

CSS- 04 - Modèles boîtes (suite)- Grid et Flex





Flex

Contrairement aux autres valeurs de [display](#) qui influencent uniquement l'affichage des éléments par rapport aux autres dans la page, la propriété `display: flex;` ou `display: inline-flex;` influence aussi l'affichage de ses enfants en les positionnant dans un corridor sur l'axe des X ou Y, en modifiant leur dimension, leur ordre, etc. afin de remplir l'espace disponible le plus adéquatement possible.

flex-direction

Comme son nom l'indique, la valeur de cette propriété définit la direction qu'auront ses enfants.

Valeurs possibles:

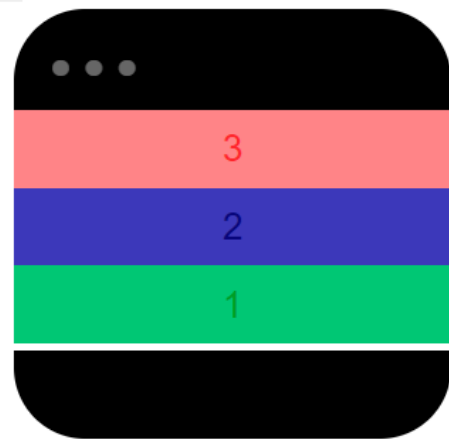
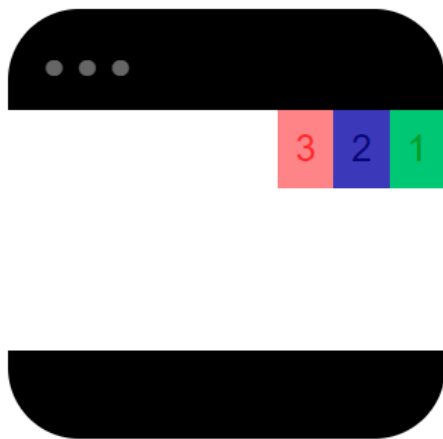
- row  (*par défaut*)
- row-reverse 
- column 
- column-reverse 

flex-direction: row vs column



À gauche, `flex-direction: row;`
À droite, `flex-direction: column;`

flex-direction: row-reverse vs column-reverse



À gauche, flex-direction: row-reverse;
À droite, flex-direction: column-reverse;

- [flex-direction](#)
- [flex-direction](#)

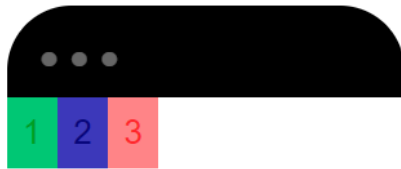
justify-content

Un peu comme Word ou Google Doc, flexbox vous permet de justifier votre contenu horizontalement ↔ afin d'atteindre l'affichage désiré.

Possibilités:

- flex-start (*par défaut*)
- flex-end
- center
- space-between
- space-around
- space-evenly
- etc.

justify-content: flex-start vs flex-end



À gauche, justify-content: flex-start;

À droite, justify-content: flex-end;

justify-content: center vs space-between



À gauche, justify-content: center;
À droite, justify-content: space-between;

justify-content: space-around vs space-evenly



À gauche, justify-content: space-around;
À droite, justify-content: space-evenly;

Ces deux valeurs peuvent se ressembler dans un espace restreint, mais remarquer comment avec justify-content: space-around; chaque élément a un espace équivalent à gauche et à droite et comment l'espace de chacun des éléments s'additionne.



Tandis qu'en justify-content: space-evenly; l'espace n'est pas additionné, mais plutôt jumelé afin de créer des espaces identiques entre chaque élément.



align-items

Dans la même lignée que [justify-content](#), mais cette fois à la verticale ⇅.

Possibilités:

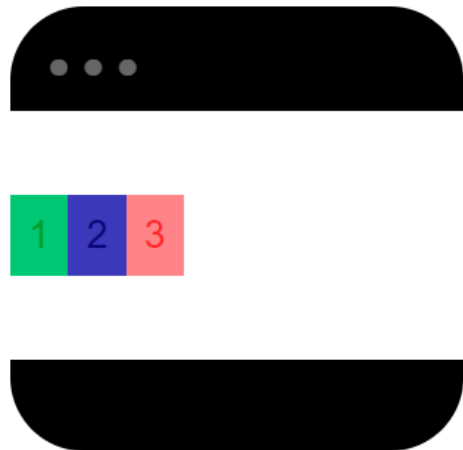
- stretch (*par défaut*)
- flex-start
- flex-end
- center
- etc.

align-items: stretch vs center



À gauche, align-items: stretch;

À droite, align-items: center;

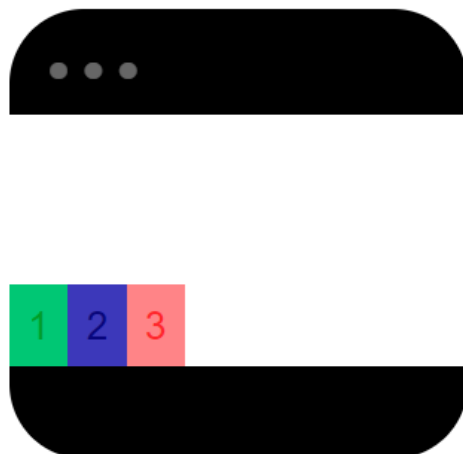


align-items: flex-start vs flex-end



À gauche, align-items: flex-start;

À gauche, align-items: flex-end;



- [align-items](#)
- [align-items](#)

justify-content et align-items sont l'équivalent de brancher une clé USB en CSS. Deux ou trois tentatives sont souvent requises avant d'obtenir le résultat espéré.

align-self

La propriété align-self est pratiquement identique à la propriété [align-items](#) à la différence qu'elle s'applique sur un élément en particulier, **et non le parent**, afin de l'aligner de façon différente aux autres.

Elle accepte aussi les valeurs:

- stretch
 - flex-start
 - flex-end
 - center
 - etc.
- [align-self](#)
 - [align-self](#)

Inspecteur

L'inspecteur est d'une aide précieuse lorsque l'on manipule les éléments flexbox. Dans le DOM tree, les éléments en display: flex; ou display: inline-flex; sont mis en évidence grâce à un badge contenant le mot "*flex*". Lorsque cliqué, ce badge met en évidence la zone prise dans la page par le flexbox en question.

Lorsque l'on examine les propriétés CSS appliquées à cet élément on remarque à la droite de la propriété display un icône rappelant une grille. Lorsque cliqué, cet icône affiche différentes propriétés en lien avec flexbox, par exemple: flex-direction, justify-content, align-items, etc. Les valeurs

disponibles pour ces propriétés sont illustrées avec des icônes permettant d'identifier ou de tester rapidement la valeur souhaitée.

Voir fichier *flex-inspecteur.mp4*

Exercice

Réaliser les 24 niveaux sur le site <https://flexboxfroggy.com/#fr>

Grid

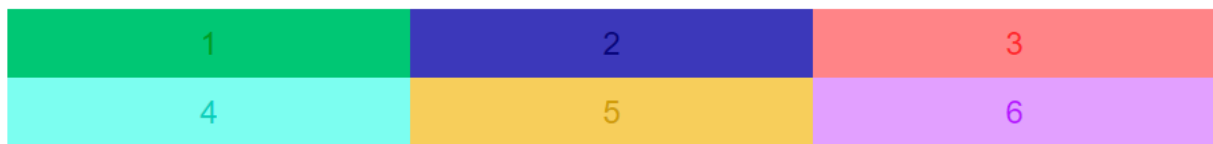
Tout comme [flexbox](#), la propriété `display: grid;`, ou sa variante `display: inline-grid;`, influence l'affichage de ses enfants. Cependant, contrairement à flexbox qui les positionne en fonction d'une seule dimension (*x ou y*), grid permet de les positionner sur une grille quadrillée en deux dimensions (*x & y*).

grid-template-columns

Permet de définir indépendamment la taille de chaque colonne d'une grille. Le nombre de colonnes correspond au nombre de valeurs spécifiées.

Par exemple, trois valeurs produisent trois colonnes:

```
.grid {  
  display: grid;  
  grid-template-columns: auto auto auto ;  
}
```



Puisqu'il y a plus d'éléments que de colonnes, une 2e rangée est automatiquement créée afin d'accueillir tous les items. La première rangée est donc explicite tandis que la deuxième est implicite.

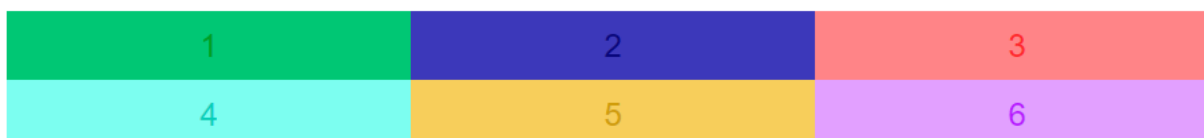
- [grid-template-columns](#)
- [grid-template-columns](#)

grid-template-rows

Permet de définir indépendamment la taille de chaque rangée d'une grille. Le nombre de rangées correspond au nombre de valeurs spécifiées.

Par exemple, deux valeurs produisent deux rangées:

```
.grid {  
  display: grid;  
  grid-template-columns: auto auto auto;  
  grid-template-rows: auto auto;  
}
```



Il aurait été possible de créer plus de rangées. Cependant, ces rangées auraient été vides puisqu'il n'y a seulement assez d'éléments pour combler deux rangées de trois colonnes.

- [grid-template-rows](#)
- [grid-template-rows](#)

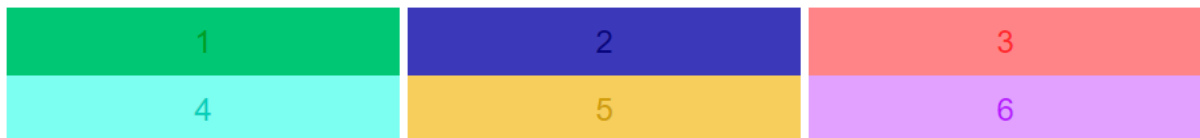
column-gap, row-gap & gap

Les propriétés column-gap, row-gap et gap permettent de définir des espaces entre les cellules d'une grille. Ces propriétés acceptent [toutes les unités de base](#).

column-gap

La propriété column-gap permet de définir l'espace entre les colonnes d'une grille.

```
.grid {  
  display: grid;  
  grid-template-columns: auto auto auto;  
  column-gap: 5px;  
}
```

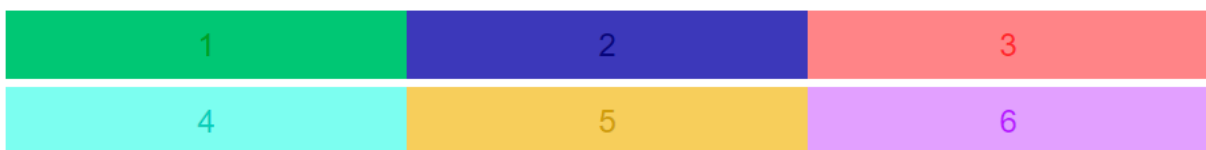


- [column-gap](#)
- [column-gap](#)

row-gap

La propriété row-gap permet de définir l'espace entre les rangées d'une grille.

```
.grid {  
  display: grid;  
  grid-template-columns: auto auto auto;  
  row-gap: 5px;  
}
```

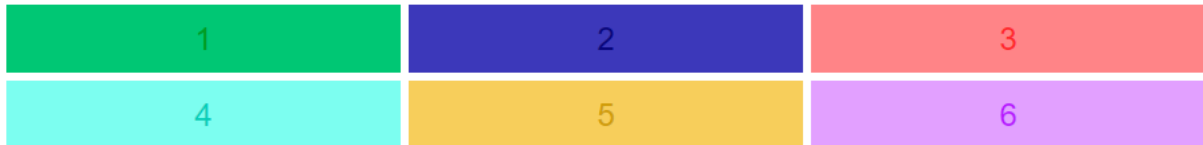


- [row-gap](#)
- [row-gap](#)

gap

La propriété gap permet de définir l'espace entre les colonnes et rangées d'une grille simultanément.

```
.grid {  
  display: grid;  
  grid-template-columns: auto auto auto;  
  gap: 5px;  
}
```



Cette propriété accepte aussi de recevoir deux valeurs. Le cas échéant, la première correspond à l'espace entre les rangées et la deuxième, à celle entre les colonnes.

Par exemple:

gap: 10px 50px;

Génère un espace de 10px entre les rangées ↑ et de 50px entre les colonnes ↔.

Pratiquement toutes les unités, **sauf les fr**, peuvent être utilisées pour la propriété de type gap.

- [gap](#)
- [gap](#)

fr

Afin de simplifier la gestion des colonnes et des rangées une nouvelle unité CSS vue le jour. Cette unité, intitulée fr pour *fraction*, permet de distribuer l'espace disponible de façon relative entre chaque élément ayant une valeur de ce type.

Par exemple, pour avoir trois colonnes identiques, il serait possible de faire:

```
grid-template-columns: 1fr 1fr 1fr;
```

Combinaison avec gap

À priori, cette unité peut sembler similaire aux pourcentages (%). Cependant, puisque les fractions basent leurs calculs sur l'espace disponible et non l'espace total de leur parent, ils peuvent-être utiliser avantageusement avec les propriétés de type gap.

Par exemple, si un column-gap: 5px est présent sur des éléments en pourcentages à gauche versus en fractions à droite.

```
.grid {
  display: grid;
  column-gap: 5px;
}

.frame.no1 .grid {
  grid-template-columns: 33.3% 33.3% 33.3%;
}


.frame.no2 .grid {
  grid-template-columns: 1fr 1fr 1fr;
}
```



The diagram illustrates the difference in spacing between two grid layouts. On the left, a grid with three columns defined by percentages (33.3%, 33.3%, 33.3%) and a column-gap of 5px is shown. The columns are colored green, blue, and red, and the gaps are visible. On the right, a similar grid is shown with columns defined by fractions (1fr, 1fr, 1fr) and a column-gap of 5px. The columns are also colored green, blue, and red, and the gaps are visible. The visual result shows that the fraction-based layout maintains consistent proportions across different container widths, while the percentage-based layout does not.

Combinaison avec unités tierce

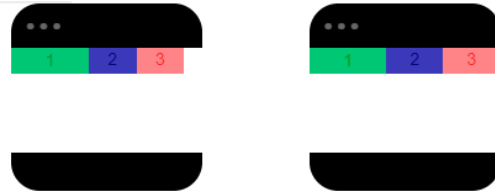
Lorsqu'une unité tierce est introduite, les pourcentages continuent de se baser sur la dimension totale du parent, peu importe la dimension prise par cette unité, contrairement aux fractions qui se séparent l'espace disponible restant après que l'unité tierce est prise son espace.

Dans les exemples ci-dessous , la première colonne à une unité tierce de 50px. À gauche, on remarque que la combinaison avec des pourcentages produit un résultat indésirable, tandis qu'à droite, les fractions séparent également l'espace restant, produisant ainsi un résultat harmonieux.

```
.grid { display: grid; }

.frame.no1 .grid {
  grid-template-columns: 50px 25% 25%;
}

.frame.no2 .grid {
  grid-template-columns: 50px 1fr 1fr;
}
```



minmax()

Comme son nom l'indique, l'unité `minmax()` permet de définir une dimension minimale et maximale à une cellule de grille. Cette unité est particulièrement utile afin de créer une mise en page responsive tout en évitant que certains éléments se retrouve trop coincés.

Par exemple, deux grilles identiques avec toutes les cellule d'une largeur de `1fr` à l'exception de la 2^e cellule verte ayant une valeur de `minmax(200px, 1fr)`.

Voir vidéo *grid-minmax.mp4*

- [minmax\(\)](#)

repeat

Spécifier individuellement chaque colonne ou rangée n'est pas un problème lorsqu'une grille est de dimension relativement modeste. Cependant, lorsqu'une grille grossie, il peut devenir rapidement lassant et mélangeant d'écrire la dimension de chaque colonne ou rangée, surtout si celle-ci est identique.

Par exemple:

`grid-template-columns: 1fr 1fr 1fr 1fr 1fr 1fr;`

Pour palier à cet enjeu, la commande `repeat()` fut créée.

L'exemple précédent pourrait donc être réécrit ainsi:

`grid-template-columns: repeat(6, 1fr);`

auto-fit & auto-fill

Afin de pouvoir réaliser une grille responsive sans avoir à écrire une multitude de [media queries](#), il est possible d'utiliser les valeurs auto-fit et auto-fill à la place d'un nombre spécifique de colonnes dans un repeat().

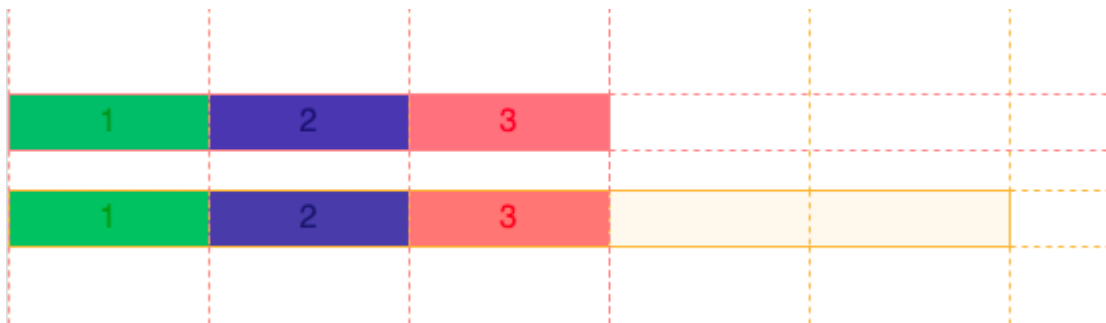
Par exemple avec auto-fit,


```
grid-template-columns: repeat(auto-fit, 150px);
```


Permet d'afficher autant d'éléments sur une rangée qu'il y a d'espace disponible.

Voir vidéo *grid-repeat-fit-content.mp4*

La différence entre auto-fit et auto-fill étant la gestion de l'espace vide restant. Avec auto-fit, aucune cellule vide supplémentaire n'est ajoutée dans la grille, même si l'espace le permet, alors qu'avec auto-fill des cellules vides sont créées. Dans la majorité des cas, le résultat sera similaire. Cependant, cette particularité peut parfois s'avérer utile lorsque combinée avec d'autres propriétés de grille.



En haut , repeat(auto-fit, ...) aucune cellule vide de créée.

En bas , repeat(auto-fill, ...) 2 cellules vides de créées.

Combinaison avec d'autres unités

Il est aussi possible de combiner `repeat()` avec d'autres unités.

Par exemple:

```
.grid {  
  display: grid;  
  grid-template-columns: 30px repeat(2, 1fr);  
}
```



- [repeat\(\)](#)

Inspecteur

L'inspecteur est d'une aide précieuse lorsque l'on manipule les éléments en grid. Dans le DOM tree, les éléments en `display: grid`; ou `display: inline-grid`; sont mis en évidence grâce à un badge contenant le mot "*grid*". Lorsque cliqué, ce badge met en surbrillance les divisions constituant la grille en question.

Dans l'onglet *layout*, il est possible d'afficher des informations supplémentaires. Notamment, étendre les divisions afin de les rendre plus visibles, afficher la taille des colonnes et rangées, etc.

Voir fichier *grid-inspector-layout.mp4*

Exercice

Réaliser les 28 niveaux sur le site <https://cssgridgarden.com/#fr>

Sources

<https://smnarnold.com/>

CSS 04 suite

Réalisation : Guillaume DELACROIX Formateur AFPA

11 janvier 2023

Afpa 