

# CSS Grid Layout, guide complet

## Introduction

CSS Grid Layout (aka “Grid”) est un système de layout bi-dimensionnel basé sur les grilles qui a pour ambition ni plus ni moins que de révolutionner la façon dont nous concevons les interfaces utilisateurs basées sur des grilles.

CSS a toujours été utilisé pour la mise en page web, mais il n’a jamais été très bon pour cela. Nous avons d’abord utilisé des tables, puis des *floats*, divers positionnements et inline-block, mais toutes ces méthodes s’apparentent fondamentalement à des rustines, et ne permettaient pas de résoudre certains problèmes récurrents comme le centrage vertical. Flexbox nous a bien aidés, mais il est conçu pour des layouts plus simples et uni-dimensionnels, et non pour des layouts bi-dimensionnels. En fait, Flexbox et Grid se complètent bien et sont faits pour travailler ensemble. Grid est le premier module CSS créé spécifiquement pour résoudre les problèmes de layout que nous avons contournés par des tripatouillages depuis que nous réalisons des sites web.

## Les bases et la compatibilité

Il est facile de se lancer dans Grid. Il suffit de définir un élément container comme une grille, via la propriété `display:grid`, de régler les dimensions des colonnes et des rangées avec `grid-template-columns` et `grid-template-rows` et de placer ses éléments enfants dans la grille avec `grid-column` et `grid-row`. Comme pour Flexbox, l’ordre des items de la grille tel qu’il apparaît dans la source n’a pas d’importance. Votre CSS peut les placer dans n’importe quel ordre, ce qui facilite la réorganisation de votre grille avec les media queries. Vous pouvez ainsi définir le layout de votre page et le réorganiser entièrement pour l’adapter à différentes tailles

d'écrans, et tout cela avec juste quelques lignes de CSS. Grid est un des modules CSS les plus puissants jamais proposés.

Une chose importante à noter toutefois : Grid n'est pas encore tout à fait prêt à être utilisé en production même si depuis mars 2017 CSS Grid est supporté par Firefox, Chrome, Opera et Safari. Pour les détails complets et à jour de la compatibilité, voir [Can I Use](#).

## Terminologie importante

Avant de plonger dans les concepts de Grid, il est important de bien comprendre la terminologie. Dans la mesure où les termes utilisés sont assez similaires, il est facile de les confondre si vous n'avez pas bien mémorisé leur signification telle qu'elle est définie par la spécification. Mais ne vous inquiétez pas, il n'y en a pas beaucoup.

### Container Grid

C'est l'élément sur lequel s'applique `display: grid`. C'est donc le parent direct de tous les items grid. Dans cet exemple, `container` est le container grid.

```
/HTML
<div class="container">
  <div class="item item-1"></div>
  <div class="item item-2"></div>
  <div class="item item-3"></div>
</div>
```

## Item Grid

L'enfant (c'est à dire le descendant direct) du container grid. Ici, les éléments `item` sont des items grid, mais ce n'est pas le cas de `sub-item`.

```
/HTML
<div class="container">
  <div class="item"></div>
  <div class="item">
    <p class="sub-item"></p>
  </div>
  <div class="item"></div>
</div>
```

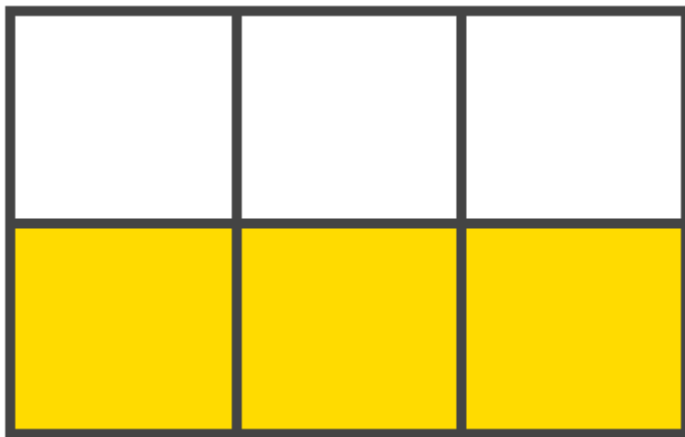
## Ligne de Grille

Les lignes qui divisent et constituent la structure de la grille. Elles peuvent être verticales (“lignes de grille de colonnes”) ou horizontales (“lignes de grille de rangées”) et sont situées d'un côté ou de l'autre d'une rangée ou d'une colonne. Ici, la ligne jaune est un exemple de ligne de grille de colonne (*column grid line*).



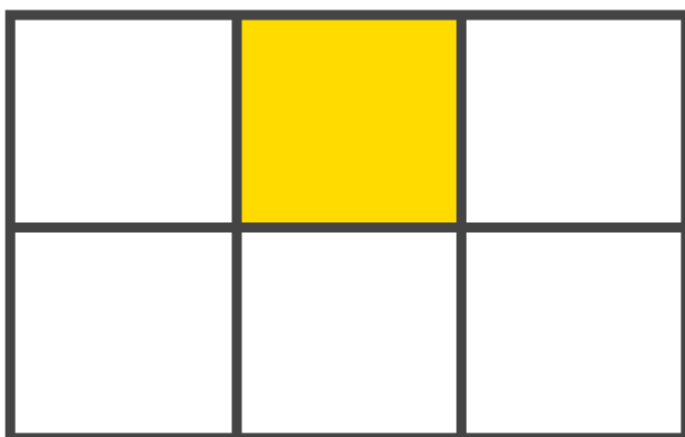
## Piste de Grille

La piste (ou plage) de grille (*grid track*) est l'espace situé entre deux lignes de grille adjacentes, en d'autres termes ce sont les colonnes ou les rangées de la grille. Ci-dessous, la piste de grille est située entre les deuxième et troisième lignes de rangée.



## Cellule de Grille

La cellule de grille (*grid cell*) est l'espace situé entre deux lignes de grille adjacentes de rangée et deux lignes de grille adjacentes de colonne. C'est une "unité" de la grille. Ci-dessous, la cellule de grille est entre les lignes de grille de rangée 1 & 2, et les lignes de grille de colonne 2 & 3. Dit autrement, une cellule est l'intersection d'une rangée et d'une colonne.



## Zone de Grille

La zone de grille (*grid area*) est l'espace entouré par quatre lignes de grille. Une zone de grille peut comprendre n'importe quel nombre de cellules. Voici la zone de grille entre les lignes de grille de rangée 1 & 3 et les lignes de grille de colonne 1 & 3.



## Les Propriétés Grid

Nous allons voir d'abord les propriétés du parent (le grid container), puis celles des enfants (les grid items).

- Propriétés du Parent (Grid Container)
- Propriétés des Enfants (Grid Items)

---

### Propriétés du Parent (Grid Container)

#### Table des matières

- display
- grid-template-columns
- grid-template-rows
- grid-template-areas
- grid-column-gap
- grid-row-gap
- grid-gap
- justify-items
- align-items
- justify-content
- align-content
- grid-auto-columns
- grid-auto-rows
- grid-auto-flow
- grid

## display

Définit l'élément en tant que container grid et établit un *nouveau contexte de formatage* de son contenu.

Valeurs :

- `grid` - génère une grille de niveau bloc
- `inline-grid` - génère une grille de niveau inline.

```
.container{  
  display: grid | inline-grid  
}
```

Remarque : `column`, `float`, `clear` et `vertical-align` n'ont aucun effet sur un container grid.

## grid-template columns, grid-template-rows

Définissent les colonnes et rangées de la grille via une liste de valeurs séparées par un espace. Les valeurs représentent la dimension de la piste (*track*) et l'espace entre les pistes représente la ligne de grille.

Valeurs :

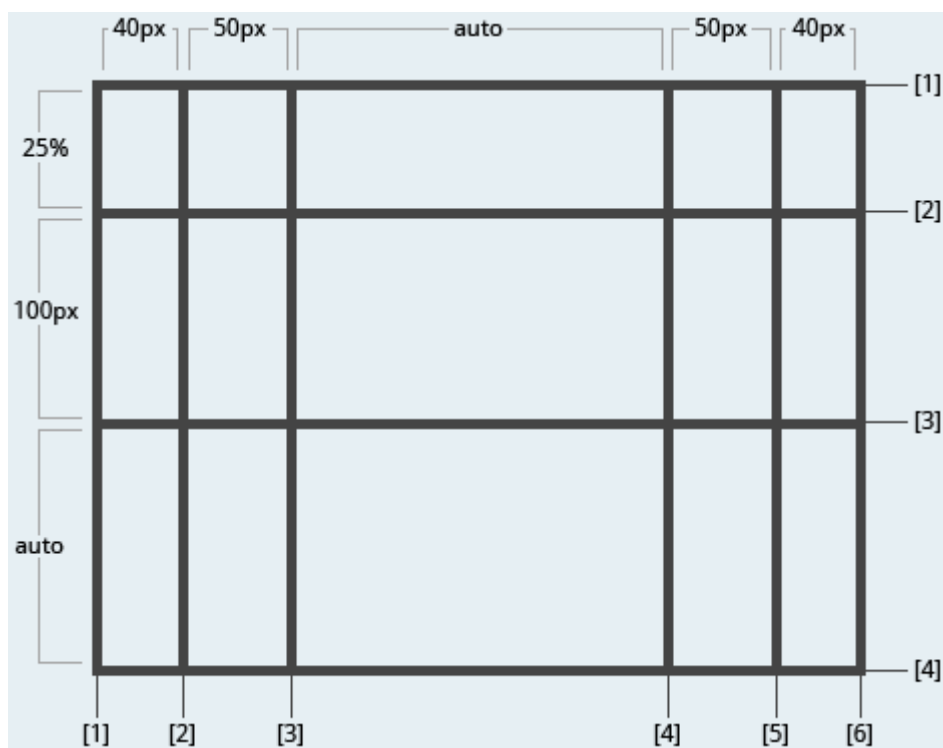
- `<track-size>` - peut être une longueur, un pourcentage, ou une fraction de l'espace libre dans la grille (via l'unité `fr`)
- `<line-name>` - un nom arbitraire choisi par vous
- `subgrid` - si votre container grid est lui-même un item grid (autrement dit, nous avons des grilles imbriquées), vous pouvez utiliser cette propriété pour indiquer que vous voulez que les dimensions de ses rangées et colonnes soient héritées de l'élément parent plutôt que d'en donner des spécifiques.

```
.container{  
  grid-template-columns: <track-size> ... | <line-name>  
<track-size> ... | subgrid;  
  grid-template-rows: <track-size> ... | <line-name>  
<track-size> ... | subgrid;  
}
```

Exemples :

Lorsque vous laissez un espace vide entre les valeurs de pistes (*track*), les lignes de grille se voient automatiquement attribuer des numéros :

```
.container{  
  grid-template-columns: 40px 50px auto 50px 40px;  
  grid-template-rows: 25% 100px auto;  
}
```

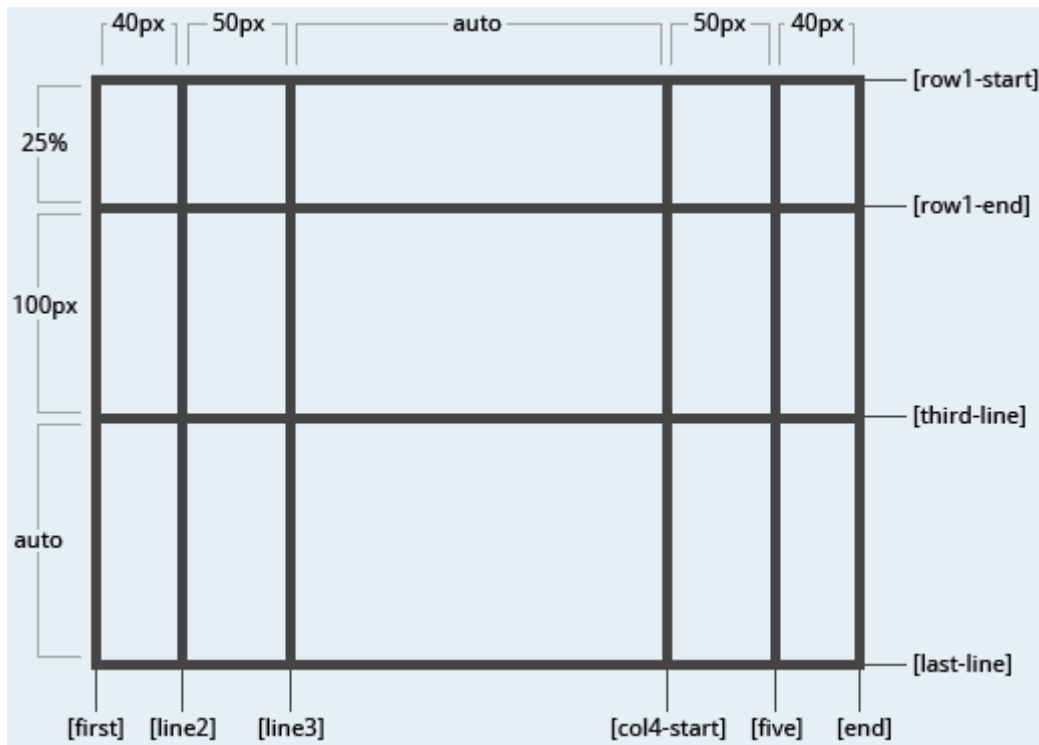


Mais vous pouvez choisir de nommer explicitement les lignes. Remarquez la syntaxe entre crochets pour les noms de lignes :

```
.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];  
}
```



}



Notez bien qu'une même ligne peut avoir plus d'un nom. Par exemple ci-dessous la deuxième ligne a deux noms, row1-end et row2-start :

```
.container{  
  grid-template-rows: [row1-start] 25% [row1-end row2-start]  
  25% [row2-end];  
}
```

Si votre définition contient des parties qui se répètent, vous pouvez utiliser la notation `repeat()` pour alléger le code :

```
.container{  
  grid-template-columns: repeat(3, 20px [col-start]) 5%;  
}
```

...ce qui est équivalent à :

```
.container{  
  grid-template-columns: 20px [col-start] 20px [col-start] 20px  
  [col-start] 5%;  
}
```

L'unité `fr` vous permet de spécifier la dimension d'une piste comme une fraction de l'espace libre du container grid. Par exemple, ce code donnera à chaque item la dimension d'1/3 du container :

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

L'espace libre est calculé après prise en compte des items non-flexibles. Dans l'exemple qui suit, la quantité d'espace libre disponible pour les unités `fr` est l'espace total déduction faite des 50px.

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

## grid-template-areas

Définit un template de grille en référençant les noms des zones de grille (*grid areas*) spécifiées par la propriété `grid-area`. Si l'on répète le nom d'une zone de grille, cela étend la surface couverte par ces cellules. Un point (.) signifie que la cellule est vide. La syntaxe elle-même fournit une visualisation de la structure de la grille.

Valeurs :

- `<grid-area-name>` - le nom de la zone de grille spécifié avec `grid-area`
- `.` - un point indique une cellule vide
- `none` - aucune zone de grille n'est définie

```
.container{  
  grid-template-areas: "<grid-area-name> | . | none | ..."  
                      "..."  
}
```

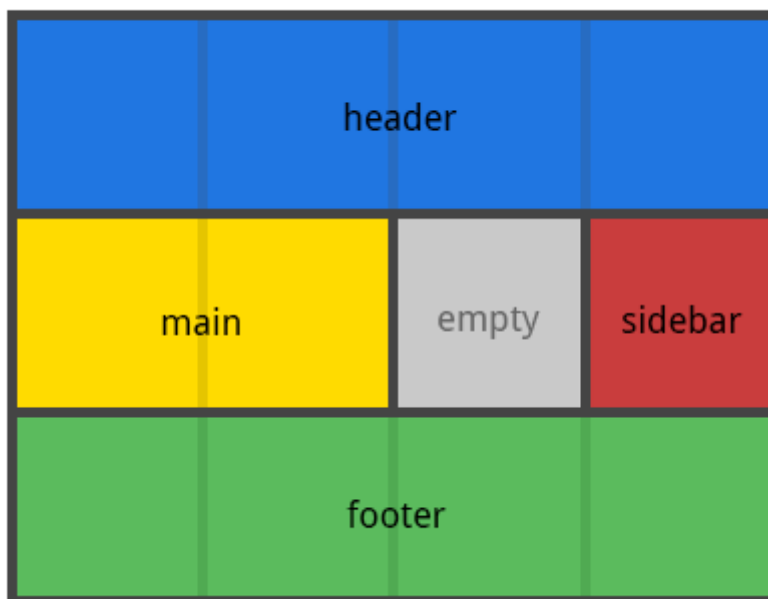
Exemple :

```
.item-a{  
  grid-area: header;  
}  
.item-b{  
  grid-area: main;  
}  
.item-c{  
  grid-area: sidebar;  
}  
.item-d{  
  grid-area: footer;  
}
```

```
.container{  
  grid-template-columns: 50px 50px 50px 50px;  
  grid-template-rows: auto;  
  grid-template-areas: "header header header header"  
                      "main main . sidebar"  
                      "footer footer footer footer"
```

}

On crée ainsi une grille de quatre colonnes sur trois rangées. La rangée supérieure tout entière constitue le header, la rangée du milieu comprend deux zones principales, une cellule vide et une zone de sidebar. La dernière rangée est le footer.



Chaque rangée de notre déclaration doit avoir le même nombre de cellules.

Vous pouvez utiliser autant de `.` points que vous voulez pour déclarer une cellule vide. Du moment que les points ne sont pas séparés par un espace, ils représentent une cellule unique.

Remarquez qu'avec cette syntaxe vous ne nommez pas les lignes mais les zones. Quand vous utilisez cette syntaxe, les lignes de chaque côté des zones sont nommées automatiquement. Si le nom de votre zone de grille est `foo`, alors le nom de la ligne de rangée (et de colonne) au début de la zone sera `foo-start` et à l'autre bout `foo-end`. En conséquence, certaines lignes pourront avoir plusieurs noms, par exemple dans notre illustration

précédente, la ligne verticale tout à gauche aura trois noms : header-start, main-start et footer-start.

## grid-column-gap et grid-row-gap

Spécifie la dimension des lignes de grille. On peut les voir comme des largeurs de gouttières entre les colonnes ou les rangées.

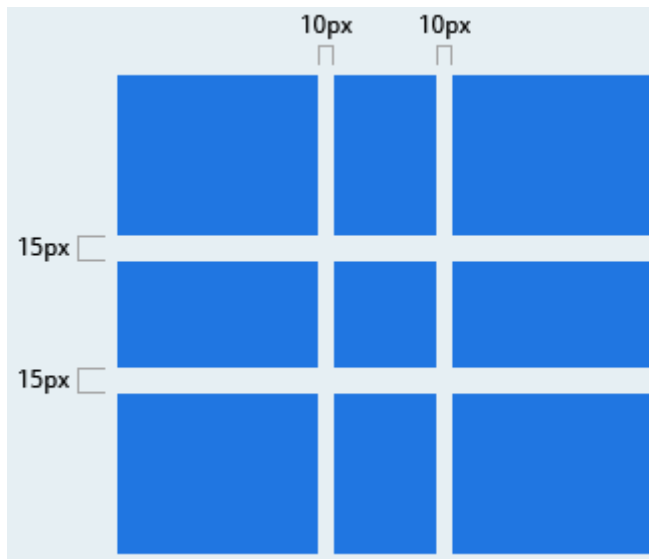
Valeurs :

- `<line-size>` - une longueur

```
.container{  
  grid-column-gap: <line-size>;  
  grid-row-gap: <line-size>;  
}
```

Exemple :

```
.container{  
  grid-template-columns: 100px 50px 100px;  
  grid-template-rows: 80px auto 80px;  
  grid-column-gap: 10px;  
  grid-row-gap: 15px;  
}
```



Les gouttières sont créées *uniquement* entre les colonnes ou les rangées, pas autour.

## grid-gap

C'est un raccourci de `grid-column-gap` et `grid-row-gap`.

Valeurs : - `<grid-column-gap> <grid-row-gap>` - longueur

```
.container{  
  grid-gap: <grid-column-gap> <grid-row-gap>;  
}
```

Exemple :

```
.container{  
  grid-template-columns: 100px 50px 100px;  
  grid-template-rows: 80px auto 80px;  
  grid-gap: 10px 15px;  
}
```

Si aucun `grid-row-gap` n'est spécifié, il prend la même valeur que `grid-column-gap`.

## justify-items

Aligne le contenu à l'intérieur d'un item grille, le long de l'axe des colonnes (par opposition à `align-items` qui les aligne le long de l'axe des rangées). Cette valeur s'applique à tous les items grid à l'intérieur du container.

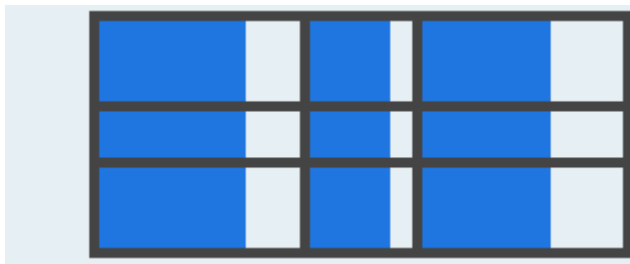
Valeurs :

- `start` - aligne le contenu à partir de la gauche de la zone de grille.
- `end` - aligne le contenu en partant de la droite de la zone de grille
- `center` - aligne le contenu au centre de la zone de grille
- `stretch` - remplit toute la largeur de la zone de grille (c'est la valeur par défaut).

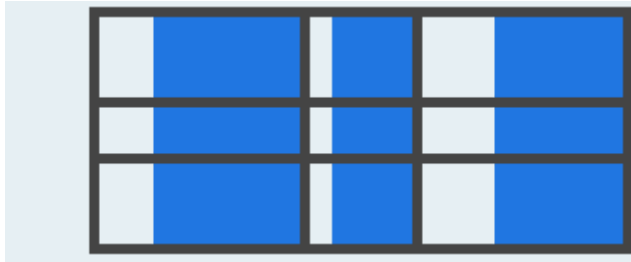
```
.container{  
  justify-items: start | end | center | stretch;  
}
```

Exemples :

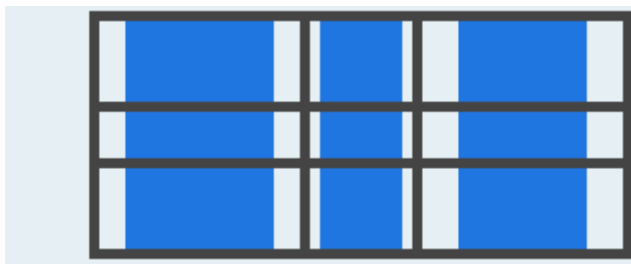
```
.container{  
  justify-items: start;  
}
```



```
.container{  
  justify-items: end;  
}
```



```
.container{  
  justify-items: center;  
}
```



```
.container{  
  justify-items: stretch;  
}
```





On peut également spécifier ce comportement pour des items de grille *individuels* via la propriété `justify-self`.

## align-items

Aligne le contenu situé à l'intérieur d'un item de grille le long de l'axe des rangées (par opposition à `justify-items` qui l'aligne le long de l'axe des colonnes). Cette valeur s'applique à tous les items grid à l'intérieur du container.

Valeurs :

- `start` - aligne le contenu à partir du sommet de la zone de grille.
- `end` - aligne le contenu en partant du bas de la zone de grille
- `center` - aligne le contenu au centre de la zone de grille
- `stretch` - remplit toute la hauteur de la zone de grille (c'est la valeur par défaut).

```
.container{  
  align-items: start | end | center | stretch;  
}
```

Exemples :

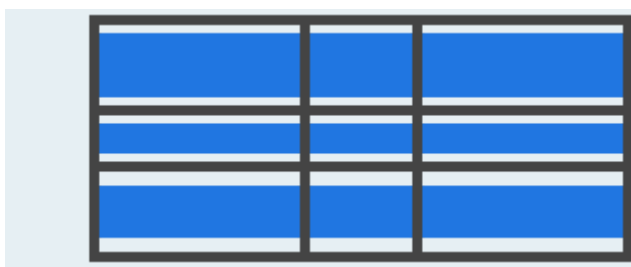
```
.container{  
  align-items: start;  
}
```



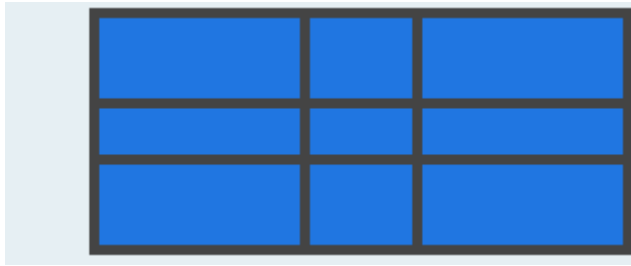
```
.container{  
  align-items: end;  
}
```



```
.container{  
  align-items: center;  
}
```



```
.container{  
  align-items: stretch;  
}
```



On peut également spécifier ce comportement pour des items de grille *individuels* via la propriété `align-self`.

## justify-content

Parfois la dimension totale de votre grille semblera inférieure à la dimension de son container grid. Cela peut arriver si tous les items grid sont dimensionnés avec des unités non-flexibles comme `px`. Dans ce cas, vous pouvez régler l'alignement de la grille à l'intérieur du container. Cette propriété aligne la grille le long de l'axe des colonnes (par opposition à `align-content` qui aligne la grille le long de l'axe des rangées).

Valeurs :

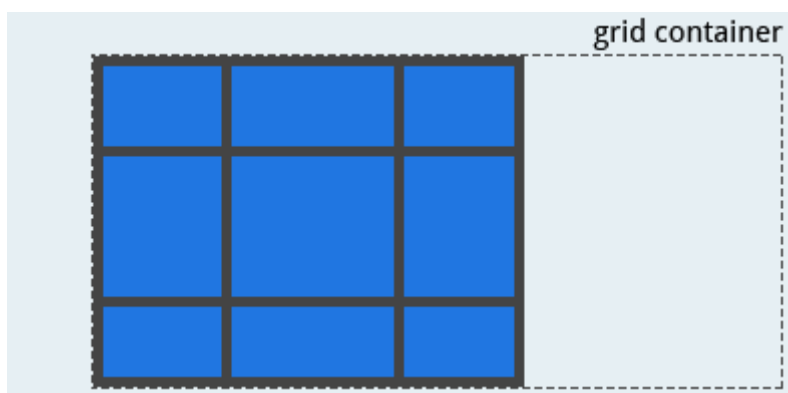
- `start` - aligne la grille à partir de la gauche de la grille container (*container grid*).
- `end` - aligne la grille à partir de la droite de la grille container.
- `center` - aligne la grille au centre de la grille container
- `stretch` - redimensionne les items pour permettre à la grille de remplir toute la largeur de la grille container.
- `space-around` - place un espace égal entre chaque item de grille, et un demi-espace aux extrémités.
- `space-between` - place un espace égal entre chaque item de grille, et aucun espace aux extrémités.
- `space-evenly` - place un espace égal entre chaque item de grille, y compris aux extrémités.

```
.container{  
  justify-content: start | end | center | stretch | space-around |
```

```
space-between | space-evenly;  
}
```

Exemples :

```
.container{  
  justify-content: start;  
}
```

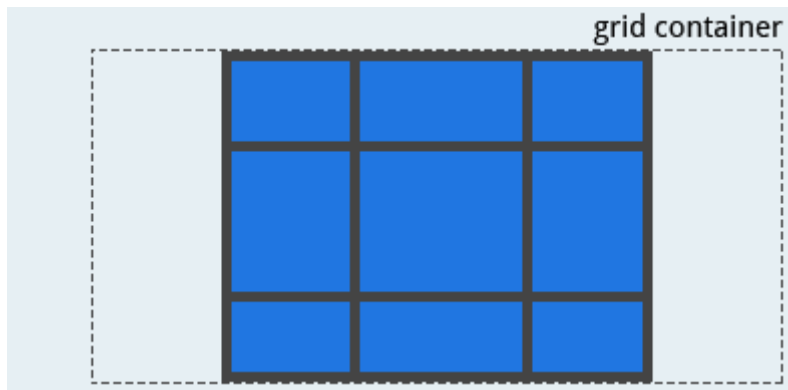


```
.container{  
  justify-content: end;  
}
```

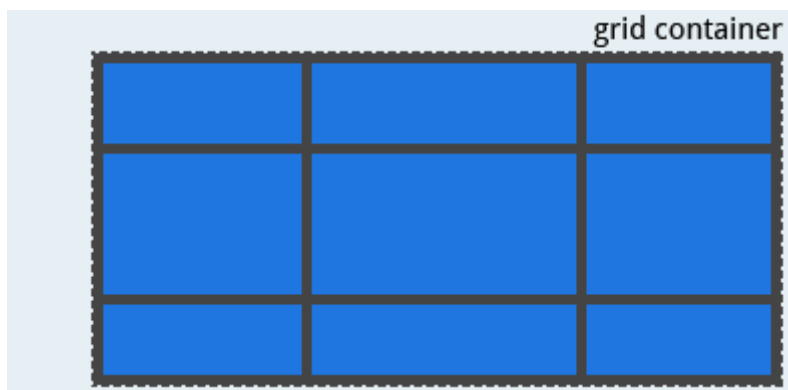


```
.container{  
  justify-content: center;  
}
```

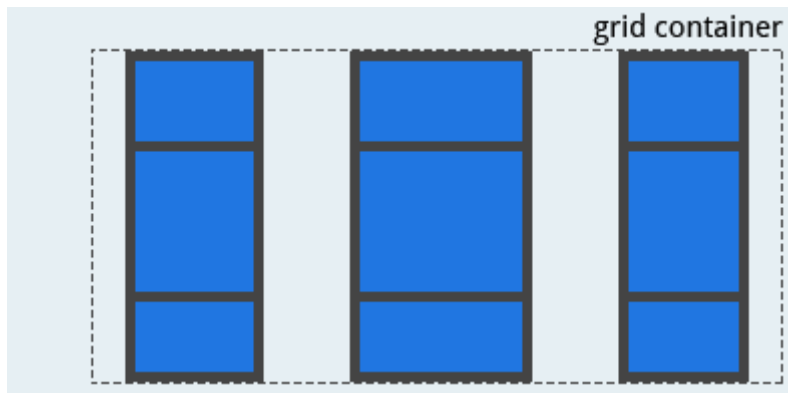
```
}
```



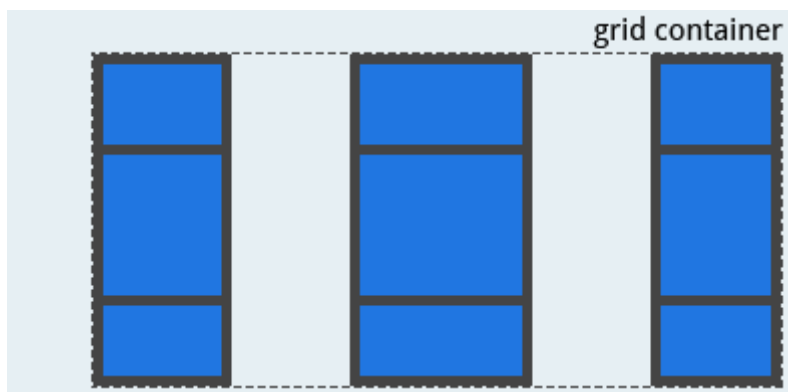
```
.container{  
  justify-content: stretch;  
}
```



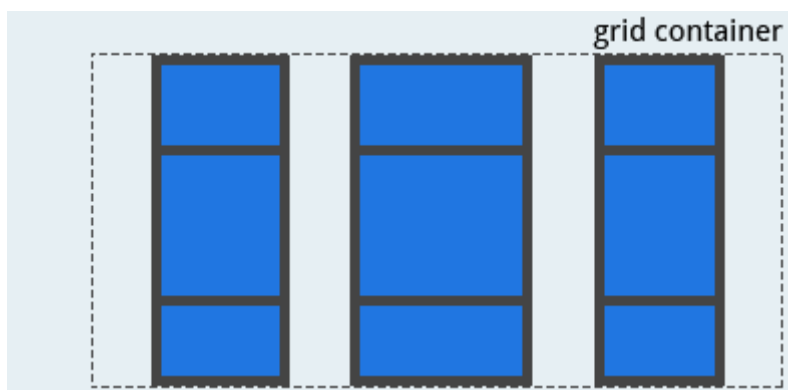
```
.container{  
  justify-content: space-around;  
}
```



```
.container{
  justify-content: space-between;
}
```



```
.container{
  justify-content: space-around;
}
```



## align-content

Parfois la dimension totale de votre grille semblera inférieure à la dimension de son container grid. Cela peut arriver si tous les items grid sont dimensionnés avec des unités non-flexibles comme `px`. Dans ce cas, vous pouvez régler l'alignement de la grille à l'intérieur du container. Cette propriété aligne la grille le long de l'axe des rangées (par opposition à `justify-content` qui aligne la grille le long de l'axe des colonnes).

Valeurs :

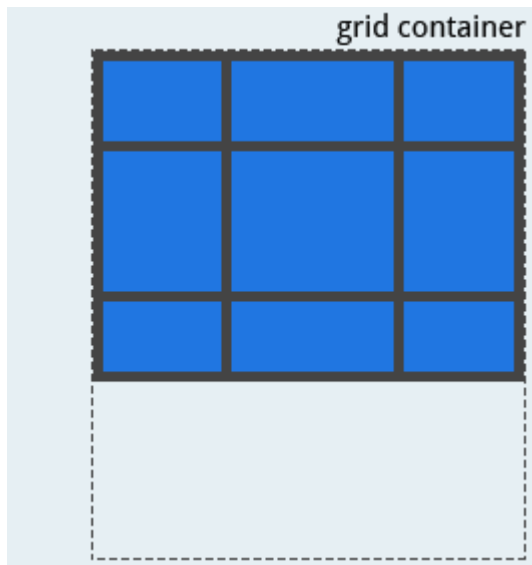
- `start` - aligne la grille à partir du sommet de la grille container (*container grid*).
- `end` - aligne la grille à partir du bas de la grille container.
- `center` - aligne la grille au centre de la grille container
- `stretch` - redimensionne les items pour permettre à la grille de remplir toute la hauteur de la grille container.
- `space-around` - place un espace égal entre chaque item de grille, et un demi-espace aux extrémités.
- `space-between` - place un espace égal entre chaque item de grille, et aucun espace aux extrémités.
- `space-evenly` - place un espace égal entre chaque item de grille, y compris aux extrémités.

```
.container{  
  align-content: start | end | center | stretch | space-around |  
  space-between | space-evenly;  
}
```

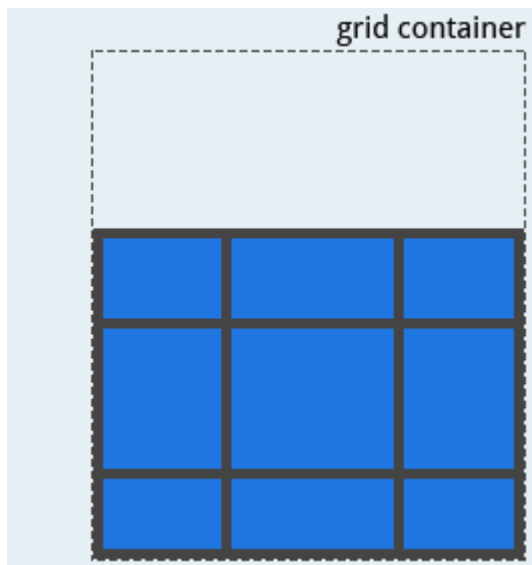
Exemples :

```
.container{  
  align-content: start;
```

```
}
```



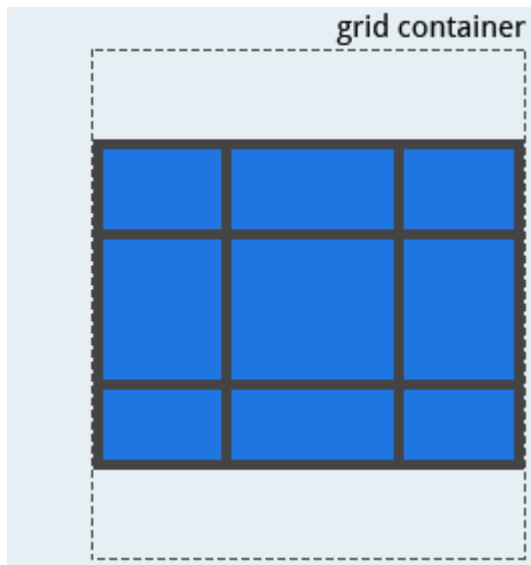
```
.container{  
  align-content: end;  
}
```



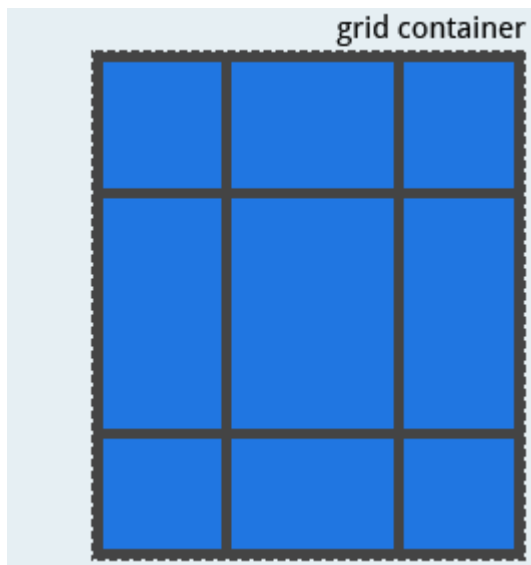
```
.container{  
  align-content: center;
```



```
}
```

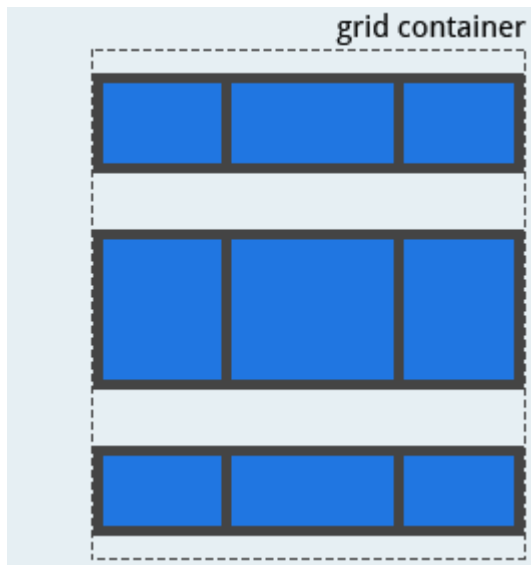


```
.container{  
  align-content: stretch;  
}
```

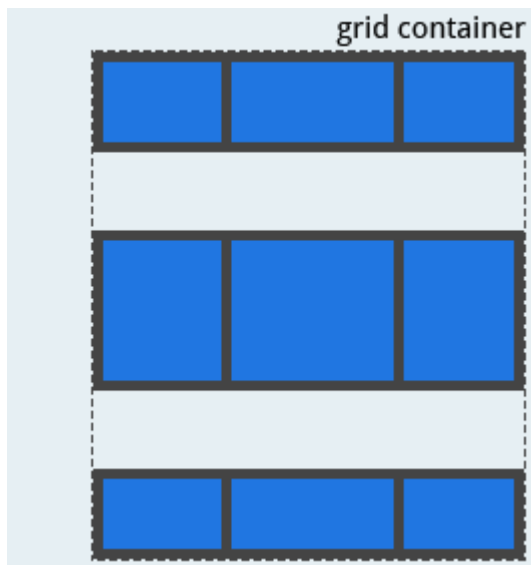


```
.container{  
  align-content: space-around;
```

```
}
```

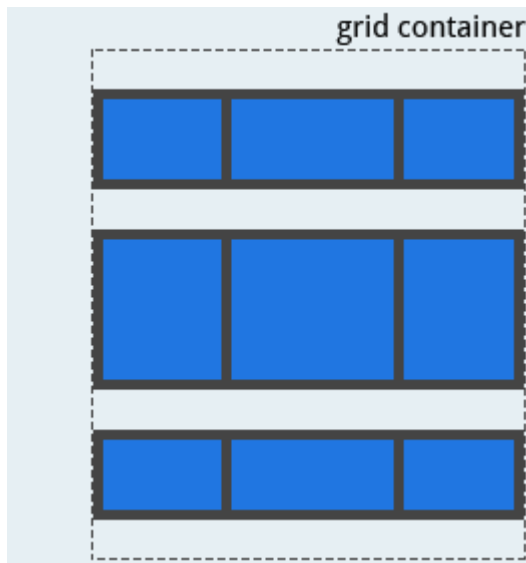


```
.container{  
  align-content: space-between;  
}
```



```
.container{  
  align-content: space-evenly;
```

}



## grid-auto-columns, grid-auto-rows

Spécifie la dimension de toute piste auto-générée (également appelée piste implicite — *implicit grid track*). Les pistes implicites sont créées lorsque vous positionnez explicitement des rangées ou des colonnes (via `grid-template-rows` ou `grid-template-column`) qui se trouvent en dehors de la grille définie.

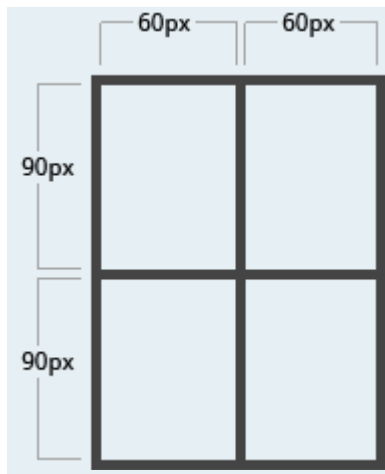
Valeurs :

- `<track-size>` - peut être une longueur, un pourcentage, ou une fraction de l'espace libre dans la grille (via l'unité `fr`)

```
.container{  
  grid-auto-columns: <track-size> ...;  
  grid-auto-rows: <track-size> ...;  
}
```

Pour illustrer la façon dont les pistes implicites sont créées, imaginons ce cas :

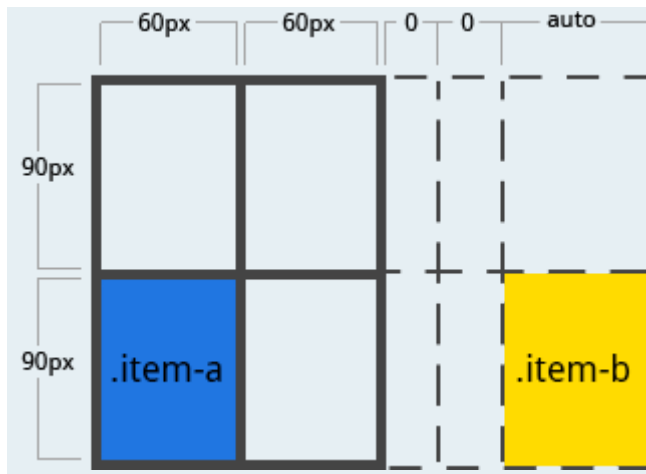
```
.container{  
  grid-template-columns: 60px 60px;  
  grid-template-rows: 90px 90px  
}
```



On crée ainsi une grille de 2 × 2.

Mais imaginons maintenant que vous utilisiez `grid-column` et `grid-row` pour positionner vos items de grille comme ceci :

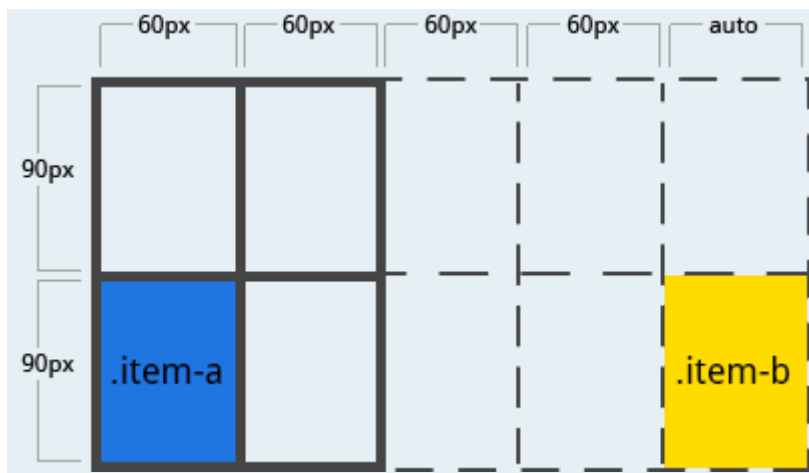
```
.item-a{  
  grid-column: 1 / 2;  
  grid-row: 2 / 3;  
}  
.item-b{  
  grid-column: 5 / 6;  
  grid-row: 2 / 3;  
}
```



Nous avons dit à `.item-b` de commencer à la ligne de colonne 5 et de se terminer à la ligne de colonne 6, *mais nous n'avons jamais défini de lignes de colonne 5 ou 6.*

Nous avons référencé des lignes qui n'existent pas, en conséquence de quoi des pistes implicites de largeur 0 sont créées pour remplir l'espace. Nous pouvons utiliser `grid-auto-columns` et `grid-auto-rows` pour spécifier la largeur de ces pistes implicites :

```
.container{
  grid-auto-columns: 60px;
}
```



## grid-auto-flow

Si vous avez des items de grille que vous ne placez pas explicitement sur la grille, l'algorithme de placement automatique intervient pour placer automatiquement les items. Cette propriété contrôle la façon dont l'algorithme de placement automatique fonctionne.

Valeurs :

- `row` - indique à l'algorithme de remplir chaque rangée tout à tour, en ajoutant de nouvelles rangées si nécessaire.
- `column` - indique à l'algorithme de remplir chaque colonne tout à tour, en ajoutant de nouvelles colonnes si nécessaire.
- `dense` - indique à l'algorithme d'essayer de remplir les trous le plus tôt possible dans la grille au cas où de plus petits items devaient apparaître plus tard.

```
.container{  
  grid-auto-flow: row | column | row dense | column dense  
}
```

Attention : `dense` pourrait faire apparaître vos items dans le désordre.

Exemples :

Considérons ce HTML :

```
<section class="container">  
  <div class="item-a">item-a</div>  
  <div class="item-b">item-b</div>  
  <div class="item-c">item-c</div>  
  <div class="item-d">item-d</div>  
  <div class="item-e">item-e</div>
```

</section>

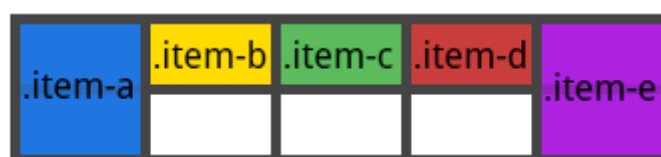
Définissons une grille de cinq colonnes sur deux rangées et réglons `grid-auto-flow` sur `row` (qui est également la valeur par défaut) :

```
.container{  
  display: grid;  
  grid-template-columns: 60px 60px 60px 60px 60px;  
  grid-template-rows: 30px 30px;  
  grid-auto-flow: row;  
}
```

Quand nous plaçons les items sur la grille, nous spécifions seulement le placement de deux d'entre eux, les `item-a` et `item-e` :

```
.item-a{  
  grid-column: 1;  
  grid-row: 1 / 3;  
}  
.item-e{  
  grid-column: 5;  
  grid-row: 1 / 3;  
}
```

Comme nous avons réglé `grid-auto-flow` sur `row`, notre grille va ressembler à ceci. Remarquez que les trois items que nous n'avons pas placés (`item-b`, `item-c` et `item-d`) se répartissent sur les rangées disponibles :



Mais si au lieu de cela nous avons réglé `grid-auto-flow` sur `column`, les `item-b`, `item-c` et `item-d` se seraient répartis sur les colonnes :

```
.container{  
  display: grid;  
  grid-template-columns: 60px 60px 60px 60px 60px;  
  grid-template-rows: 30px 30px;  
  grid-auto-flow: column;  
}
```



## grid

`grid` est un raccourci permettant de définir toutes les propriétés suivantes en une seule déclaration : `grid-template-rows`, `grid-template-columns`, `grid-template-areas`, `grid-auto-rows`, `grid-auto-columns`, et `grid-auto-flow`. Il donne également à `grid-column-gap` et `grid-row-gap` leur valeur initiale, même si elles ne peuvent pas être explicitement définies par cette propriété.

Valeurs :

- `none` - règle toutes les sous-propriétés à leur valeur initiale.
- `subgrid` - règle `grid-template-rows` et `grid-template-columns` sur `subgrid` et toutes les autres sous-propriétés à leur valeur initiale.
- `<grid-template-rows> / <grid-template-columns>` - règle `grid-template-rowset` `grid-template-columns` à leur valeur spécifiée, respectivement, et toutes les autres sous-propriétés à leur valeur initiale.



- `<grid-auto-flow> [ <grid-auto-rows> [ / <grid-auto-columns> ] ]` - accepte toutes les valeurs de `grid-auto-flow`, `grid-auto-rows` et `grid-auto-columns`, respectivement. Si `grid-auto-columns` est omis, il prend la valeur donnée à `grid-auto-rows`. Si les deux sont omis, ils sont réglés sur leur valeur initiale.

```
.container{  
  grid: none | subgrid | <grid-template-rows> /  
<grid-template-columns> | <grid-auto-flow> [ <grid-auto-rows>  
[/ <grid-auto-columns> ]];  
}
```

Exemples :

Les deux blocs de code suivants sont équivalents :

```
.container{  
  grid: 200px auto / 1fr auto 1fr;  
}
```

est l'équivalent de :

```
.container{  
  grid-template-rows: 200px auto;  
  grid-template-columns: 1fr auto 1fr;  
  grid-template-areas: none;  
}
```

De même, les deux blocs de code suivants sont équivalents :

```
.container{  
  grid: column 1fr / auto;
```

```
}
```

est l'équivalent de :

```
.container{  
  grid-auto-flow: column;  
  grid-auto-rows: 1fr;  
  grid-auto-columns: auto;  
}
```

grid accepte également une syntaxe plus complexe mais très pratique pour régler toutes les propriétés en une seule fois. Vous spécifiez `grid-template-areas`, `grid-auto-rows` et `grid-auto-columns`, et toutes les autres sous-propriétés sont réglées sur leur valeur initiale. Vous spécifiez les noms de lignes et les dimensions de pistes en ligne avec leur zones de grille respective. C'est plus facile à décrire avec un exemple :

```
.container{  
  grid: [row1-start] "header header header" 1fr [row1-end]  
        [row2-start] "footer footer footer" 25px [row2-end]  
        / auto 50px auto;  
}
```

C'est l'équivalent de :

```
.container{  
  grid-template-areas: "header header header"  
                      "footer footer footer";  
  grid-template-rows: [row1-start] 1fr [row1-end row2-start]  
25px [row2-end];  
  grid-template-columns: auto 50px auto;  
}
```

---

## Propriétés des Enfants (Grid Items)

### Table des matières

- [grid-column-start](#)
- [grid-column-end](#)
- [grid-row-start](#)
- [grid-row-end](#)
- [grid-column](#)
- [grid-row](#)
- [grid-area](#)
- [justify-self](#)
- [align-self](#)

### grid column-start, grid-column-end, grid-row-start, grid-row-end

Déterminent l'endroit où se trouve un item de grille en référence à des lignes de grille spécifiques. `grid-column-start` / `grid-row-start` est la ligne où commence l'item, `grid-column-end` / `grid-row-end` est la ligne où il finit.

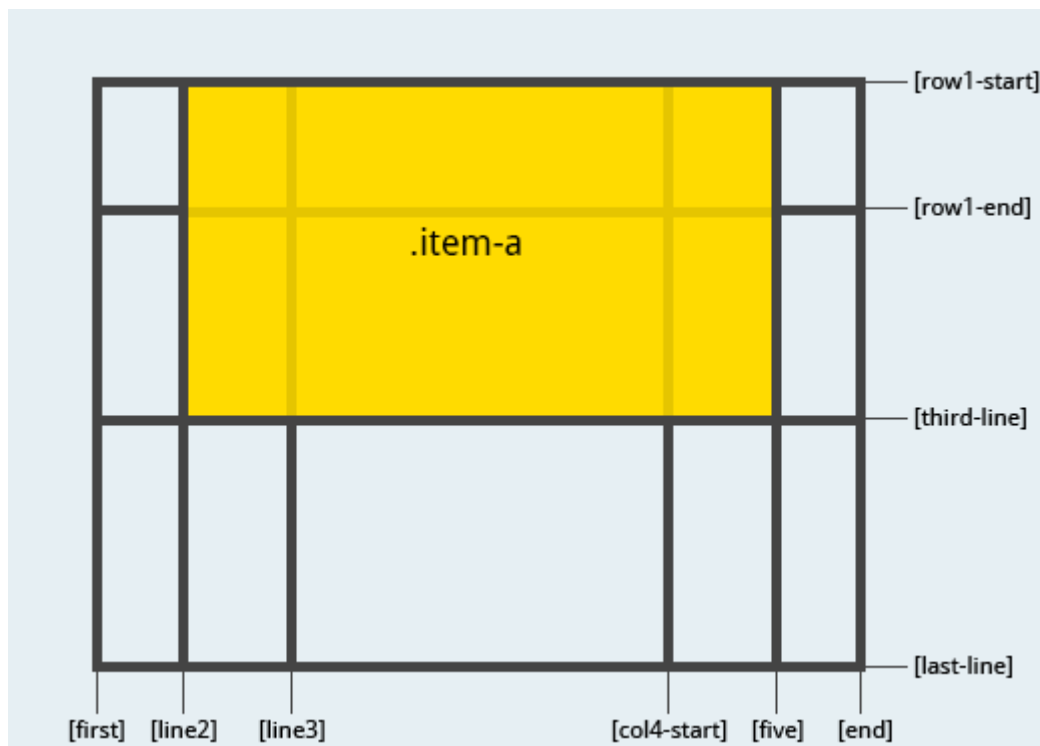
Valeurs :

- `<line>` - peut être un numéro pour se référer à une ligne de grille numérotée, ou un nom pour se référer à une ligne de grille nommée.
- `span <number>` - l'item s'étendra (*span*) sur le nombre spécifié de pistes de grille.
- `span <name>` - l'item s'étendra jusqu'à rencontrer le ligne contenant le nom fourni.
- `auto` - indique un auto-placement, une extension (*span*) automatique, ou le span par défaut

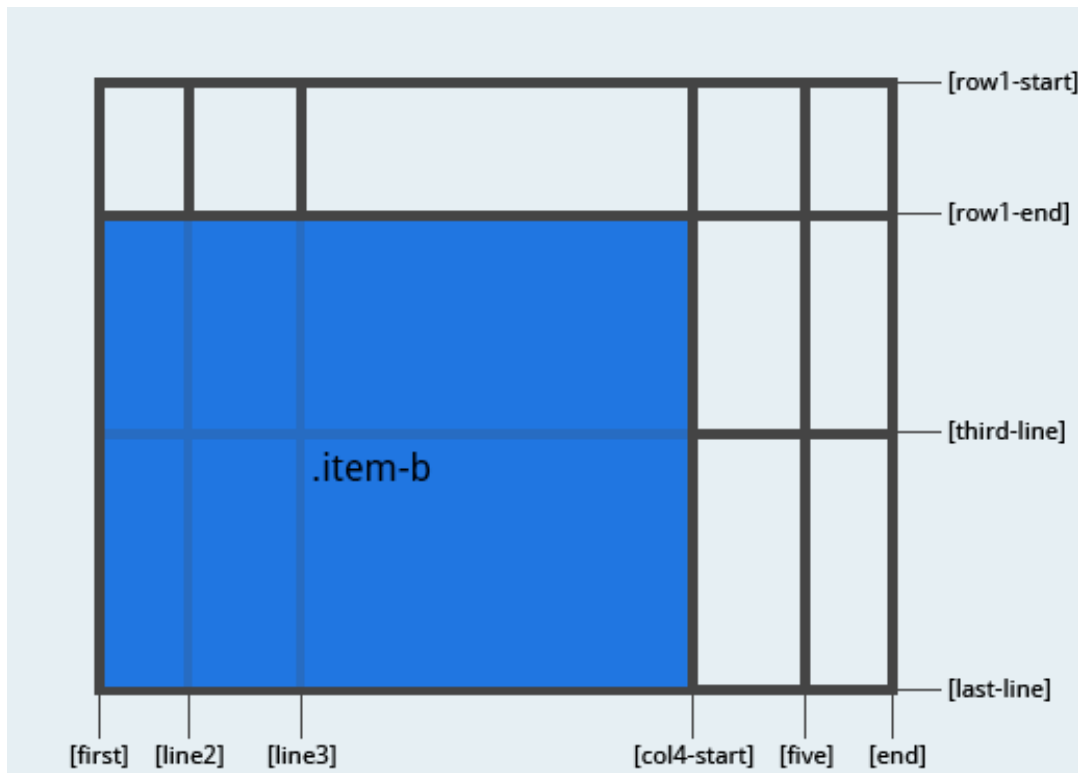
```
.item{
  grid-column-start: <number> | <name> | span <number> |
span <name> | auto
  grid-column-end: <number> | <name> | span <number> |
span <name> | auto
  grid-row-start: <number> | <name> | span <number> | span
<name> | auto
  grid-row-end: <number> | <name> | span <number> | span
<name> | auto
}
```

Exemples :

```
.item-a{
  grid-column-start: 2;
  grid-column-end: five;
  grid-row-start: row1-start
  grid-row-end: 3
}
```



```
.item-b{  
  grid-column-start: 1;  
  grid-column-end: span col4-start;  
  grid-row-start: 2  
  grid-row-end: span 2  
}
```



Si aucune `grid-column-end`/ `grid-row-end` n'est déclarée, l'item s'étendra sur 1 piste par défaut.

Les items peuvent se recouvrir. Vous pouvez utiliser `z-index` pour contrôler leur empilement.

## grid-column, grid-row

Raccourci pour `grid-column-start` + `grid-column-end`, et `grid-row-start` + `grid-row-end`, respectivement.

Valeurs :

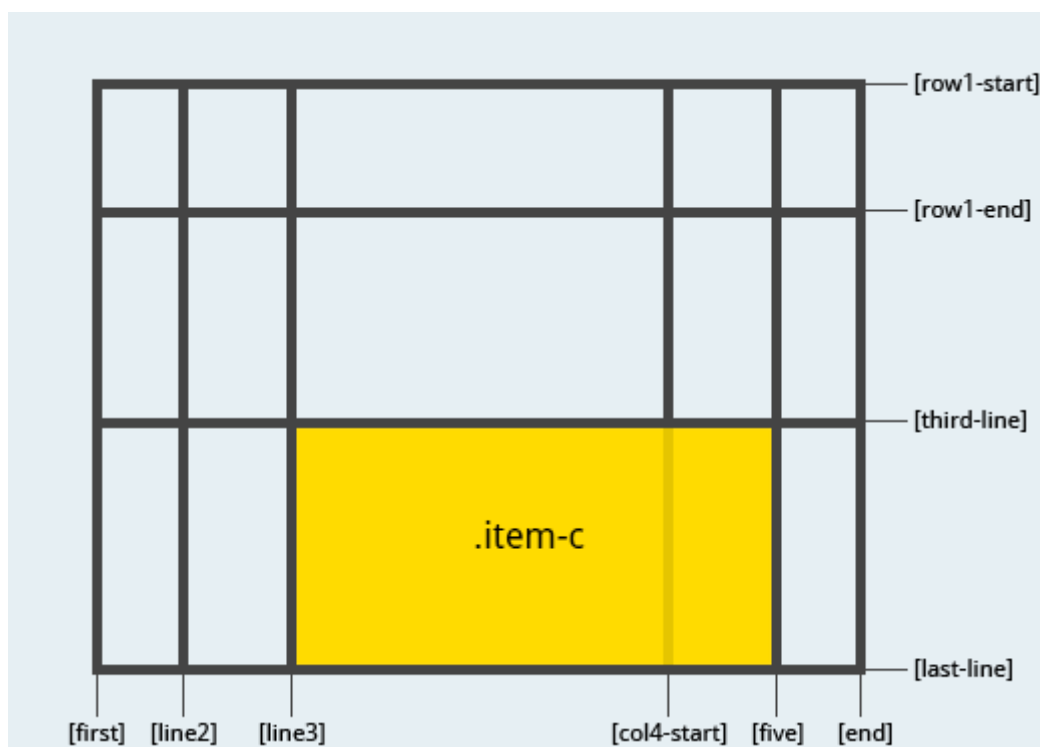
- `<start-line>` / `<end-line>` - chacun accepte les mêmes valeurs que la version longue, y compris `span`.

```
.item{
  grid-column: <start-line> / <end-line> | <start-line> / span
```

```
<value>;  
  grid-row: <start-line> / <end-line> | <start-line> / span  
<value>;  
}
```

Exemple :

```
.item-c{  
  grid-column: 3 / span 2;  
  grid-row: third-line / 4;  
}
```



Si aucune valeur de fin de ligne n'est déclarée, l'item s'étendra sur 1 piste par défaut.

## grid-area

Donne un nom à un item pour qu'il puisse être référencé par un template créé avec la propriété `grid-template-areas`. Par ailleurs, cette propriété peut être utilisée comme raccourci encore plus court pour `grid-row-start` + `grid-column-start` + `grid-row-end` + `grid-column-end`.

Valeurs :

- `<name>` - un nom, choisi par vous.
- `<row-start> / <column-start> / <row-end> / <column-end>` - peuvent être des nombres ou des lignes nommées.

```
.item{  
  grid-area: <name> | <row-start> / <column-start> /  
<row-end> / <column-end>;  
}
```

Exemples :

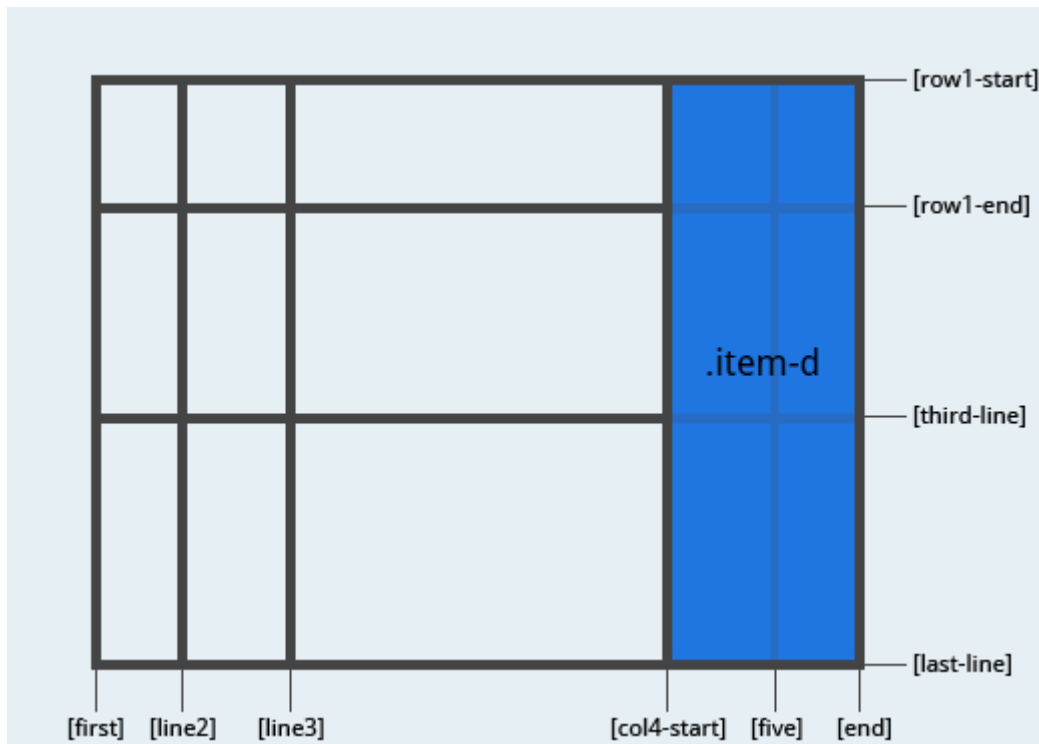
`grid-area` pour attribuer un nom à l'item :

```
.item-d{  
  grid-area: header  
}
```

`grid-area` comme raccourci de `grid-row-start` + `grid-column-start` + `grid-row-end` + `grid-column-end` :

```
.item-d{  
  grid-area: 1 / col4-start / last-line / 6  
}
```





## justify-self

Aligne le contenu d'un item de grille sur l'axe des colonnes (par opposition à `align-self` qui l'aligne le long de l'axe des rangées). Cette propriété s'applique au contenu d'un item de grille et uniquement à lui.

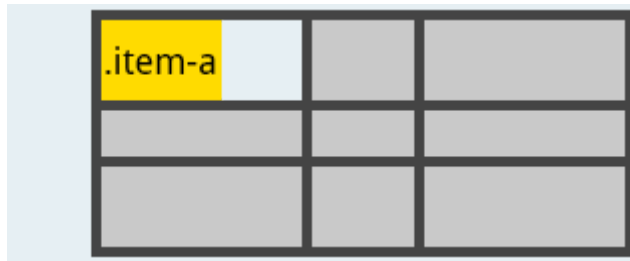
Valeurs :

- `start` - aligne le contenu sur la gauche de la zone de grille
- `end` - aligne le contenu sur la droite de la zone de grille
- `center` - aligne le contenu au centre de la zone de grille
- `stretch` - Remplit toute la largeur de la zone de grille (c'est la valeur par défaut).

```
.item{
  justify-self: start | end | center | stretch;
}
```

Exemples :

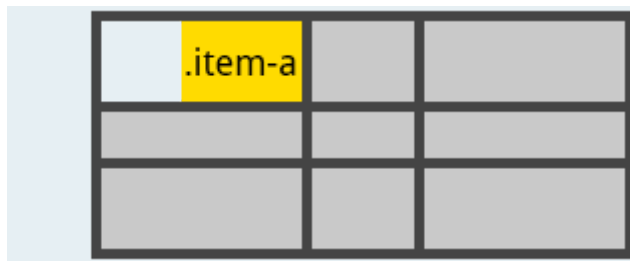
```
.item-a{  
  justify-self: start;  
}
```



A 3x3 grid with a light blue background. The first cell of the first row is highlighted in yellow and contains the text ".item-a". The text is aligned to the left (start) of the cell.

.item-a		

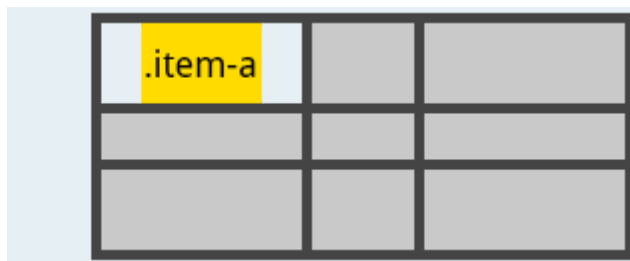
```
.item-a{  
  justify-self: end;  
}
```



A 3x3 grid with a light blue background. The first cell of the first row is highlighted in yellow and contains the text ".item-a". The text is aligned to the right (end) of the cell.

.item-a		

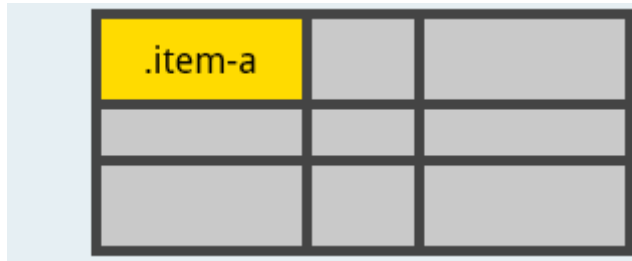
```
.item-a{  
  justify-self: center;  
}
```



A 3x3 grid with a light blue background. The first cell of the first row is highlighted in yellow and contains the text ".item-a". The text is centered within the cell.

.item-a		

```
.item-a{  
  justify-self: stretch;  
}
```



Pour régler l'alignement de *tous* les items dans une grille, ce comportement peut être défini sur le container grid via la propriété `justify-items`.

## align-self

Aligne le contenu d'un item de grille sur l'axe des rangées (par opposition à `justify-self` qui l'aligne le long de l'axe des colonnes). Cette propriété s'applique au contenu d'un item de grille et uniquement à lui.

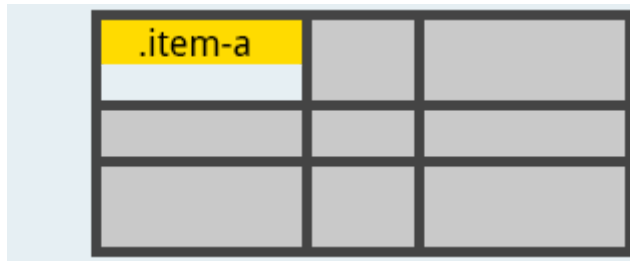
Valeurs :

- `start` - aligne le contenu sur le sommet de la zone de grille
- `end` - aligne le contenu sur le bas de la zone de grille
- `center` - aligne le contenu au centre de la zone de grille
- `stretch` - Remplit toute la hauteur de la zone de grille (c'est la valeur par défaut).

```
.item{  
  align-self: start | end | center | stretch;  
}
```

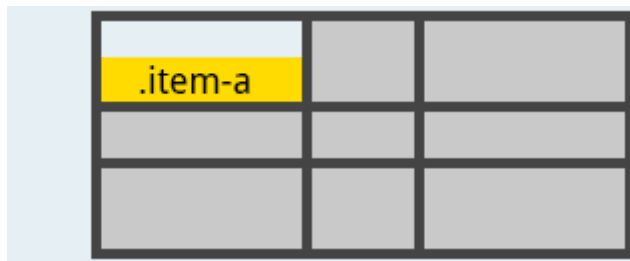
Exemples :

```
.item-a{  
  align-self: start;  
}
```



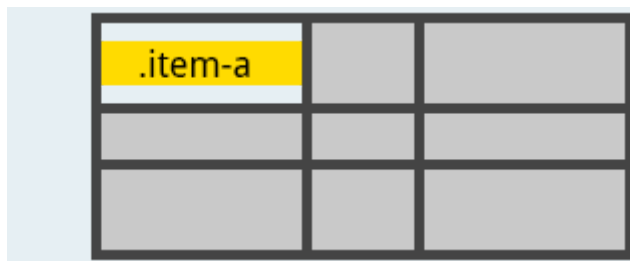
.item-a		

```
.item-a{  
  align-self: end;  
}
```



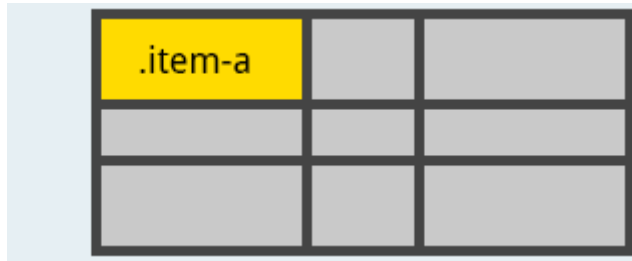
.item-a		

```
.item-a{  
  align-self: center;  
}
```



.item-a		

```
.item-a{  
  align-self: stretch;  
}
```



Pour régler l'alignement de *tous* les items dans une grille, ce comportement peut être défini sur le container grid via la propriété align-items.