AP1 : prise en main d'un site WEB existant, ajout de contenu et développement personnel

B1.3 Développer la présence en ligne
B1.6 Organiser son développement personnel

Objet du document : Apprentissage des éléments principaux de HTML 5 et CSS 3 afin de pouvoir réaliser un site web

Prérequis : Editeur de texte Notepad++ ou VS code Navigateur Firefox et Chrome

Chapitre 18 : Faites votre mise en page avec Flexbox

Il y a plusieurs façons de mettre en page un site. Au fil du temps, plusieurs techniques ont existé :

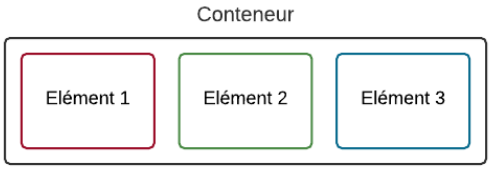
1. Au début, les webmasters utilisaient des tableaux HTML pour faire la mise en page (berk).
2. Puis, CSS est apparu et on a commencé à faire une mise en page à l'aide de la propriété float (bof).
3. Cette technique avait des inconvénients. Une autre, plus pratique, a consisté à créer des éléments de type inline-block sur la page (mouais).
4. Aujourd'hui, une bien meilleure technique encore existe : **Flexbox** ! Elle permet toutes les folies (ou presque), et c'est celle que je vous recommande d'utiliser si vous en avez la possibilité, lorsque vous créez un nouveau site. Flexbox est désormais reconnu par tous les navigateurs récents !

Un conteneur, des éléments

Le principe de la mise en page avec Flexbox est simple : vous définissez un conteneur, et à l'intérieur vous placez plusieurs éléments. Imaginez un carton dans lequel vous rangez plusieurs objets : c'est le principe !

Sur une même page web, vous pouvez sans problème avoir plusieurs conteneurs (plusieurs cartons, si vous préférez). Ce sera à vous d'en créer autant que nécessaire pour obtenir la mise en page que vous voulez.

Commençons par étudier le fonctionnement d'un carton (euh pardon, d'un conteneur).



Un conteneur et ses éléments

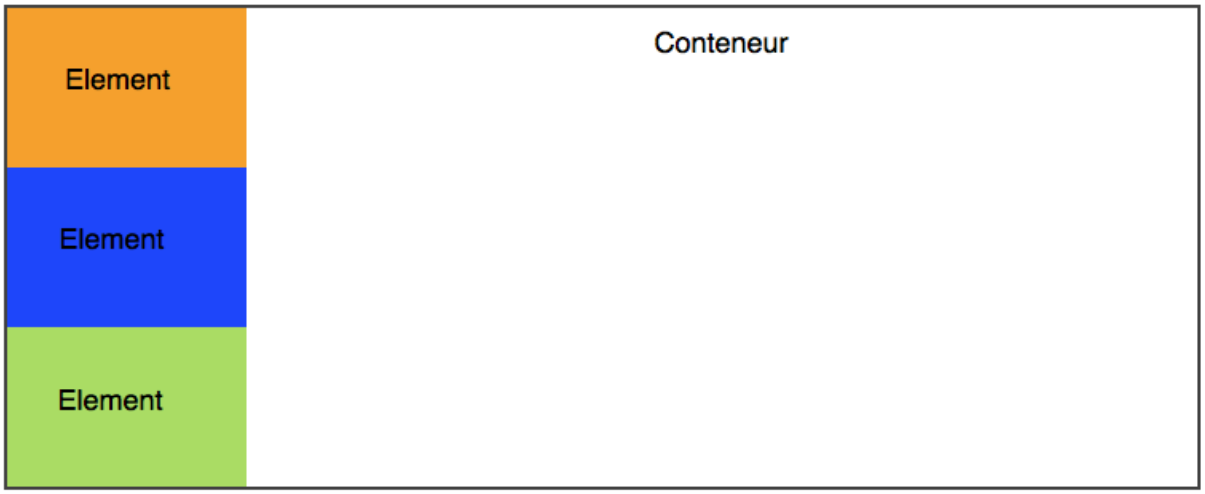
Le conteneur est une balise HTML, et les éléments sont d'autres balises HTML à l'intérieur :

```
<div id="conteneur">
    <div class="element 1">Element </div>
    <div class="element 2">Element </div>
    <div class="element 3">Element </div>
</div>
```

OK, jusque-là, vous devriez suivre.

Mais si je fais ça, par défaut, mes éléments vont se mettre les uns en dessous des autres, non ? Ce sont des blocs, après tout !

Oui, tout à fait. Si je mets une bordure au conteneur, une taille et une couleur de fond aux éléments, on va vite voir comment ils s'organisent :



Par défaut, les blocs se placent les uns en dessous des autres

Rien de bien nouveau, c'est le comportement normal dont nous avons l'habitude.

Je vous montrerai un peu plus tard comment attribuer une couleur et une bordure différente aux éléments avec la pseudo-classe :nth-child. Pour l'instant, j'ai mis de la couleur sur chacun des éléments, pour que vous puissiez les distinguer facilement dans mon exemple !

Soyez flex !

Découvrons maintenant Flexbox. Si je mets une (une seule !) propriété CSS, tout change. Cette propriété, c'est `flex` , et je l'applique au conteneur :

```
#conteneur
{
    display: flex;
}
```

... alors les blocs se placent par défaut côte à côte. Magique !

Un coup de flex, et les blocs se positionnent côte à côte !

La direction

Flexbox nous permet d'agencer ces éléments dans le sens que l'on veut.
Avec `flex-direction` , on peut les positionner verticalement ou encore les inverser. Il peut prendre les valeurs suivantes :

- row : organisés sur une ligne (par défaut) ;
- column : organisés sur une colonne ;
- row-reverse : organisés sur une ligne, mais en ordre inversé ;
- column-reverse : organisés sur une colonne, mais en ordre inversé.

Exemple :

```
#conteneur
{
    display: flex;
    flex-direction: column;
}
```

Les éléments sont disposés en colonne

Mais mais... c'est pareil qu'au début, non ? On avait ce résultat sans Flexbox, après tout !

C'est vrai. Mais maintenant que nos éléments sont *flex*, ils ont tout un tas d'autres propriétés utiles que nous allons voir juste après, on va y revenir.

Essayez aussi de tester l'ordre inversé, pour voir :

```
#conteneur
{
    display: flex;
    flex-direction: column-reverse;
}
```

Les éléments sont en colonne... dans l'ordre inverse !

Regardez bien la différence : les blocs sont maintenant dans l'ordre inverse ! Je n'ai pas du tout changé le code HTML, qui reste le même depuis le début.

Le retour à la ligne

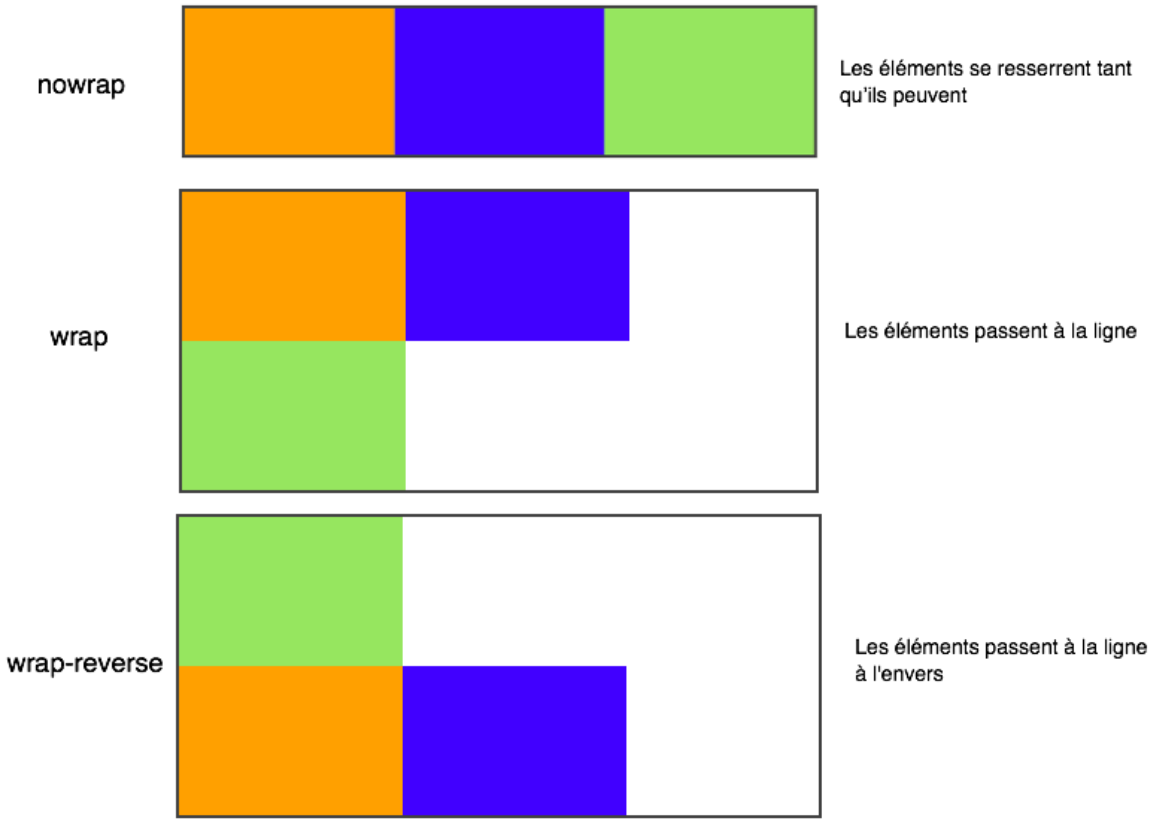
Par défaut, les blocs essaient de rester sur la même ligne s'ils n'ont pas la place (ce qui peut provoquer des bugs de design, parfois). Si vous voulez, vous pouvez demander à ce que les blocs aillent à la ligne lorsqu'ils n'ont plus la place, avec `flex-wrap` qui peut prendre ces valeurs :

- nowrap : pas de retour à la ligne (par défaut) ;
- wrap : les éléments vont à la ligne lorsqu'il n'y a plus la place ;
- wrap-reverse : les éléments vont à la ligne, lorsqu'il n'y a plus la place, en sens inverse.

Exemple :

```
#conteneur
{
    display: flex;
    flex-wrap: wrap;
}
```

Voici l'effet que prennent les différentes valeurs sur une même illustration :



Gestion du retour à la ligne avec flex-wrap

Alignez-les !

Reprenons. Les éléments sont organisés soit horizontalement (par défaut), soit verticalement. Cela définit ce qu'on appelle **l'*axe principal***. Il y a aussi un axe secondaire (*cross axis*) :

- si vos éléments sont organisés horizontalement, l'axe secondaire est l'axe vertical ;
- si vos éléments sont organisés verticalement, l'axe secondaire est l'axe horizontal.

Pourquoi je vous raconte ça ? Parce que nous allons découvrir comment aligner nos éléments sur l'axe principal *et* sur l'axe secondaire.

Aligner sur l'axe principal

Pour faire simple, partons sur des éléments organisés horizontalement (c'est le cas par défaut).

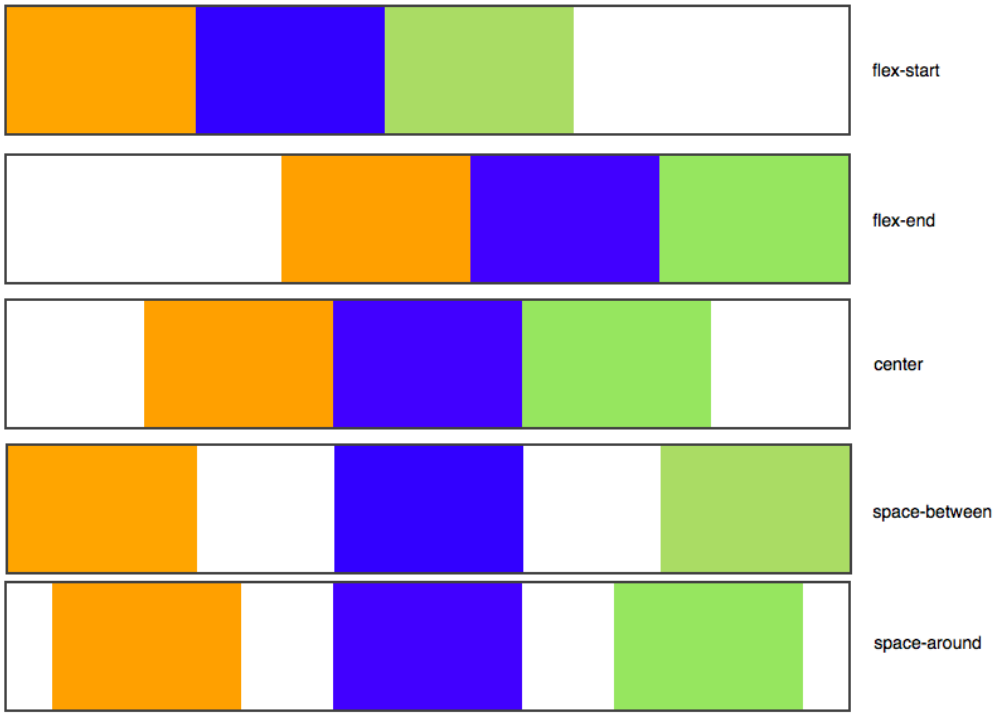
Pour changer leur alignement, on va utiliser `justify-content` , qui peut prendre ces valeurs :

- flex-start : alignés au début (par défaut) ;
- flex-end : alignés à la fin ;
- center : alignés au centre ;
- space-between : les éléments sont étirés sur tout l'axe (il y a de l'espace entre eux) ;
- space-around : idem, les éléments sont étirés sur tout l'axe, mais ils laissent aussi de l'espace sur les extrémités.

Par exemple :

```
#conteneur
{
    display: flex;
    justify-content: space-around;
}
```

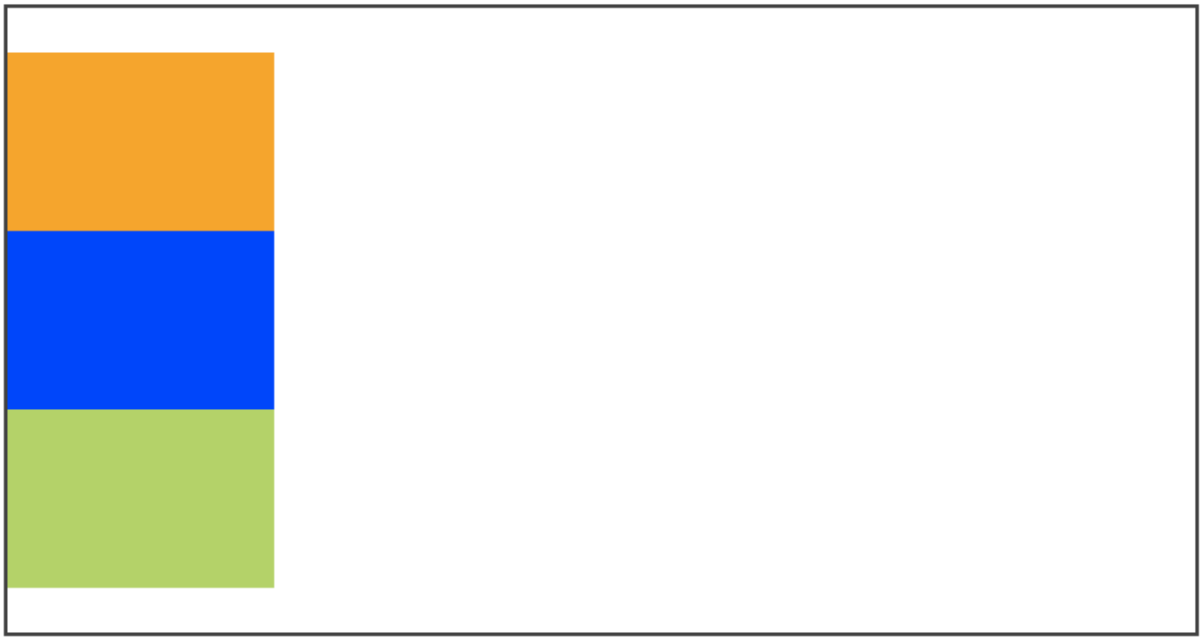
Le mieux est encore de tester toutes les valeurs possibles pour voir ce que ça donne, vous pensez pas ?



Les différentes valeurs possibles pour l'alignement avec justify-content
Vous voyez comment les éléments s'alignent différemment selon les cas ? Avec une simple propriété, on peut intelligemment agencer nos éléments comme on veut !

Maintenant, voici ce qu'il faut bien comprendre : **ça marche aussi si vos éléments sont dans une direction verticale**. Dans ce cas, l'axe vertical devient l'axe principal, et `justify-content` s'applique aussi :

```
#conteneur
{
    display: flex;
    flex-direction: column;
    justify-content: center;
    height: 350px; /* Un peu de hauteur pour que les éléments aient la place de bouger */
}
```



Avec une direction verticale (column), le centrage fonctionne de la même façon, cette fois en hauteur !

Aligner sur l'axe secondaire

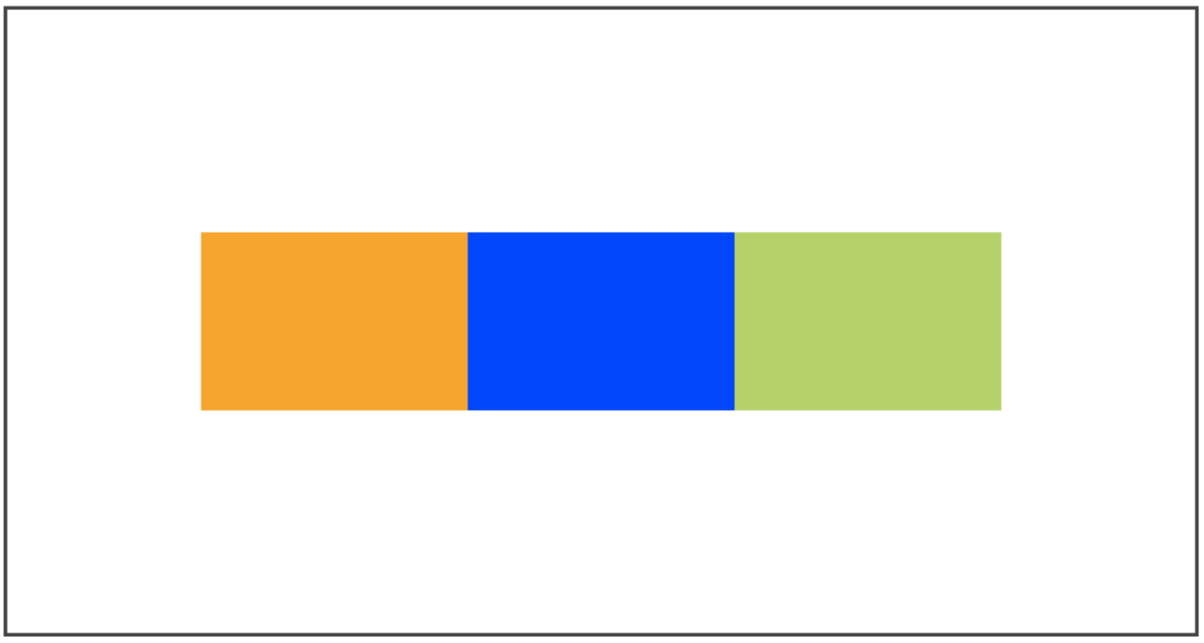
Comme je vous le disais, si nos éléments sont placés dans une direction horizontale (ligne), l'axe secondaire est... vertical. Et inversement : si nos éléments sont dans une direction verticale (colonne), l'axe secondaire est horizontal.

Avec `align-items` , nous pouvons changer leur alignement sur l'axe secondaire. Il peut prendre ces valeurs :

- stretch : les éléments sont étirés sur tout l'axe (valeur par défaut) ;
- flex-start : alignés au début ;
- flex-end : alignés à la fin ;
- center : alignés au centre ;
- baseline : alignés sur la ligne de base (semblable à flex-start).

Pour ces exemples, nous allons partir du principe que nos éléments sont dans une direction horizontale (mais n'hésitez pas à tester aussi dans la direction verticale !).

```
#conteneur
{
    display: flex;
    justify-content: center;
    align-items: center;
}
```



Un alignement sur l'axe secondaire avec align-items nous permet de centrer complètement l'élément dans le conteneur !

Saint Graal du développeur web, le centrage vertical et horizontal peut d'ailleurs être obtenu encore plus facilement. Dites que votre conteneur est une flexbox et établissez des marges automatiques sur les éléments à l'intérieur. C'est tout ! Essayez !

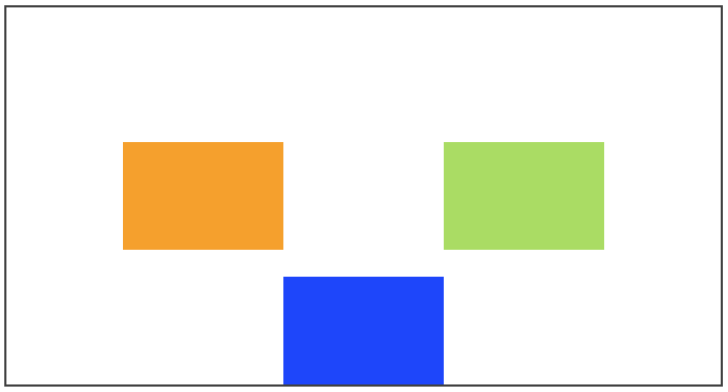
```
#conteneur
{
    display: flex;
}

.element
{
    margin: auto;
}
```

Aligner un seul élément

Il est possible de faire une exception pour un seul des éléments sur l'axe secondaire avec `align-self` :

```
#conteneur
{
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;
}
.element:nth-child(2) /* On prend le deuxième bloc élément */
{
    background-color: blue;
    align-self: flex-end; /* Seul ce bloc sera aligné à la fin */
}
```



Un élément aligné différemment des autres avec align-self. Tiens, je crois que j'ai dessiné une tête en pixel art !

Répartir plusieurs lignes

Si vous avez plusieurs lignes dans votre Flexbox, vous pouvez choisir comment celles-ci seront réparties avec `align-content` .

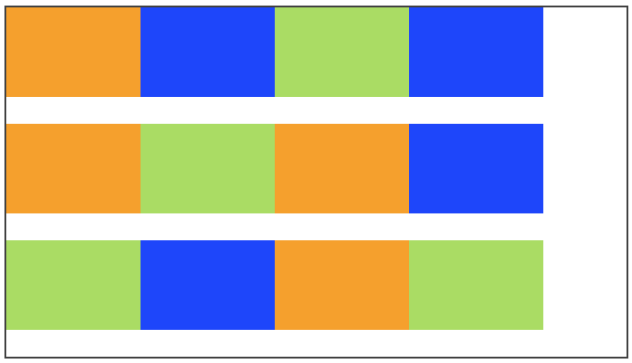
Cette propriété n'a aucun effet s'il n'y a qu'une seule ligne dans la Flexbox.

Prenons donc un cas de figure où nous avons plusieurs lignes. Je vais rajouter des éléments :

```
<div id="conteneur">
    <div class="element"></div>
    <div class="element"></div>
    <div class="element"></div>
    <div class="element"></div>
    <div class="element"></div>
    <div class="element"></div>
    <div class="element"></div>
    <div class="element"></div>
    <div class="element"></div>
    <div class="element"></div>
    <div class="element"></div>
    <div class="element"></div>
</div>
```

J'autorise mes éléments à aller à la ligne avec `flex-wrap` :

```
#conteneur
{
    display: flex;
    flex-wrap: wrap;
}
```

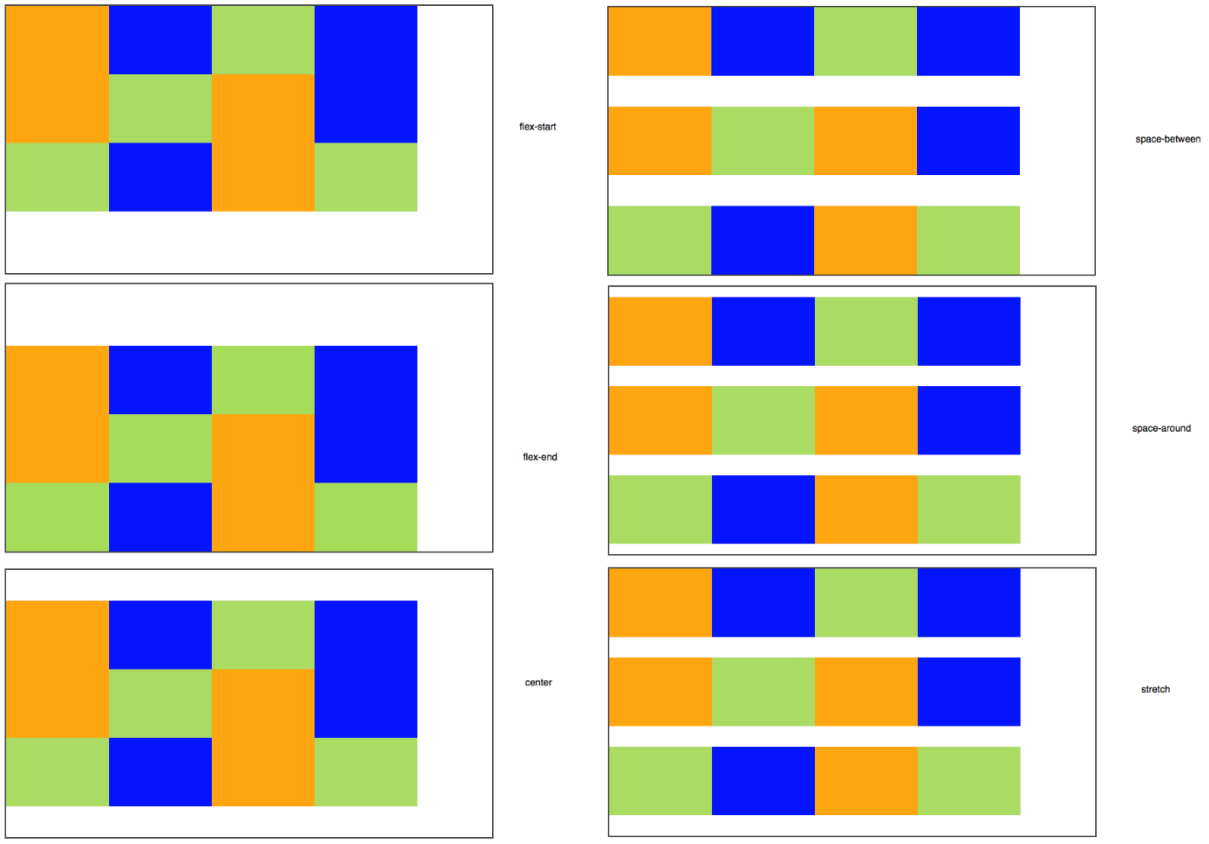


Plusieurs lignes dans une Flexbox

Jusque là, rien de vraiment nouveau. Voyons voir comment les lignes se répartissent différemment avec la nouvelle propriété `align-content` que je voulais vous présenter. Elle peut prendre ces valeurs :

- flex-start : les éléments sont placés au début ;
- flex-end : les éléments sont placés à la fin ;
- center : les éléments sont placés au centre ;
- space-between : les éléments sont séparés avec de l'espace entre eux ;
- space-around : idem, mais il y a aussi de l'espace au début et à la fin ;
- stretch (par défaut) : les éléments s'étirent pour occuper tout l'espace.

Voici ce que donnent les différentes valeurs :



Les lignes sont placées différemment avec align-content

Rappel à l'ordre

Sans changer le code HTML, nous pouvons modifier l'ordre des éléments en CSS grâce à la propriété `order` . Indiquez simplement un nombre, et les éléments seront triés du plus petit au plus grand nombre.

Reprenons une simple ligne de 3 éléments :

```
#conteneur
{
    display: flex;
}
```


Une ligne de 3 éléments

Si je dis que le premier élément sera placé en 3e position, le second en 1re position et le troisième en 2de position, l'ordre à l'écran change !

```
.element:nth-child(1)
{
    order: 3;
}
.element:nth-child(2)
{
    order: 1;
}
.element:nth-child(3)
{
    order: 2;
}
```

Avec order, nous pouvons réordonner les éléments en CSS

Encore plus flex : faire grossir ou maigrir les éléments

Allez, encore une dernière technique, après on passe à la pratique.

Avec la propriété `flex` , nous pouvons permettre à un élément de grossir pour occuper tout l'espace restant.

```
.element:nth-child(2)
{
    flex: 1;
}
```

Le deuxième élément s'étire pour prendre tout l'espace

Le nombre que vous indiquez à la propriété flex indique dans quelle mesure il peut grossir par rapport aux autres.

```
.element:nth-child(1)
{
    flex: 2;
}
.element:nth-child(2)
{
    flex: 1;
}
```

Ici, le premier élément peut grossir 2 fois plus que le deuxième élément :

Le premier élément peut grossir deux fois plus que le deuxième élément

La propriété `flex` est en fait une super-propriété qui combine `flex-grow` (capacité à grossir), `flex-shrink` (capacité à maigrir) et `flex-basis` (taille par défaut). J'utilise simplement flex comme je vous l'ai montré ici, mais si vous voulez en savoir plus, je vous invite à vous renseigner sur ces autres propriétés.

Deux liens pour apprendre en s'amusant !

https://flexboxfroggy.com/#fr

https://cssgridgarden.com/#fr

EN RÉSUMÉ : NOTEZ ICI LES POINTS IMPORTANTS DU CHAPITRE