

Initiation au modèle flexbox

JACQUA David

Sommaire

I. Première approche	4	III. Les règles des flex-items	21
A. Utilisation simple	4	A. Distribution différentielle des flex-items	21
B. Qu'est-ce qu'inline-flex ?	5	B. Taille des flex-items	23
C. Le container-flex et les flex-items.....	6	C. L'ordre des éléments	30
II. Les règles du container-flex	6	IV. Conclusion	32
A. Le flux : direction et comportement des lignes ou colonnes.....	6		
B. Distribution sur l'axe principale	11		
C. Distribution sur l'axe secondaire avec axe principal multiple	19		

Crédits des illustrations:
© Skill & You et Focolla

Les repères de lecture



Retour au chapitre



Définition



Objectif(s)



Espace Élèves



Vidéo /
Audio



Point important /
À retenir



Remarque(s)



Pour aller
plus loin



Normes et lois



Quiz

Introduction

Comme nous l'avons évoqué précédemment, le modèle flexbox est un véritable module de mise en page proposant de nombreuses solutions et combinaisons s'agissant d'alignement, de distribution, d'agencement et de centrage des éléments.

S'agissant d'un modèle complexe à appréhender, il faut l'envisager pas à pas et avec méthode. En particulier, on distinguera deux types de règles, celles portant sur le conteneur (ou flex-container) et celles portant sur les éléments eux-mêmes (ou - [flex-items]). Il faut dans un premier temps comprendre les premières avant d'aborder les secondes.

```
1 <style>
2 @media only screen and (min-width: 768px) {
3   .content {
4     width: 98% !important;
5     float: left !important;
6     margin: 0 5% !important;
7     min-width: 265px !important;
8   }
9   img {
10    width: 100% !important;
11    display: block !important;
12    height: auto !important;
13  }
14 }
15 </style>
16 </head>
```

Fig. 1

I. Première approche

A. Utilisation simple

Prenons l'exemple d'un menu simple, composé de cinq liens. Jusqu'à présent on utilisait des flottements ou une méthode avec `display:inline-block`, par exemple). Dans le premier cas, il nous faut mettre fin au flottement en utilisant diverses méthodes (`div.clear`, `.clearfix`, `.overflow`), dans le second il faut pouvoir supprimer l'espace généré par `inline-block` (par exemple à l'aide des commentaires). Les contraintes imposées par l'un ou l'autre de ces choix peuvent être évitées si l'on opte pour flexbox. En effet, le modèle flexbox permet très simplement de modifier l'ordre d'apparition des éléments de vertical à horizontal par exemple et ce, grâce au simple ajout d'une propriété sur les parents des éléments. Ce parent nous l'appelons le flex-container, les enfants directs seront alors des flex-items. Nous allons donc aboutir au HTML suivant :

```
<nav class="main-nav">
  <ul class="flex-container">
    <li class="flex-item"><a href="#">item 1</a></li>
    <li class="flex-item"><a href="#">item 2</a></li>
    <li class="flex-item"><a href="#">item 3</a></li>
    <li class="flex-item"><a href="#">item 4</a></li>
    <li class="flex-item"><a href="#">item 5</a></li>
  </ul>
</nav>
```

Fig.2

Pour le CSS, on aura simplement :

```
.flex-container {
  display: flex;
}
.flex-item a{
  display: block;
  border: 1px solid #625B48;
  padding: 5px 15px;
  text-decoration: none;
  color: #E1CE9A;
  text-align: center;
  background-color: #AE8964;
}
.flex-item a:hover {
  color: #AE8964;
  background-color: #E1CE9A;
}
```

Fig.3

Avec comme rendu (roll-over sur item 3) :



Fig. 4

Voilà, c'est très simple. Il suffit de placer la règle `display:flex` sur le parent (ou flex-container).

B. Qu'est-ce qu'inline-flex ?

Nous venons de voir la valeur `flex`, mais il en existe une autre un peu plus rare que nous allons explorer tout de suite : `inline-flex`. Par défaut, un flex-container est un élément de bloc, avec `inline-flex` on va simplement demander à notre flex-container de se comporter comme un élément `inline`. Imaginons que nous ayons un `` avant notre liste dans le HTML :

```
<nav class="main-nav">
  <span>ça, navigation ! </span>
  <ul class="flex-container">
    <li class="flex-item"><a href="#">item 1</a></li>
    <li class="flex-item"><a href="#">item 2</a></li>
    <li class="flex-item"><a href="#">item 3</a></li>
    <li class="flex-item"><a href="#">item 4</a></li>
    <li class="flex-item"><a href="#">item 5</a></li>
```

Fig. 5

Si nous ne modifions pas le CSS, on obtient :



Fig. 6

Par contre si nous indiquons à notre flex-container :

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

Fig. 7

On a :



Fig. 8

Cette propriété rappelle donc `inline-block` car on retrouve, dans le rapport entre `inline-flex` et `flex`, quelque chose du rapport entre `inline-block` et `block`.

C. Le container-flex et les flex-items

Pour explorer le modèle flexbox il va falloir distinguer les règles qui se placent sur le flex-container et celles qui vont se placer sur les flex-items. Dans tous les cas, c'est bien le comportement des flex-items qui sera modifié, et si l'on veut comprendre ce modèle de mise en page, cette distinction est primordiale. En effet, certaines règles rentrent en conflit avec d'autres ou les rendent inopérantes. Maîtriser en premier lieu celles qui se placent sur le container-flex est donc primordiale avant d'explorer celles qui se placent sur les flex-items afin de pouvoir s'y retrouver.

II. Les règles du container-flex

A. Le flux : direction et comportement des lignes ou colonnes

Par défaut, le modèle flexbox distribue les flex-items sur une seule rangée au point de dépasser du flex-container s'ils sont trop nombreux. Prenons le HTML suivant :

```
<nav class="main-nav">
  <ul class="flex-container">
    <li class="flex-item"><a href="#">item 1</a></li>
    <li class="flex-item"><a href="#">item 2</a></li>
    <li class="flex-item"><a href="#">item 3</a></li>
    <li class="flex-item"><a href="#">item 4</a></li>
    <li class="flex-item"><a href="#">item 5</a></li>
    <li class="flex-item"><a href="#">item 6</a></li>
    <li class="flex-item"><a href="#">item 7</a></li>
    <li class="flex-item"><a href="#">item 8</a></li>
    <li class="flex-item"><a href="#">item 9</a></li>
    <li class="flex-item"><a href="#">item 10</a></li>
  </ul>
</nav>
```

Fig.9

Et le CSS :

```
#wrapper {
  margin: 0 auto;
  width: 50vw;
  background-color: #fff;
  padding: 15px;
  border: 1px solid #d5f;
}

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

Fig.10

Le rendu :

item 1	item 2	item 3	item 4	item 5	item 6	item 7	item 8	item 9	item 10
--------	--------	--------	--------	--------	--------	--------	--------	--------	---------

Fig. 11

Le flex-containeur que l'on visualise grâce à la bordure rose n'est pas assez large pour contenir tous les flex-items, bien que ceux-ci se soient réduit au maximum de leur possibilité comme en témoigne le rejet du chiffre sur une seconde ligne. Mais il n'y a tout simplement pas assez de place.

Il n'y a qu'une manière de modifier ce comportement ; utiliser `flex-wrap` pour indiquer aux flex-items de passer sur une seconde rangée, avec la valeur `wrap`, soit :

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

Fig. 12

Le rendu :

item 1	item 2	item 3	item 4	item 5	item 6
item 7	item 8	item 9	item 10		

Fig. 13

Petite particularité on peut demander aux flex-items de « passer à la ligne », et également, si on le souhaite, inverser l'ordre d'apparition des rangées avec `wrap-reverse`.

```
.flex-container {  
    display: flex;  
    flex-wrap: wrap-reverse;  
}
```

Fig. 14

Le rendu :

item 7	item 8	item 9	item 10
item 1	item 2	item 3	item 4

Fig. 15

Ce n'est pas l'ordre des flex-items qui est inversé, mais l'ordre des rangées de retour à la ligne.

Si l'on voulait forcer les flex-items à tenir sur une ligne on spécifierait la valeur par défaut de **flex-wrap** qui est **nowrap**, on reviendrait à notre premier exemple où les items débordent du flex-container.

Maintenant on peut également demander à nos flex-items d'apparaître non pas de gauche à droite mais de bas vers le haut en utilisant **flex-direction** pour lequel on associera la valeur **column**.

```
.flex-container {  
    display: flex;  
    flex-direction: column;  
}
```

Fig. 16

Le rendu :

item 1
item 2
item 3
item 4
item 5
item 6
item 7
item 8
item 9
item 10

Fig. 17

Si vous êtes en **flex-direction: column**, alors la valeur de **flex-wrap** importe peu. Dans cette configuration, en effet, le flex-item s'adapte à la largeur du flex-container et ne peut par conséquent jamais déborder. Le flex-item suivant se contente de s'empiler à la suite de son frère.

A nouveau on peut inverser l'empilement des flex-item en utilisant **column-reverse**:

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

Fig.18

Le rendu :

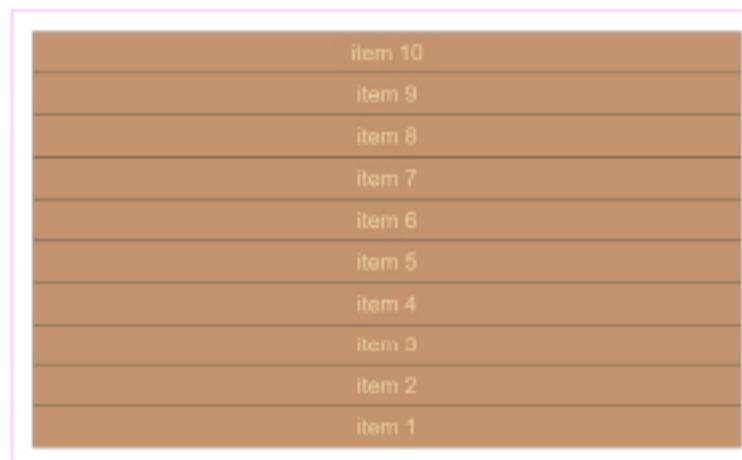


Fig.19

Par défaut, la valeur de **flex-direction** est **row** (ce qui nous renvoie à la première copie d'écran de cette partie), et tout comme pour les colonnes on peut inverser l'ordre des éléments en utilisant **row-reverse**. Mais plutôt que nous attarder sur les valeurs de **flex-direction**, nous allons aborder le raccourci **css flex-flow**, combinant **flex-direction + flex-wrap**. Par exemple :

```
.flex-container {  
    display: flex;  
    flex-flow: row-reverse wrap;  
}
```

Fig.20

Le rendu :

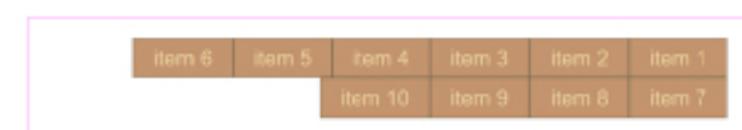


Fig.21

Il est donc courant de n'utiliser que le raccourci flex-flow, par exemple pour styliser les colonnes inversées :

```
.flex-container {  
    display: flex;  
    flex-flow: column-reverse wrap;  
}
```

Fig.22

Le rendu :

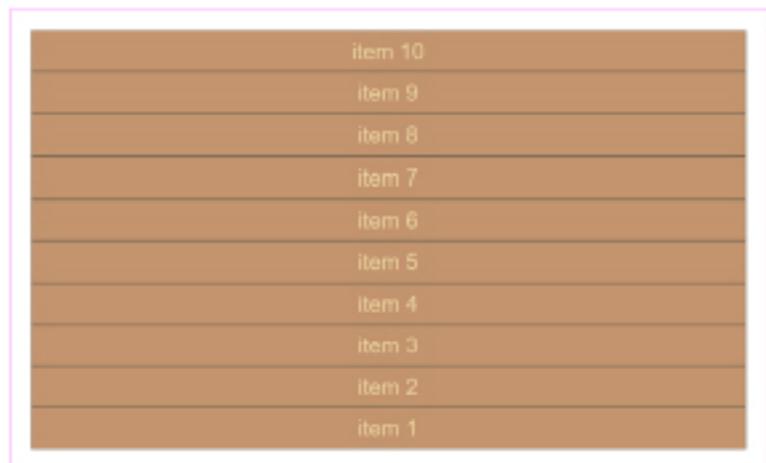


Fig.23

Tableau n°1 Récapitulatif

DESCRIPTION	PROPRIÉTÉ	VALEURS	EXEMPLE
Passage ou non à la ligne	flex-wrap	nowrap wrap wrap-reverse	flex-wrap : wrap
Axe vertical ou horizontal	flex-direction	row row-reverse column column-reverse	flex-direction : row
Raccourci	flex-flow	'flex-direction' + 'flex-wrap'	flex-flow : nowrap row

B. Distribution sur l'axe principale

Il faut bien comprendre que l'axe principal peut tout aussi bien être horizontal (c'est le comportement par défaut équivalent à `flex-direction: row | row-reverse`) que vertical (`flex-direction: column | column-reverse`). Cette notion est fondamentale pour appréhender la distribution des éléments.

Regardons le comportement par défaut des flex-items en diminuant le nombre de flex-items :

```
<nav class="main-nav">
  <ul class="flex-container">
    <li class="flex-item"><a href="#">item 1</a></li>
    <li class="flex-item"><a href="#">item 2</a></li>
    <li class="flex-item"><a href="#">item 3</a></li>
    <li class="flex-item"><a href="#">item 4</a></li>
  </ul>
</nav>
```

Fig.24

Par défaut les flex-items se collent sur la gauche du flex-conteneur. On peut évidemment leur demander de se placer tout à droite. Par exemple :

```
.flex-container {
  display: flex;
  flex-flow: row nowrap;
  justify-content: flex-end;
}
```

Fig.25

Le rendu :

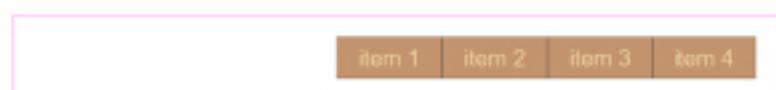


Fig.26

On peut centrer les 4 flex-items dans le conteneur :

```
.flex-container {
  display: flex;
  flex-flow: row nowrap;
  justify-content: center;
}
```

Fig.27



item 1 item 2 item 3 item 4

Fig.28

Mais également opérer un centrage vertical, si notre axe principal est vertical :

```
.flex-container {
    display: flex;
    flex-flow: column nowrap;
    justify-content: center;
    height: 50vh;
    /*hauteur exprimé en unité viewport
    en gros en pourcentage de la fenêtre */
}
```

Fig.29

On modifie les axes verticaux et horizontaux avec `flex-flow` puis on donne une taille supérieure (bien sûr) au `flex-container`, sinon on ne s'apercevrait pas du centrage vertical. Résultat.

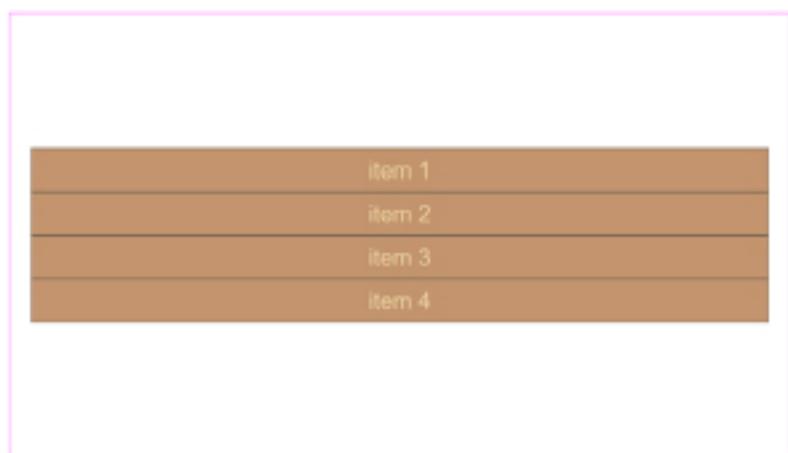


Fig.30

On peut demander aux flex-items de se retrouver à la fin de l'axe principal, c'est-à-dire à droite lorsqu'on est en `row` ou en bas quand on est en `column` (comme ici).

Par exemple :

```
.flex-container {  
    display: flex;  
    flex-flow: column nowrap;  
    justify-content: flex-end;  
    height: 50vh;  
}
```

Fig. 31

Ce qui nous donne :

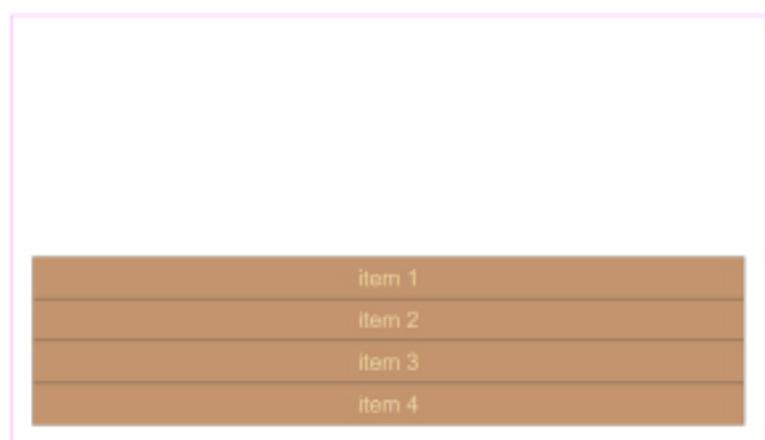
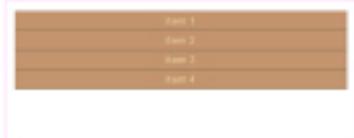
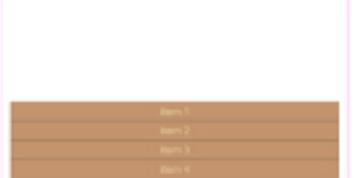
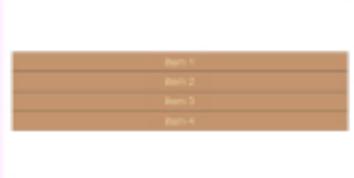


Fig. 32

Mais la propriété **justify-content** a aussi de nombreuses capacités en termes de distribution des éléments. Ici, il faut bien comprendre comment celle-ci fonctionne. Il s'agit de répartir le total de la place qu'il reste moins celle des flex-items, c'est donc l'espace restant qui est dispatché. Avec **flex-start** ou **flex-end**, l'espace restant est placé après ou avant les flex-items. Avec **center** il est réparti de manière équivalente après et avant (ici, en haut et en bas).

Tableau n°2

FLEX-START	FLEX-END	CENTER
		
		

La distribution peut être plus complexe encore :

Avec **space-between** il est répartit équitablement entre les flex-items sachant que les deux extrémités sont placées contre les bord du flex-container.

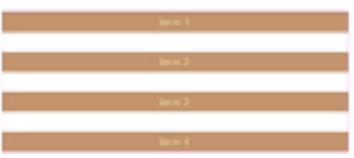
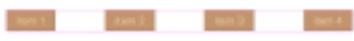
Avec **space-around** l'espace vacant est distribué autour de chaque flex-item de manière égale à droite et à gauche de l'élément, indépendamment de leur position.

Enfin avec **space-evenly**, tous les espaces entre les flex-items comme entre les flex-items et les extrémités sont égaux. Cf. Tableau n°3.

Récapitulatif :

justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly;

Tableau n°3

SPACE BETWEEN	SPACE-AROUND	SPACE-EVENLY
		
		

Distribution sur l'axe secondaire avec l'axe principal unique

Après avoir détaillé l'axe principal, intéressons-nous à l'axe secondaire. Il peut lui aussi être horizontal (si l'axe principal est vertical) ou vertical (si l'axe principal est horizontal). Cette notion est fondamentale pour appréhender la distribution des éléments.

Regardons la distribution par défaut des flex-items sur l'axe secondaire, le HTML reste le même. On va placer les flex-items sur une rangée et demander une répartition en space-between :

```
.flex-container {  
    display: flex;  
    flex-flow: row nowrap;  
    justify-content: space-between;  
    height: 50vh;  
}
```

Fig. 3.3

Résultat :

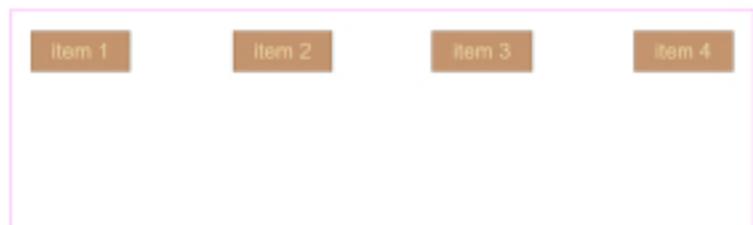


Fig. 3.4

On peut modifier l'axe secondaire en demandant à nos flex-items de se retrouver à la fin de l'axe secondaire via la propriété align-items (c'est-à-dire, dans notre cas, en bas du bloc, puisque l'axe principal étant horizontal, le secondaire est vertical) :

```
.flex-container {  
    display: flex;  
    flex-flow: row nowrap;  
    justify-content: space-between;  
    align-items: flex-end;  
    height: 50vh;  
}
```

Fig. 3.5



Fig.36

On retrouve alors certaines mêmes propriétés déjà vues pour **justify-content** (**flex-start**, **flex-end**, **center**) et deux valeurs que nous allons regarder de plus près **baseline** et **stretch**. Mais avant cela il nous faut intervenir sur nos flex-items pour comprendre ce qu'il se produit.

```
<nav class="main-nav">
  <ul class="flex-container">
    <li class="flex-item"><a href="#">item 1</a></li>
    <li class="flex-item big"><a href="#">item 2 </a></li>
    <li class="flex-item bigger"><a href="#">item 3</a></li>
    <li class="flex-item"><a href="#">item 4</a></li>
  </ul>
</nav>
```

Fig.37

On ajoute la classe **big** et **bigger** aux deux flex-items du centre. Pour bien visualiser ce que l'on fait on ajoute couleur et bordure aux flex-items et on augmente la taille et le padding des classes **big** et **bigger**:

```
.flex-container {
  display: flex;
  flex-flow: row nowrap;
  justify-content: space-between;
  align-items: flex-end;
  height: 50vh;
}

.flex-item {
  border: 1px solid #625B48;
  background-color: #AE8964;
}
.big {
  padding: 8px 20px;
  height: 25px;
}
.bigger {
  padding: 32px 24px;
  height: 50px;
}
```

Fig.38

Le résultat :

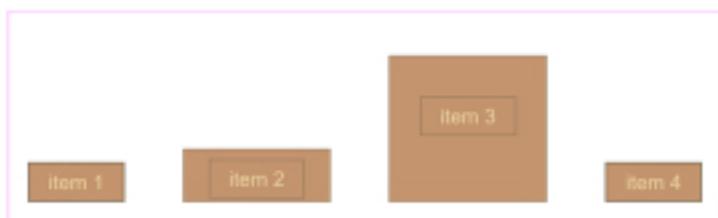


Fig. 39

On distingue désormais clairement nos flex-items qui ont des tailles différentes.

Regardons d'abord la valeur `baseline`.

```
.flex-container {  
    display: flex;  
    flex-flow: row nowrap;  
    justify-content: space-between;  
    align-items: baseline;  
    height: 50vh;  
}
```

Fig. 40

Le rendu :

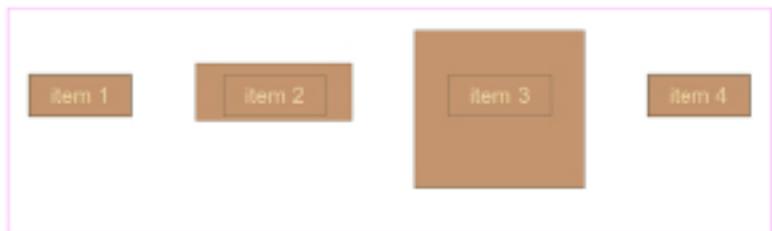


Fig. 41

Quelle est la clef pour comprendre ce comportement. C'est tout simplement la première ligne du texte qui est utilisée comme repère de l'alignement vertical.

```
.flex-container {  
    display: flex;  
    flex-flow: row nowrap;  
    justify-content: space-between;  
    align-items: stretch;  
    height: 50vh;  
}
```

Fig.42

Le rendu :

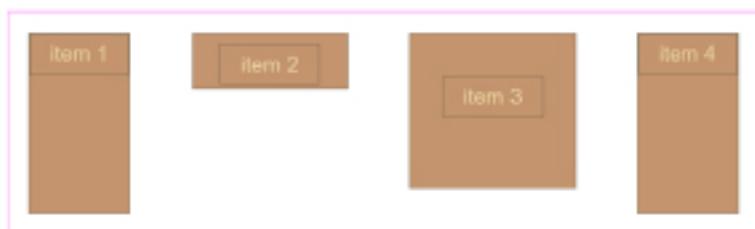


Fig.43

Le premier et le dernier élément prennent toute la hauteur. Mais que se passe-t-il pour les deux autres ? Comme ils possèdent une hauteur ils ne sont pas soumis à stretch. Si nous retirons ces indications de hauteur, quel sera leur comportement ?

```
.big {  
    padding: 8px 20px;  
    /* height: 25px; */  
}  
.bigger {  
    padding: 32px 24px;  
    /* height: 50px; */  
}
```

Fig.44

On a placé les deux hauteurs en commentaire pour les neutraliser. Le résultat :

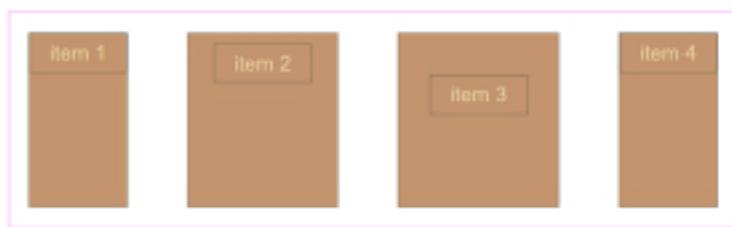


Fig.45

Récapitulatif :

align-items: flex-start | flex-end | center | baseline | stretch;

C. Distribution sur l'axe secondaire avec axe principal multiple

Lorsqu'on a plusieurs lignes, la propriété align-items n'est pas suffisante. Regardons en effet ce qu'il se produit avec 12 flex-items aux largeurs différentes :

```
<nav class="main-nav">
  <ul class="flex-container">
    <li class="flex-item"><a href="#">item 1</a></li>
    <li class="flex-item big"><a href="#">item 2 </a></li>
    <li class="flex-item bigger"><a href="#">item 3</a></li>
    <li class="flex-item"><a href="#">item 4</a></li>
    <li class="flex-item"><a href="#">item 5</a></li>
    <li class="flex-item big"><a href="#">item 6 </a></li>
    <li class="flex-item bigger"><a href="#">item 7</a></li>
    <li class="flex-item"><a href="#">item 8</a></li>
    <li class="flex-item"><a href="#">item 9</a></li>
    <li class="flex-item big"><a href="#">item 10 </a></li>
    <li class="flex-item bigger"><a href="#">item 11</a></li>
    <li class="flex-item"><a href="#">item 12</a></li>
  </ul>
</nav>
```

Fig.46

Le CSS :

```
.flex-container {
  display: flex;
  flex-flow: row wrap;
  justify-content: flex-start;
  align-items: flex-end;
  height: 80vh;
}

.flex-item {
  border: 1px solid #625848;
  background-color: #AE8964;
}

.big {
  width: 100px;
}
.bigger {
  width: 150px;
}
```

Fig.47

On demande à nos éléments de passer à la ligne flex-flow: `row nowrap`. Sur l'axe principal nos flex-items vont se placer sur la gauche (l'axe principal est horizontal) avec `justify-content: flex-start`. Sur l'axe secondaire (ici vertical) on veut que les flex-items se retrouvent en bas du flex-containeur `align-items: flex-end`.

Rendu :



Fig.48

Ce qui fonctionne parfaitement lorsqu'on a une seule ligne de flex-items, n'est pas tout à fait satisfaisant. Par défaut les trois rangées sont réparties surtout la hauteur. Ici, `align-items` est peu concluant. On va donc utiliser `align-content` pour piloter l'affichage des flex-items. Le CSS :

```
.flex-container {  
    display: flex;  
    flex-flow: row wrap;  
    justify-content: flex-start;  
    align-items: flex-end;  
    align-content: flex-end;  
    height: 80vh;  
}
```

Fig.49

Le rendu :

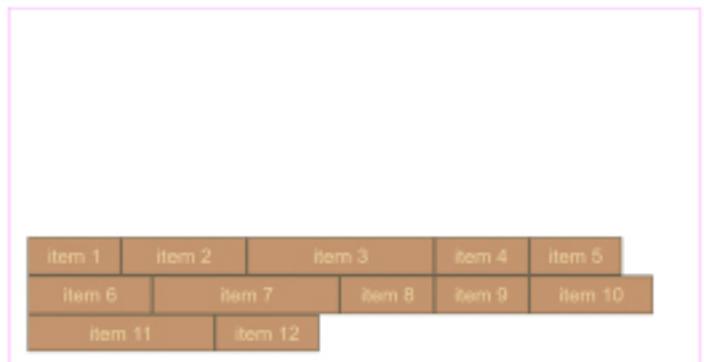


Fig.50

Je vous invite à faire l'expérience de modifier les valeurs de `align-items`. Vous observerez qu'elles ne sont d'aucune utilité dans cette situation précise. Cela ne serait pas le cas si l'axe principal était vertical.

Récapitulatif :

`align-content: flex-start | flex-end | center | space-between | space-around | stretch;`

III. Les règles des flex-items

L'une des particularités du modèle flexbox est que les règles peuvent entrer en conflit les unes avec les autres ou ne pas fonctionner dans certains cas. Il faut donc procéder par ordre, d'abord s'intéresser au flex-container, puis à l'alignement sur l'axe principal, ensuite sur l'axe secondaire et si celui-ci contient plus d'une rangée ou plus d'une colonne aux comportements de ces rangées ou colonnes. Ensuite seulement, on va s'intéresser aux flex-items.

Nous allons passer en revue rapidement les trois principales familles de règles des flex-items : la distribution différentielle, les tailles, les agrandissements-réductions, et enfin l'ordonnancement.

A. Distribution différentielle des flex-items

On peut demander à un flex-items de ne pas se comporter comme les autres sur l'axe secondaire. Prenons l'exemple suivant :

```
<nav class="main-nav">
  <ul class="flex-container">
    <li class="flex-item"><a href="#">item 1</a></li>
    <li class="flex-item big"><a href="#">item 2 </a></li>
    <li class="flex-item bigger"><a href="#">item 3</a></li>
    <li class="flex-item"><a href="#">item 4</a></li>
  </ul>
</nav>
```

Fig.51

Le CSS :

```
.flex-container {
  display: flex;
  flex-flow: row nowrap;
  justify-content: space-between;
  align-items: flex-end;
  height: 50vh;
}

.flex-item {
  border: 1px solid #625B48;
  padding: 5px 15px;
  background-color: #AE8964;
}
```

Fig.52

Le rendu :



Fig.53

Les éléments sont placés en bas de bloc sur l'axe secondaire via `align-items: flex-end`. Grâce aux deux classes `.big` et `.bigger` placées sur les flex-items 2 et 3, on va pouvoir modifier leur comportement, par exemple :

```
.big {  
    padding: 8px 20px;  
    height: 25px;  
    align-self: flex-start;  
}  
.bigger {  
    padding: 32px 24px;  
    height: 50px;  
    align-self: center;  
}
```

Fig.54

Le rendu :

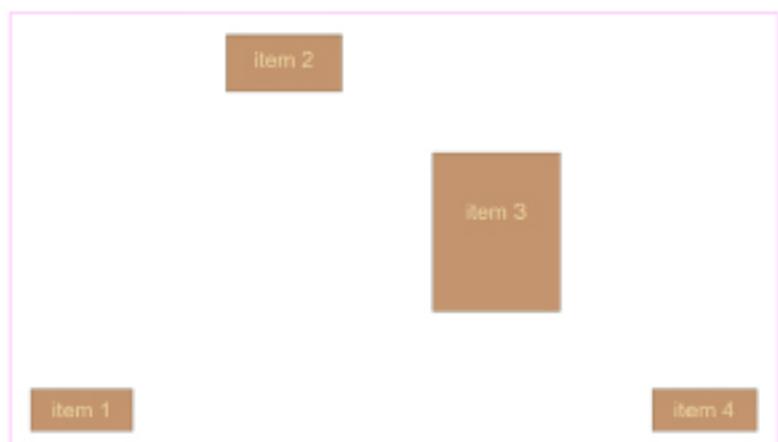


Fig.55

Si on modifie l'axe principal et secondaire en demandant des colonnes avec `flex-flow: column nowrap`; et que l'on demande passe la valeur d'`align-self` sur `stretch`, le résultat est le suivant :

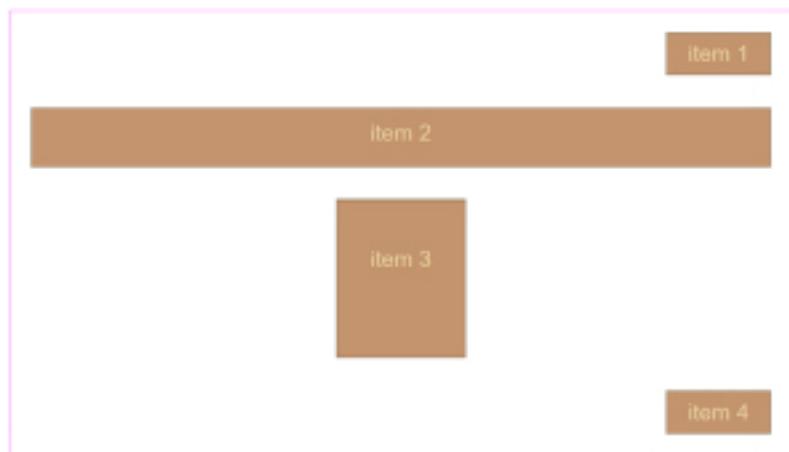


Fig.56

Si vous ne l'aviez pas encore constaté, les valeurs d'`align-self` sont les mêmes que celles de `align-items`, excepté la valeur `auto` (par défaut). Il s'agit simplement d'individualiser le comportement du flex-item.

Récapitulatif :

`align-self: auto | flex-start | flex-end | center | baseline | stretch;`

B. Taille des flexs-items

Le modèle flexbox permet non seulement d'indiquer la taille par défaut d'un flex-item mais également de piloter leur rétrécissement ou leur agrandissement selon la taille disponible.

1. Taille par défaut

On utilise `flex-basis` pour indiquer la taille par défaut. Par exemple :

```
.flex-container {  
    display: flex;  
    flex-flow: row nowrap;  
    justify-content: space-between;  
    align-items: flex-end;  
    height: 50vh;  
}  
  
.flex-item {  
    border: 1px solid #625B48;  
    padding: 5px 15px;  
    background-color: #AE8964;  
}  
.bigger {  
    flex-basis: 210px;  
}
```

Fig.57

Ici, je demande à ce que la taille par défaut de mon troisième flex-item ait une largeur par défaut de 210px.

Le rendu :



Fig.58

Cette valeur par défaut ne veut pas dire que notre élément conserve cette largeur. Une des propriétés des flex-items étant de s'adapter à la place qu'il dispose, ainsi si je réduis la fenêtre :

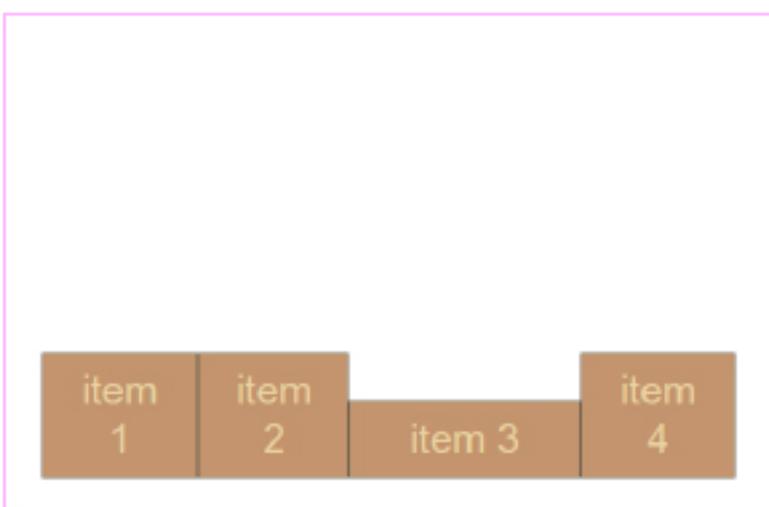


Fig.59

La largeur du flex-item est réduite.

Notez que `width` ne modifiera pas ce comportement, contrairement à `min-width` qui est prioritaire sur `flex-basis`. Ainsi :

```
.bigger {  
    flex-basis: 210px;  
    min-width: 210px;  
}
```

Fig.60

Donnera pour le même exemple (à fenêtre réduite) :



Fig.61

Récapitulatif :
flex-basis: auto | <largeur>;

2. Agrandissement

Lorsque tous les flex-items ont plus de place qu'ils en ont besoin, on peut piloter un ratio d'agrandissement. Regardons le code CSS suivant :

```
.flex-item {  
    border:1px solid #625B48;  
    padding: 5px 15px;  
    background-color: #AE8964;  
}  
  
.big {  
    flex-grow: 2;  
}  
  
.bigger {  
    flex-grow: 4;  
}
```

Fig.62

Le rendu :



Fig.63

Ici on remarque que le flex-item 3 (avec la class .bigger) est plus grand que le le flex-item 2 (avec la class .bigger), lui-même plus grand que les 2 autres. Comme il s'agit d'un ratio d'agrandissement (dont les calculs sont complexes), il fonctionne relativement aux autres. Si on indique un flex-grow de 1 pour les autres flex-items, comme ceci :

```
.flex-item {  
    border:1px solid #625B48;  
    padding: 5px 15px;  
    background-color: #AE8964;  
    flex-grow:1  
}
```

Fig.64

On aura :



Fig.65

Pourquoi ? Parce que `flex-grow:1` autorise le flex-item à prendre toute la place qu'il peut. Ainsi :

```
.flex-item {  
    border:1px solid #625B48;  
    padding: 5px 15px;  
    background-color: #AE8964;  
    flex-grow:1  
}  
  
.big {  
    flex-grow: 1;  
}  
  
.bigger {  
    flex-grow: 1;  
}
```

Fig.66

Donne :



Fig.67

Alors que :

```
.flex-item {  
    border:1px solid #625B48;  
    padding: 5px 15px;  
    background-color: #AE8964;  
    flex-grow:0;  
}  
  
.big {  
    flex-grow: 0;  
}  
  
.bigger {  
    flex-grow: 0;  
}
```

Fig.68

Donne :



Fig.69

En effet **flex-grow: 0** est la valeur par défaut.

Récapitulatif :

flex-grow: <chiffre>; [valeur par défaut : 0]

3. Rétrécissement

On observe peu ou prou la même chose s'agissant du rétrécissement. On va indiquer tout de même une valeur de base pour visualiser le ratio de réduction.

Ainsi :

```
.flex-item {  
    border:1px solid #625B48;  
    padding: 5px 15px;  
    background-color: #AE8964;  
    flex-shrink: 3;  
}  
  
.big {  
    /*rien*/  
}  
.bigger {  
    flex-shrink: 0;  
    flex-basis: 50%;  
}
```

Fig.70

On fait en sorte que `flex-item.bigger` soit bien plus grand que les autres, cela donne :



Fig.71

Et lorsqu'on réduit la fenêtre :

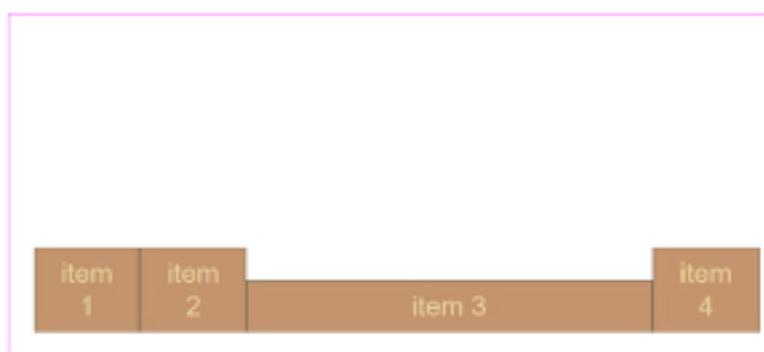


Fig.72

Sachant que la valeur par défaut est 1.

Récapitulatif :

`flex-shrink <chiffre>; [valeur par défaut : 1]`

4. Raccourcis

Il est plus intéressant de travailler avec le raccourci regroupant les trois valeurs (taille, agrandissement et réduction) tant elles sont liées dans le calcul de l'espace occupé.

Ainsi on pourra avoir :

```
.flex-item {  
    border:1px solid #625B48;  
    padding: 5px 15px;  
    background-color: #AE8964;  
    flex-shrink: 3;  
    flex: 0 2 auto;  
}  
  
.big {  
    flex: 3 1 auto;  
}  
.bigger {  
    flex: 4 1 40%;  
}
```

Fig.73

Qui donne (grande fenêtre) :



Fig.74

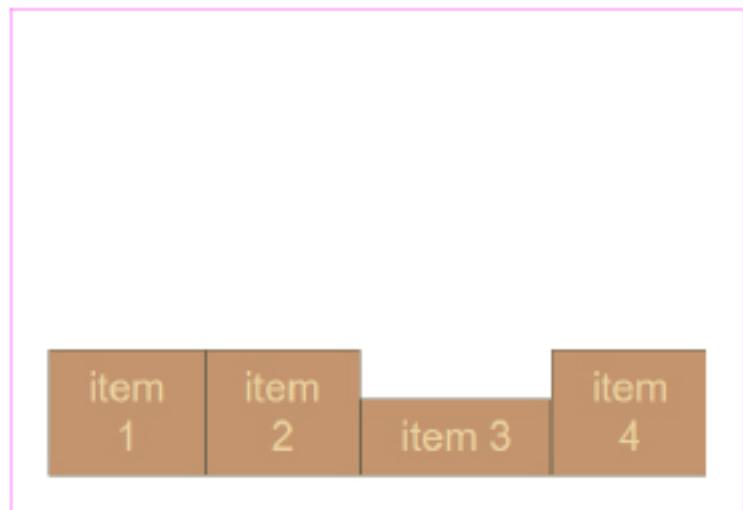


Fig.75

Récapitulatif:

`flex: flex-grow flex-shrink flex-basis | none [valeur par défaut: 0 1 autos]`

C. L'ordre des éléments

Une dernière propriété du modèle flexbox, très à simple à appréhender, concerne l'ordre des éléments. Nous avons déjà observé qu'il était possible d'inverser l'ordre des flex-items à partir du flex-container en utilisant `flex-direction: row-reverse` (inversion du flux horizontal), `flex-direction: column-reverse` (inversion du flux vertical) ou `flex-wrap: wrap-reverse` (inversion des rangées lorsqu'on en a plus d'une).

Mais flexbox nous permet également de piloter beaucoup plus finement l'ordre via la propriété `order`. Pour simplifier la démonstration on va ajouter des classes au HTML pour expliciter l'ordonnancement:

```
<nav class="main-nav">
  <ul class="flex-container">
    <li class="flex-item un"><a href="#">item 1</a></li>
    <li class="flex-item deux"><a href="#">item 2 </a></li>
    <li class="flex-item trois"><a href="#">item 3</a></li>
    <li class="flex-item quatre"><a href="#">item 4</a></li>
  </ul>
</nav>
```

Fig.76

On va piloter très facilement les éléments :

```
.flex-container {  
    display: flex;  
    flex-flow: wrap nowrap;  
    justify-content: center;  
    min-height: 30vh;  
}  
.un {  
    order: 2;  
}  
.deux {  
    order: 1;  
}  
.trois {  
    order: 4;  
}  
.quatre {  
    order: 3;  
}
```

Fig.77

Le résultat :

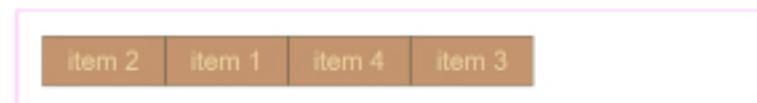


Fig.78

Pardéfaut la valeur ordre d'un flex-item est égale à zéro. Plus le chiffre est petit plus l'élément se trouve disposé sur la gauche (si les diverses règles d'inversion vues en début de cette section ne sont pas utilisées), plus le chiffre est élevé et plus l'élément se trouve déplacé vers la droite. Bien sûr cela fonctionne également pour une présentation en colonnes. Par exemple :

```
.flex-container {  
    display: flex;  
    flex-flow: column nowrap;  
    /*on passe en colonne*/  
    align-items: center;  
    /*on centre sur l'axe secondaire*/  
    min-height: 30vh;  
}  
.un {  
    order: 2;  
}  
.deux {  
    order: 1;  
}  
.trois {  
    order: 4;  
}  
.quatre {  
    /* rien */  
}
```

Fig.79

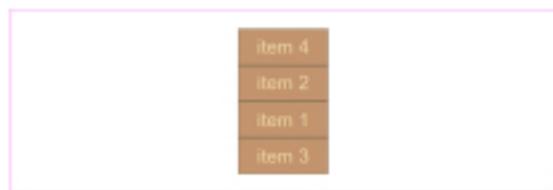


Fig. 80

N'ayant indiqué aucune valeur au flex-item quatre, celui-ci apparaît en premier, en effet la valeur de la propriété `order` est égale à zéro.

Dernière petite précision, il est possible d'utiliser des chiffres négatifs.

Récapitulatif

`order: <chiffre positif ou négatif>; [valeur par défaut : 0]`

IV. Conclusion

Le modèle flexbox est un système de mise en page à une dimension. Etant complexe à maîtriser, il est préférable dans un premier temps de travailler sur les règles du flex-container et de n'user des règles sur les flex-items qu'avec parcimonie. Néanmoins, il permet de résoudre de nombreuses problématiques de mise en page jusqu'alors compliquées à mettre en œuvre. En attendant que le *CSS Grid Layout* soit déployé, qui, lui, offre un système de mise en page à deux dimensions, on peut déjà préconiser le modèle flexbox comme choix préférentiel de mise en page.