

**CSS**

**CSS**





# Introduction au CSS:

Css signifie Cascading Style Sheets, traduit cela signifie fiche de style en cascade, voyons pourquoi il se nomme ainsi :

Ses propriétés :

- 📌 Le CSS est un langage de mise en page de document html.
- 📌 Il permet une définition homogène des styles d'une page web.

Il y a trois façons pour définir ces styles :

- 📌 en utilisant l'attribut style dans des balises HTML classiques
- 📌 au sein de la section `<head>` par une balise `<style>`
- 📌 dans un fichier d'extension .css appelé dans une page par la balise `<link>`



# Evolution du CSS:

CSS1 : introduit en 1996. Début très difficile à cause de la guerre entre les navigateurs

CSS2 : apparue en 1998, environ 70 nouvelles propriétés par rapport à CSS1

CSS3 : version officiellement non finalisée, mais plusieurs nouvelles propriétés prises en charge par les navigateurs.



Première méthode :

```
<tag style="property: value;"></tag>
```

Exemple :

```
<p style="color:red;"></p>
```

Le texte de ce paragraphe aura un arrière plan rouge.



## Deuxième méthode :

```
<style type="text/css">  
  selector {  
    property: value;  
  }  
</style>
```



Exemple :

```
<style type="text/css">
  p {
    color: red;
  }
</style>
```

Tous les paragraphes de mon document seront affichés en rouge.

Avec HTML 5

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

Plus besoin de préciser le type pour la balise <style>.



# Troisième méthode :

Dans le fichier CSS (file.css) :

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

Dans le fichier HTML:

```
<head>  
  <link rel="stylesheet" type="text/css" href="file.css" />  
</head>
```

Tous les paragraphes de mon document seront affichés en rouge.





```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>cours</title>
  </head>
  <body>
    <p style="color: red">je m'affiche en rouge</p>
    <p style="color: blue">je m'affiche en bleu</p>
    <p style="color: purple">je m'affiche en violet</p>
  </body>
</html>
```



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>cours</title>
    <style>
      h1 {
        color: blue;
      }
      h2 {
        color: red;
      }
      h3 {
        color: purple;
      }
    </style>
  </head>
  <body>
    <h1>je m'affiche en bleu</h1>
    <h2>je m'affiche en rouge</h2>
    <h3>je m'affiche en violet</h3>
  </body>
</html>
```



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>cours</title>
    <style>
      h1 {
        color: blue;
      }
      h2 {
        color: red;
      }
      h3 {
        color: purple;
      }
    </style>
  </head>
  <body>
    <h1>je m'affiche en bleu</h1>
    <h2>je m'affiche en rouge</h2>
    <h3>je m'affiche en violet</h3>
  </body>
</html>
```



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>cours</title>
    <link rel="stylesheet" href="file.css" />
  </head>
  <body>
    <h1>je m'affiche en bleu</h1>
    <h2>je m'affiche en rouge</h2>
    <h3>je m'affiche en violet</h3>
  </body>
</html>
```



```
h1 {  
    color: blue;  
}  
h2 {  
    color: red;  
}  
h3 {  
    color: purple;  
}
```

# Règles CSS :



```
selecteur {  
    propriete: valeur;  
}
```





Une règle de css s'écrit comme ceci, elle comporte une partie sélecteur, une partie propriété, et une valeur.

- 📌 Le sélecteur sert à choisir sur quels éléments de notre page html sera appliquer la règle.
- 📌 La propriété sera la partie que nous souhaitons changer, il en existe un très grand nombre, et ils dépendent de l'élément sélectionné.
- 📌 La valeur est tout simplement la valeur que prendra la propriété de l'élément sélectionné.

# Sélecteur :

Les sélecteurs définissent les éléments sur lesquelles s'applique un ensemble de règles CSS.

Ils peuvent sélectionner un élément a partir :

-  d'une balise HTML
-  d'une classe
-  d'un identifiant
-  d'une combinaison de sélecteur.



# Balises HTML :

Concerne tout les éléments de la page html partageant la même balise.

```
h1 {  
    color: blue;  
}
```

De cette manière tout les balises h1 de mon document html seront en bleu.





# Les classes :

Concerne tout les éléments de la page qui partagent la même classe.

```
<h1 class="bleu">je m'affiche en bleu</h1>
```

De cette manière la balise h1 appartient a la classe “bleu”.

Nous pouvons donc sélectionner cette balise par sa classe.

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



# Les identifiant :

Concerne les élément de la page avec le même id.

```
<h1 id="bleu">je m'affiche en bleu</h1>
```

De cette manière la balise h1 aura l'identifiant "bleu".

Nous sélectionnerons nos balises via leur id en utilisant le #.

Nous le ferons de cette manière.

```
#bleu {  
    color: blue;  
}
```



# Les sélecteurs :

*	tous les élément	<code>&lt;body&gt;...&lt;/body&gt;</code>
<code>h1, p</code>	les h1 et p	<code>&lt;h1&gt;...&lt;p&gt;..&lt;/p&gt;...&lt;/h1&gt;</code>
<code>div p</code>	les p dans h1	<code>&lt;div&gt;oui&lt;p&gt;...&lt;span&gt;non&lt;p&gt;</code>
<code>p#gras</code>	les p portant l'id gras	<code>&lt;p id="gras"&gt;oui&lt;p&gt;...&lt;span&gt;mon&lt;p&gt;</code>
<code>#gras</code>	tout les élément avec l'id gras	<code>&lt;p id="gras"&gt;oui&lt;p&gt;...&lt;span&gt;mon&lt;p&gt;</code>
<code>p.bleu</code>	les p portant la classe bleu	<code>&lt;p class="bleu"&gt;oui&lt;p&gt;...&lt;span&gt;mon&lt;p&gt;</code>
<code>h1 + p</code>	les p directement après h1	<code>&lt;h1&gt; &lt;p&gt;oui&lt;/p&gt;&lt;p&gt;non&lt;/p&gt;&lt;/h1&gt;</code>
<code>div &gt; p</code>	les p enfant direct de div	<code>&lt;div&gt;...&lt;p&gt;oui&lt;/p&gt;&lt;/div&gt; &lt;div&gt;...&lt;span&gt;..&lt;p&gt; non&lt;/p&gt;&lt;/span&gt;&lt;/div&gt;</code>
<code>div ~ p</code>	les p précédés par div avec p et div ont le même parent	<code>&lt;ul&gt;...&lt;/ul&gt;&lt;p&gt;non&lt;/p&gt; &lt;div&gt;...&lt;/div&gt;&lt;p&gt;oui&lt;/p&gt;</code>



# Les autres sélecteurs :

<code>input [type=text]</code>	les input de type text	<code>&lt;input type='text'/&gt;</code>
<code>img[title~=red]</code>	les images avec un titre contenant le mot red	<code>&lt;img title='red bull'/&gt;</code> oui <code>&lt;img title='redbull'/&gt;</code> non
<code>img[title*=red]</code>	les image avec un titre contenant le mot red	<code>&lt;img title='red bull'/&gt;</code> oui <code>&lt;img title='redbull'/&gt;</code> oui
<code>[lang =en]</code>	les éléments avec un attribut lang commençant par en	<code>&lt;div lang="en-us"&gt;</code> oui <code>&lt;div lang="fr"&gt;</code> non
<code>img[src\$=.jpg]</code>	les images ayant un nom se terminant par .jpg	<code>&lt;img src='1.jpg'/&gt;</code> oui <code>&lt;img src='jpg.png'/&gt;</code> non



# Les pseudo sélecteur :

Un pseudo-sélecteur nous permet d'affiner nos sélections, et de prendre “uniquement” ce que nous souhaitons.

La syntaxe se fait de cette manière :

```
selecteur:pseudoselecteur {  
    propriete: valeur;  
}
```

Voici différents pseudo-sélecteurs :



balise	exemple
p:first-letter	la première lettre de chaque élément p
p:first-line	la première ligne de chaque élément p
p:before	le contenu avant chaque élément p
p:after	le contenu après chaque élément p
p:lang(en)	les p avec un attribut lang dont la valeur commence par 'en'
td[colspan='3']	les cases d'un tableau qui sont sur 3 colonnes



<b>li:not(:first-child)</b>	<b>tous les éléments qui ne sont pas les premiers</b>
p:nth-child(2)	les p qui sont les seconds enfants de leurs parents
p:nth-last-of-type(2)	les p qui sont les seconds enfants de leurs parents en commençant par le dernier
div ul.menu li.entree	les li de classe 'entree' qui sont dans 'ul' de classe 'menu', qui sont dans des div
a:hover	les liens survoles
a:visited	les liens visites
input:optional	les inputs sans la propriété required
input:required	les inputs avec la propriété required
input:read-only	les inputs avec la propriété readonly



# Quelques exemples :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>cours</title>
    <link rel="stylesheet" href="file.css" />
  </head>
  <body>
    <h1 class="rouge">premier h1 de class rouge</h1>
    <h1 class="bleu">second h1 class bleu</h1>
    <h1 class="bleu">troisieme h1 class bleu</h1>
  </body>
</html>
```





```
h1:first-of-type {  
    background-color: black;  
}
```

```
h1:nth-of-type(2) {  
    background-color: brown;  
}
```

```
.rouge {  
    color: red;  
}
```

```
.bleu::first-letter {  
    color: yellow;  
}
```

```
.bleu {
```



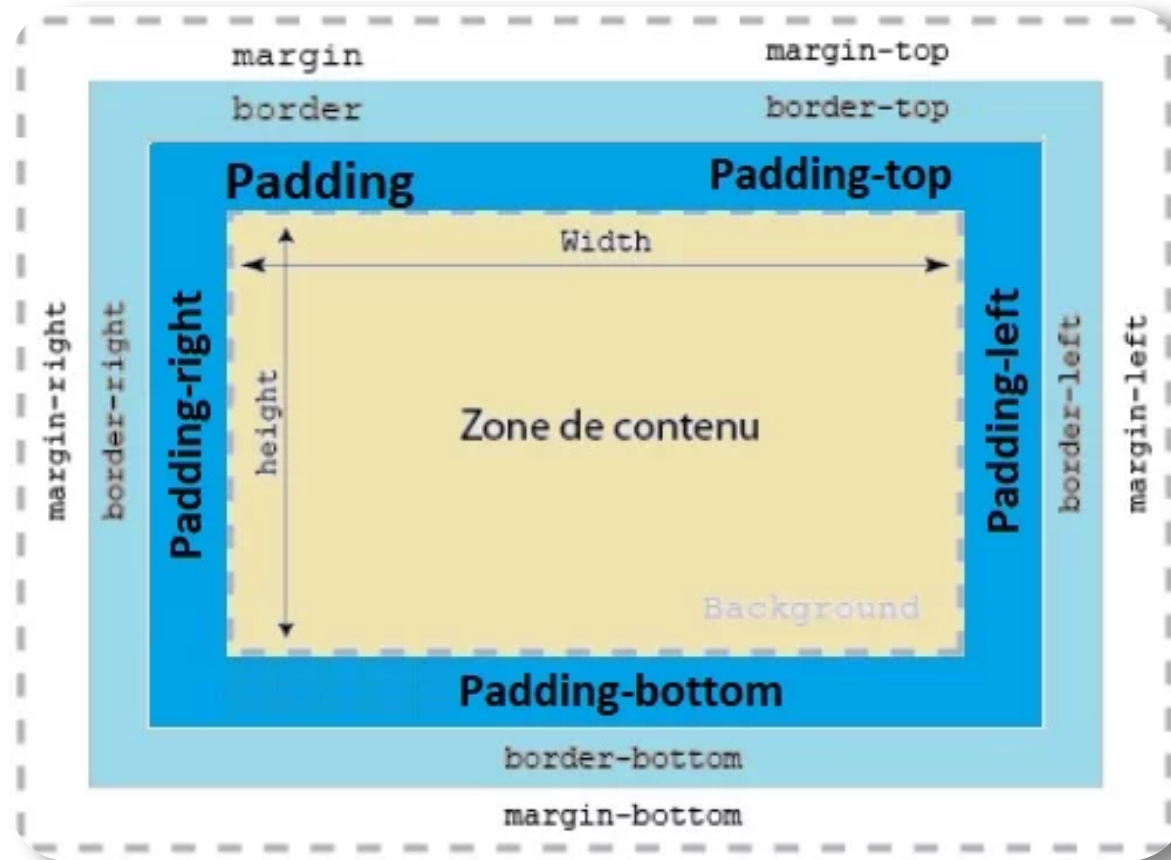
# Pour s'exercer :

<https://flukeout.github.io/>



# Modèle de boîte :







**Selon la spécification W3C, toutes les balises HTML se présentent sous cette forme :**





# Les attributs de notre boite :

les dimensions :

-  width : la largeur de notre zone de contenu
-  height : la hauteur de notre zone de contenu
-  min-width : largeur minimale
-  max-width : largeur maximale
-  min-height : hauteur minimale
-  max-height : hauteur maximale.








Ils ne représentent pas la taille de la boîte, celle ci se calcule en ajoutant la taille de la marge et de la bordure !

elles se définissent de cette manière :





```
#box {  
    height: 50px;  
    width: 50px;  
}
```

# Le padding:

-  padding : défini la distance entre la zone de contenu et la bordure
-  padding-top : défini uniquement la zone de padding du dessus
-  padding-right : défini uniquement la zone de padding de droite
-  padding-left : défini uniquement la zone de padding de gauche
-  padding-bottom : défini uniquement la zone de padding du dessous

```
#box {  
  padding: 5px;  
  padding-left: 5px;  
  ...;  
}
```

# La bordure:

-  `border` : définit la bordure qui entoure la zone de contenu
  -  `border-radius` : définit la courbure des angles de la zone de contenu
  -  `border-style` : définit le style de la bordure
  -  `border-color` : définit la couleur de la bordure
- N'oubliez pas de définir votre bordure, sinon elle ne s'affichera pas.





Bien sur nous pourrions comme a chaque fois sélectionner la zone a définir : top/right/left/bottom

```
#box {  
  border: solid;  
  border: 5px;  
  border-radius: 10px;  
  border-style: dotted;  
}
```



# La marge:

**margin** : permet d'espacer notre élément

Bien sur nous pourrons comme a chaque fois sélectionner la zone a définir : top/right/left/bottom

```
#box {  
    margin: 15px;  
}
```

# Propriété multivaluée:



Il est possible pour certains attributs de prendre plusieurs valeurs d'entrée à la suite respectant une certaine logique.

par exemple:

border

- 📌 border-width : épaisseur
- 📌 border-color : couleur
- 📌 border-style : plusieurs valeurs possibles : none, solid, dashed, dotted...

```
#box {  
    border: 1px red dotted;  
}
```



# Les différentes propriétés pour formater du texte:

Les valeurs de la propriété de font-style :

 italic

 normal

 oblique

Les valeurs de la propriété font-weight :

 normal


 bold

Les valeurs de la propriété text-decoration :

 underline





 overline

 line-through

 none

# L'overflow:





Et si notre texte ne rentre pas dans la boîte, on peut utiliser la propriété overflow pour indiquer ce qu'il faut faire :

-  hidden : cacher le texte qui dépasse
-  visible : le texte reste visible à l'extérieur de la boîte
-  scroll : ajouter une barre de défilement pour parcourir le texte qui dépasse
-  auto : le navigateur décide de rajouter une barre de défilement si nécessaire

# Les positions :



Définir la position nous permet de mettre des objets à des endroits précis :

-  relative : L'élément est positionné dans le flux normal du document puis décalé, par rapport à lui-même.
-  absolute : position absolue définie par rapport au block englobant (un élément positionne en absolu ne se réfère pas à son élément parent direct, mais au premier ancêtre positionne qu'il rencontre.).
-  fixed : position qui reste fixe, l'élément est toujours positionne par rapport à la fenêtre du navigateur même si on descend dans la page (avec un scroll par exemple).
-  sticky (collant) : position dépendante de défilement de l'utilisateur



# La propriété z-index :

z-index prend une valeur numérique. L'élément avec un plus grand z-index sera placé au-dessus des autres.

Il ne fonctionne que sur des éléments ayant une position absolute, relative ou fixed.



# Float :




La propriété float permet de déclarer et choisir l'emplacement d'un objet flottant.

Un objet flottant ne dépends plus du flux pour se positionner.

Les valeurs de la propriété float :

 left

 right

Les valeurs de la propriété clear : pour annuler float :

 left

 right

 both



```
<!DOCTYPE html>
<html>
  <head>
    <title>float et clear</title>
    <link rel="stylesheet" href="file.css" />
  </head>
  <body>
    <div class="container">
      <div class="d1">1</div>
      <div class="d2">2</div>
      <div class="d3">3</div>
      <div class="d4">4</div>
    </div>
  </body>
</html>
```



```
.container > div {  
  text-align: center;  
  height: 100px;  
  font-size: xx-large;  
  color: white;  
  align-items: center;  
  display: flex;  
  justify-content: center;  
  text-shadow: #000000 0px 0px 10px;  
}  
  
.d1 {  
  background-color: #256d85;  
}  
  
.d2 {  
  background-color: #47b5ff;  
}  
  
.d3 {  
  background-color: #004e8a;  
}  
  
.d4 {  
  background-color: #06283d;  
}
```



1

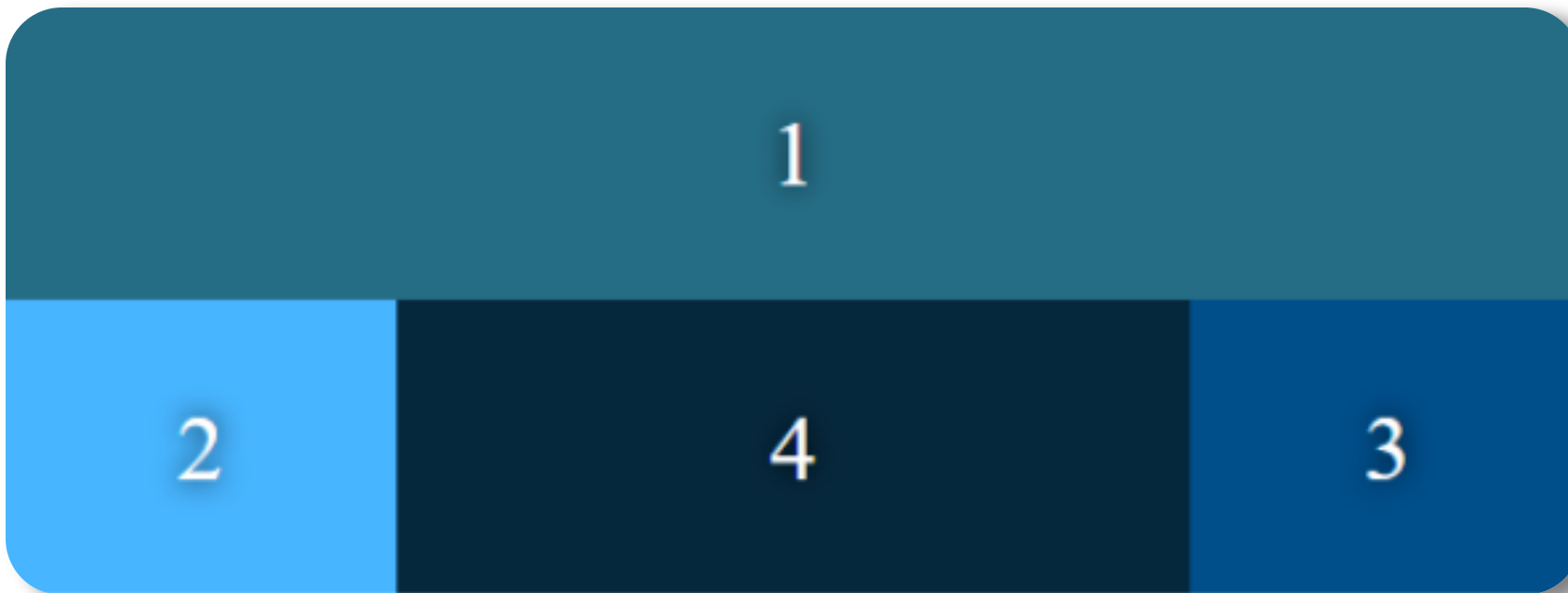
2

3

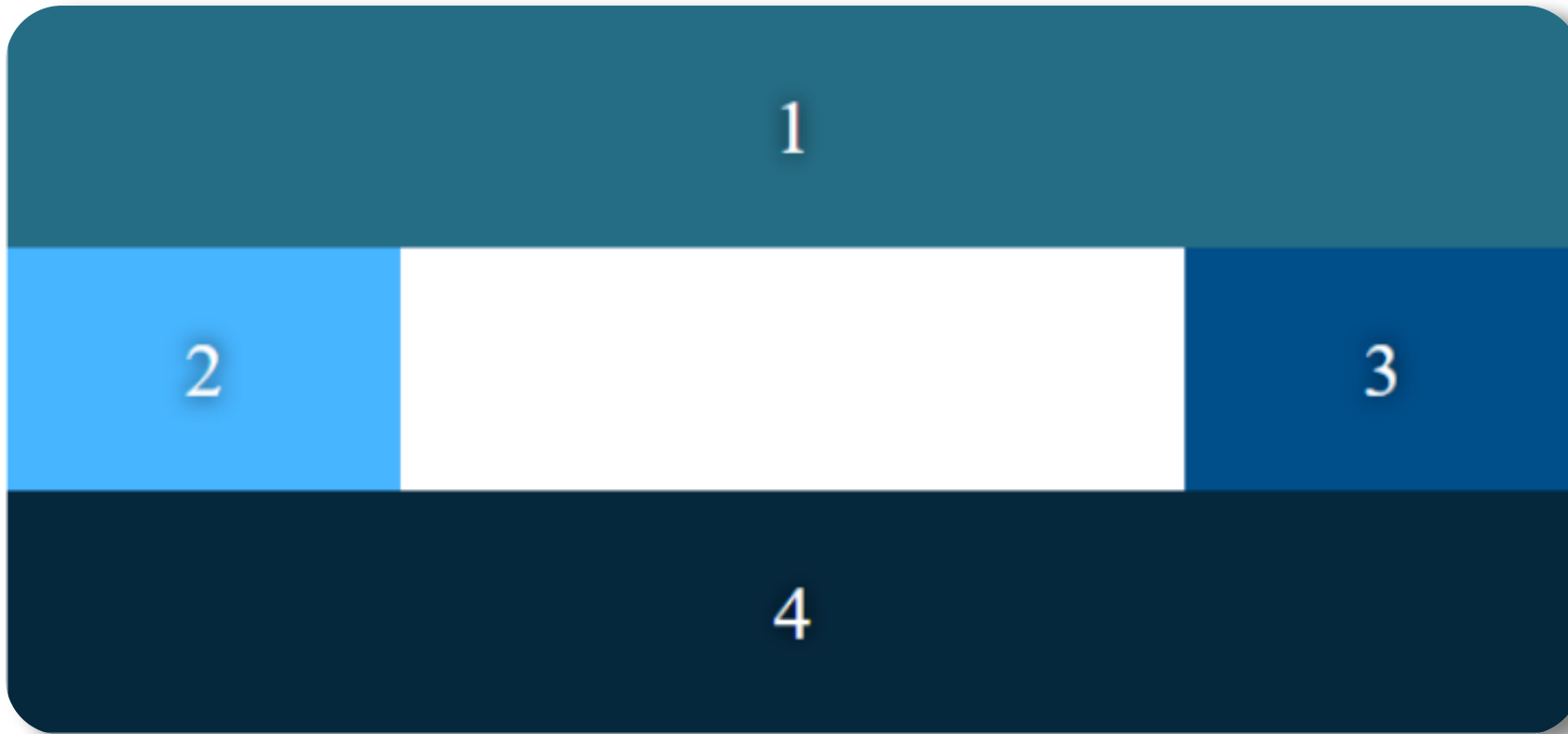
4



**Maintenant, nous souhaitons  
obtenir ce résultat :**



Puis celui-ci :





```
.container > div {  
  text-align: center;  
  height: 100px;  
  font-size: xx-large;  
  color: white;  
  align-items: center;  
  display: flex;  
  justify-content: center;  
  text-shadow: #000000 0px 0px 10px;  
}  
  
.d1 {  
  background-color: #256d85;  
}  
  
.d2 {  
  background-color: #47b5ff;  
  float: left;  
  width: 25%;  
}  
  
.d3 {  
  background-color: #004e8a;  
  float: right;  
  width: 25%;  
}  
  
.d4 {  
  background-color: #06283d;  
  /* clear: both; */  
}
```

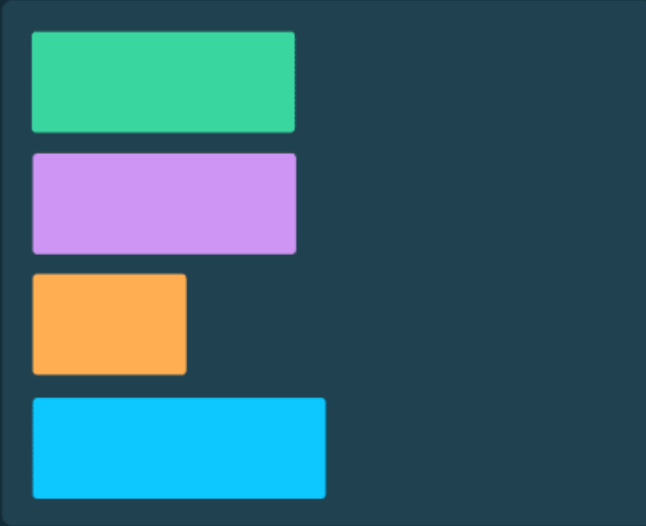


# La propriété display :

- 📌 inline-block : permet de placer des éléments inline tout en préservant leurs capacités d'éléments block, tel que la possibilité de définir une largeur et une hauteur, des marges et padding top et bottom, ...



## BLOCK VS INLINE



**display: block;**

Block elements stack, regardless of their width.







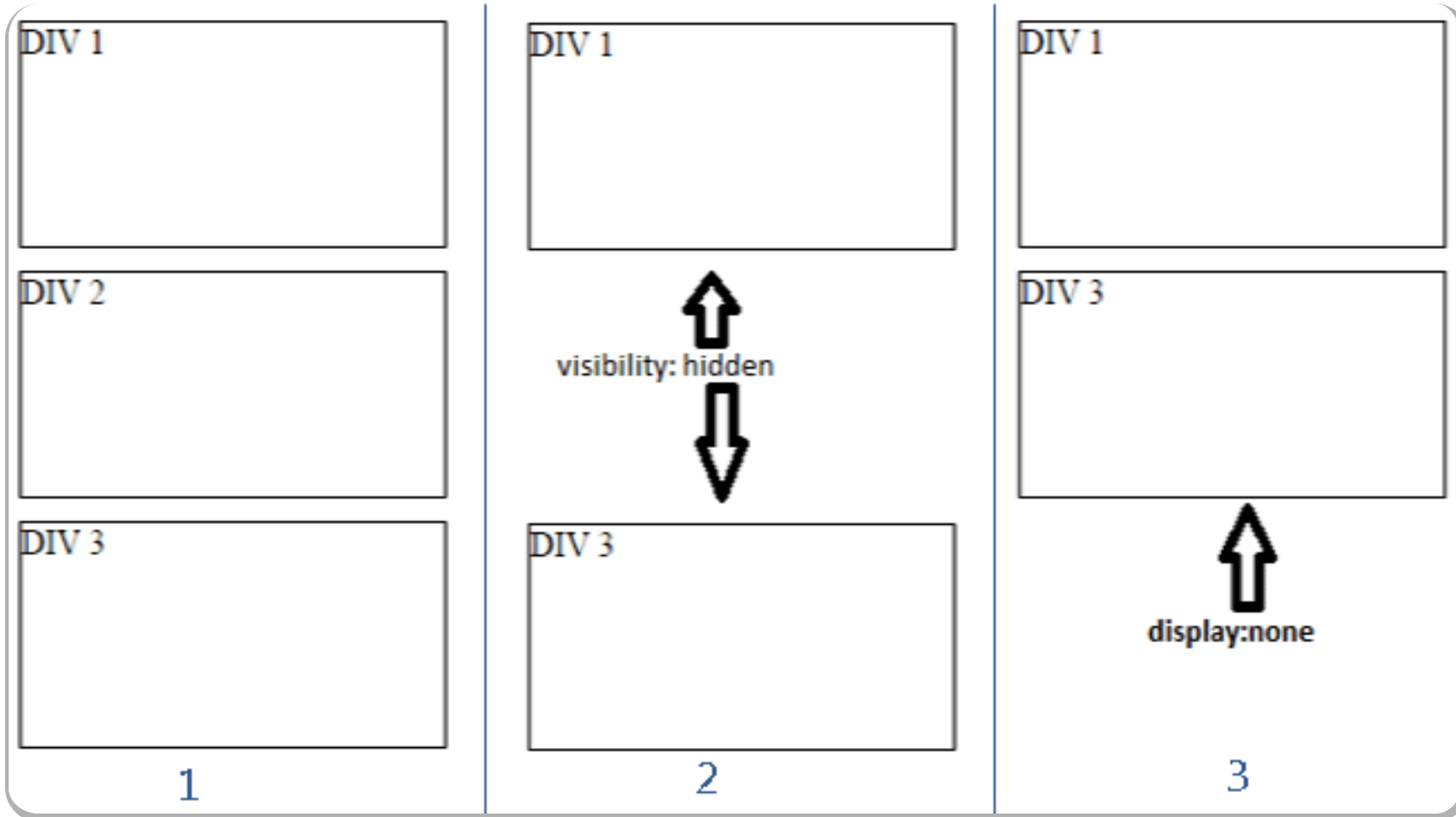
**display: inline;**

Inline elements flow from one line to the next.









# La propriété visibility :

-  visible : l'élément est visible
-  hidden : l'élément est caché
-  collapse : l'élément est caché, mais il garde sa place dans le layout
-  inherit : l'élément hérite de la valeur de son parent



# Display:

-  flex : l'élément est affiché en flex
-  grid : l'élément est affiché en grid
-  list-item : l'élément est affiché comme un élément de liste
-  run-in : l'élément est affiché comme un élément de run-in
-  contents : l'élément est affiché comme un élément de contenu
-  inherit : l'élément hérite de la valeur de son parent

# Flexbox



La propriété `display: flex` permet de créer un conteneur flex.

Quand on applique cette propriété à un élément, tous ses enfants immédiats deviennent des éléments flex.





flexbox permet de créer des layouts flexibles et responsives.

Cela permet de créer des layouts en 1 dimension (ligne ou colonne) ou en 2 dimensions (ligne et colonne).

exemple :

```
.container {  
  display: flex;  
  flex-direction: row;  
  justify-content: space-between;  
  align-items: center;  
}
```

# Flex-direction :

-  row : définit la direction du flux principal comme étant de gauche à droite
-  row-reverse : définit la direction du flux principal comme étant de droite à gauche
-  column : définit la direction du flux principal comme étant de haut en bas
-  column-reverse : définit la direction du flux principal comme étant de bas en haut



exemple :

```
.container {  
  display: flex;  
  flex-direction: row;  
  align-items: center;  
  height: 100px;  
  width: 300px;  
}
```



1

2

3

4





```
.container {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  height: 100px;  
  width: 300px;  
}
```



1

2

3

4



```
.container {  
  display: flex;  
  flex-direction: row-reverse;  
  align-items: center;  
  height: 100px;  
  width: 300px;  
}
```








4

3

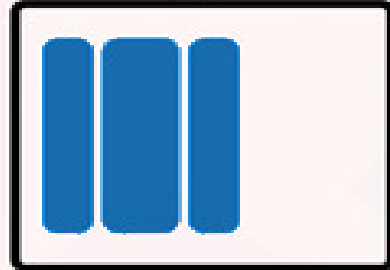
2

1

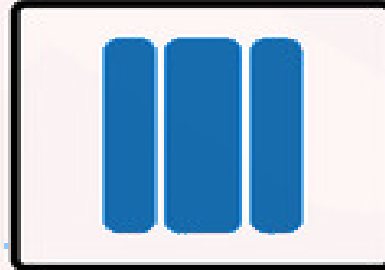
# Justify-content :

-  flex-start : aligne les éléments au début de la ligne
-  flex-end : aligne les éléments à la fin de la ligne
-  center : aligne les éléments au centre de la ligne
-  space-between : aligne les éléments avec un espace égal entre eux
-  space-around : aligne les éléments avec un espace égal autour d'eux

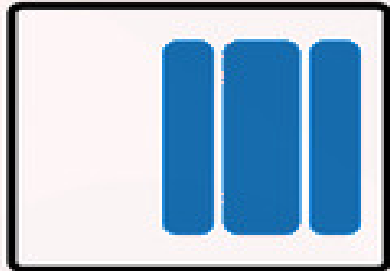
## justify-content



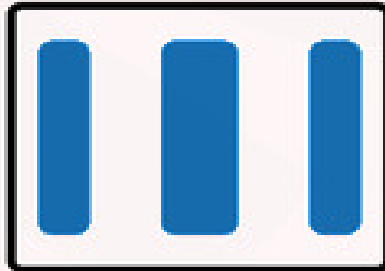
**flex-start**



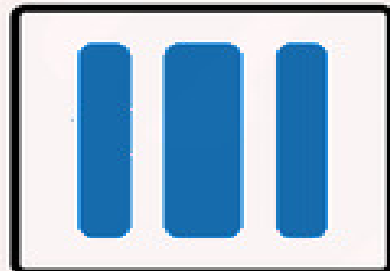
**center**



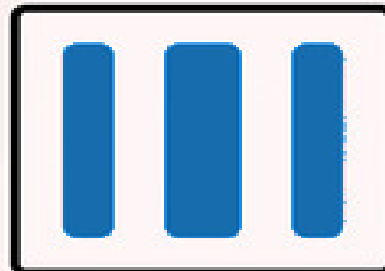
**flex-end**



**space-between**








**space-around**



**space-evenly**

# Align-items :

-  stretch : étire les éléments pour qu'ils remplissent l'espace disponible
-  flex-start : aligne les éléments au début de la ligne
-  flex-end : aligne les éléments à la fin de la ligne
-  center : aligne les éléments au centre de la ligne
-  baseline : aligne les éléments sur leur ligne de base

## align-items



flex-start



center









flex-end



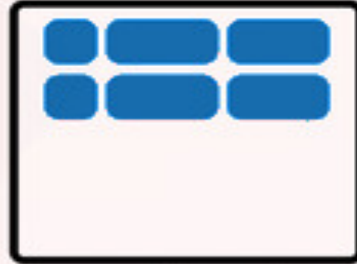
stretch



# Align-content :

-  stretch : étire les lignes pour qu'elles remplissent l'espace disponible
-  flex-start : aligne les lignes au début de la ligne
-  flex-end : aligne les lignes à la fin de la ligne
-  center : aligne les lignes au centre de la ligne
-  space-between : aligne les lignes avec un espace égal entre elles
-  space-around : aligne les lignes avec un espace égal autour d'elles

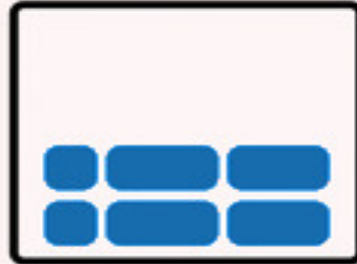
## align-content



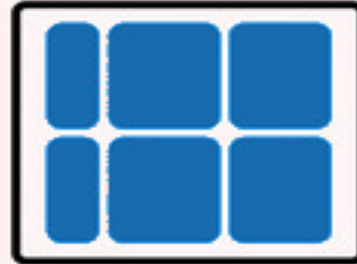
flex-start



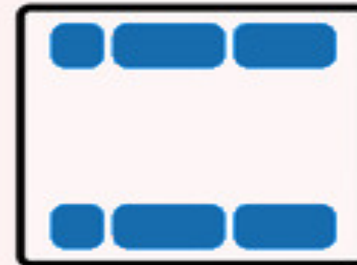
center



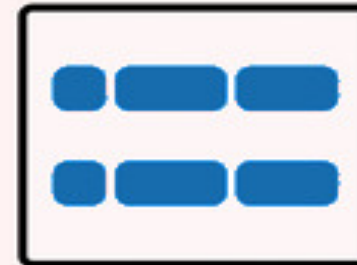
flex-end



stretch



space-between



space-around

# Flex-wrap :

- 📌 flex-wrap: nowrap; permet de placer les éléments les uns à la suite des autres, de gauche à droite
- 📌 flex-wrap: wrap; permet de placer les éléments les uns à la suite des autres, de droite à gauche
- 📌 flex-wrap: wrap-reverse; permet de placer les éléments les uns à la suite des autres, de droite à gauche



**flex-wrap: nowrap**

1 2 3 4 5

**flex-wrap: wrap**

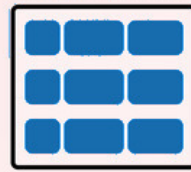
1 2 3 4  
5

**flex-wrap: wrap-reverse**

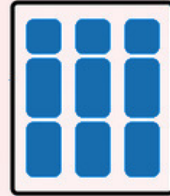
5 4 3 2  
1

## CSS Flexbox

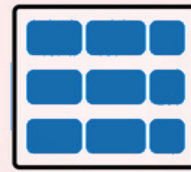
### flex-direction



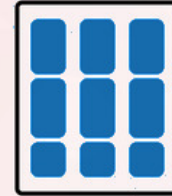
row



column



row-reverse



column-reverse

### align-items



flex-start



center



flex-end



stretch

### justify-content



flex-start



center



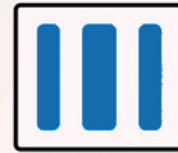
flex-end



space-between

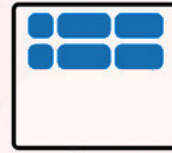


space-around

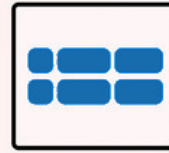


space-evenly

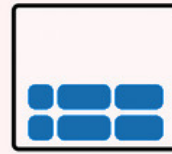
### align-content



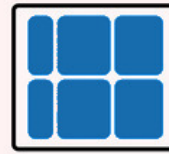
flex-start



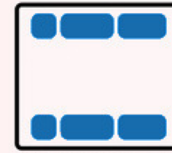
center



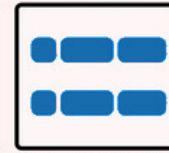
flex-end



stretch



space-between



space-around



# Grid

La propriété `display: grid` permet de créer un conteneur grid.

Quand on applique cette propriété à un élément, tous ses enfants immédiats deviennent des éléments grid.

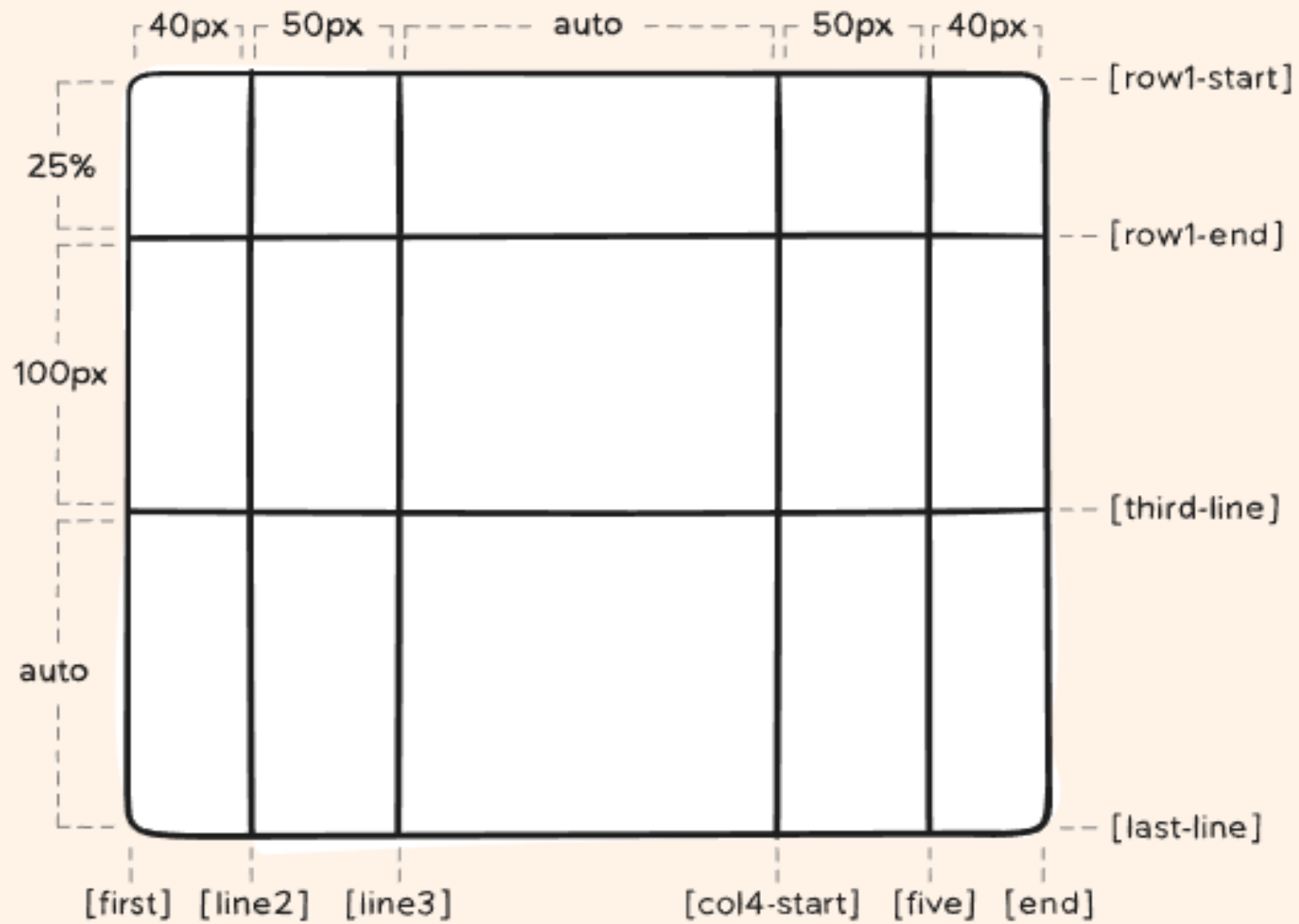
grid permet de créer des layouts en 2 dimensions (ligne et colonne).

nous pouvons définir le nombre de lignes et de colonnes, leur taille, leur positionnement, etc.

voici un exemple de layout en 2 dimensions :



```
.container {  
  grid-template-columns:  
    [first] 40px [line2] 50px  
    [line3] auto [col4-start] 50px  
    [five] 40px [end];  
  grid-template-rows:  
    [row1-start] 25% [row1-end] 100px  
    [third-line] auto [last-line];  
}
```







# Grid-template-columns :

- 📌 `grid-template-columns: 100px;` crée une colonne de 100px
- 📌 `grid-template-columns: 100px 200px;` crée deux colonnes de 100px et 200px
- 📌 `grid-template-columns: 100px 200px 300px;` crée trois colonnes de 100px, 200px et 300px
- 📌 `grid-template-columns: 1fr 2fr 1fr;` crée trois colonnes de 1fr, 2fr et 1fr. 1fr représente une fraction de la largeur restante disponible après avoir retiré le nombre de pixels spécifié pour les autres colonnes.



exemple :

```
.container {  
  display: grid;  
  grid-template-columns: 100px 200px;  
  grid-template-rows: 100px 200px;  
}
```

# Grid-template-rows :

- 📌 `grid-template-rows: 100px;` crée une ligne de 100px
- 📌 `grid-template-rows: 100px 200px;` crée deux lignes de 100px et 200px
- 📌 `grid-template-rows: 100px 200px 300px;` crée trois lignes de 100px, 200px et 300px
- 📌 `grid-template-rows: 1fr 2fr 1fr;` crée trois lignes de 1fr, 2fr et 1fr.  
1fr représente une fraction de la hauteur restante disponible après avoir retiré le nombre de pixels spécifié pour les autres lignes.



exemple :

```
.container {  
  display: grid;  
  grid-template-columns: 100px 200px;  
  grid-template-rows: 100px 200px;  
}
```

# Grid-template-areas :

- 📌 grid-template-areas: "header header header" "main main main" "footer footer footer"; crée trois lignes et trois colonnes, et place les éléments dans les lignes et colonnes correspondantes
- 📌 grid-template-areas: "header header header" ". main ." "footer footer footer"; crée trois lignes et trois colonnes, et place les éléments dans les lignes et colonnes correspondantes. La ligne et la colonne centrale sont vides.

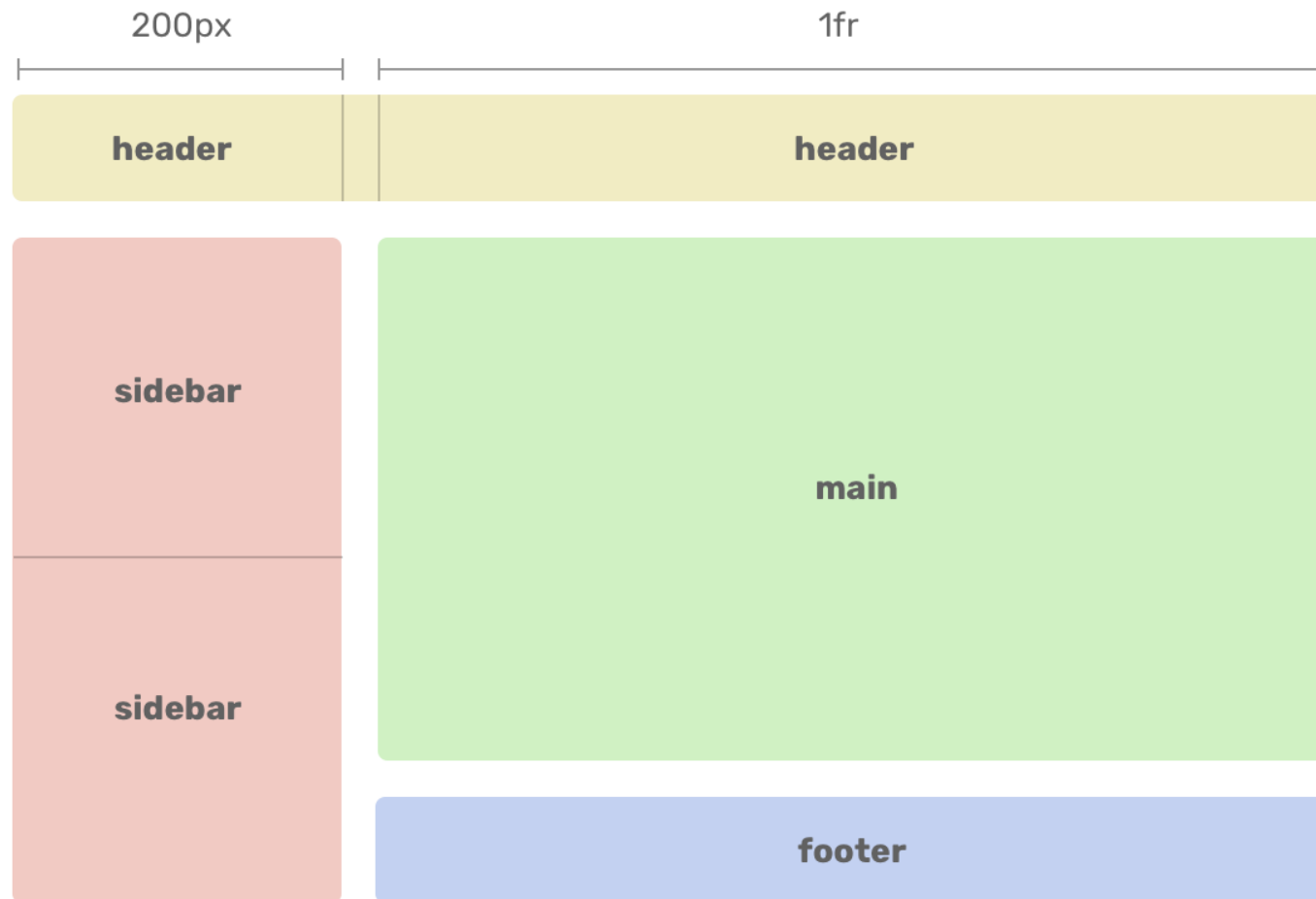


exemple :

```
.container {  
  display: grid;  
  grid-template-columns: 200px 1fr;  
  grid-template-areas: "header header" "sidebar main" "sidebar footer";  
}
```



```
grid-template-columns: 200px 1fr;  
grid-template-areas: "header header"  
                    "sidebar main"  
                    "sidebar footer";
```





# Grid-template :

- 📌 grid-template: 100px 200px / 100px 200px; crée trois lignes et trois colonnes, et place les éléments dans les lignes et colonnes correspondantes.





exemple :

```
.container {  
  display: grid;  
  grid-template: 100px 200px / 100px 200px;  
}
```



# Grid-column-gap :

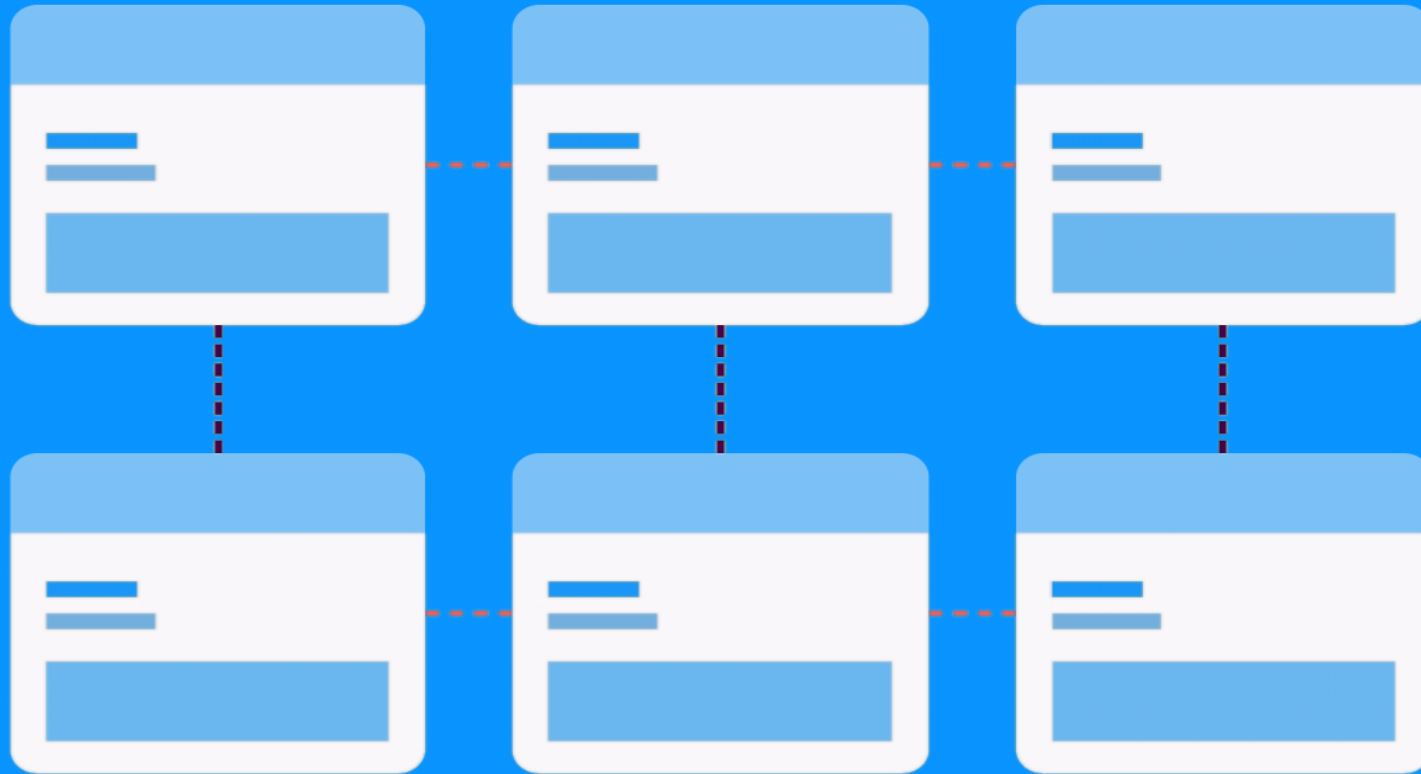
📌 grid-column-gap: 10px; crée un espace de 10px entre les colonnes



exemple :



```
.container {  
  display: grid;  
  grid-template: 100px 200px / 100
```

----- row-gap  
----- column-gap





# Flexbox vs Grid :

-  Flexbox : pour créer des layouts en 1 dimension (ligne ou colonne)
-  Grid : pour créer des layouts en 2 dimensions (ligne et colonne)



Pour aller plus loin :

<http://cssgridgarden.com/#fr>



# Media Queries :

Les media queries permettent de définir des règles CSS spécifiques à un type d'appareil ou à une taille d'écran. Les media queries sont très utiles pour créer des sites web responsives.

exemple :

```
@media screen and (max-width: 600px) {  
    body {  
        background-color: lightblue;  
    }  
}
```

## Desktop



@media screen and  
(min-width: 1024px)  
{...}

## Tablet



@media screen and  
(min-width: 768px) and  
(max-width: 1023px)  
{...}

## Smartphone



@media screen and  
(max-width: 767px)  
{...}





# Pseudo-classes :

Les pseudo-classes permettent de définir un état particulier d'un élément.










Les pseudo-classes sont très utiles pour créer des effets de survol sur des liens, des boutons, etc.

exemple :

```
a:hover {  
    background-color: yellow;  
}
```



# Les erreurs courantes dans les fichiers html et css :

-  Erreur de fermeture de balise
-  Erreur de nom de balise
-  Erreur de nom de classe ou d'id
-  Erreur de sélecteur
-  Erreur de propriété
-  Erreur de valeur
-  Erreur de chemin d'accès
-  Erreur de positionnement
-  Erreur de typo



# Erreur de syntaxe html :

Les erreurs de syntaxe html sont les erreurs les plus courantes. Il est important de vérifier que le code html est valide avant de passer à la suite.

Nous pouvons vérifier la validité du code html en utilisant le validateur w3c : <https://validator.w3.org/>

Ces erreurs entraînent des problèmes d'affichage du site web, ou des problèmes de positionnement des éléments.



# Erreur de syntaxe css :

Les erreurs de syntaxe css aussi tres courantes.

Pour vérifier la validité du code css, nous pouvons utiliser le validateur w3c : <https://jigsaw.w3.org/css-validator/>

Ces erreurs entraînent des problemes d'affichage et de style du site web.

Certaines erreurs peuvent être difficiles à détecter, car elles ne sont pas visibles immédiatement.



# Les erreurs de propriété / valeur :

Ces erreurs se produisent lorsque la propriété ou la valeur n'existe pas.

Ces erreurs peuvent facilement être évitée en regardant la documentation de la propriété, ou en utilisant un validateur css.



# Les erreurs de chemin d'accès :

Ces erreurs se produisent lorsque le chemin d'accès vers un fichier est incorrect.

Ces erreurs peuvent être évitées en vérifiant que le chemin d'accès de chaque fichier est correct.

Une bonne pratique est de placer tous les fichiers dans un dossier nommé "ressources".



# Les outils pour créer un site web :

- 📌 Un éditeur de texte (Visual Studio Code, Sublime Text, Notepad++, etc.)
- 📌 Un navigateur internet (Google Chrome, Firefox, Safari, etc.)
- 📌 Un validateur html (<https://validator.w3.org/>)
- 📌 Un validateur css (<https://jigsaw.w3.org/css-validator/>)
- 📌 Un préprocesseur css (Sass, Less, Stylus, etc.)



# Les erreurs d'image html :

Ces erreurs se produisent lorsque le chemin d'accès vers une image est incorrect.

Nous pouvons éviter ces erreurs en utilisant la propriété "alt" pour décrire l'image.

Cette propriété pourrait être utilisée par les moteurs de recherche pour décrire l'image si elle n'est pas affichée.





# Les erreurs de colorimétrie :

Ces erreurs se produisent lorsque la couleur n'est pas correctement définie.

Les sites web sont affichés sur des écrans de différentes couleurs, et les couleurs peuvent être différentes d'un navigateur à l'autre.

Une bonne pratique est d'utiliser des couleurs nommées pour les couleurs de base, et des couleurs hexadécimales pour les couleurs personnalisées.

Des outils pour éviter ces erreurs seront [colorpicker.com](https://colorpicker.com), et <https://colorhunt.co/>.



# Les erreurs de version de navigateur:

Ces erreurs se produisent lorsque le code html ou css n'est pas compatible avec une version de navigateur.

Ces erreurs peuvent être évitées en utilisant des préfixes pour les propriétés, et en utilisant des outils comme [caniuse.com](https://caniuse.com) pour vérifier la compatibilité des propriétés.



# Les erreurs d'inattention :

Ces erreurs se produisent lorsque nous ne faisons pas attention à certains détails.

Comme par exemple, oublier de fermer une balise, ou de fermer une parenthèse, une accolade, ou une guillemet.

```
  
<p>Mon texte
```

# CSS

**Merci pour votre  
attention !!**

