**LE MODÈLE DES BOÎTES**

* Une page web peut être vue comme une succession et un empilement de boîtes, qu'on appelle « **blocs** ». La plupart des éléments vus au chapitre précédent sont des blocs : <**header**> , <**article**> , <**nav**> … Mais nous connaissions déjà d'autres blocs : les paragraphes <**p**> , les titres <**h1**> …
* Dans ce chapitre, nous allons apprendre à manipuler ces blocs comme de véritables boîtes. Nous allons leur donner des dimensions, les agencer en jouant sur leurs marges, mais aussi apprendre à gérer leur contenu… pour éviter que le texte ne dépasse de ces blocs.
* Ce sont des notions fondamentales dont nous allons avoir besoin pour mettre en page notre site web.

**LES BALISES DE TYPE BLOCK ET INLINE :**

* En HTML, la plupart des balises peuvent se ranger dans l'une ou l'autre des deux catégories :

Les balises **inline** : c'est le cas par exemple des liens <**a**></**a**>.

Les balises **block** : c'est le cas par exemple des paragraphes <**p**></**p**>.

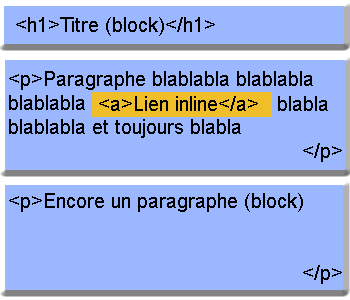
* Il existe en fait plusieurs autres catégories très spécifiques, par exemple pour les cellules de tableau (type **table-cell**  ) ou les puces (type **list-item**  ). Nous n'allons pas nous y intéresser pour le moment, car ces balises sont minoritaires.

Comment reconnaître une balise **inline** d’une balise **block** ?

**Block** : Une balise de type **block** sur sa page web crée automatiquement un retour à la ligne avant et après. Il suffit d'imaginer tout simplement un bloc. Notre page web sera en fait constituée d'une série de blocs les uns à la suite des autres. Mais on verra qu'en plus, il est possible de mettre un bloc à l'intérieur d'un autre, ce qui va augmenter considérablement nos possibilités pour créer le design de notre site.

**Inline** : Une balise de type **inline** se trouve obligatoirement à l'intérieur d'une balise **block**. Une balise **inline** ne crée pas de retour à la ligne, le texte qui se trouve à l'intérieur s'écrit donc à la suite du texte précédent, sur la même ligne (c'est pour cela que l'on parle de balise « **en ligne** »).

* Depuis **HTML5,** la catégorisation des différents éléments est un peu plus complexe que cela. Cependant, cette petite simplification va vous permettre de bien comprendre la différence entre le concept de "**bloc**" et le concept de "**en ligne**".
* Pour bien visualiser le concept, voici en figure suivante un petit schéma.



* Sur fond bleu, tout ce qui est de type **block**.
* Sur fond jaune, tout ce qui est de type **inline**.

Comme on peut le voir, les blocs sont les uns en dessous des autres. On peut aussi les imbriquer les uns à l'intérieur des autres (se souvenir que nos blocs <**section**> contiennent par exemple des blocs <**aside**>).

La balise **inline** <**a**></**a**> , elle, se trouve à l'intérieur d'une balise **block** et le texte vient s'insérer sur la même ligne.

#### **Quelques exemples**

* Afin de mieux aider à assimiler quelles balises sont inline et quelles balises sont **block**, voici un petit tableau dressant la liste de quelques balises courantes.

| **Balises block** | **Balises block** |
| --- | --- |
| <p> | <em> |
| <footer> | <strong> |
| <h1> | <mark> |
| <h2> | <a> |
| <article> | <img /> |
| ... | ... |

Ce tableau n'est pas complet, loin de là. Si l’on veut avoir la liste complète des balises qui existent et savoir si elles sont de type **inline** ou **block**, il faut se reporter à l'[annexe donnant la liste des balises HTML](https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3/memento-des-balises-html).

#### **Les balises universelles**

* On les connaît déjà car elles ont été présentées il y a quelques chapitres. Ce sont des balises qui n'ont aucun sens particulier (contrairement à <**p**> qui veut dire « paragraphe » ,<**strong**> , « import‌ant », etc.).
* Le principal intérêt de ces balises est que l'on peut leur appliquer une class (ou un id ) pour le CSS quand aucune autre balise ne convient.
* Il existe deux balises génériques et, comme par hasard, la seule différence entre les deux est que l'une d'elle est **inline** et l'autre est **block** :

<**span**></**span**> : **inline** ;

<**div**></**div**> : **block** .

#### **Respectez la sémantique**

* Les balises universelles sont « pratiques » dans certains cas, certes, mais attention à ne pas en abuser. Je tiens à en avertir de suite : beaucoup de webmasters mettent des <**div**> et des <**span**> trop souvent et oublient que d'autres balises plus adaptées existent.

Voici deux exemples :

**Exemple d'un span inutile** : <**span** class="important"> . Je ne doit jamais voir ceci dans un de vos codes, alors qu'il existe la balise <**strong**> qui sert à indiquer l'importance.

**Exemple d'un div inutile** : <**div** class="titre"> . Ceci est complètement absurde puisqu'il existe des balises faites spécialement pour les titres (<**h1**>, <**h2**> …).

* Oui, on peut dire qu'au final le résultat (visuel) est le même. Je suis tout à fait d'accord. Mais les balises génériques n'apportent aucun sens à la page et ne peuvent pas être comprises par l'ordinateur. Il faut utiliser toujours d'autres balises plus adaptées quand c'est possible. Google lui-même le conseille pour vous aider à améliorer la position de vos pages au sein de ses résultats de recherche.

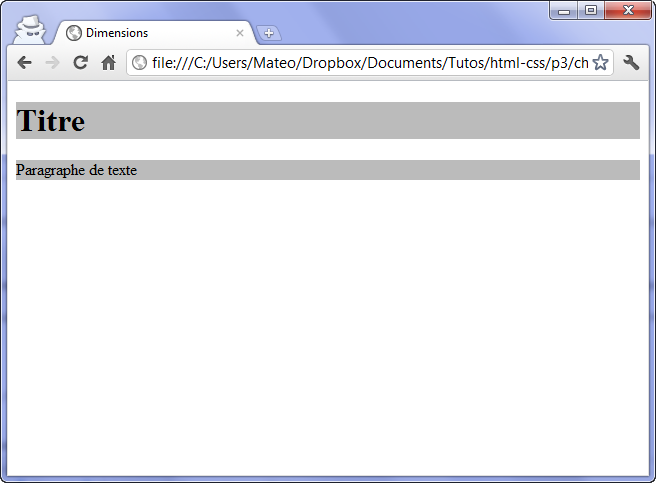
**Les Dimensions**

* Ici le travaille sera uniquement sur des balises de type **block**.
* Pour commencer, intéressons-nous à la taille des blocs. Contrairement à un **inline**, un **bloc** a des dimensions précises. Il possède une largeur et une hauteur. Ce qui fait, ô surprise, qu'on dispose de deux propriétés CSS :

**width** : C'est la largeur du bloc, à exprimer en pixels (**px**) ou en pourcentage (**%**) ;

**height** : C'est la hauteur du bloc, là encore on l'exprime soit en pixels (**px**), soit en pourcentage (**%**).

* Pour être exact, **width** et **height**  représentent la largeur et la hauteur du contenu des blocs. Si le bloc a des marges (on va découvrir ce principe un peu plus loin), celles-ci s'ajouteront à la largeur et à la hauteur.
* Par défaut, un bloc prend **100 %** de la largeur disponible. On peut le vérifier en appliquant à nos blocs des bordures ou une couleur de fond (figure suivante).



* Maintenant, rajoutons un peu de CSS afin de modifier la largeur des paragraphes. Le CSS suivant dit : « Je veux que tous mes paragraphes aient une largeur de 50 % ».

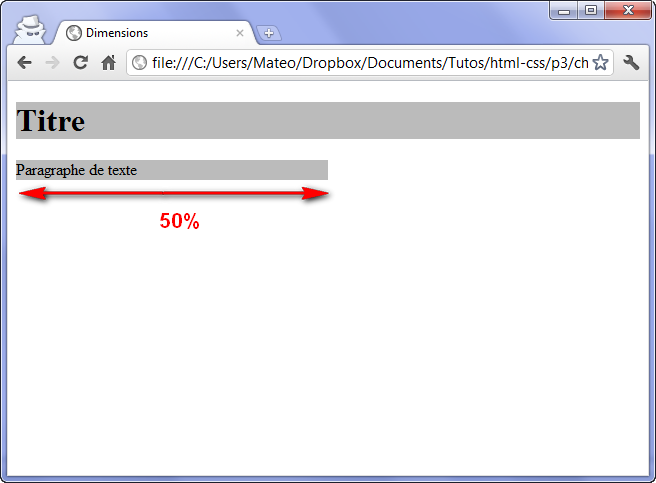
p

{

**width**: 50%;

}

Le résultat est visible à la figure suivante.



* Les pourcentages seront utiles pour créer un design qui s'adapte automatiquement à la résolution d'écran du visiteur.Toutefois, il se peut que vous ayez besoin de créer des blocs ayant une dimension précise en pixels :

p

{

**width**: 250px;

}

#### **Minimum et maximum**

* On peut demander à ce qu'un bloc ait des dimensions minimales et maximales. C'est très pratique, car cela nous permet de définir des dimensions « limites » pour que notre site s'adapte aux différentes résolutions d'écran de nos visiteurs :

**min-width** : Largeur minimale ;

**min-height** : Hauteur minimale ;

**max-width** : Largeur maximale ;

**max-height** : Hauteur maximale ;

* Par exemple, on peut demander à ce que les paragraphes occupent 50 % de la largeur *et* exiger qu'il fassent au moins 400 pixels de large dans tous les cas :

p

{

**width**: 50%;

**min-width**: 400px;

}

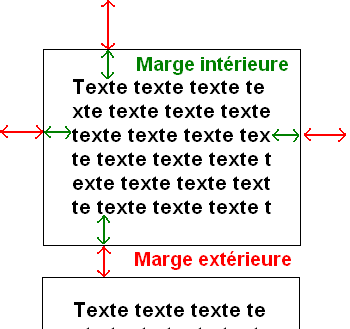
* Il faut observer le résultat en modifiant la largeur de la fenêtre de son navigateur. On va voir que, si celle-ci est trop petite, le paragraphe se force à occuper au moins 400 pixels de largeur.

**Les Marges**

* Il faut savoir que tous les blocs possèdent des **marges**. Il existe deux types de marges :

**Les marges intérieures** ;

**Les marges extérieures** .

* Il faut bien regarder le schéma qui se trouve à la figure suivante.
* Sur ce bloc, j’ai mis une bordure pour qu'on repère mieux ses frontières.

L'espace entre le texte et la bordure est la marge intérieure (**en vert**).

L'espace entre la bordure et le bloc suivant est la marge extérieure (**en rouge**).

* En CSS, on peut modifier la taille des marges avec les deux propriétés suivantes :

**padding** : Indique la taille de la marge intérieure. À exprimer en général en pixels (**px**) ;

**margin** : Indique la taille de la marge extérieure. Là encore, on utilise le plus souvent des pixels (**px**) .

Les balises de type inline possèdent également des marges. Vous pouvez donc aussi essayer ces manipulations sur ce type de balises.

* Pour bien voir les marges, prenons deux paragraphes auxquels j'applique simplement une petite bordure (figure suivante) :

p

{

**width**: 350px;

**border**: 1px solid black;

**text-align**: justify;

}



Marges par défaut sur les paragraphes.

* Comme on peut le constater, il n'y a par défaut pas de marge intérieure (**padding**). En revanche, il y a une marge extérieure (**margin**). C'est cette marge qui fait que deux paragraphes ne sont pas collés et qu'on a l'impression de « sauter une ligne ».

Les marges par défaut ne sont pas les mêmes pour toutes les balises de type block. Essayez d'appliquer ce CSS à des balises <div> qui contiennent du texte, par exemple : vous verrez que, dans ce cas, il n'y a par défaut ni marge intérieure, ni marge extérieure.

* Supposons que je veuille rajouter une marge intérieure de **12 px** aux paragraphes (figure suivante) :

p

{

width: 350px;

border: 1px solid black;

text-align: justify;

padding: 12px; /\* Marge intérieure de 12px \*/

}



* Maintenant, je veux que mes paragraphes soient plus espacés entre eux. Je rajoute la propriété **margin** pour demander à ce qu'il y ait **50 px** de marge entre deux paragraphes (figure suivante) :

p

{

width: 350px;

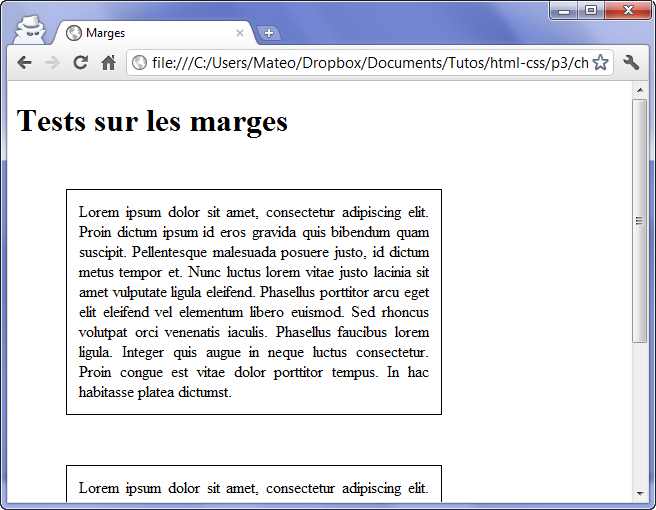
border: 1px solid black;

text-align: justify;

padding: 12px;

margin: 50px; /\* Marge extérieure de 50px \*/

}



* Mais ? Une marge s'est rajoutée à gauche aussi !
* Eh oui, **margin** (comme **padding** , d'ailleurs) s'applique aux quatre côtés du bloc.

Si l’on veut spécifier des marges différentes en haut, en bas, à gauche et à droite, il va falloir utiliser des propriétés plus précises… Le principe est le même que pour la propriété **border**, on va voir !

#### 

#### **En haut, à droite, en bas, à gauche… et on recommence**

* L'idéal serait de retenir les termes suivants en anglais :

**top** : haut ;

**bottom** : bas ;

**left** : gauche ;

**right** : droite.

* Ainsi, on peut retrouver toutes les propriétés de tête. On va quand même vous faire la liste des propriétés pour **margin** et **padding** , histoire que l’on soit sûr d'avoir compris le principe.
* Voici la liste pour **margin** :

**margin-top** : Marge extérieure en haut ;

**margin-bottom** : Marge extérieure en bas ;

**margin-left** : Marge extérieure à gauche ;

**margin-right** : Marge extérieure à droite .

* La liste pour **padding**  :

**padding-top** : Marge intérieure en haut ;

**padding-bottom** : Marge intérieure en bas ;

**padding-left**  : Marge intérieure à gauche ;

**padding-right** : Marge intérieure à droite.

* Il y a d'autres façons de spécifier les marges avec les propriétés margin et padding . Par exemple : margin: 2px 0 3px 1px; signifie « 2 px de marge en haut, 0 px à droite (le px est facultatif dans ce cas), 3 px en bas, 1 px à gauche ». Autre notation raccourcie : margin: 2px 1px; signifie « 2 px de marge en haut et en bas, 1 px de marge à gauche et à droite ».

**Centrer des blocs**

* Il est tout à fait possible de centrer des blocs. C'est même très pratique pour réaliser un design centré quand on ne connaît pas la résolution du visiteur.
* Pour centrer, il faut respecter les règles suivantes :

Donnez une largeur au bloc (avec la propriété **width**  ) ;

Indiquez que l’on veut des marges extérieures automatiques, comme ceci : **margin:** **auto;** .

* Essayons cette technique sur nos petits paragraphes (lignes 3 et 4) :

p

{

**width**: 350px; /\* On a indiqué une largeur (obligatoire) \*/

**margin**: auto; /\* On peut donc demander à ce que le bloc soit centré avec auto \*/

**border**: 1px solid black;

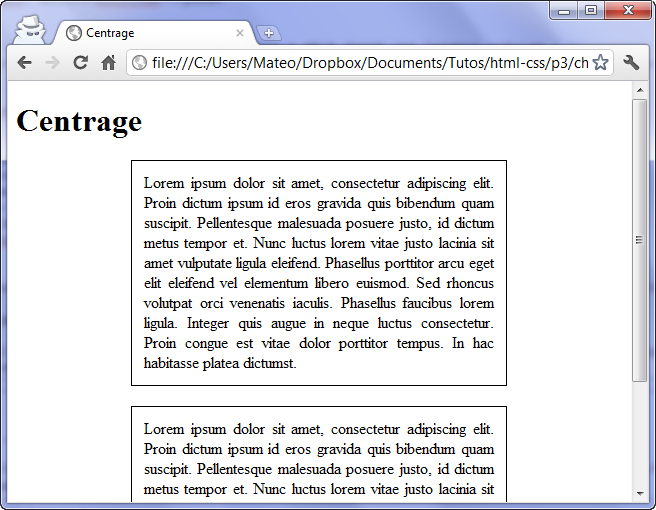
**text-align**: justify;

**padding**: 12px;

**margin-bottom**: 20px;

}

Et voici le résultat à la figure suivante.



Ainsi, le navigateur centre automatiquement nos paragraphes !

Il n'est cependant pas possible de centrer verticalement un bloc avec cette technique. Seul le centrage horizontal est permis.

**QUAND CA DEPASSE…**

* Lorsque l’on commence à définir des dimensions précises pour nos blocs, comme on vient de le faire, il arrive qu'ils deviennent trop petits pour le texte qu'ils contiennent. Les propriétés CSS que nous allons voir ici ont justement été créées pour contrôler les dépassements… et décider quoi faire si jamais cela devait arriver.

**overflow** : couper un bloc

* Si l’on a un long paragraphe et que l’on veut (pour une raison qui ne regarde que nous) qu’il fasse **250px** de large et **110px** de haut. Ajoutons-lui une bordure et remplissons-le de texte… à ras-bord (figure suivante) :

p

{

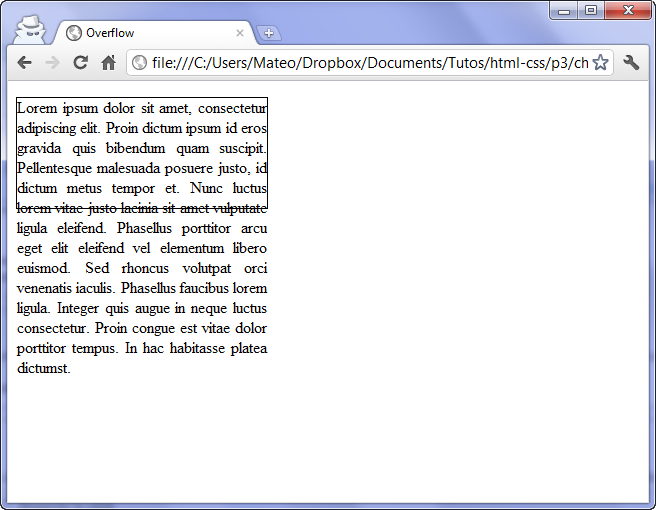
**width**: 250px;

**height**: 110px;

**text-align**: justify;

**border**: 1px solid black;

}



Horreur ! Le texte dépasse des limites du paragraphe !

* Si on a demandé des dimensions précises, on les a eues, mais le texte ne tient pas à l’intérieur d’un si petit bloc.
* Si l’on veut que le texte ne dépasse pas des limites du paragraphe, il va falloir utiliser la propriété **overflow**. Voici les valeurs qu'elle peut accepter :

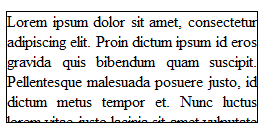
**visible** (par défaut) : Si le texte dépasse les limites de taille , il reste visible et sort volontairement du bloc ;

**hidden** : Si le texte dépasse les limites, il sera tout simplement coupé. On ne pourra pas voir tout le texte ;

**scroll** : Là encore, le texte sera coupé s'il dépasse les limites. Sauf que cette fois, le navigateur mettra en place des barres de défilement pour qu'on puisse lire l'ensemble du texte. C'est un peu comme un cadre à l'intérieur de la page.

**auto** : C’est le mode “pilote automatique”. En gros, c’est le nav qui décide de mettre ou non des barres de défilement (il n’en mettra que si c’est nécessaire). C’est la valeur qui est conseillée d’utiliser le plus souvent.

* Avec **overflow** : **hidden**; le texte est donc coupé (on ne peut pas voir la suite), comme sur la figure suivante : (le texte est coupé aux limites du paragraphe).



* Essayons maintenant overflow: auto; avec le code CSS suivant (résultat à la figure suivante) :

p

{

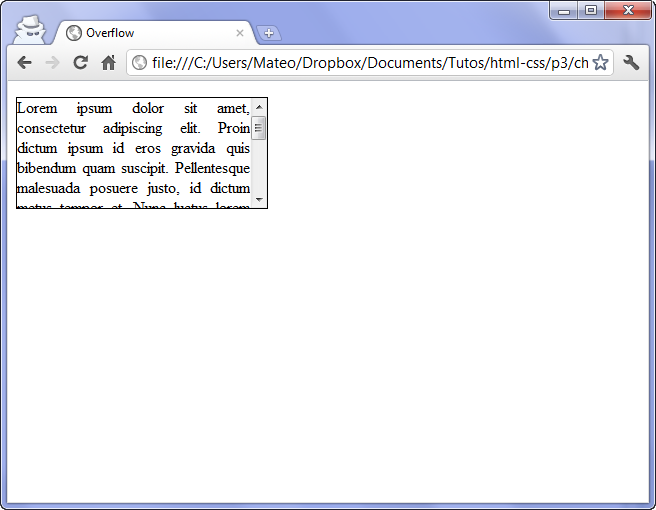
width: 250px;

height: 110px;

text-align: justify;

border: 1px solid black;

overflow: auto;

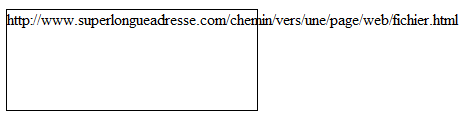
}

* (des barres de défilement sont ajoutées au paragraphe)
* Des barres de défilement nous permettent maintenant de consulter le contenu qui n'était pas visible.

Il existe une ancienne balise HTML, <**iframe**> , qui donne à peu près le même résultat. Cependant, l'usage de cette balise est déconseillé aujourd'hui. Elle permet de charger tout le contenu d'une autre page HTML au sein de votre page.

**word-wrap** : couper les textes trop larges

* Si l’on doit placer un mot très long dans un bloc, qui ne tient pas dans la largeur, on va adorer **word-wrap**. Cette propriété permet de forcer la césure des très longs mots (généralement des adresses un peu longues).
* La figure suivante représente ce que l'on peut avoir quand on écrit un URL un peu long dans un bloc. (le texte déborde en largeur)



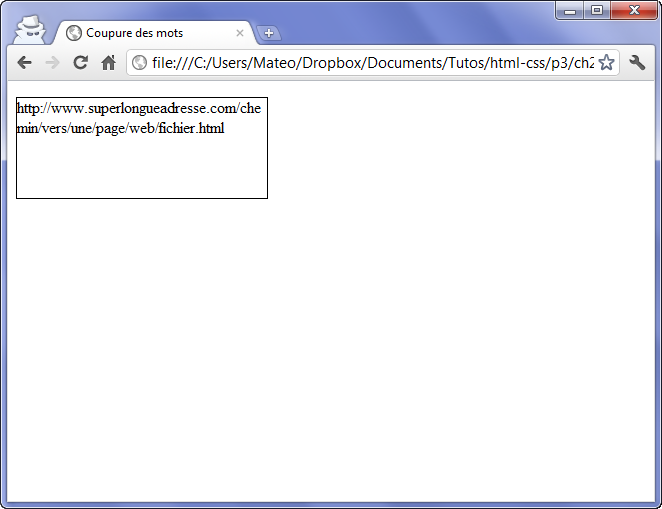
* L’ordinateur, ne sait pas “couper” l’adresse car il n’y a ni espace ni tiret. Il ne sait pas faire la césure.
* Avec le code suivant, la césure sera forcée dès que le texte risque de dépasser (figure suivante).

p

{

**word-wrap**: break-word;

}



(Le texte est coupé pour ne pas déborder)

* Il est conseillé d'utiliser cette fonctionnalité dès qu'un bloc est susceptible de contenir du texte saisi par des utilisateurs (par exemple sur les forums de votre futur site). Sans cette astuce, on peut « casser » facilement le design d'un site (en écrivant par exemple une longue suite de « aaaaaaaaaaa »).

**Résumé**

* On distingue deux principaux types de balises en HTML :
  + le type block ( <**p**> , <**h1**> …) : ces balises créent un retour à la ligne et occupent par défaut toute la largeur disponible. Elles se suivent de haut en bas ;
  + le type inline ( <**a**> , <**strong**> …) : ces balises délimitent du texte au milieu d'une ligne. Elles se suivent de gauche à droite.
* On peut modifier la taille d'une balise de type block avec les propriétés CSS width (largeur) et height (hauteur).
* On peut définir des minima et maxima autorisés pour la largeur et la hauteur : **min-width** , **max-width** , **min-height** , **max-height** .
* Les éléments de la page disposent chacun de marges intérieures ( **padding** ) et extérieures ( **margin** ).
* S'il y a trop de texte à l'intérieur d'un bloc de dimensions fixes, il y a un risque de débordement. Dans ce cas, il peut être judicieux de rajouter des barres de défilement avec la propriété **overflow** , ou de forcer la césure avec **word-wrap** .

**Faire sa mise en page avec Flexbox**

* Il est temps d'apprendre à mettre en page notre site. Vous savez ? Placer un en-tête, des menus sur le côté, choisir où apparaît une information, etc. C'est la pièce manquante du puzzle pour que nous puissions enfin créer notre site !
* Il y a plusieurs façons de mettre en page un site. Au fil du temps, plusieurs techniques ont existé :

Au début, les webmasters utilisaient des tableaux HTML pour faire la mise en page (berk).

Puis, CSS est apparu et on a commencé à faire une mise en page à l'aide de la propriété **float** (bof).

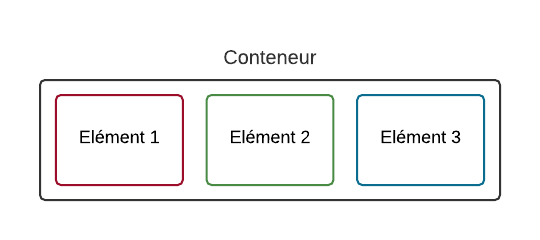
Cette technique avait des inconvénients. Une autre, plus pratique, a consisté à créer des éléments de type **inline-block** sur la page (mouais).

Aujourd'hui, une bien meilleure technique existe encore : **Flexbox** Elle permet toutes les folies (ou presque), et c'est celle qui est recommandé d'utiliser si on en a la possibilité, lorsque l’on crée un nouveau site. [Flexbox est désormais reconnu par tous les navigateurs récents](https://caniuse.com/#feat=flexbox) !

Nous découvrirons donc le fonctionnement de Flexbox dans ce chapitre. Pour ceux qui ont besoin d'informations sur les autres techniques de mise en page plus anciennes, je vous invite à consulter le chapitre "Quelques autres techniques de mise en page". Cela peut toujours vous être utile.

**UN CONTENEUR, DES ÉLÉMENTS**

* Le principe de la mise en page avec Flexbox est simple : on définit un conteneur, et à l'intérieur on y place plusieurs éléments. On imagine un carton dans lequel on range plusieurs objets : c'est le principe.
* Sur une même page web, nous pouvons sans problème avoir plusieurs conteneurs (plusieurs cartons, si l’on préfère). Ce sera à nous d'en créer autant que nécessaire pour obtenir la mise en page que l’on veut.
* Commençons par étudier le fonctionnement d'un conteneur.



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

<**div id**="conteneur">

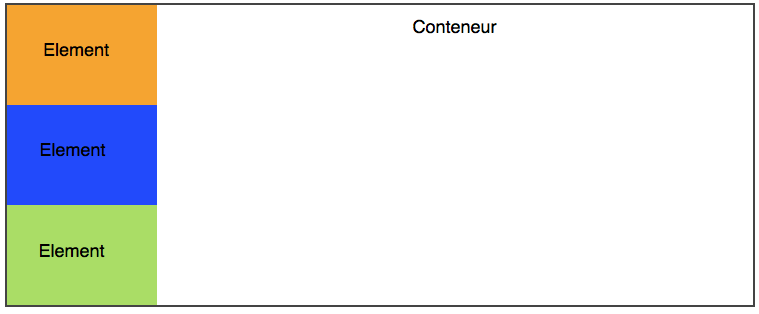
<**div class**="element 1">Element </div>

<**div class**="element 2">Element </div>

<**div class**="element 3">Element </div>

</**div**>

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**](https://developer.mozilla.org/fr/docs/Web/CSS/Pseudo-classes) [**:nth-child**](https://developer.mozilla.org/fr/docs/Web/CSS/: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**

On va découvrir 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, c’est magique



**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.

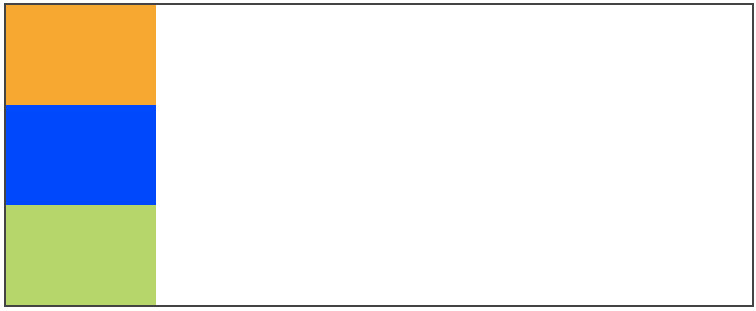
Exemple :

#conteneur

{

display: flex;

flex-direction: column;

}

(Les éléments sont disposé en colonne)

On peut également tester l’ordre inversé, pour voir :

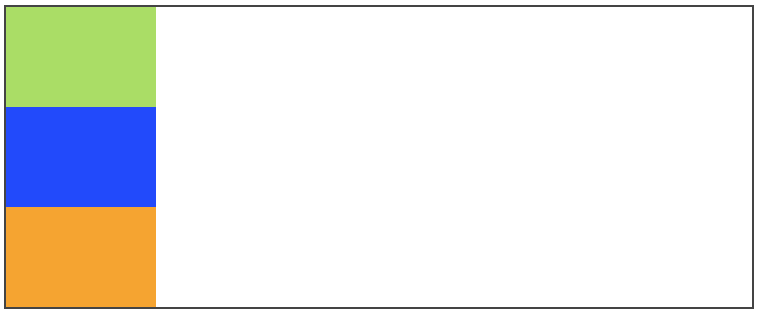
#conteneur

{

display: flex;

flex-direction: column-reverse;

}



(Les éléments sont en colonne... dans l’ordre inverse)

* Il faut bien regarder 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 A 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 l’on veut, on peut 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 de place ;

**wrap-reverse** : Les éléments vont à la ligne, lorsqu’il n’y a plus la place, en sens inverse.

Exemple :

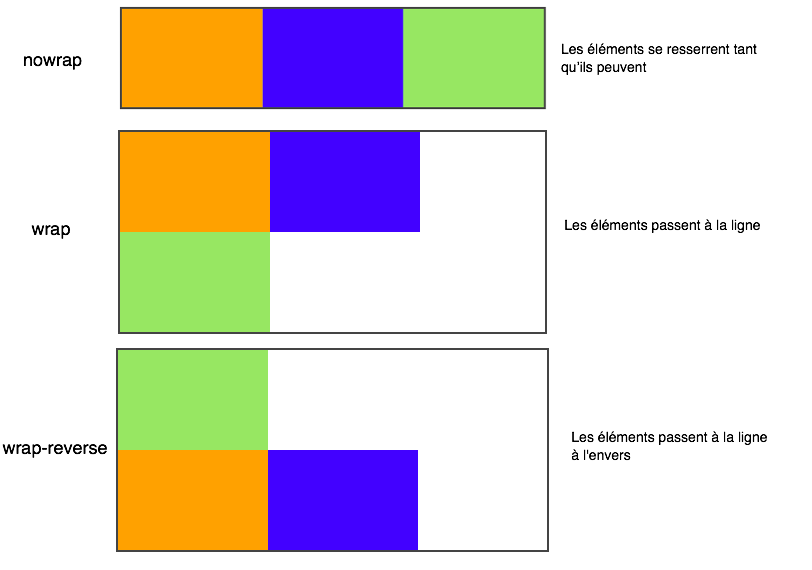
#conteneur

{

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 nos éléments sont organisés horizontalement, l'axe secondaire est l'axe vertical ;
* Si nos éléments sont organisés verticalement, l'axe secondaire est l'axe horizontal.

Pourquoi on parle de 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

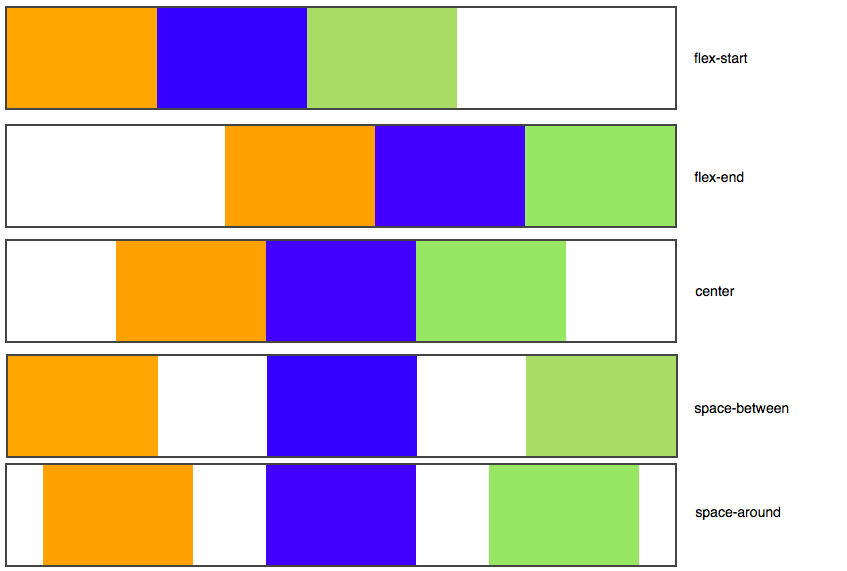
{

display: flex;

justify-content: space-around;

}

Le mieux est encore de tester toutes les valeurs possibles pour voir ce que ça donne.

(Les différentes valeurs possibles pour l’alignement avec justify-content)

On voit 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 nos éléments sont dans une direction verticale. Dans ce cas, l’axe vertical devient l’axe principal et justify-content s’applique aussi :

#conteneur

{

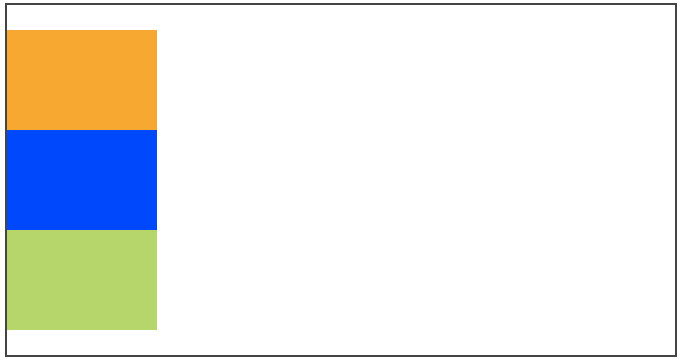
display: flex;

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**

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

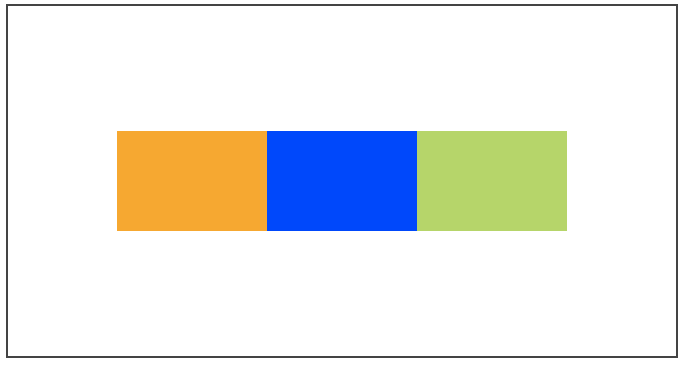
{

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.

#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 \*/

{

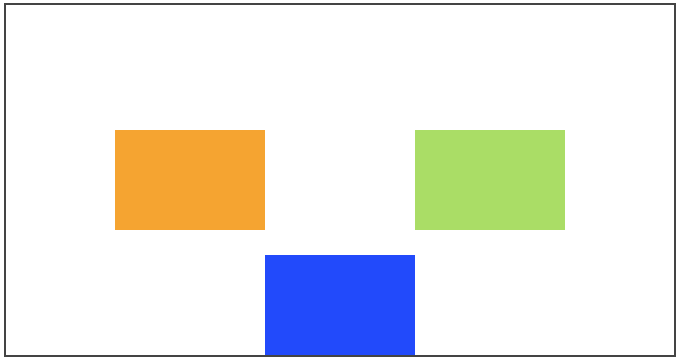
background-color: blue;

align-self: flex-end; /\* Seul ce bloc sera aligné à la fin \*/

}

/\* ... \*/

Résultat :

(Un élément aligné différemment des autres avec **align-self**.)

**RÉPARTIR PLUSIEURS LIGNES**

Si on a plusieurs lignes dans notre Flexbox, on peut 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 dans 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 les éléments à aller à la ligne **flex-wrap** :

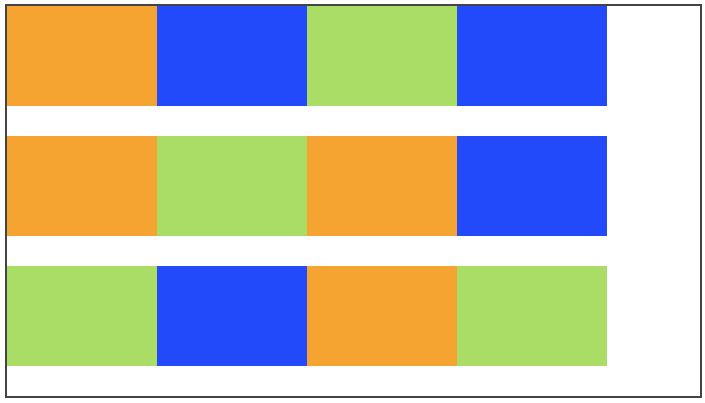
#conteneur

{

display: flex;

flex-wrap: wrap;

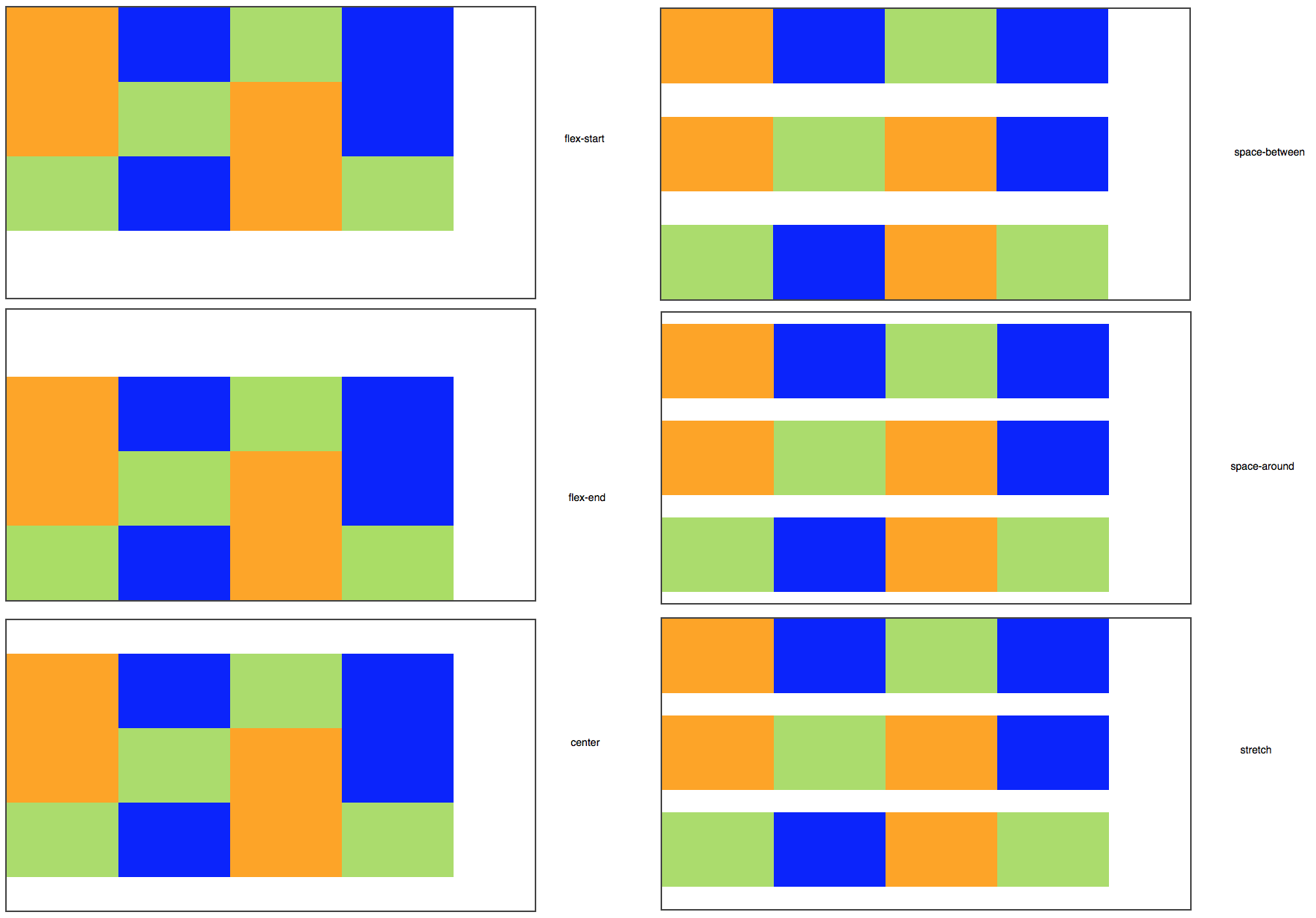
}

(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 veux 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**. Il faut indiquer 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 on dit que le premier 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.

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;

}

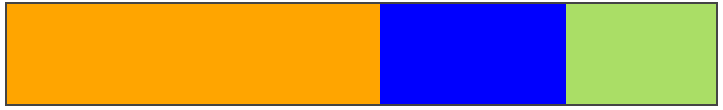
.element:nth-child(2)

{

flex: 1;

}

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

* 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. 

**RÉSUMÉ :**

* Il existe plusieurs techniques pour positionner les blocs sur la page. Flexbox est la technique la plus récente et de loin la plus puissante, que je vous recommande d'utiliser.
* Le principe de Flexbox est d'avoir un conteneur, avec plusieurs éléments à l'intérieur. Avec **display: flex;** sur le conteneur, les éléments à l'intérieur sont agencés en mode Flexbox (horizontalement, par défaut).
* Flexbox peut gérer toutes les directions. Avec **flex-direction** , on peut indiquer si les éléments sont agencés horizontalement (par défaut) ou verticalement. Cela définit ce qu'on appelle l'*axe principal*.
* L'alignement des éléments se fait sur l'axe principal avec **justify-content** , et sur l'axe secondaire avec **align-items**  .
* Avec **flex-wrap**  , on peut autoriser les éléments à revenir à la ligne s'ils n'ont plus d'espace.
* S'il y a plusieurs lignes, on peut indiquer comment les lignes doivent se répartir entre elles avec **align-content** .
* Chaque élément peut être réagencé en CSS avec **order** (pas besoin de toucher au code HTML !).
* Avec la super-propriété **flex**  , on peut autoriser nos éléments à occuper plus ou moins d'espace restant.
* Flexbox, c'est cool.