-delay : le délais avant exécution
- animation-timing-function : l'easing à utiliser pour l'animation

Let's Play : Awesome Cube

- Créer un cube entièrement en HTML /CSS3.
- Chacune des 6 faces doit être numérotée de **1 à 6**
- La position des numéros doit respecter celle d'un **dé à 6 faces**
- Le cube doit tourner de façon à afficher chacune des faces dans **l'ordre croissant** des numéros
- Lorsque le cube affiche une face, le numéro doit être **lisible dans le bon sens**
- Une fois sur la face 6, l'animation reprend depuis la face numéro 1
- Animer l'effet lumineux





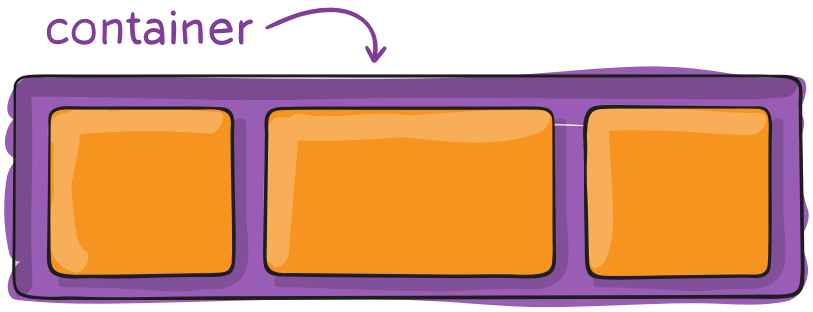
VI

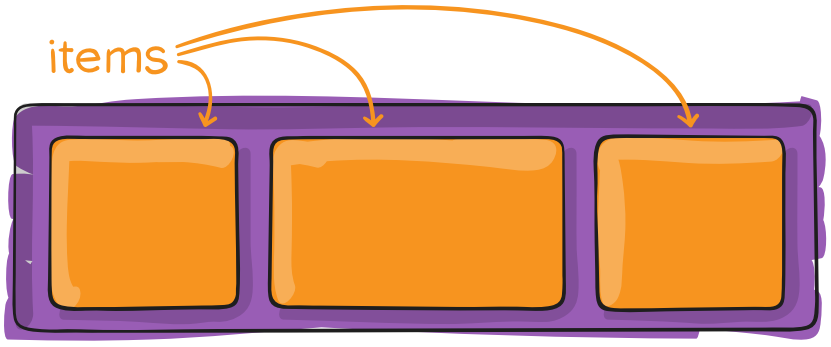
Interfaces avec Flexbox

Introduction au Flexbox

La spécification CSS Flexbox Layout fournit un moyen efficace de d'agencer et de dimensionner les éléments d'une interface au sein d'un document HTML.

La spécifications définit des **conteneurs** (container) encapsulant des **éléments** (items) qui peuvent être agencés en fonction de l'espace disponible au sein du conteneur.

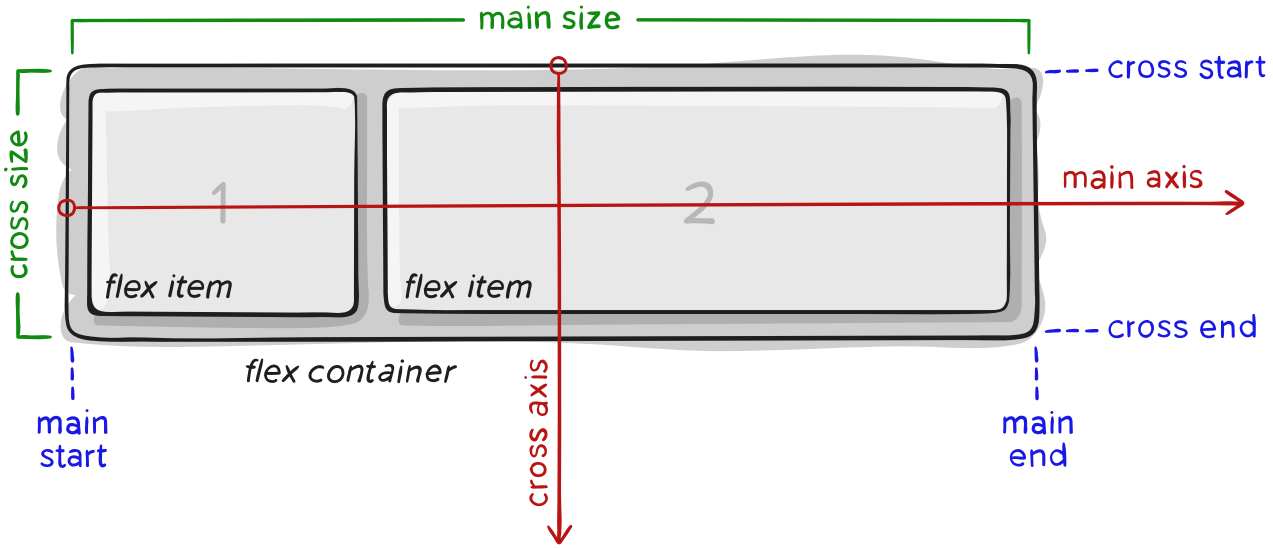




Introduction au Flexbox

La spécifications Flexbox plusieurs propriétés CSS. Une partie de ces propriétés s'appliquent aux containers et l'autre aux items.

Au sein d'un container, les items vont s'agencer selon un axe principal, le **main-axis.** L'axe opposé est le **cross-axis.**



Introduction au Flexbox

La première chose à faire pour utiliser Flexbox est de déclarer un container.

La propriété CSS **display: flex** permet d'initialiser un container Flexbox. Ainsi, **tous les éléments** HTML présents dans le containers seront des items.

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

Introduction au Flexbox

La seconde chose est de définir le sens de l'axe principal **main-axis** en utilisant la propriété flex-drection :

- *row (par défaut)* : affiche les items de gauche à droite
- *row-reverse* : affiche les items de droite à gauche
- *column* : affiche les items de haut en bas
- *column-reverse* : affiche les items bas en haut

```
.container {
  flex-direction: row | row-reverse | column | column-reverse;
}
```

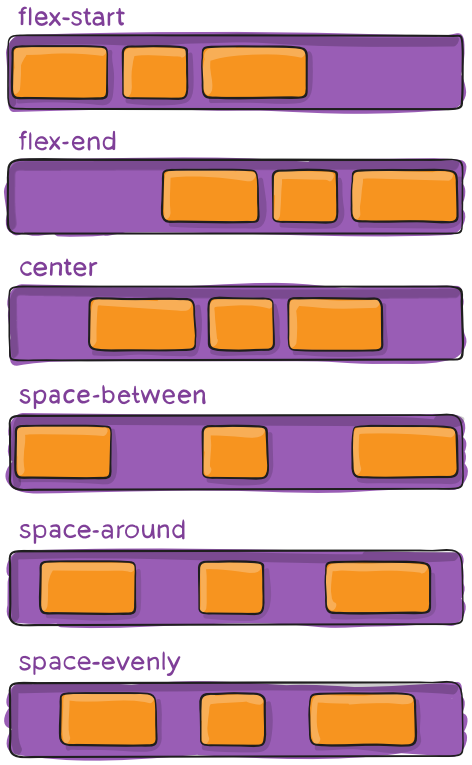
Introduction au Flexbox

justify-content permet d'aligner les items le long du main-axis tant que l'espace disponible au sein du container le permet.

- *flex-start (par défaut)* : positionne les items au début du main-axis
- *center* : positionne les items au centre du main-axis
- *flex-end* : positionne les items à la fin du main-axis
- *space-between* : espace les items uniquement entre eux de manière égale.
- *space-around* : met un espace égale autour de chaque élément
- *space-evenly* : met un espace identique entre les items et les bords du container

Introduction au Flexbox

justify-content permet d'aligner les items le long du main-axis tant que l'espace disponible au sein du container le permet.



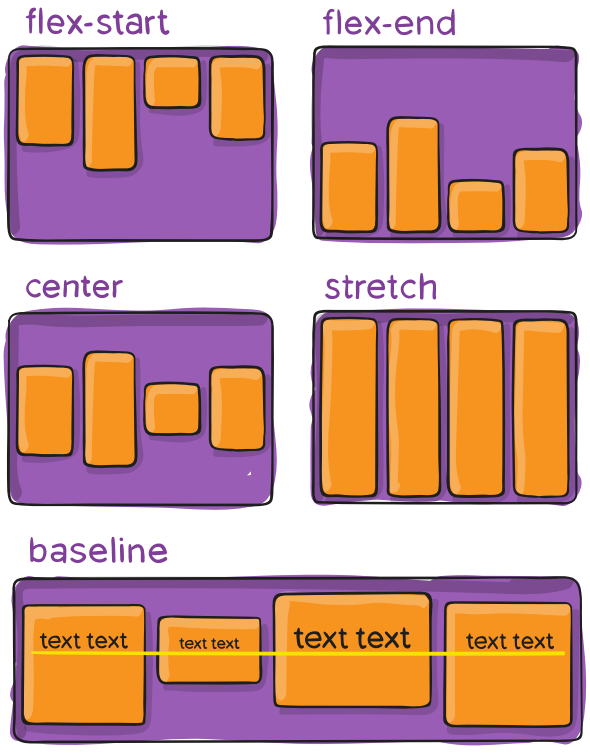
Introduction au Flexbox

align-items permet d'aligner les items le long du cross-axis tant que l'espace disponible au sein du container le permet.

- *stretch (par défaut)* : les items occupent la totalité de l'espace du cross-axis
- *flex-start* : positionne les items au début du cross-axis
- *center* : positionne les items au centre du cross-axis
- *flex-end* : positionne les items à la fin du cross-axis
- *baseline* : aligne les items en fonction de leur alignement de base, de façon à garder le contenu des items aligné

Introduction au Flexbox

align-items permet d'aligner les items le long du cross-axis tant que l'espace disponible au sein du container le permet.



Introduction au Flexbox

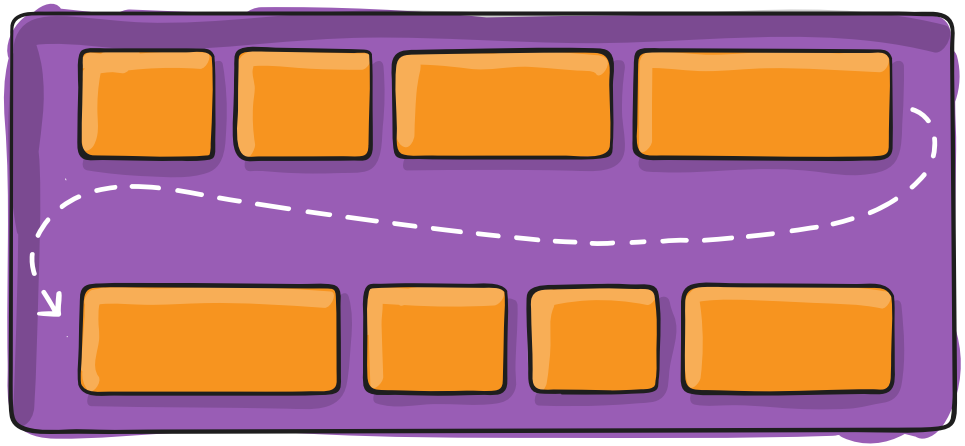
flex-wrap permet de spécifier le comportement des items lorsqu'il n'y a plus d'espace disponible sur le main-axis. Par défaut, les items essaieront au maximum de tenir sur une ligne, quitte à réduire la taille de ces derniers.

- *nowrap (par défaut)* : les items resteront sur une seule ligne, celle correspondant au main-axis
- *wrap*: les items s'étaleront sur plusieurs ligne, favorisant un retour à la ligne lorsqu'il n'y a plus de place sur le main-axis
- *wrap-reverse* : idem que *wrap*, sauf que le retour à la ligne est effectué dans le sens opposé.

Introduction au Flexbox

flex-wrap permet de spécifier le comportement des items lorsqu'il n'y a plus d'espace disponible sur le main-axis.

Par défaut, les items essaieront au maximum de tenir sur une ligne, quitte à réduire la taille de ces derniers.



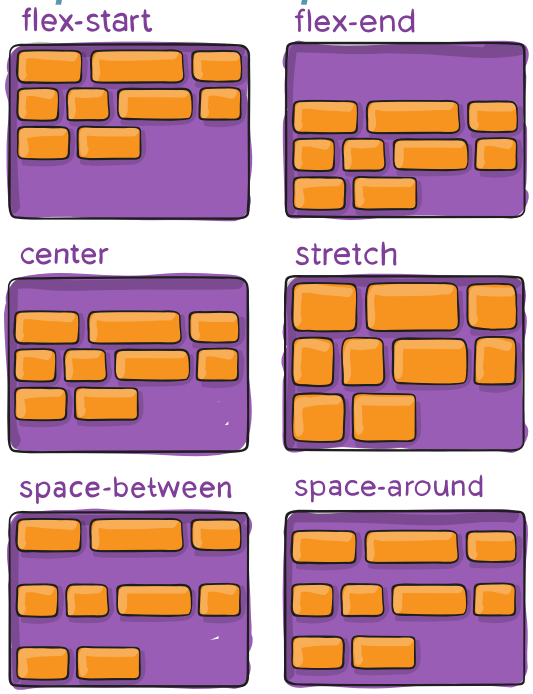
Introduction au Flexbox

align-content va permettre d'agencer les lignes d'items les unes par rapport au autres. Ces lignes sont créées uniquement lorsque la propriété flex-wrap est à *wrap* ou *wrap-reverse*.

- *normal (par défaut)* : les lignes restent dans leur position initiale
- *flex-start* : les lignes seront groupées au début du cross-axis
- *center* : les lignes seront groupées au centre du cross-axis
- *flex-end* : les lignes seront groupées à la fin du cross-axis
- *space-between* : l'espace est distribué également entre les lignes
- *space-around* : l'espace est distribué également autour des lignes
- *flex-evenly* : l'interligne est égale partout
- *stretch*: la taille des lignes sera maximisée pour ne laisser aucun espace vide

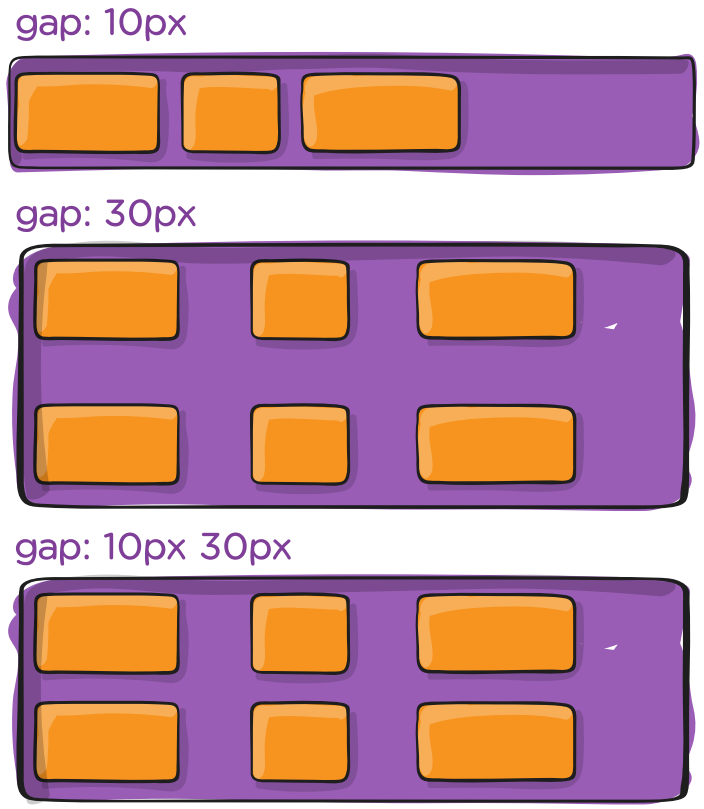
Introduction au Flexbox

align-content va permettre d'agencer les lignes d'items les unes par rapport au autres. Ces lignes sont créées uniquement lorsque la propriété flex-wrap est à *wrap* ou *wrap-reverse*.



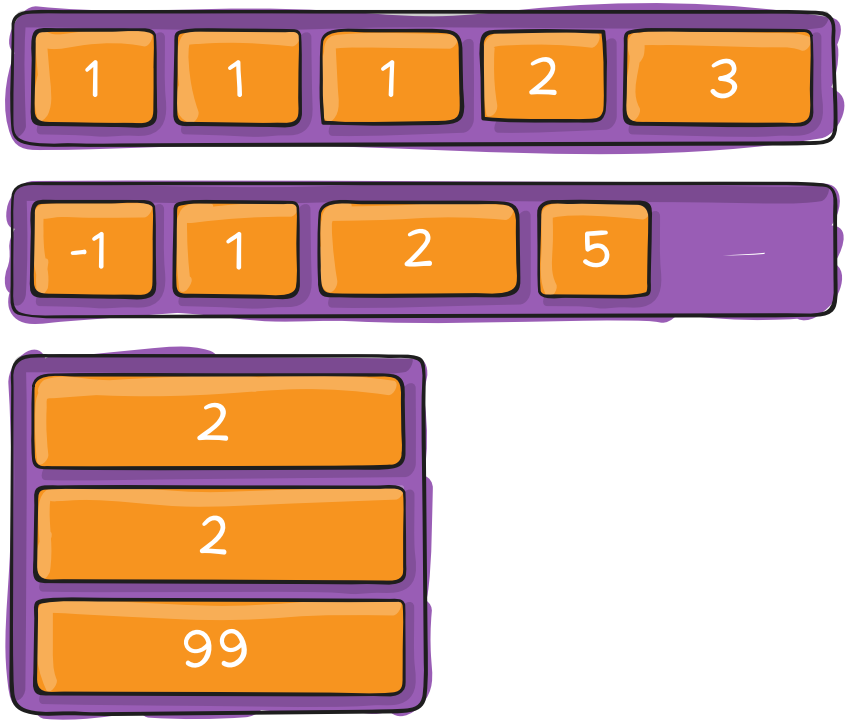
Introduction au Flexbox

gap permet de fixer l'espacement entre les items.



Introduction au Flexbox

order permet d'indiquer dans quel ordre l'item doit être affiché :



Introduction au Flexbox

flew-grow va contrôler la taille d'un item de façon à ce qu'il occupe tout l'espace disponible.

La valeur n'a pas d'unité mais doit être entière et supérieure ou égale à 0 (par défaut).

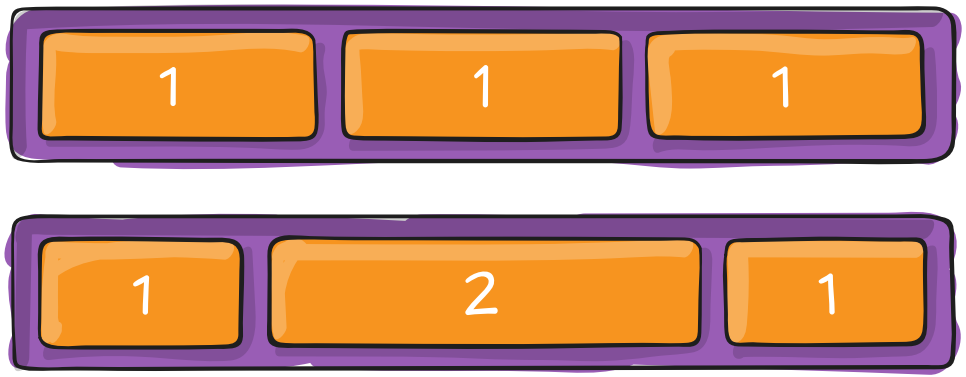
```
.item {
  flex-grow: 1;
}
```

Avec flex-grow supérieur à 0, l'item occupera 100% de l'espace disponible si il est seul.

Introduction au Flexbox

Lorsqu'il y a plusieurs items, l'espace occupé va dépendre par flex-grow dépend de l'espace pris par les autres items voisin.

Si tous les items sont à 1, alors ils auront tous la même taille.
Si un item est à 2, alors il sera 2 fois plus grand que ceux qui sont à 1,



Introduction au Flexbox

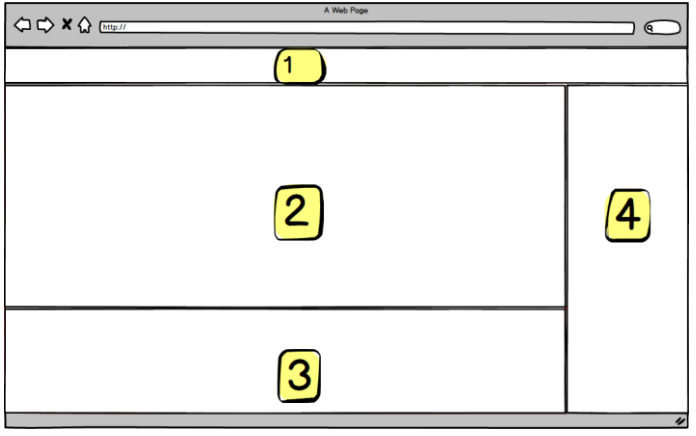
La propriété flex est une propriété qui combine flex-grow ainsi que d'autres propriétés (non présentées dans le cours).

Il est recommandé d'utiliser directement la propriété flex à la place de flex-grow:

```
.item {
  flex: 1
}
```

Let’s Play : Le jeu de paires partie 1

- Designer l’interface d’un jeux de paires à l’aide de Flexbox
- L’interface doit contenir un header **(1)**, une zone pour les cartes **(2)**, une zone pour les informations sur la partie en cours **(3)** ainsi qu’une zone pour le résultat des jeux précédents **(4).**
- Au clic sur un carte, elle doit être en mesure de révéler le numéro sur sa face cache via un « flip ».
- Toutes les zones, hormis le header, doivent être scrollable.



VII

Responsive Design

Définition du Responsive Design

Le « responsive design » désigne l'ensemble des techniques permettant de concevoir des interfaces web capables d'agencer leur contenu en fonction de la taille de l'écran.

Le **responsive design** est une approche qui garanti une **expérience utilisateur** optimale, quelle que soit la plateforme sur lequel l'utilisateur navigue.

Définition du Responsive Design

Pour gérer les différents périphérique, CSS dispose des ***media queries*** qui permettent de définir des règles CSS uniquement sur certain supports.

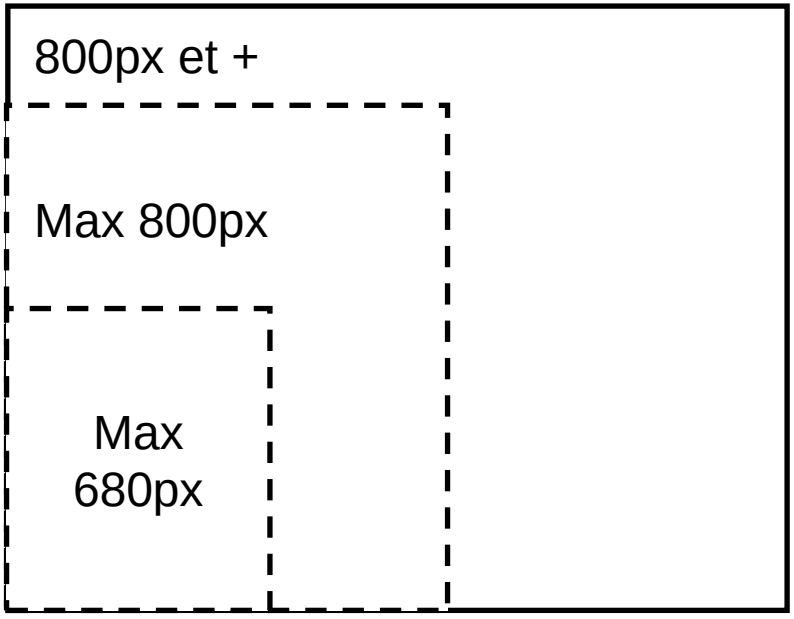
Les ***media queries*** ne servent pas seulement pour le responsive design, c'est un moyen d'appliquer des règles en fonction du type du support, et de certaines de ses propriété, comme la taille de l'écran.

Il est donc possible d'avoir de la CSS spécifique pour l'impression, pour le mode sombre ou limitée à une certaine résolution d'écran.

Définition du Responsive Design

L'emploi de la *media query* se fait avec le mot clé **@media** suivi de la caractéristique sur laquelle est s'applique :

```
font-size: 20px;

@media (max-width: 800px) {
  font-size: 18px;
}

@media (max-width: 680px) {
  font-size: 16px;
}
```



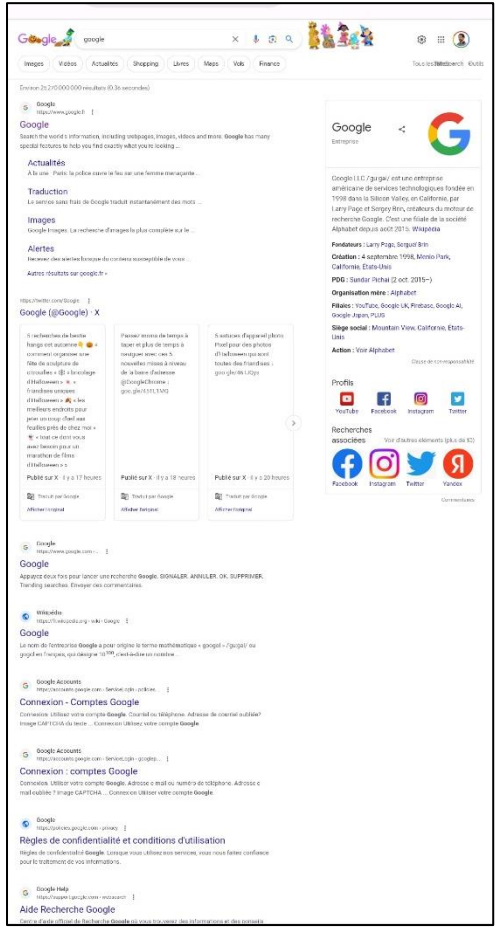
Le Viewport

Le ***viewport*** correspond à toute la zone visible dans laquelle la page s'affiche.

Sur mobile la taille du *viewport* est décorrélée de la taille réelle de l'écran.

Le viewport est souvent supérieur à la taille de l'écran.

Le problème est que dans ce scénario, les *media queries* ne seront jamais déclenchées.



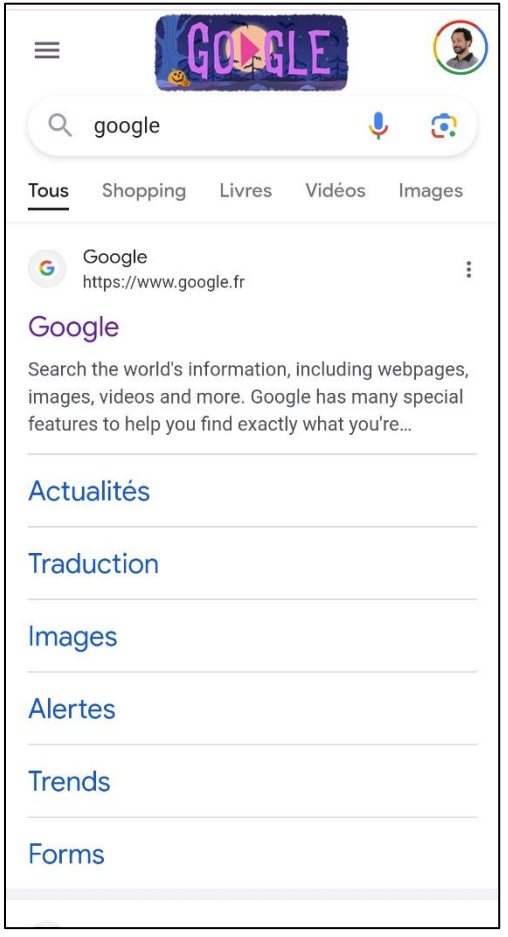
La balise meta viewport

Pour forcer un mobile à utiliser un viewport égale à la taille de l'écran, il faut passer la balise meta qui doit être situé dans le header du document :

```
<meta name="viewport" content="width=device-width">
```

Responsive Design et mobile

Lorsque le viewport correspond à la taille de l'écran, les *media queries* vont pouvoir être déclenchée comme attendu.



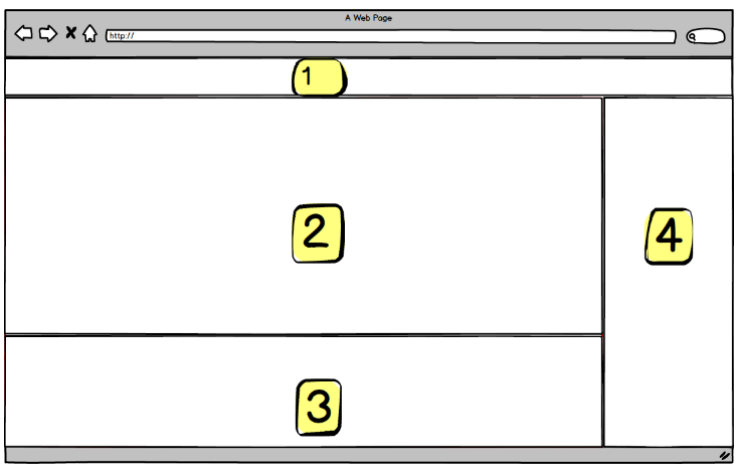
La balise meta viewport

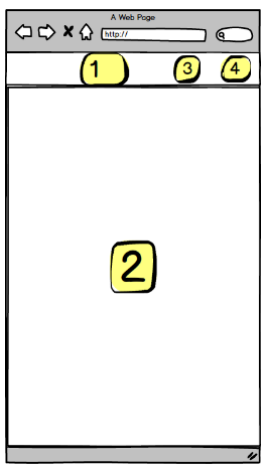
La balise meta viewport peut également spécifier le comportement à adopter en cas de *pinch-to-zoom :*

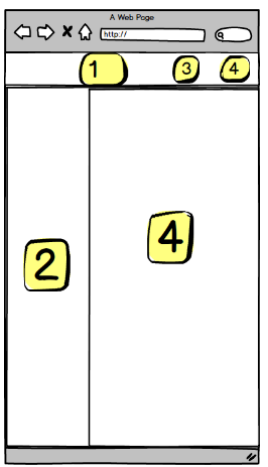
```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=yes">
```

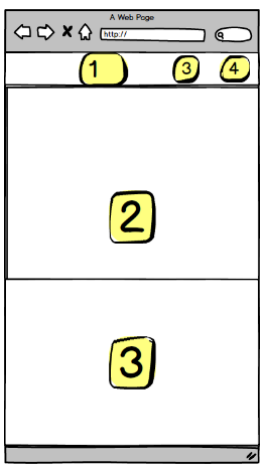
Let’s Play : Le jeu de paires partie 2

- Rendre l’interface du jeux de paire responsive.
- Les deux menus **(3)** et **(4)** doivent être masqués en version mobile.
- Deux boutons doivent permettent d’ouvrir le menu **(3)** et le menu **(4).**
- Un menu ouvert passe par dessus le contenu **(2)**.
- Le menu **(2)** à une largeur fixe, le menu **(3)** à une hauteur fixe.









VIII

CSS and Beyond

Les variables CSS

Ajout des variables en CSS et de la pseudo classe :root :

```css
:root {
  --main-bg-color: brown;
}

body {
  background: var(--main-bg-color);
}
```

Les nested declarations

Ajout des déclarations imbriquées :

```
.square {
  width: 250px;
  height: 250px;
}

.square .rounded {
  border-radius: 25px;
}
```

```
.square {
  width: 250px;
  height: 250px;

  .rounded {
    border-radius: 25px;
  }
}
```