Flexbox, guide complet

01 FEBRUARY 2014 on CSS, Design, Responsive Web Design, Intermédiaire, Flexbox

Par Chris Coyier

☞ *NdT : Ce tutoriel est la réunion de quatre articles de Chris Coyier formant une introduction à Flexbox. Vous pouvez ensuite consulter tous les articles sur Flexbox traduits dans la Cascade notamment les exemples concrets d'implémentation et les astuces techniques.*

Le module CSS3 `Flexbox Layout` fournit une façon efficace de disposer, aligner et distribuer l'espace entre les items d'un container, même lorsque leurs dimensions sont inconnues et/ou dynamiques - d'où le terme "flex".

L'idée principale est de donner à un élément contenant (container) la possibilité de changer les largeur et hauteur des éléments contenus (items), afin de remplir au mieux l'espace disponible, et de s'adapter à tous les terminaux et toutes les tailles d'écrans. Un container flexible permet aux items de s'étendre pour occuper la place disponible ou au contraire les réduit pour leur éviter de déborder.

Le plus important à retenir c'est qu'avec Flexbox la disposition n'est pas rigidement directionnelle, contrairement à ce que nous connaissons habituellement en CSS — où Block est basé sur un schéma vertical et Inline sur un schéma horizontal. Cela fonctionne bien pour les pages, mais ça manque de... flexibilité lorsqu'il s'agit d'applications complexes, en particulier lorsqu'il faut s'adapter aux changements d'orientation de device, redimensionner, étendre ou réduire l'espace, etc.

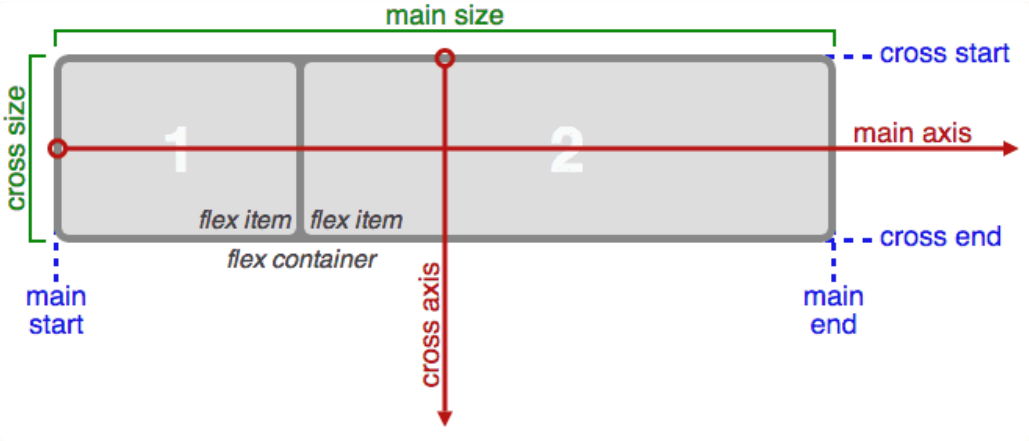
Note importante : Flexbox est plutôt adapté aux composants d'une application, de petite échelle, alors que les grilles conviennent à des mises en page complexes et à grande échelle. CSS Grid et Flexbox ont des usages différents mais sont faits pour travailler ensemble !

☞ Nous allons commencer par décrire la mécanique, mais vous pouvez souhaiter regarder d'abord les exemples d'usage de flexbox.

Les bases

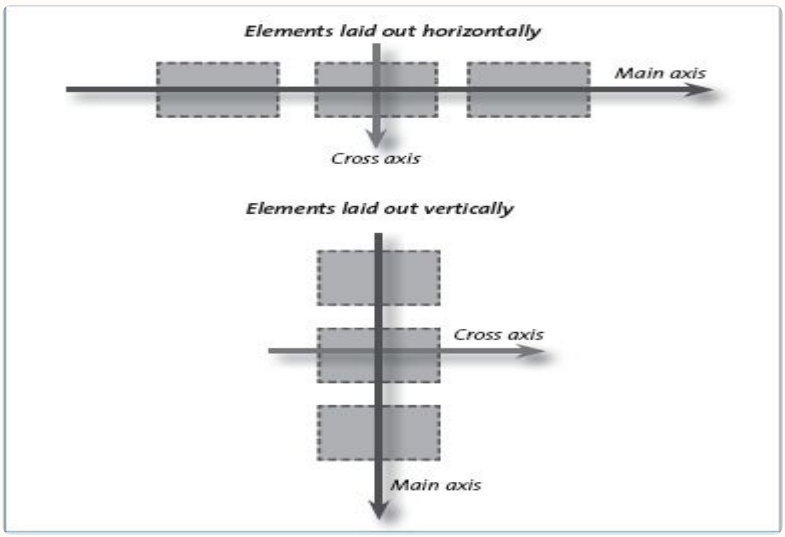
Flexbox est un module, pas une propriété, cela entraîne pas mal de conséquences, entre autres que Flexbox a des propriétés bien à lui. Certaines concernent le container (l'élément parent, qu'on appelle "flex container"), d'autres concernent le ou les enfants (les "flex items").

Alors que le positionnement CSS habituel est basé sur les directions de flux block et inline, le positionnement flex, lui, est basé sur les directions "flex-flow". L'illustration ci-dessous, tirée des spécifications, explique l'idée qui est à la base du positionnement flex :



Les items seront disposés soit sur l'axe principal (`main axis`) depuis `main-start` ou `main-end` , ou sur l'axe perpendiculaire (`cross axis`) en partant de `cross-start` ou de `cross-end` . Le flex-flow suit donc l'axe principal ou l'axe perpendiculaire.

- **main axis** - L'axe principal d'un container flex est l'axe primaire sur lequel les items sont disposés. Attention, il n'est pas forcément horizontal, tout dépendra de la propriété `justify-content` (voir ci-dessous).
- **main-start | main-end** - À l'intérieur du container, les items flex sont placés entre un point de départ (main-start) et un point d'arrivée (main-end).
- **main size** - La taille d'un item flex qui se trouve dans l'axe principal, qu'il s'agisse de la hauteur ou de la largeur, est la taille principale (main size). La propriété main size est soit "width", soit "height", selon l'orientation.
- **cross axis** - L'axe perpendiculaire à l'axe principal est appelé cross axis. Sa direction dépend de la direction de l'axe principal.
- **cross-start | cross-end** - Les lignes sont remplies avec les items et sont placées dans le container en partant du côté cross-start et en allant vers cross-end.
- **cross-size** - La largeur ou la hauteur d'un item, selon la dimension dans laquelle on se trouve (même principe que main size).



Le schéma ci-dessus montre bien que l'axe principal peut être horizontal ou vertical. Dans le premier cas (éléments disposés horizontalement), on pourrait penser par exemple à des articles disposés en colonnes sur une page, ou à un groupe de photos

(voir un exemple plus bas), ou encore à une barre de navigation horizontale. Dans le deuxième cas (éléments disposés verticalement), on peut imaginer une sidebar, des liens de navigation (voir un exemple plus bas), etc.

Propriétés

☞ **Commençons par les propriétés qui s'appliquent à l'élément parent, le container**.

La première chose à faire est de définir un contexte général d'affichage. Le module flexbox fontionne à l'intérieur de ce contexte.

display: flex|inline-flex;

C'est ainsi qu'on définit un container flex, il est block par défaut ou inline selon la valeur donnée. Cela cée un contexte flex pour tous les descendants directs.

```
display: flex | inline-flex;
```

- `flex` : Cette valeur génère un container flex, de niveau block, à l'intérieur de l'élément.
- `inline-flex` : Cette valeur génère un container flex, de niveau inline, à l'intérieur de l'élément.

Notez que :

- les propriétés `columns-` du module multi-colonnes (https://developer.mozilla.org/fr/docs/CSS/Colonnes_CSS3) n'ont pas d'effet sur un container flex
- `float` , `clear` et `vertical-align` n'ont pas d'effet sur un item flex.

Pour comprendre en se divertissant, le mieux est de jouer avec ces valeurs sur des outils en ligne tels que flexplorer (http://bennettfeely.com/flexplorer/) (voir liste en fin d'article).

flex-direction

La propriété flex-direction établit l'axe principal.

```
flex-direction: row | row-reverse | column | column-reverse
```

- `row` (valeur par défaut): de gauche à droite si la lecture se fait dans ce sens, de droite à gauche dans le cas inverse (1).
- `row-reverse` : inverse le sens
- `column` : comme `row` mais du haut vers le bas
- `column-reverse` : comme `row-reverse` mais du bas vers le haut

flex-wrap

Cette propriété définit si le container comprend une seule ligne ou plusieurs et la direction sur l'axe perpendiculaire (cross-axis), qui détermine la direction dans laquelle les nouvelles lignes seront empilées.

```
flex-wrap: nowrap | wrap | wrap-reverse
```

- `nowrap` : (valeur par défaut) sur une seule ligne, de gauche à droite dans un système `ltr`, sinon l'inverse. La ligne peut déborder de son contenant.
- `wrap` : multiligne, de gauche à droite dans un système `ltr`, sinon l'inverse. Pas de débordement, on passe à la ligne.
- `wrap-reverse` : multiligne, de droite à gauche dans un système `ltr`, sinon l'inverse.

flex-flow

Cette propriété est un raccourci des propriétés "flex-direction" et "flex-wrap" qui ensemble définissent les axes "main" et "cross" du container flex. La valeur par défaut est `row nowrap`.

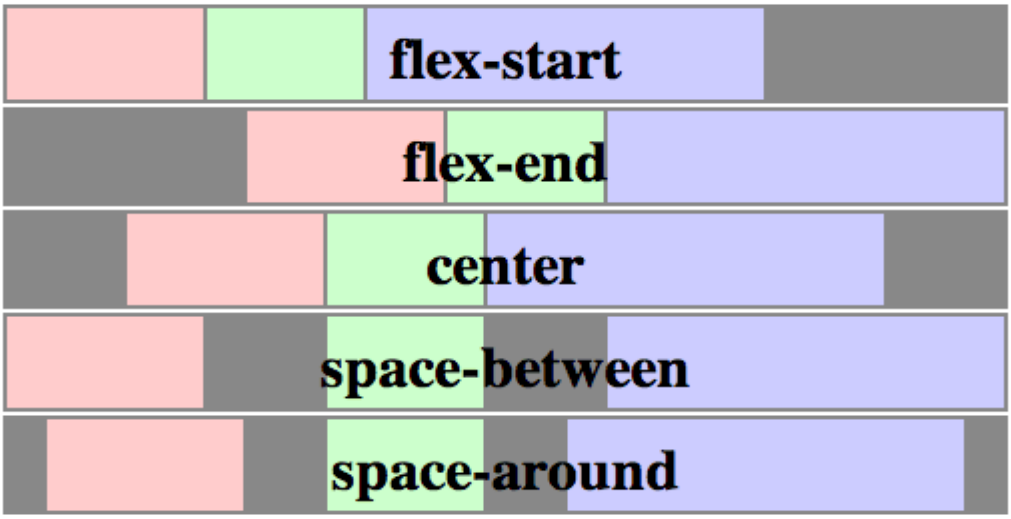
```
flex-flow: <'flex-direction'> || <'flex-wrap'>
```

justify-content

La propriété justify-content définit l'alignement le long de l'axe principal. Elle permet de distribuer l'espace excédentaire lorsque tous les items flex sur une ligne sont inflexibles ou lorsqu'ils ont atteint leur taille maximale. Elle contrôle aussi l'alignement des items lorsqu'ils débordent.

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

- `flex-start` (par défaut) : les items sont regroupés en début de ligne
- `flex-end` : les items sont regroupés en fin de ligne
- `center` : les items sont centrés le long de la ligne
- `space-between` : les items sont répartis sur la ligne; le premier est collé du côté start, le dernier du côté end.
- `space-around` : les items sont répartis sur la ligne avec un espacement égal autour de chacun.



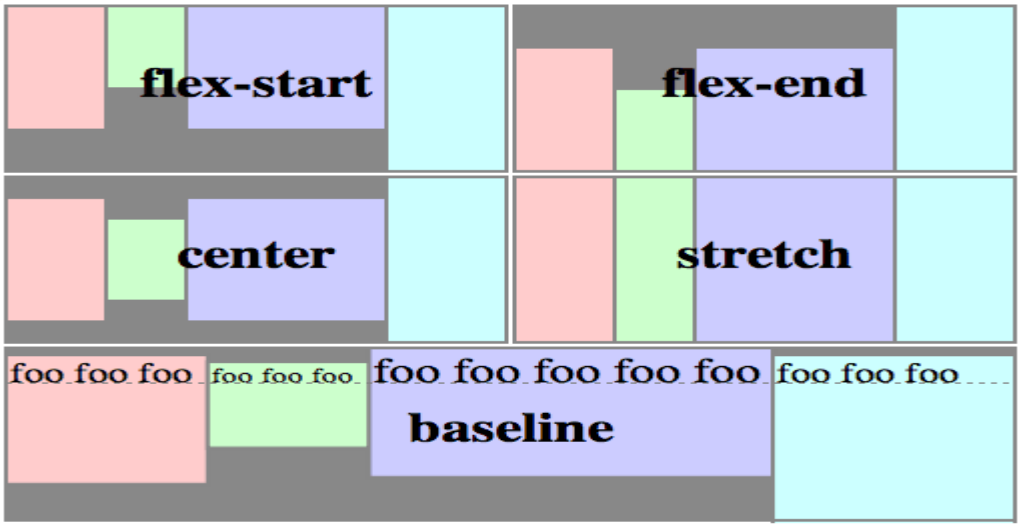
☞ *NdT : pour plus de détails sur space-between, vous pouvez lire l'article Guide de Flexbox : space-between, l'oublié dans la Cascade.*

align-items

La propriété align-items définit la façon dont les items d'une ligne sont disposés le long de l'axe "cross". On peut le voir comme la version de `justify-content` pour "cross axis".

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

- `flex-start` : l'item est placé au début de la ligne cross-start.
- `flex-end` : la marge "cross-end" de l'item est placée sur la ligne cross-end
- `center` : les items sont centrés sur l'axe cross
- `baseline` : les items sont alignés sur leur ligne de base
- `stretch` (par défaut) : les items sont étirés jusqu'à remplir le container (tout en respectant min-width/max-width)



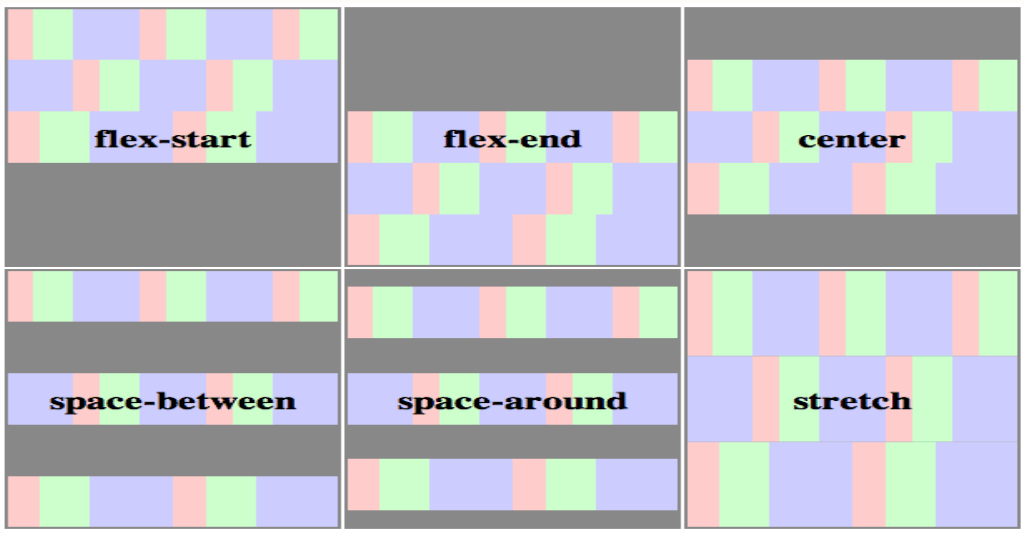
align-content

La propriété align-content aligne les lignes d'un container flex à l'intérieur de l'espace où il reste de l'espace sur l'axe cross, un peu comme `justify-content` aligne les items sur l'axe principal.

Note : cette propriété n'a pas d'effet quand la flexbox n'a qu'une seule ligne.

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

- `flex-start` : lignes regroupées au début du container
- `flex-end` : lignes regroupées à la fin du container
- `center` : lignes regroupées au centre du container
- `space-between` : les lignes sont réparties, la première est collée du côté start, la dernière du côté end.
- `space-around` : les lignes sont réparties avec un espacement égal autour de chacune.
- `stretch` (par défaut) : les lignes s'étirent pour remplir tout l'espace.



☞ **Passons maintenant aux propriétés qui s'appliquent aux éléments enfants, les items flex**.

order

Par défaut, les items flex sont disposés par ordre d'arrivée. Cependant, la propriété `order` permet de contrôler l'ordre dans lequel ils apparaissent dans le container.

```
order: <nombre entier>
```

flex-grow

La propriété flex-grow définit la possibilité pour un item de grandir, si nécessaire. Elle accepte une valeur sans unité qui sert de proportion. Elle dicte l'espace que peut prendre l'item à l'intérieur de l'espace disponible dans le flex container.

Si tous les items ont `flex-grow` défini à 1, chaque enfant aura le même espace dans le container. Si vous donnez à l'un des enfants une valeur de 2, cet enfant prendra deux fois plus de place que les autres.

```
flex-grow: <nombre entier> (par défaut = 0)
```

Les chiffres négatifs ne sont pas valides.

flex-shrink

La propriété flex-shrink définit la possibilité pour un item flex de rétrécir si nécessaire.

```
flex-shrink: <nombre entier> (par défaut = 1)
```

Les chiffres négatifs ne sont pas valides.

☞ *NdT : pour plus de détails sur cette propriété, vous pouvez lire l'article Guide de Flexbox : n'oubliez pas flex-shrink dans la Cascade.*

flex-basis

La propriété flex-basis définit la taille par défaut d'un élément avant que l'espace restant soit réparti.

```
flex-basis: <longueur> | auto (par défaut = auto)
```

☞ *NdT : pour mieux comprendre cette propriété un peu compliquée, vous pouvez vous reporter à l'article Layout sur 12 colonnes avec Flexbox dans La Cascade.*

flex

Cette propriété est le raccourci de `flex-grow`, `flex-shrink` et `flex-basis`. Les deuxième et troisième paramètres sont optionels. La valeur par défaut est `0 1 auto`.

```
flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
```

align-self

La propriété align-self permet à des items flex de passer outre aux alignement par défaut ou à ceux spécifiés par `align-items`. Les valeurs sont les mêmes que pour ce dernier.

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

Exemples

Commençons avec un exemple extrêmement simple, qui résout un problème que nous rencontrons tous les jours : le centrage parfait. Rien de plus simple avec Flexbox.

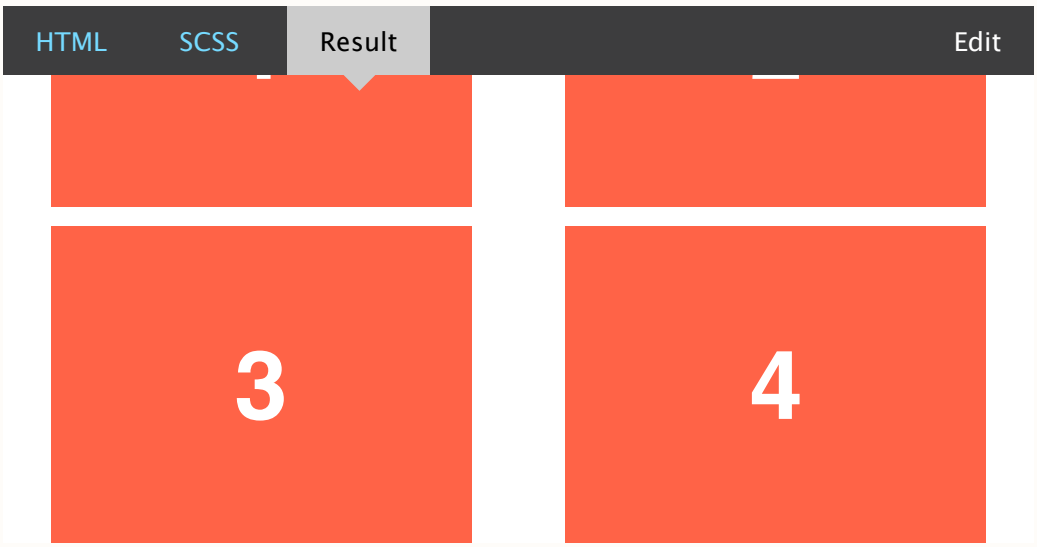
```
.parent {
        display: flex;
        height: 300px; /* Ou ce que vous voulez */
}

.child {
        width: 100px;  /* Ou ce que vous voulez */
        height: 100px; /* Ou ce que vous voulez */
        margin: auto;  /* Magique ! */
}
```

Tout vient de ce qu'une marge réglée sur "auto" dans un container flex absorbe l'espace supplémentaire. Une marge verticale "auto" centrera l'item parfaitement sur les deux axes.

Utilisons maintenant quelques propriétés supplémentaires. Prenons une liste de 6 items, de dimension fixe, mais ils pourraient tout aussi bien être ajustables. Nous voulons qu'ils soient disposés de manière régulière sur l'axe horizontal, de façon à ce que lorsque nous modifions la taille de l'écran les items se répartissent sans avoir recours à des media queries.

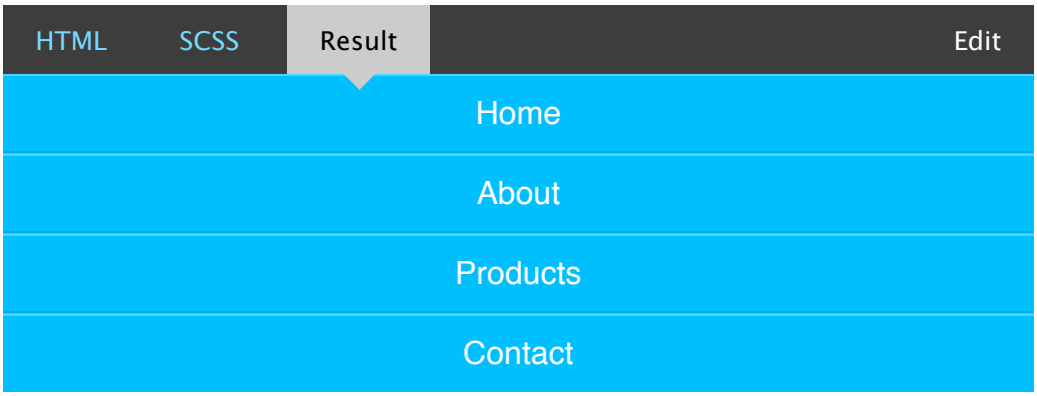
```
.flex-container {
        /* On crée d'abord un context flex */
        display: flex;

        /* Puis on définit la direction du flux et le wrap
        * Rappelez-vous que c'est la même chose que :
        * flex-direction: row;
        * flex-wrap: wrap;
        */
        flex-flow: row wrap;

        /* Puis on définit la façon dont se répartit
l'espace restant */
        justify-content: space-around;
}
```

Et voilà. Le reste n'est plus qu'une question de style. Ci-dessous un exemple de cette répartition. Le mieux est de le regarder sur CodePen (https://codepen.io/HugoGiraudel/pen/LklCv) et de faire varier la taille de l'écran pour voir ce qui se passe.



Essayons quelque chose d'autre. Admettons que nous ayons une barre de navigation alignée en haut à droite de notre page, mais nous souhaiterions qu'elle soit centrée pour les écrans de taille moyenne et sur une colonne pour des petits écrans. Facile (https://codepen.io/HugoGiraudel/pen/pkwqH) :

```
/* Grand écran */
.navigation {
        display: flex;
        flex-flow: row wrap;
        /* Aligne les items à end line sur main-axis */
        justify-content: flex-end;
}

/* Écrans moyens */
@media all and (max-width: 800px) {
        .navigation {
        /* on centre en répartissant l'espace entre les
items */
        justify-content: space-around;
        }
}

/* Petit écran */
@media all and (max-width: 500px) {
        .navigation {
        /* On n'utilise plus row mais column */
        flex-direction: column;
        }
}
```



Essayons encore mieux, en jouant avec la flexibilité des items flex ! Par exemple un layout mobile-first (https://codepen.io/HugoGiraudel/pen/qIAwr) sur 3 colonnes, avec un header et un footer pleine largeur. Le tout indépendamment de l'ordre des sources.

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

/* Tous les items sont à 100% width */
.header, .main, .nav, .aside, .footer {
        flex: 1 100%;
}

/* We rely on source order for mobile-first approach
* in this case:
* 1. header
* 2. nav
* 3. main
* 4. aside
* 5. footer
*/

/* Écrans moyens */
@media all and (min-width: 600px) {
        /* Les 2 sidebars partagent une rangée */
        .aside { flex: 1 auto; }
}

/* Grands écrans */
@media all and (min-width: 800px) {
        /* on intervertit l'ordre de la 1ère sidebar et de
main
        * et on dit à l'élément de prendre 2 fois plus de
place en largeur que les 2 sidebars
        */
        .main { flex: 2 0px; }

        .aside-1 { order: 1; }
        .main    { order: 2; }
        .aside-2 { order: 3; }
        .footer  { order: 4; }
}
```

HTML SCSS Result Edit

Header

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

Aside 1

Voyons maintenant la façon dont flexbox résout le problème suivant : comment faire pour qu'un nombre arbitraire de boîtes remplissent l'espace disponible dans la boîte parent et si nécessaire s'étendent au-delà (c'est à dire : ne s'écrasent pas pour tenir dans la boîte). Vous pouvez regarder cette vidéo (https://css-tricks.com/video-screencasts/131-tinkering-flexbox/) (en anglais) qui illustre ce que je veux dire par là.

Par "boîte", j'entends simplement un élément de niveau block. Div, section, article, ce que vous voulez.

La hauteur par défaut d'une boîte est déterminée par son contenu. Les boîtes s'empilent les unes sur les autres. La hauteur de leur boîte parent pourrait également être déterminée par la somme de leurs hauteurs, mais il pourait en être autrement, par exemple si elle est définie explicitement ou si elle est soumise à quelque chose de variable, comme la taille de la fenêtre du navigateur. Si la boîte parent a une hauteur plus importante, il y aura simplement de l'espace vide en bas.

Pouvons-nous forcer les boîtes à se répartir l'espace disponible de façon égale ? Avec flexbox, oui.

À gauche, par défaut. À droite, ce que nous voulons

Admettons que le HTML soit :

```
<section class="fill-height-or-more">
        <div>
                <!-- stuff -->
        </div>
        <div>
                <!-- stuff -->
        </div>
        <div>
                <!-- stuff -->
        </div>
</section>
```

On démarre flexbox dans l'élément parent :

```
// dans CSS

.fill-height-or-more {
        display: flex;
}
```

et on empile les boîtes sur un axe vertical (column) plutôt qu'horizontal (row) qui est la valeur par défaut :

```
.fill-height-or-more {
        display: flex;
        flex-direction: column;
}
```

Avec cela, le rendu sera le même que si nous n'avions rien fait. Mais maintenant nous allons appliquer la propriété flex aux enfants, et ils vont remplir l'espace :

```
.fill-height-or-more > div {
        /* ce sont les items flex */
        flex: 1;
}
```

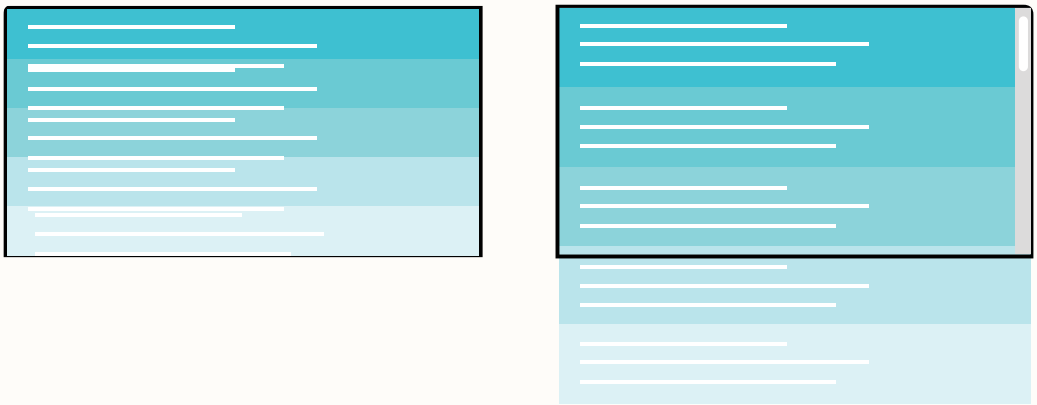
Et c'est fait =)

Pour détailler : `flex: 1` est l'équivalent de `flex: 1 1 auto`, c'est le raccourci de trois propriétés différentes :

```
flex-grow: 1;
flex-shrink: 1;
flex-basis: auto;
```

Ce qui est sympa avec flexbox c'est que tout ça marcherait avec n'importe quel nombre de boîte, ça pourrait être une boîte unique ou 100 boîtes. S'il reste de l'espace, il sera partagé, sinon, pas de problème.

Dans un monde antérieur à flexbox, nous aurions sans doute essayé de savoir combien de boîtes il y avait, puis nous aurions fixé leur hauteur en poucentage. Ça fonctionne pour remplir l'espace, mais s'il y avait trop de boîtes elles seraient écrasées. Voilà encore un truc sympa avec flexbox, il ne va pas écraser nos boîtes au point qu'elles n'affichent plus leur contenu :



À gauche, utilisation de pourcentages. À droite, le comportement souhaité

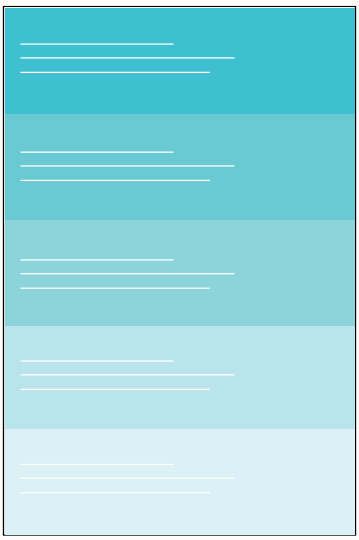
Si nous voulons aller encore un peu plus loin, nous pouvons utiliser flexbox pour centrer le contenu à l'intérieur des boîtes. C'est là qu'on devient fou avec CSS. Le centrage vertical est difficile. Même avec flexbox ici, il va nous falloir transformer chacun de nos items en containers, puis utiliser un contenant interne qui devient l'item flex que nous centrons. Donc, oui, un élément de plus.

```
// dans HTML

<section class="fill-height-or-more">
        <div>
                <div>
                        <!-- stuff -->
                </div>
        </div>
    ...

// dans CSS

.fill-height-or-more > div {
        /* un flex item... */
        flex: 1;

        /* ...qui est aussi un flex container */
        display: flex;
}
```

Puis pour le centrer nous définissons la direction verticale à nouveau (column) et nous utilisons une autre propriété flexbox, `justify-content` :

```
.fill-height-or-more > div {
        flex: 1;

        display: flex
        justify-content: center;
        flex-direction: column;
}
```

Voici ce que ça donne :



D'un côté, c'est logique. Pour centrer quelque chose, il faut bien un élément dans lequel le centrer. D'un autre côté, avec une table et des cellules, on n'aurait pas eu besoin d'un élément supplémentaire... Bon enfin, voici la démo (https://codepen.io/chriscoyier/pen/Jeryz) !

HTML SCSS Result Edit

Boxes That Fill Height (or more)

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vestibulum tortor quam, feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.

Two

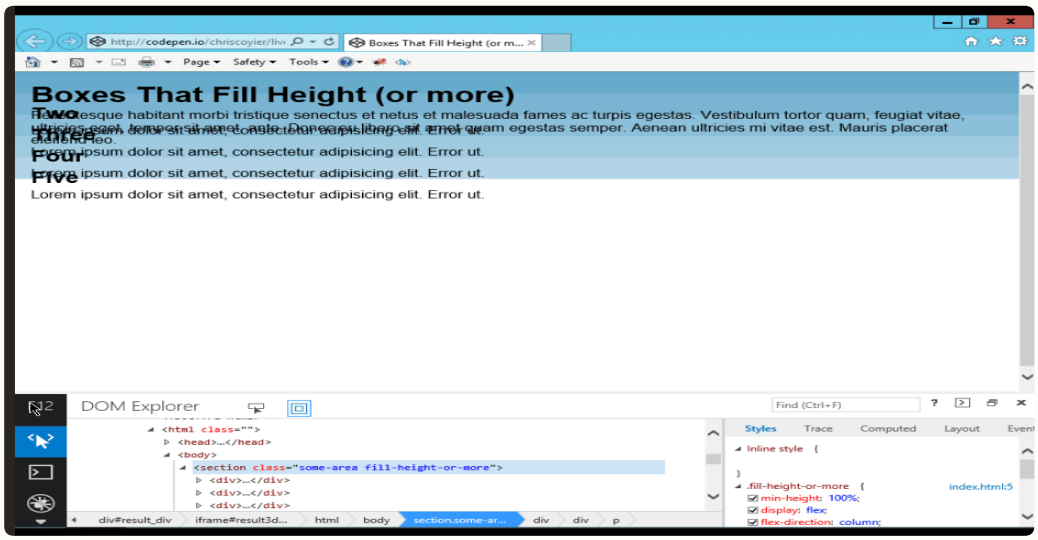
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Error ut.

Three

Compatibilité navigateurs

J'ai utilisé uniquement la dernière syntaxe ici. Les versions actuelles de Chrome, Opera sont compatibles. Les prochaines versions de Firefox et Android le seront aussi. Safari et iOS supportent la nouvelle syntaxe, avec des préfixes -webkit-. Can I Use (https://caniuse.com/flexbox) vous racontera toute l'histoire.

IE est bizarre. IE 10 est compatible avec l'ancienne version de flexbox, IE 11 avec la nouvelle. Pour cette démo pourtant il y a quelque chose qui ne fonctionne pas. La hauteur de `.fill-height-or-more` est bien rendue en utlisant `min-height`, mais les boîtes sont écrasées.



Voici à quoi ressemble notre petit exemple lorsqu'on lui ajoute les préfixes constructeurs :

```
.fill-height-or-more {
        display: -webkit-box;
        display: -webkit-flex;
        display: -moz-box;
        display: -ms-flexbox;
        display: flex;
        -webkit-box-orient: vertical;
        -webkit-box-direction: normal;
        -webkit-flex-direction: column;
        -moz-box-orient: vertical;
        -moz-box-direction: normal;
        -ms-flex-direction: column;
        flex-direction: column;
}

.fill-height-or-more > div {
        -webkit-box-flex: 1;
        -webkit-flex: 1;
        -moz-box-flex: 1;
        -ms-flex: 1;
        flex: 1;
        display: -webkit-box;
        display: -webkit-flex;
        display: -moz-box;
        display: -ms-flexbox;
        display: flex;
        -webkit-box-pack: center;
        -webkit-justify-content: center;
        -moz-box-pack: center;
        -ms-flex-pack: center;
        justify-content: center;
        -webkit-box-orient: vertical;
        -webkit-box-direction: normal;
        -webkit-flex-direction: column;
        -moz-box-orient: vertical;
        -moz-box-direction: normal;
        -ms-flex-direction: column;
        flex-direction: column;
}
```

Mon conseil : utilisez une syntaxe moderne, comme je l'ai fait précédemment, puis laissez Autoprefixer s'occuper du reste, il règlera non seulement le problème des préfixes constructeurs mais aussi celui de l'ancienne syntaxe.

Flexbox ancien et nouveau

Le flexbox que nous connaissons a subi de nombreux changements. Si vous cherchez flexbox dans Google, vous trouverez beaucoup d'informations dépassées. Si vous trouvez quelque chose comme `display: box;` ou une propriété `box-{*}`, c'est la

vieille version de 2009. Si vous trouvez `display: flexbox;` ou la fonction `flex()`, c'est une phase bizarre de flexbox en 2011. Si vous voyez `display: flex` et `flex-{*}`, vous êtes sur la spécification actuelle.

Notes du Traducteur :

(1) Pour rappel, vous pouvez définir le sens de lecture dans CSS avec la propriété `direction` qui accepte les valeurs `ltr` - de gauche à droite (*left to right*) - et `rtl` - de droite à gauche ↩

Intéressé par Flexbox ? Retrouvez tous les articles parus dans La Cascade.

Vous pouvez également consulter le Dico-CSS sur chacune des propriétés de Flexbox, par exemple justify-content, align-content, align-items, flex-direction...

Ressources complémentaires en anglais

W3C, recommandation Flexbox (https://www.w3.org/TR/css3-flexbox/)
Flexbox (http://tympanus.net/codrops/css_reference/flexbox/), in Codrops CSS reference, long article détaillé de Sara Soueidan
Using Flexbox today (http://www.chriswrightdesign.com/experiments/using-flexbox-today/), par Chris Wright
A designer's guide to Flexbox (http://demosthenes.info/blog/780/A-Designers-Guide-To-Flexbox), de Philip Walton
Solved by Flexbox (https://philipwalton.github.io/solved-by-flexbox/), de Philip Walton, une démonstration par l'exemple de tout les problèmes que Flexbox peut résoudre.
Flexbox cheatsheet (http://www.sketchingwithcss.com/samplechapter/cheatsheet.html), l'antisèche de flexbox, un excellent complément à cet article, beaucoup d'exemples pratiques de positionnement.
Flexbox Cheatsheet Cheatsheet (http://jonibologna.com/flexbox-cheatsheet/), par Joni Trythall, une superbe antisèche à télécharger, super pratique !
Flexbox, The next generation of CSS layout has arrived (http://blog.teamtreehouse.com/flexbox-next-generation-css-layout-arrived?utm_source=CSS-Weekly&utm_campaign=Issue-109&utm_medium=email), par Nick Pettit (Treehouse)

Outils pour expérimenter Flexbox en ligne :

Flexbox Froggy (http://flexboxfroggy.com/), un jeu absolument génial pour apprendre Flexbox !
Flexplorer (http://bennettfeely.com/flexplorer/)

Flexbox adventures (http://www.chriswrightdesign.com/experiments/flexbox-adventures/), par Chris Wright
CSS Flexbox Please! (http://demo.agektmr.com/flexbox/)
Flexy Boxes (http://the-echoplex.net/flexyboxes/)
Un terrain de jeu (http://devbryce.com/site/flexbox/) pour comprendre Flexbox.

Ressources complémentaires en français

CSS3 Flexbox Layout module (http://www.alsacreations.com/tuto/lire/1493-css3-flexbox-layout-module.html), par Raphael Goetter
Utiliser Flexbox, par Sean Fioritto (ici-même) : Sean répond ici à la question “quand pourrai-je utiliser Flexbox dans le monde réel ?”
Exercices Flexbox (http://cyrilvernier.net/teaching/draft/exercices-flexbox.html), excellent site pédagogique de Cyril Vernier

---

Articles originaux parus ou mis à jour le 30 janvier 2014 (https://css-tricks.com/boxes-fill-height-dont-squish/) , le 20 janvier 2014 (https://css-tricks.com/snippets/css/a-guide-to-flexbox/) , le 18 février 2013 (https://css-tricks.com/using-flexbox/) et le 7 août 2012 (https://css-tricks.com/old-flexbox-and-new-flexbox/) dans CSS-Tricks (https://css-tricks.com).

Sur l'auteur : Chris Coyier (https://profiles.google.com/101987072260327250694/about) *est designer, blogger, conférencier. Créateur de CSS-Tricks, de CodePen, animateur du Podcast Shop Talk (http://shoptalkshow.com), il est l'auteur avec Jeff Starr (https://digwp.com/jeff-starr/) de "Digging into WordPress (https://digwp.com)". Vous pouvez le retrouver sur Twitter (https://twitter.com/intent/user?screen_name=chriscoyier), Google+ (https://plus.google.com/101987072260327250694).*

NdT : CSS-Tricks est un site ressource d'une richesse exceptionnelle, axé sur CSS et le web design en général. Vous pouvez suivre son actualité sur Twitter (https://twitter.com/real_css_tricks), Facebook (https://www.facebook.com/CSSTricks), YouTube (https://www.youtube.com/realcsstricks), GitHub (https://github.com/CSS-Tricks/). Mélange d'articles, de vidéos, de forums, de ressources diverses (jetez un coup d'oeil à son "Almanac (https://css-tricks.com/almanac/)"), il est très populaire sur le web anglophone.

CodePen (https://codepen.io/about) est un site où vous pouvez tester votre code, faire des maquettes, proposer vos dernières créations et recueillir les observations de vos pairs. C'est aussi une source d'inspiration pour vos propres designs.