Positionner le contenu

L'un des avantages du CSS est qu'il nous permet de positionner le contenu et les éléments d'une page de presque n'importe quelle manière imaginable, en structurant nos conceptions et en aidant à rendre le contenu plus digeste.

Il existe plusieurs types de positionnement différents au sein du CSS, et chacun a sa propre application. Dans ce chapitre, nous allons examiner quelques cas d'utilisation différents - création de mises en page réutilisables et positionnement unique d'éléments uniques - et décrire quelques façons de procéder pour chacun.

Positionnement avec float

Une façon de positionner les éléments sur une page est avec la propriété `float`. La propriété `float` est assez polyvalente et peut être utilisée d'un grand nombre de façons différentes.

La propriété `float` nous permet essentiellement de retirer un élément du flux "normal" d'une page et de le positionner vers la gauche ou la droite de son parent. Tous les autres éléments de la page vont alors circuler autour de l'élément flottant. Par exemple, un élément image flottant -balise `<img>`- à côté de quelques paragraphes de texte fera que les paragraphes vont entourer l'image.

Lorsque la propriété `float` est utilisée sur plusieurs éléments en même temps, elle offre la possibilité de créer une mise en page par éléments flottants directement à côté ou à l'opposé l'un de l'autre, comme on peut le voir dans les mises en page à plusieurs colonnes.
La propriété `float` accepte quelques valeurs, les plus populaires sont `left` et `right` qui permettent aux éléments à flotter à gauche ou à droite de leur élément parent.

```
img {
  float: left;
}
```

Float en pratique

Nous allons créer une mise en page classique avec un entête en haut, deux colonnes dans le centre, et un pied de page en bas. Idéalement cette page sera marquée en utilisant les éléments `<header>`,`<section>`, `<aside>` et `<footer>`. Dans l'élément `<body>`, le code HTML sera celui ci:

```
<header>...</header>
<section>...</section>
<aside>...</aside>
<footer>...</footer>
```

Fig 1 Affichage des éléments sans "float"

Les éléments, `<section>` et `<aside>` comme tout élément de niveau bloc, seront empilées les uns sur les autres par défaut. Cependant nous voulons que ces éléments se placent côte à côte. En faisant "flotter" `<section>` vers la gauche et `<aside>` à droite, nous pouvons les positionner comme deux colonnes se faisant face. Notre CSS devrait ressembler ceci:

```
section{
  float: left;
}
aside{
  float: right;
}
```

Pour référence, quand un élément est flottant, il flottera jusqu'au bord de son élément parent. S'il n'y a pas d'élément parent, l'élément flottant sera alors flotter jusqu'au bord de la page.

Lorsque nous faisons flotter un élément, nous le retirons du flux normal du document HTML. Cela fait que la largeur de cet élément prend par défaut la largeur de son contenu. Exemple lorsque nous créons des colonnes pour une mise en page réutilisables, ce comportement est souhaité. Elle peut être corrigée en ajoutant une largeur fixe valeur à chaque colonne. En outre, pour empêcher les éléments flottants de se coller les uns aux autres, nous pouvons utiliser la propriété de marge pour créer un espace entre les éléments.

Ici, nous améliorons le bloc de code précédent en ajoutant une `margin` et une `width` de chaque colonne pour mieux façonner notre résultat.

```css
section {
  float: left;
  margin: 0 1.5%;
  width: 63%;
}
aside{
  float: right;
  margin: 0 1.5%;
  width: 30%;
}
```

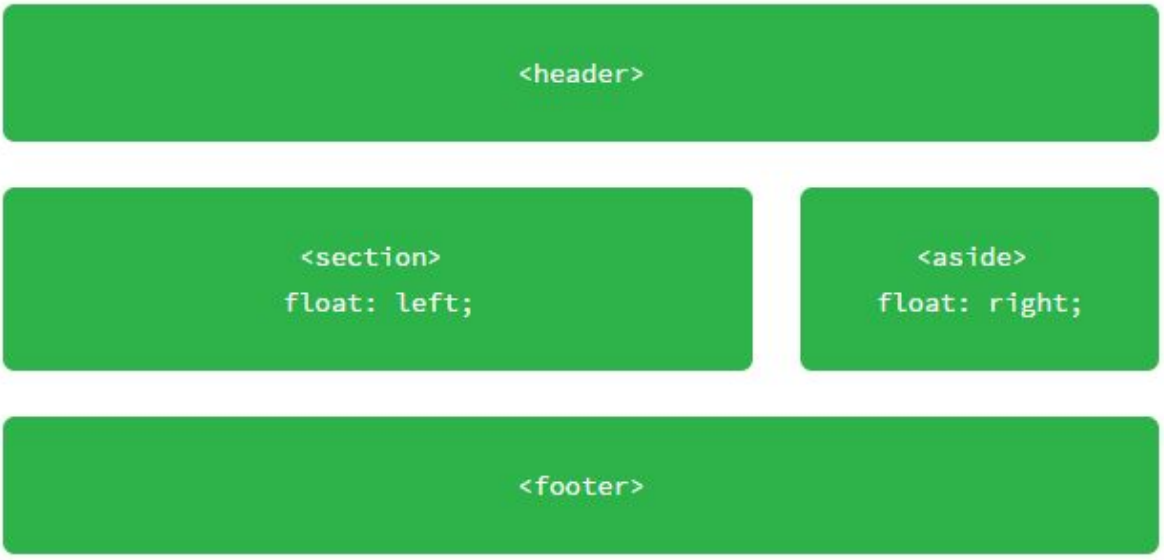

Fig 2 Affichage des éléments avec "float"

Float peut changer la valeur display d'un élément

Lorsque vous faites flotter un élément, il est également important de comprendre qu'un élément est retiré du flux normal d'une page, ce qui peut modifier la valeur d'affichage par défaut de l'élément. La propriété float repose sur un élément ayant une valeur d'affichage de block, et peut modifier la valeur d'affichage par défaut d'un élément s'il n'est pas déjà affiché en tant qu'élément de niveau du bloc.
Par exemple, un élément avec une valeur d'affichage inline, tel que l'élément `<span>`, ignore toutes les valeurs de propriétés de hauteur ou de largeur. Cependant, si cet élément de niveau en ligne "flotte", sa valeur d'affichage sera changée en bloc, et il peut alors accepter des valeurs de propriétés de hauteur ou de largeur.
Lorsque nous faisons "flotter" des éléments nous devons garder un oeil sur la façon dont leurs valeurs de propriété d'affichage sont affectées.

Avec deux colonnes nous pouvons faire flotter une colonne vers la gauche et l'autre vers la droite, mais avec plus colonnes nous devons changer notre approche. Disons, par exemple, que nous aimerions avoir une rangée de trois colonnes entre nos éléments `<header>` et `<footer>`. Si nous supprimons `<aside>` et que l'on utilise trois éléments `<section>`, notre HTML pourrait ressembler à ceci:

```
<header>...</header>
<section>...</section>
<section>...</section>
<section>...</section>
<footer>...</footer>
```

Pour positionner ces trois éléments `<section>` dans une rangée de trois colonnes, au lieu de faire flotter une colonne vers la gauche et une colonne à droite, nous ferons flotter les trois éléments `<section>` à la gauche. Nous devons aussi ajuster la largeur des `<section>` éléments pour tenir compte des colonnes supplémentaires et d'amener à se positionner l'un à côté de l'autre.

```
section {
  float: left;
  margin: 0 1.5%;
  width: 30%;
}
```

Ici nous avons trois colonnes, toutes les valeurs de largeur et marge égales et toutes flottent à gauche.

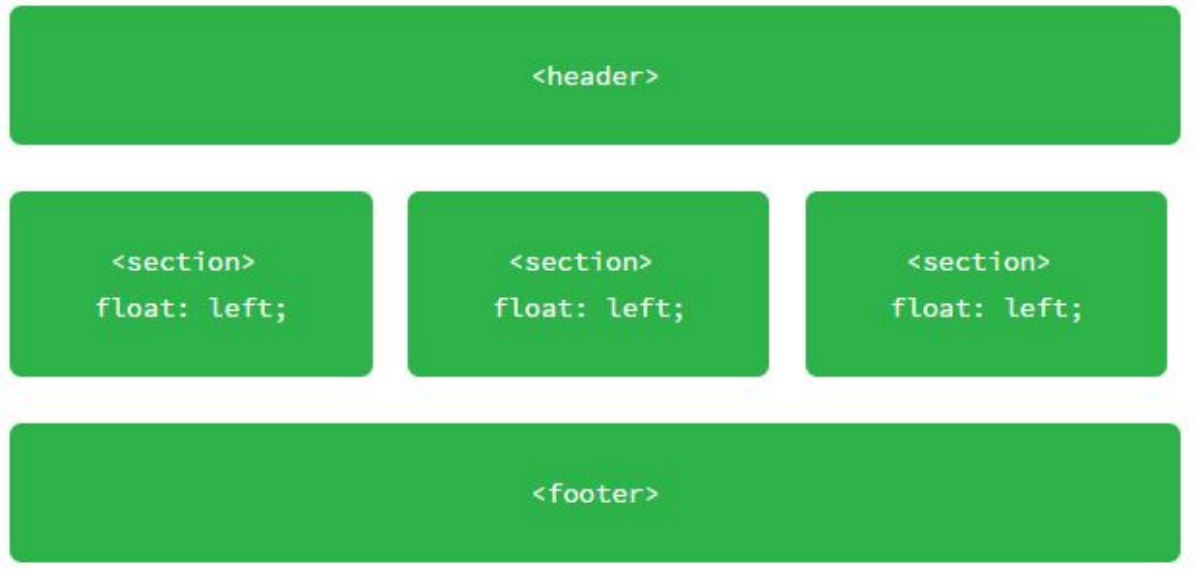

Fig 3 Affichage trois colonnes avec "float"

Gérer les effets indésirables des float

La propriété `float` a été initialement conçue pour permettre au contenu d'épouser les formes d'une images. Une image peut être déclarée `float`, et tout le contenu entourant cette image va alors couler naturellement autour d'elle. Bien que cela fonctionne très bien pour les images, la propriété `float` n'a jamais été réellement destinée à être utilisé à des fins de mise en page et de positionnement, et donc il faut faire attention à quelques pièges.

L'un de ces pièges est que parfois les styles appropriés ne rendront pas correctement sur un élément qui sera positionné à côté ou contenu dans un élément parent d'un élément flottant. Quand un élément est flottant, il est retiré du flux normal de la page, et, par conséquent, les styles des éléments autour cet élément flottant peuvent être affectés négativement.
Souvent les valeur des propriété `margin` et `padding` ne sont pas interprétés correctement, provoquant un effondrement dans l'élément flottant; d'autres propriétés peuvent aussi être affectées.
Un autre écueil est que le contenu parfois indésirable commence à s'enrouler autour d'un élément flottant. Retirer un élément du flux du document fait que tous les éléments autour de l'élément flottant vont l'envelopper et vont supprimer tout l'espace disponible autour de l'élément flottant, ce qui n'est généralement pas souhaité.
Avec notre exemple précédent à deux colonnes, après avoir fait flotter les éléments `<section>` et `<aside>`, et avant de mettre une valeur de propriété `width` sur les deux, le contenu de l'élément `<footer>` aurait débordé entre les deux éléments flottant situé au dessus, remplissant tout l'espace disponible. La conséquence est que l'élément `<footer>` sera assis dans le caniveau entre la `<section>` et `<aside>` éléments, consommation l'espace disponible.

Fig 4 Mise en page sans clear ou contrainte de float

Pour empêcher le contenu de déborder autour éléments flottants, nous devons effacer, ou contenir, ces float en redonnant à la page à son flux normal. Nous allons voir comment l'on peut effacer les effets de la propriété float, et ensuite nous allons jeter un coup d'oeil à la façon de contenir l'effet float.

Clear Float

Supprimer le “float” se fait en utilisant la propriété `clear,` qui accepte quelques valeurs, les plus couramment utilisées étant `left`, `right,` et `both`.

```
div {
  clear: left;
}
```

La valeur `left` efface le float à gauche, tandis que la valeur `right` efface le float à droite. La valeur `both` effacer à la fois le float à gauche et à droite. Ce sera souvent la valeur idéale à utiliser.
Pour en revenir à notre exemple précédent, si nous utilisons la propriété `clear` avec la valeur `both` sur l'élément `<footer>,` nous sommes en mesure d’effacer le “float”. Il est important que ce soit appliqué à un élément apparaissant après les éléments flottants, pas avant, afin de rendre à la page son flux normal.

```
footer {
  clear: both;
}
```

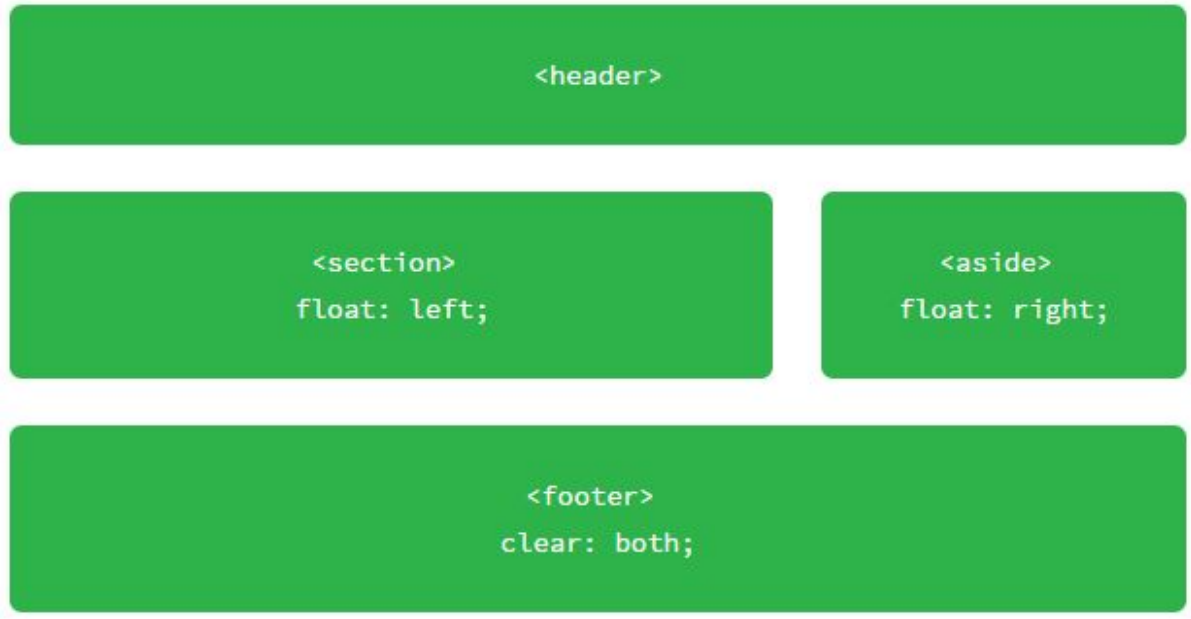

Fig 5 Mise page avec clear et float

Contenir le float

Plutôt que de supprimer le “float”, une autre option est de contenir ce “float”. Le résultat sera presque le même que d’utiliser le “clear”, cependant, contenir le “float” permet de s'assurer que tous nos styles seront rendu correctement.
Pour contenir le “float”, les éléments flottants doivent résider dans un élément parent. L'élément parent agira comme un conteneur, qui assurera que le flux du document restera tout fait normal en dehors de cet élément.

Le CSS pour cet élément de parent, représenté par la classe `group` dessous, sera le suivant :

```
.group:before,.group:after {
  content: "";
  display: table;
}

.group:after{
  clear: both;
}

.group {
  clear: both;
}
```

Il se passe pas mal de choses ici, mais ce que le CSS fait essentiellement, c'est effacer tous les éléments flottants dans l'élément avec la classe `group` et ramener le flux du document à la normale.

Plus précisément, les pseudo- éléments `:before` et `:after`, tel que mentionné dans la leçon précédente, sont générés dynamiquement pour l'élément avec la classe `group.` Ces éléments ne comprennent pas de contenu et sont affichés comme des éléments de niveau `table`, comme un éléments de niveau bloc. L'élément généré dynamiquement après l'élément avec la classe `group`, sur la pseudo classe `:after`, compense le “float” dans l'élément avec la classe `group,` tout comme le `clear` de la pseudo classe `:before`. Et enfin, l'élément avec la classe `clegrouparfix` lui même efface également tous les “float” qui peuvent apparaître dessus, dans cas où un float gauche ou droite peut exister.
Cela fait plus de code que la simple déclaration `clear: both;` seule, mais il cela peut révéler très utile surtout avec des navigateurs plus anciens.

Si l’on regarde notre mise page avec deux colonnes, nous pourrions conclure qu’il faut pour les éléments `<section>` et `<aside>` un élément parent. Cet élément parent devra alors contenir les float. Le code va ressembler à ceci:

HTML

```
<header>...</header>
<div class="group">
  <section>...</section>
  <aside>...</aside>
</div>
<footer>...</footer>
```

CSS

```css
.group:before,.group:after {
  content: "";
  display: table;
}
.group:after {
  clear: both;
}
.group {
  clear: both;
}
section {
  float: left;
  margin: 0 1.5%;
  width: 63%;
}
aside {
  float: right;
  margin: 0 1.5%;
  width: 30%;
}
```

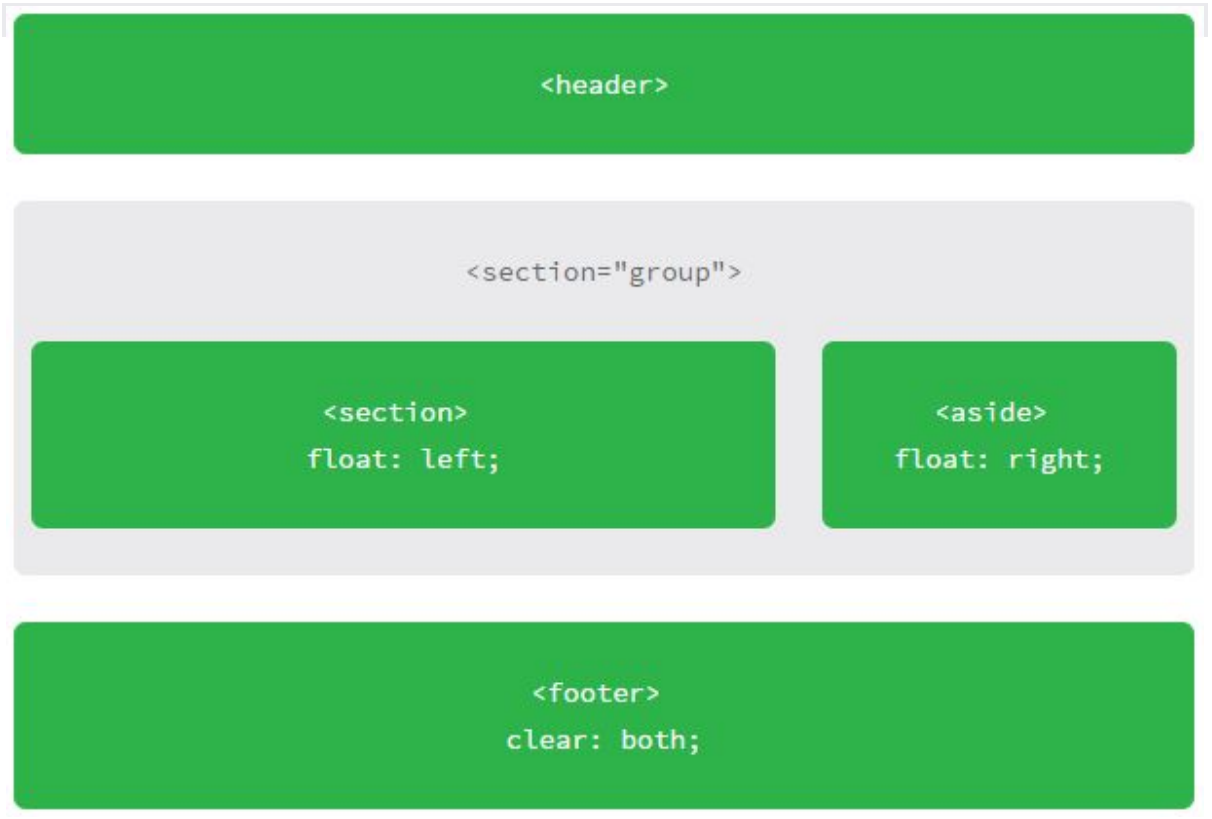

Fig 6 *Mise page avec contrainte du float*

La technique présentée ici pour contenir les éléments est connue comme un «clearfix» et sera utilisée dans de nombreux sites Web avec le nom de classe `clearfix` ou `cf`.

Nous avons choisi d'utiliser le nom de classe de `group`, car dans notre exemple il représente un groupe d'éléments, et donc cela explique mieux le contenu, soucis de sémantique of course.

Au fur et à mesure que les éléments sont flottants, il est important de noter comment ils affectent le flux d'une page et de s'assurer que le flux d'une page est réinitialisé en effaçant ou en contenant les float si nécessaire. Bien effacer des float peut causer quelques maux de tête, d'autant plus si les pages commencent à avoir de multiples rangées de colonnes.

Dans la pratique

Revenons à notre site pour essayer de faire “flotter” un peu notre contenu.
Tout d’abord, avant de commencer flottante des éléments , nous allons fournir un moyen de contenir ces flotteurs en ajoutant le clearfix à notre CSS.
Dans le fichier `style.css`, juste en dessous nos styles de grille, nous allons ajouter le clearfix sous la classe `group`.

```
/ *
  === ====================================
  Clearfix
  ============ ============================
* /
.group:before, .group:after {
  content: "";
  display: table;
}
.group:after {
  clear: both;
}
.group {
  clear: both;
}
```

Maintenant que nous pouvons contenir les float, nous allons faire flotter à gauche le premier `<h1>` au sein de l'élément `<header>` et permettre à tous les autres contenus dans l’entête de l’envelopper par la droite.

Pour ce faire, nous allons ajouter une classe `logo` à l’élément `<h1>`. Ensuite au sein de notre CSS, nous allons ajouter une nouvelle section de styles pour le header.

Dans cette section nous allons sélectionner le `<h1>` avec la classe `logo`, puis `float:left` pour qu’il flotte vers la gauche.

HTML

```
<h1 class="logo">
  <a href="index.html">Cours HTML / CSS</a>
</h1>
```

CSS

```
/ *
  ==================================== ===
  Entête
  ========================================
* /

.logo {
  float: left;
}
```

Pendant que nous y sommes, nous allons ajouter un peu plus de détails à notre logo. Nous allons commencer en plaçant un élément `<br>`, ou saut de ligne, entre le mot «Cours» et les mots «HTML/CSS» pour forcer le texte de notre logo à se mettre sur deux lignes.

Dans le CSS, nous allons ajouter une bordure en haut de notre logo et du `padding` vertical pour donner de la respiration au logo.

HTML

```
<h1 class="logo">
  <a href="index.html">Cours <br> HTML/CSS</a>
</h1>
```

CSS

```
.logo {
  border-top: 4px solid #648880;
  padding: 40px 0 22px 0;
  float: left;
}
```

Parce que nous avons rendu flottant l'élément `<h1>`, nous voulons contenir ce `float`. L'élément parent plus le plus proche de l'élément `<h1>` est l'élément `<header>`, donc nous allons ajouter la classe `group` à l'élément `<header>`.

En faisant cela, les styles de clearfix que nous avons mis en place plus tôt seront appliqués à l'élément `<header>`.

```
<header class="group container">
  ...
</header>
```

L'élément `<header>` prend forme, nous allons donc jeter un oeil à l'élément `<footer>`. Tout comme nous avons fait avec l'élément `<header>`, nous faisons flotter notre droit d'auteur à la gauche dans le `<small>` et laissons tous les autres éléments l'envelopper par la droite.

Contrairement à l'élément `<header>`, cependant, nous sommes ne va pas utiliser une classe directement sur l'élément flottant. Cette fois nous allons appliquer une classe à l'élément parent de l'élément flottant et utiliser un sélecteur CSS unique pour le sélectionner et le faire flotter.

Commençons par ajouter la classe `footer-principal` à l'élément `<footer>`. Parce que nous savons que nous allons faire flotter un élément dans l'élément `<footer>`, nous allons également ajouter la classe `group`.

```
<footer class="group container footer-principal">
  ...
</ footer>
```

Maintenant que la classe de `footer-principal` est sur `<footer>`, l'élément nous pouvons utiliser cette classe pour préqualifier la `<small>` élément avec CSS. Nous voulons sélectionner et flotter le `<small>` élément à la gauche.Il ne faut pas oublier de créer une nouvelle section au sein de notre fichier `style.css` pour ces styles de bas de page.

```
/ *
  ================== ======================
  Footer principal
  ========================== ==============
* /

.footer-principal small{
  float: right;
}
```

Pour passer en revue, ici nous sélectionnons l'élément `<small>`, qui doit résider dans un élément avec la valeur d'attribut de classe `footer-principal`,tels que notre élément `<footer>` par exemple,.
Enfin, nous allons mettre un peu de padding sur le dessus et le dessous de l'élément `<footer>` pour le séparer un peu plus du reste de la page. Nous pouvons faire directement en utilisant la classe `footer-principal`.

```
.footer-principal {
  padding-bottom: 44px;
  padding-top: 44px;
}
```

Tous ces changements pour les éléments `<header>` et `<footer>`, seront à faire sur chacunes des pages, et pas seulement la page `index.html`.

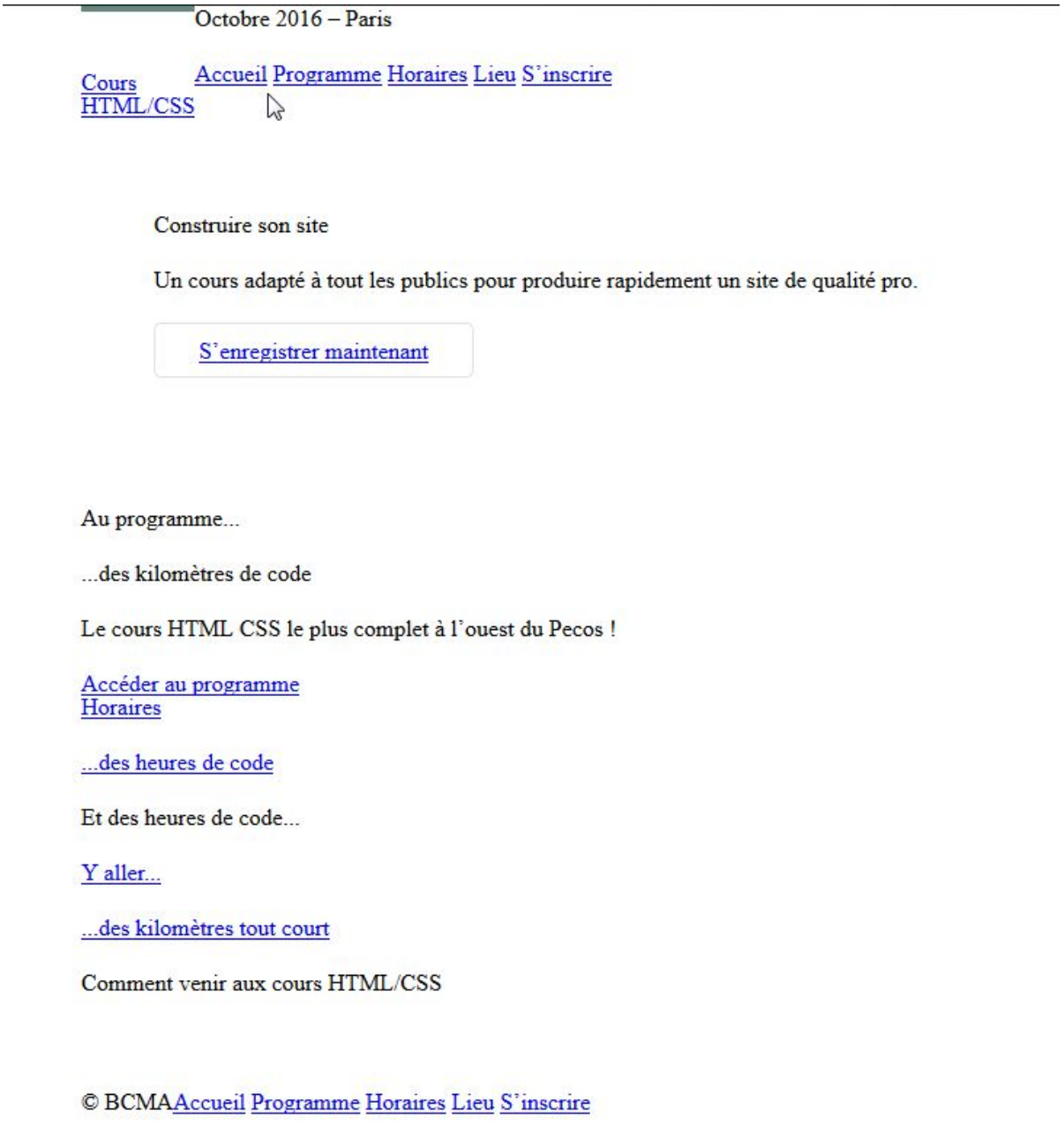

Fig 7 Avec quelques float sur nos éléments `<header>` et `<footer >` de la page d'accueil

Positionnement avec inline-block

En plus d'utiliser des float, une autre façon de positionner le contenu est d'utiliser la propriété `display` en conjonction avec la valeur `inline-block.` La méthode inline-block, comme nous verrons, est surtout utile pour faire une mise en page ou pour placer des éléments les uns à côté des autres sur une ligne.
Rappelons que la valeur `inline-block` pour la propriété `display` affiche tous les éléments sur une ligne, permettant ainsi d'accepter toutes les propriétés du modèle de boîte, y compris les `height`, `width`, `padding`, `border` et `margin.` L'utilisation d'éléments inline-block nous permet de tirer pleinement parti du modèle de boîte sans avoir à se soucier du comportement particulier du `float`, qu'il faut rétablir avec un cle`a`r.

Inline-Block en pratique

Jetons un coup d'oeil à notre exemple précédent de mis ene page trois colonnes. Nous allons commencer par garder notre HTML tel qu'il est:

```
<header>...</header>
<section>...</section>
<section>...</section>
<section>...</section>
<footer>...</footer>
```

Maintenant lieu de faire flotter nos trois éléments `<section>,` nous allons changer leurs valeurs de `display` à `inline-block,` tout en laissant les propriétés `margin` et `width` précédentes.
Notre CSS ressemblera alors à ceci:

```
section {
  display: inline-block;
  margin: 0 1.5%;
  width: 30%;
}
```

Malheureusement, ce code ne suffit pas tout à fait car on s'aperçoit le dernier `<section>` élément est poussé à une nouvelle ligne. Rappelez vous, parce que les éléments inline-block sont affichés sur la même ligne que l'autre, ils comprennent un seul espace entre eux. Lorsque la taille de chaque espace unique est ajoutée aux valeurs `width` et `margin` de tous les éléments de la ligne, la largeur totale devient trop grande, poussant ainsi le dernier élément`<section>` sur une nouvelle ligne. Pour afficher tous les `<section>` sur la même ligne, l'espace blanc entre chaque `<section>` élément doit être retiré.

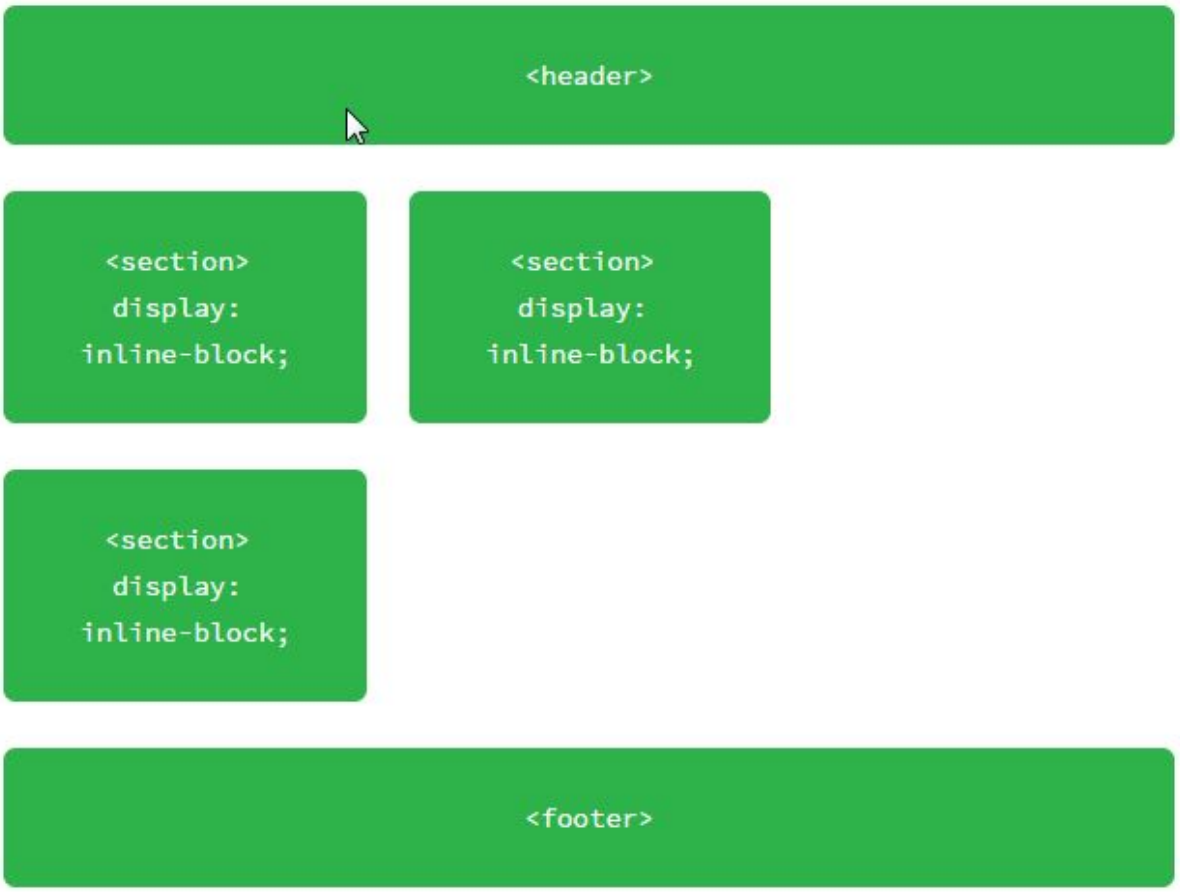

Fig 8 Eléments inline-block avec les espaces entre les éléments

Retrait des espaces entre des éléments inline-block

Il y a un certain nombre de façons de supprimer l'espace entre des éléments `inline-block`, sachant que certains sont plus complexes que autres. Nous allons mettre l'accent sur deux des moyens les plus faciles, deux que l'on peut réaliser dans le code HTML.

La première solution est de mettre à chaque nouvelle `<section>` la balise d'ouverture de l'élément sur la même ligne que la balise fermeture élément `<section>` de précédent. Plutôt que d'utiliser une nouvelle ligne pour chaque élément, nous allons finir et commencer éléments sur la même ligne. Notre HTML ressemblera à ceci:

```
<header>...</header>
<section>
  ...
</section> <section>
  ...
</section> <section>
  ...
</section>
<footer>...</footer>
```

En rédigeant les éléments `inline-block` de cette façon, cela assure qu'il n'y aura pas d'espace entre les éléments `inline-block` au sein du HTML et par conséquent aucun espace n’apparaîtra pas lorsque la page sera affichée.

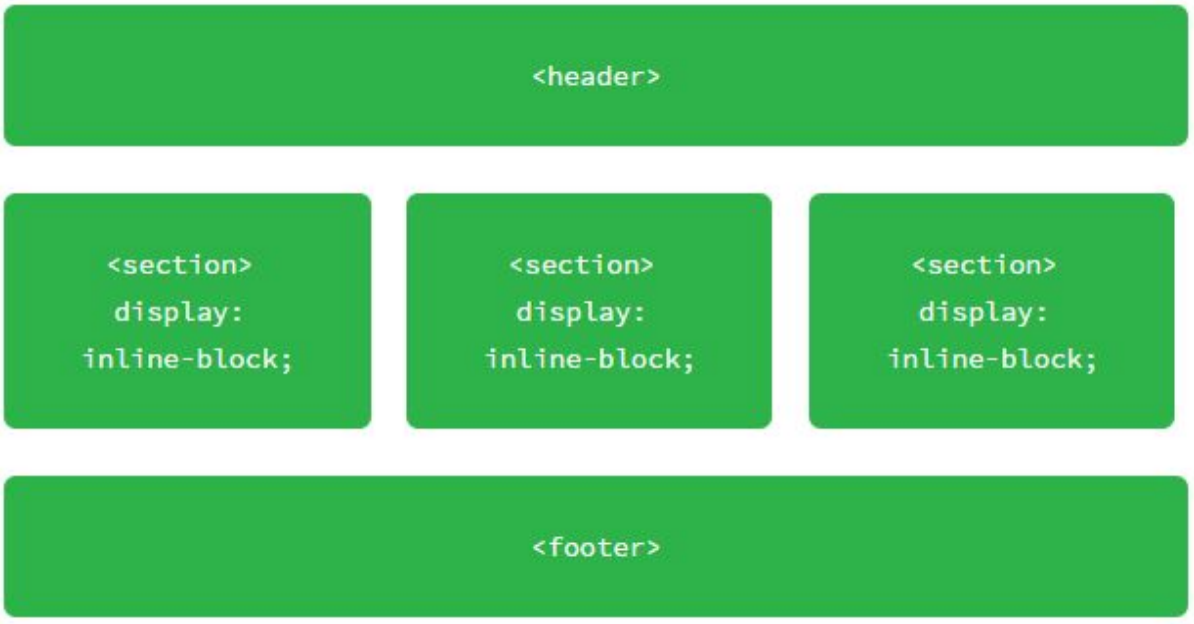

Fig 9 Eléments inline-block sans les espaces entre les éléments

Une autre façon de supprimer l'espace blanc entre des éléments `inline-block` est d'ouvrir un commentaire HTML directement après la fermeture de la balise d'un élément `inline-block`. Ensuite, fermez le commentaire HTML immédiatement avant la balise d'ouverture de l'élément `inline-block` suivant. Faire cela permet aux éléments `inline-block` de commencer et terminer sur des lignes séparées de HTML et "commenter" tous les espaces possibles entre les éléments. Le code ressemblera alors à ceci:

```
<header>...</ header>
<section>
  ...
</ section><!-- Commentaire 1
--><section>
  ...
</ section><!--Commentaire 2
--><section>
  ...
 </ section>
 <footer>...</ footer>
```

Aucune de ces options n’est parfaite, mais elles sont utiles. Je tends à favoriser l’utilisation des commentaires pour une meilleure organisation du code, mais ce n’est qu’un avis.

Création de mise en pages réutilisables

Lors de la construction d'un site Web, il est toujours préférable d'écrire des styles modulaires qui peuvent être réutilisés ailleurs, et les mises en page réutilisables sont des éléments de code réutilisable très importants. Les mise en pages peuvent être créées en utilisant soit des classes avec "float", soit des éléments avec des "display: inline-block", mais laquelle choisir pour que cela fonctionne au mieux ?

Qu'il soit préférable d'utiliser des "float" plutôt que des éléments en "inline-block" pour positionner la structure d'une page est un débat sans fin. L'approche que je vais privilégier consiste à utiliser des éléments inline-block pour créer la grille, ou mise page, d'une page et d'utiliser ensuite les float quand je veux que le contenu se positionnent aux côtés d'un élément donné (comme l'était l'utilisation initiale de la valeur d'affichage float avec les images).

Cela dit, tout est possible et si vous êtes plus à l'aise avec une approche que l'autre, alors foncez.

En pratique

Avec une solide compréhension des schémas réutilisables, le temps est venu de mettre en page de notre site.
Nous allons créer une mise en trois colonnes réutilisable en privilégiant des éléments `inline-block`. Nous ferons d'une manière qui nous permettra d'avoir trois colonnes de largeur égale ou deux colonnes qui auront pour dimension 2 tiers de la largeur totale pour la première et un tiers pour l'autre.
Pour commencer, nous allons créer des classes qui définissent la `width` de ces colonnes. Les deux classes que nous créons sont `col-1-3,` pour un tiers, et `col-2-3,` pour deux tiers. Dans la section de grille de notre fichier `style.css,` nous allons définir ces classes et leurs largeurs correspondantes.

```
.col-1-3 {
  width : 33%;
}
.col-2-3 {
  width : 66,66%;
}
```

Nous voulons deux colonnes à afficher tant qu'éléments `inline-block`. Nous devons nous assurer que leur alignement vertical est réglé en haut de chaque colonne, aussi.
Nous allons créer deux nouveaux sélecteurs qui partagent les styles d'affichage et la propriété d'alignement vertical.

```
.col-1-3, 2-3-.col {
  display: inline-block;
  vertical-align: top;
}
```

En regardant le nouveau CSS, nous avons créé deux sélecteurs de classe, `col-1-3` et `col-2-3,` qui sont séparés par une virgule. La virgule à la fin du premier sélecteur signifie qu'un autre sélecteur va suivre. Le second sélecteur est suivie par l'accolade d'ouverture " { " qui signifie qu'une ou des déclarations de style sont à suivre. En séparant les sélecteurs par des virgules, nous pouvons lier les mêmes styles à plusieurs sélecteurs.

Nous voulons mettre peu d'espace entre chacune des colonnes pour aérer le contenu. Nous pouvons accomplir cela en mettant un `padding` horizontal sur chacune des colonnes.
Cela fonctionne Cependant, lorsque deux colonnes sont sises l'une à côté de l'autre, la largeur de l'espace entre eux sera le double de celui de l'espace à partir des colonnes extérieures au bord de la rangée. Pour équilibrer cela nous allons mettre toutes nos colonnes dans une grille et ajouter le même `padding` à toutes nos colonnes de cette grille.
Nous allons utiliser un nom de classe de `grid` pour identifier notre grille, puis nous allons identifier le même `padding` horizontal pour notre grille, les colonnes de classes `col-1-3` et `col-2-3.` Notre CSS ressemblera à ceci:

```
.grid, .col-1-3, .col-2-3 {
  padding-left: 15px;
  padding-right: 15px;
}
```

Lorsque nous mettons en place le `padding` horizontal, nous devons faire attention. Rappelez vous, dans la dernière leçon nous avons créé un élément conteneur, connu sous le nom de classe `container,` pour centrer l'ensemble de notre contenu sur une page dans un élément de `960` pixel de large. Actuellement si nous devions mettre un élément avec la classe `grid` à l'intérieur d'un élément avec la classe de `container,` leurs `padding` horizontal s'ajouteraient les uns aux autres, et nos colonnes ne seraient pas proportionnelles à la largeur du reste de la page.
Pour que cela ne se produise pas, nous allons devoir partager quelques unes des règles fixées dans la classe `container` avec celles de la classe `grid` .Plus précisément, nous avons besoin de partager la propriété et les valeurs de `width` (pour vous assurer que la taille de votre page restera fixée à `960` pixels de large) et la propriété et la valeur de `margin` (pour centrer sur la page tout les élément dans la classe `grid`).
Nous allons accomplir cela en modifiant la classe `container` comme suit:

```
.container, .grid {
  margin: 0 auto;
  width: 960px;}
```

```
.container {
  padding-left: 30px;
  padding-right: 30px;
}
```

Maintenant tout élément avec la classe de `container` ou `grid` sera de `960` pixels de large et centrée sur la page. En plus, nous avons conservé le padding horizontal existant pour tout élément avec la classe de `container` en déplaçant dans un nouveau jeu de règles distinct.
Tout le travail nécessaire pour faire en sorte que nos styles de grille soient réutilisables est terminé. Maintenant il est temps de travailler notre HTML et de voir comment ces classes vont s'y insérer.
Nous allons commencer avec l'accroche sur la page d'accueil, dans notre fichier `index.html`, en alignant les sections en trois colonnes. Actuellement, elles sont insérées dans un élément `<section>` avec la classe `container`.Nous voulons changer cette classe de `container` à `grid` afin que nous puissions commencer placer nos colonnes à l'intérieur.

```
<section class="grid">
  ...
</section>
```

Ensuite, nous allons voulons ajouter une classe de `col-1-3` pour chacune des `<section>` éléments dans l'élément `<section class="grid">` avec la classe `grid`.

```
<section class="grid">
  section class="col-1-3">
    ...
  </section>
  <section class="col-1-3">
    ...
  </section>
  <section class="col-1-3">
    ...
  </section>
</section>
```

Et enfin, parce que chacune de nos colonnes est un élément `inline-block`, nous voulons nous assurer que nous enlevons l'espace blanc vide entre eux. Pour ce faire nous allons utiliser les commentaires et nous allons ajouter un peu de documentation notant chaque section venir alors que nous sommes à elle pour mieux organiser notre code.

```
<section class="grid">
  <section class="col-1-3">
    ...
  </section><!--
  Programme
  --><section class="col-1-3">
    ...
  </section><!--
   Horaires
  --><section class="col-1-3">
    ...
  </section>
</section>
```

Pour l'examen, surligne 3 nous laissons un commentaire identifiant la section "Programme" à suivre. A la fin de ligne 6, nous ouvrons un commentaire immédiatement après la fermeture de la balise `</section>`. Dans ce commentaire, sur la ligne 8 nous identifions la section "Horaires" à venir. Nous fermons alors le commentaire au début de ligne 11, juste avant l'ouverture de la balise `<section>`. Cette même structure de remarque réapparaît sur lignes suivantes entre les deux éléments `<section>`, juste avant la section "Y venir". En tout, nous avons commenté tous les espaces potentiels entre les colonnes tout en utilisant ces commentaires pour identifier nos sections.

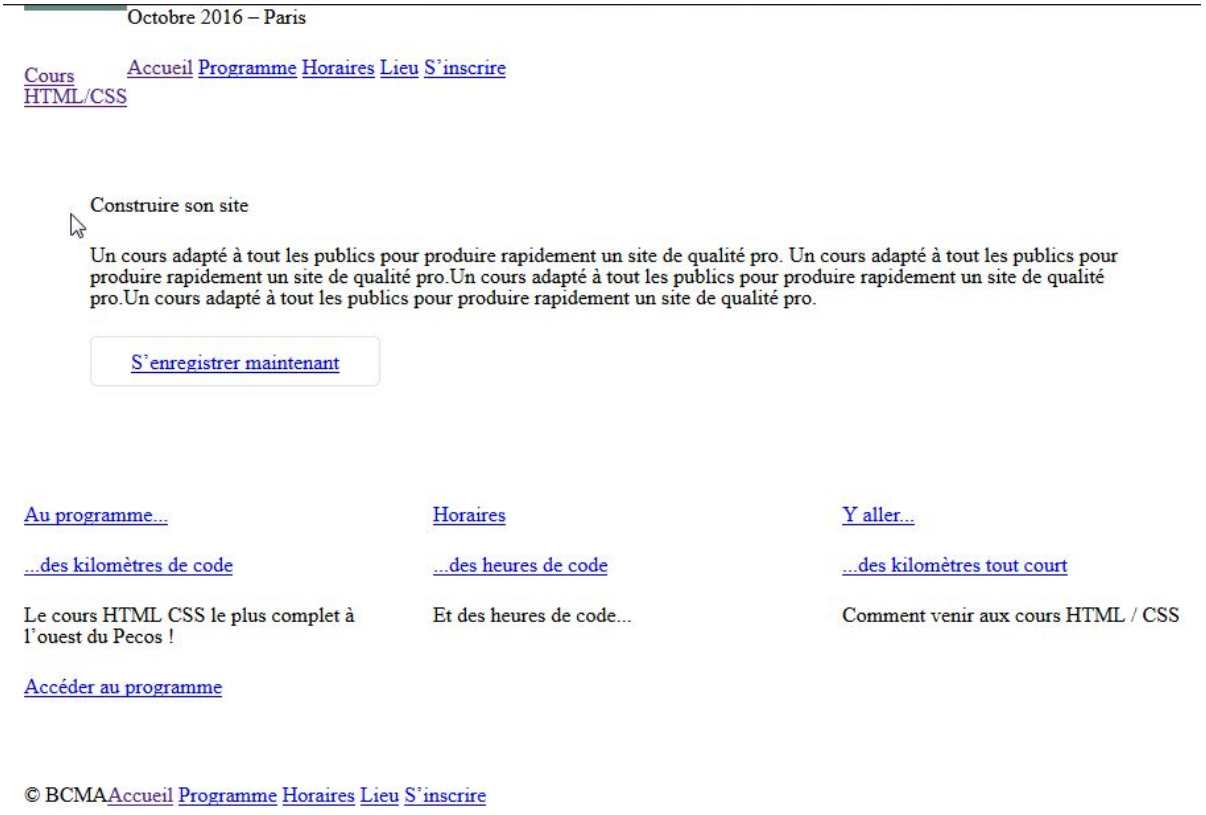

Fig 10 Notre site s'affiche désormais avec une disposition en trois colonnes

Flexbox

Flexbox est une propriété CSS qui nous permet de gérer simplement la mise en page d'une série d'éléments au sein d'un élément parent et plus précisément de :

- Mettre en place plusieurs éléments sur une ligne sans avoir à se soucier de la largeur de chacun d'eux.
- Modifier rapidement le sens de lecture vertical/horizontal.
- Aligner les éléments à gauche, à droite ou au centre du parent.
- Modifier l'ordre d'affichage des différents éléments.
- Déployer les éléments dans le parent sans problème pour gérer les marges ou la taille des éléments.

Créons notre classe

Tout d'abord il faut créer un conteneur pour nos éléments flex :

```
.flexcontainer {
    display: flex;
}
```

Flexbox utilise un principe d'axes. Si on veux des éléments alignés verticalement, on utilise une colonne (`column`), pour des éléments horizontaux (par défaut) on utilise une ligne (`row`). Nous aurons donc 2 axes appelés "*Main axis*", définis par le sens de lecture, et "*Cross Axis*" qui est perpendiculaire au premier.

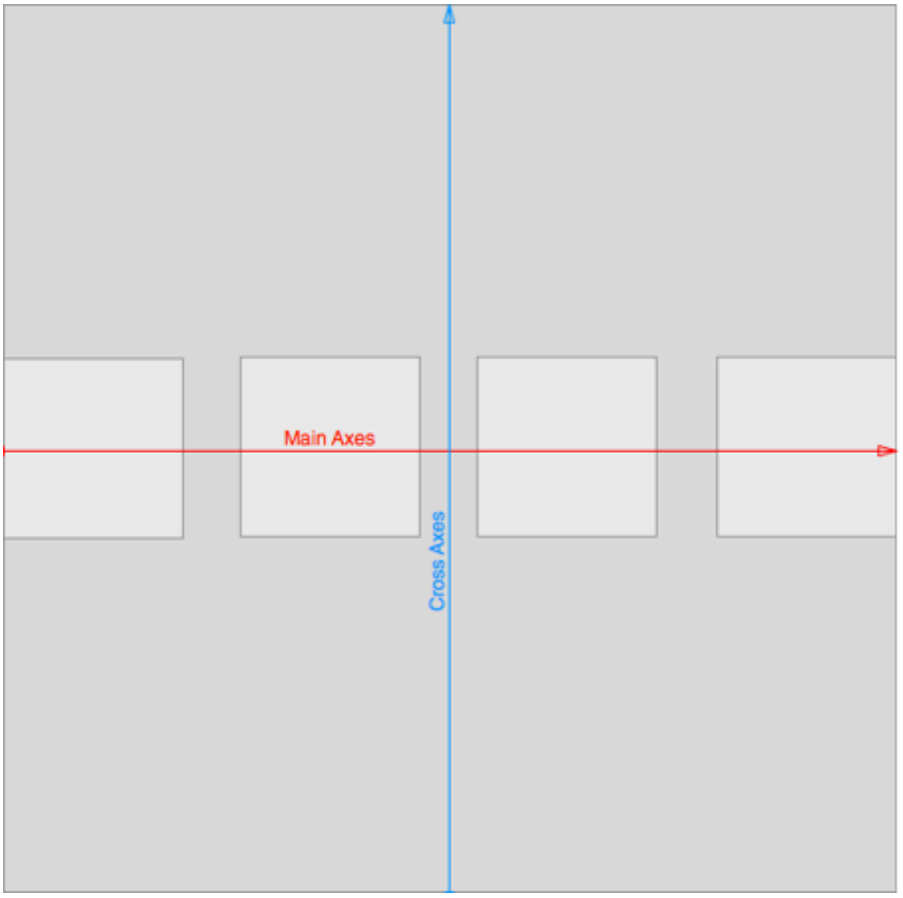

Fig 11 Main axis et cross axis

```
.flexcontainer {
   display: flex;
   flex-flow: column;
   /* ou */
   flex-flow: row;
   /* flex-flow peut aussi s'écrire flex-direction */
}
```

Fig 12 Column

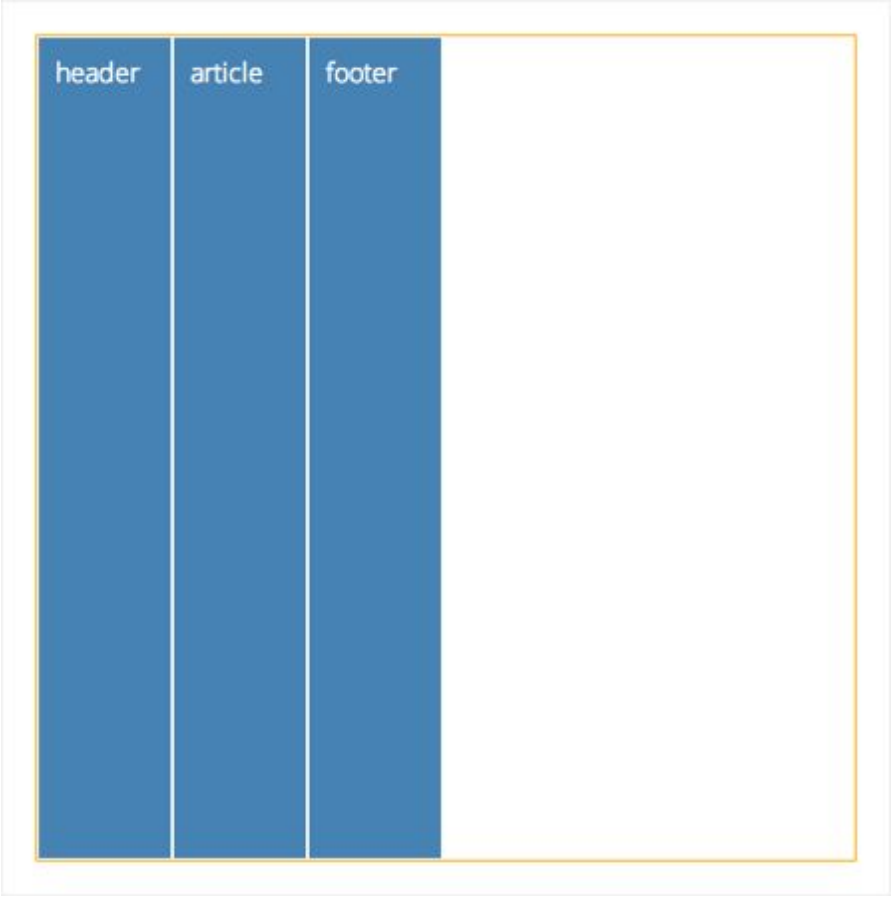

Fig 13 Row

Pour définir l'ordre d'affichage des éléments directement depuis CSS nous pouvons utiliser `row-reverse` ou `column-reverse`.

Utilisons les axes

Nous avons la possibilité de placer précisément les éléments dans notre flexbox, c'est ici que le principe d'axe vertical et horizontal prend son importance. Pour aligner les éléments à partir du début du main-axis, nous ajouterons la propriété `justify-content: flex-start`.

A retenir :

- **justify-content : main-axis**
- **align-items : cross-axis**

```
.flexcontainer {
    ...
    flex-flow: row;
    align-items: flex-end;
}
```

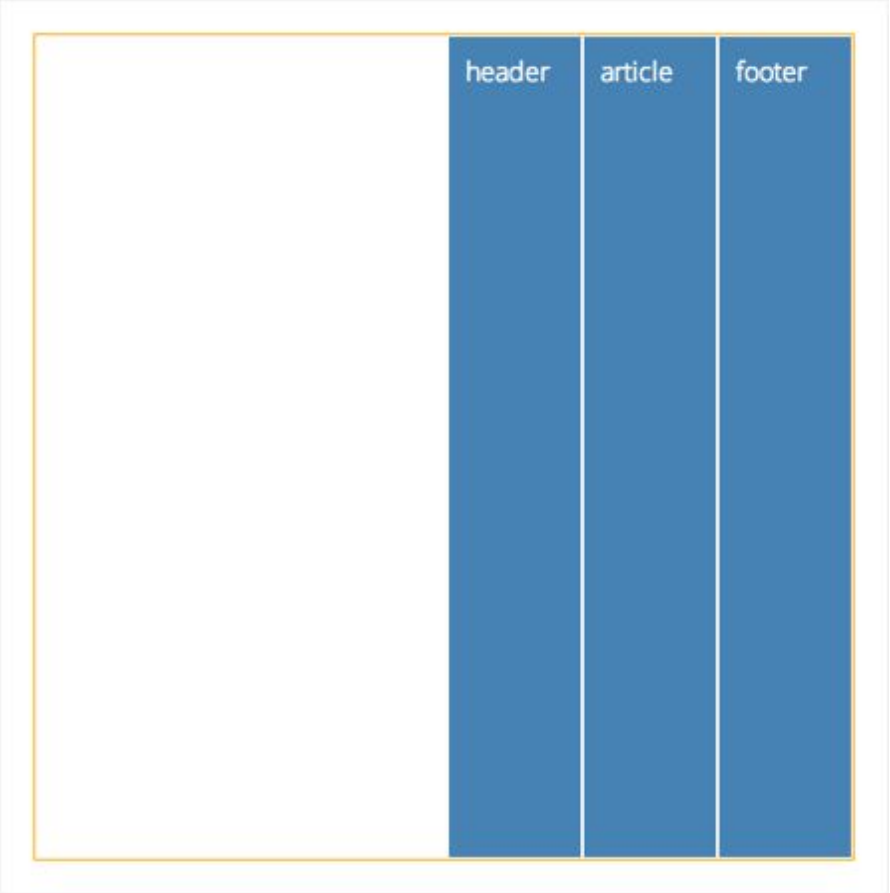

Fig 14 Alignement horizontal

Grâce à flexbox, nous avons enfin la possibilité de centrer un élément au sein d'un autre aussi bien horizontalement que verticalement.

```
.flexcontainer {
    ...
    flex-flow: column;
    justify-content: center;
}
```

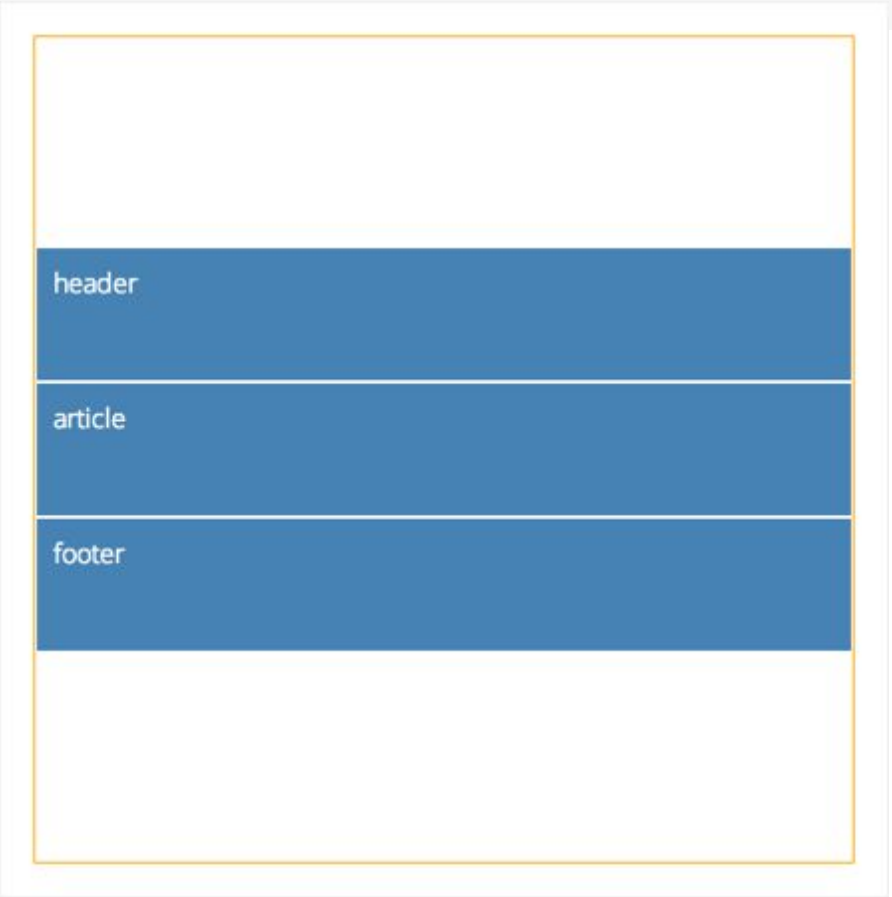

Fig 15 Alignement vertical

Aligner verticalement devient un jeu d'enfant !

Il est aussi possible de mettre des marges entre nos éléments, afin de remplir un espace donné ou simplement de séparer différents éléments. Pour cela nous utilisons à nouveau la propriété `justify-content` avec pour argument `space-around`. Les éléments de notre classe `.flexcontainer` seront séparés d'une marge identique, et d'une demi-marge sur les cotés. L'argument `space-between` aura le même effet sans les marges exterieures aux éléments.

Pour le cross-axis nous avons la possibilité de définir un alignement `flex-end`, `flex-start, center et stretch`. Ces 2 premiers arguments positionnent les éléments sur les extrémités de l'axe. `Center` place les éléments au centre du `.flexcontainer` et l'argument `strech` nous permet d'étirer les éléments afin qu'ils prennent la largeur ou hauteur du `.flexcontainer`.

Définir une taille

C’est bien pratique toutes ces nouvelles propriétés mais j’ai souvent besoin d’éléments plus grands les uns que les autres. Voyons comment nous pouvons donner une taille à nos différents éléments dans la flexbox.

```
#first {
    flex: 1;
}

#second {
    flex: 2;
}

#third {
    flex: 1;
}
```

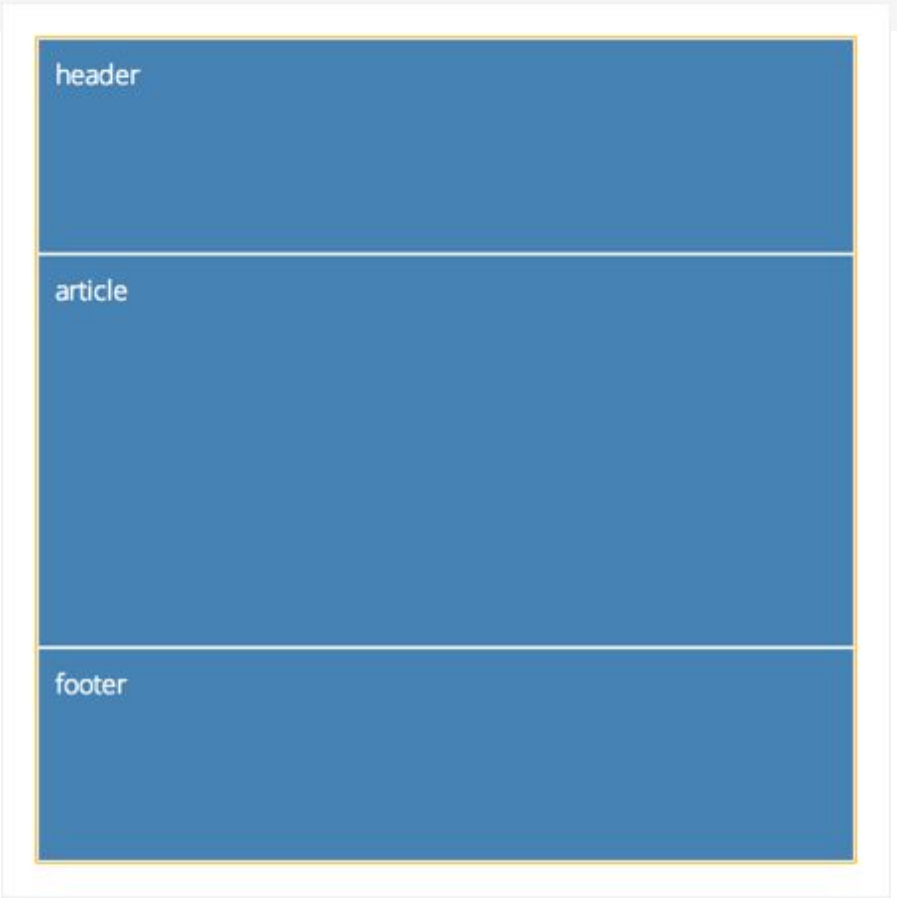

Fig 16 Taille

Le premier argument correspond au `flex-grow`. Il n'y a pas d'unité, nous utilisons des proportions. Si ces 3 éléments ont `flex: 1`; alors ils feront tous la même taille. Si l'un des ces éléments avait pour valeur 2, il prendrait 2 fois plus de place que les autres.

Il est aussi possible de définir une taille de base grâce à `flex-basis`. Lorsque le navigateur calculera la taille des éléments dans le .flexcontainer, l'espace restant après l'addition des différents tailles des éléments sera divisé proportionnellement à la propriété `flex-grow`.

Il existe enfin la propriété `flex-shrink` qui fait la même chose que `flex-grow` mais pour réduire les éléments dans une flexbox plus petite.

À retenir : La propriété *`flex`* est un raccourci pour les propriétés :

- `flex-grow`
- `flex-shrink`
- `flex-basis`

```
flex: (grow) (shrink) (basis)
```

```
.element {
   flex: 1 1 100px;
}
```

En pratique

Le temps est venu de mettre en pratique cette techno alléchante sur notre site. Pour cela nous allons dupliquer la zone de présentation à savoir la section de class `grid` et son contenu -les trois sections de présentation. Pour ne pas écraser ce que nous avons déjà réalisé, nous allons mettre en remarque html la section de classe `grid` et donc son contenu avec.

Maintenant nous allons changer le nom de classe grid en grid-flex même si on est d'accord que ce n'est pas vraiment une bonne pratique et l'on va supprimer les class col-1-3 des sections contenus dans la section avec la classe `grid-flex`. On va donc retrouver le code suivant :

```
<section class="grid-flex">
  <section>
    ...
  </section>
  <section>
    ...
  </section>
  <section>
    ...
  </section>
</section>
```

Maintenant interéssons nous au code CSS. Dans style.css nous allons ajouter deux nouvelles règles. La première concerne la classe `grid-flex` Nous allons donc définir cette classe comme devant avoir un contenu centré horizontalement et verticalement. Chacun des contenus lui aura la même taille

```
.grid-flex {
  padding: 5% 12%;
  display: flex;
  justify-content: center;
  align-items: center;
}

.grid-flex section{
  flex: 1 1 0;
  margin: 15px;
}
```

Conclusion

Parmi les points les plus novateurs de l'amélioration du CSS ces dernières années, je retiendrais avant tout les flexbox. Elles nous permettrons de nous passer de la plupart des système de grille existant, qui sont souvent lourds, peu sémantique et parfois encombrants. Une logique relativement simple, une gestion naturelle du responsive, des possibilités de mise en page bien plus avancées que celles que nous

connaissions. La nouvelle spécification **CSS *Grid*** vous aidera à faire des mises en pages encore plus nettes.

Positionnement d’un élément unique

Maintenant que faire si nous voulons positionner avec précision un élément ? “Float” qui supprime un élément de l'écoulement d'une page, produisent souvent des effets indésirables sur les éléments qui circulent autour. Les éléments `inline-block` peuvent être assez difficiles à positionner. Pour ces situations nous pouvons utiliser la propriété `position` en relation avec le décalage de boîte.

La propriété `position` identifie la *manière dont* un élément est positionné sur une page et si oui ou non il apparaîtra dans le flux normal d'un document. Ceci est utilisé en conjonction avec les propriétés de décalage de boîte `top`, `right`, `bottom`, et `left` qui identifient exactement *où* un élément sera positionné par éléments dans un certain nombre de directions différentes.

Par défaut chaque élément pour valeur de la propriété `position` la valeur `static`, qui signifie qu'il existe dans le flux normal d'un document et ne pas accepter les propriétés décalage de boîte. La valeur `static` est souvent remplacée par une `relative`, `absolute` ou `fixed`, valeurs que nous allons présenter maintenant.

Positionnement relatif

La valeur `relative` pour la propriété `position` permet aux éléments apparaissant dans le flux normal d’une page, de créer un décalage un espace pour un élément sans pour autant permettre aux autres éléments de s’écouler autour de celui ci. Par exemple, considérons le code HTML suivant et CSS:

HTML

```
<div>...</div>
<div class="offset">...</div>
<div>...</ div>
```

CSS

```
div{
  width: 100px;
  height: 100px;
  backgound-color: gray;
}
.offset{
  left: 20px;
  position: relative;
  top:20px;
```

```
    backgound-color: green;
}
```

Fig 17 Positionnement relatif d'un élément

Le deuxième élément `<div>`, l'élément avec la classe `offset`, a une valeur de `position` à `relative` et deux propriétés décalage de boîte, `left` et `top`. Cela permet de conserver la position initiale de l'élément par rapport aux autres éléments, ils ne sont pas autorisés à se déplacer dans cet espace. Les propriétés de décalage de boîte vont repositionner l'élément en le poussant 20 pixels de la gauche et 20 pixels du haut de son emplacement d'origine.
Avec des éléments en position relative, il est important de savoir que les propriétés de décalage de boîte et d'identifier où l'élément sera déplacé compte tenu de sa position d'origine. Ainsi, la propriété `left` avec une valeur de `20` pixels poussera effectivement l'élément vers la droite, depuis la gauche, `20` pixels. La propriété `top` d'une valeur de `20` pixels, alors, va pousser un élément vers le bas, depuis le haut de `20` pixels.
Lorsque nous positionnerons l'élément en utilisant les propriétés de décalage de boîte, l'élément chevauchera l'élément ci dessous plutôt que de déplacer cet élément vers le bas que les propriétés de marge ou de padding feraient.

Positionnement absolu

La valeur `absolute` de la propriété `position` est différente de la valeur `relative` en ce qu'un élément d'une valeur de position absolute n'apparaîtra pas dans le flux normal d'un document, et l'espace origine et la position de l'élément ne sera pas conservé.
De plus, l'éléments de position absolue sont déplacés par rapport à leur élément parent le plus proche de position relative. Si il n'existe pas d'élément parent de

position relative, l'élément de position absolue sera positionné par rapport à l'élément `<body>`. Exemple :

HTML

```
<section>
  <div class="offset">...</ div>
</section>
```

CSS

```
section {
  position: relative;
  backgound-color: gray;
  height: 400px;
  margin-top: 50px;
}

div{
  position: absolute;
  right: 20px;
  top:20px;
  backgound-color: green;
}
```

Fig 18 Positionnement absolu d'un élément

Dans cet exemple l'élément `<section>` est positionné relativement mais ne comprend pas de propriétés de décalage de boîte. Par conséquent sa position ne change pas. L'élément `<div>` avec une classe `offset` comprend une propriété `position` avec la valeur `absolute`.Parce que l'élément `<section>` est l'élément le plus proche parent positionné relativement à l'élément `<div>`, celui ci sera positionné par rapport à l'élément<section>.

Avec des éléments positionnés de façon relative, les propriétés de décalage de boîte identifierons dans quelle direction l'élément sera déplacé par rapport à lui même. Avec les éléments en position absolue, les propriétés de décalage de boîte seront appliqués dans la direction vers laquelle un élément sera déplacé par rapport à son élément parent de position relative le plus proche.

En raison du décalage vers la droite et le haut, l'élément apparaîtra à 20 pixels la à la droite et 20 pixels à partir du haut de l'élément `<section>`.

Parce que l'élément `<div>` est en position absolue, il n'est pas dans le flux normal de la page et il chevauchera tous les éléments environnants. En outre, la position initiale de la `<div>` n'est pas conservé, et les autres éléments sont capables d'occuper cet espace. En règle générale, la plupart positionnement peut être manipulé sans l'utilisation de la propriété `position` et le décalage des propriétés de boîte, mais dans certains cas ils peuvent être extrêmement utile.

Positionnement fixed

Un élément positionné en `fixed` est positionné par rapport à la fenêtre du navigateur, ce qui signifie qu'il reste toujours à la même place même si la page défile. De la même manière qu'avec un élément positionné en relative, nous pouvons utiliser les propriétés `top`, `right`, `bottom` et `left`. Voici le CSS :

HTML

```
<section>
  <div class="fixed">...</ div>
</section>
```

CSS

```
.fixed{
  position: fixed;
  right: 0;
  bottom:0;
  width: 100px;
}
```

Un élément positionné en `fixed` ne laisse aucun espace dans la page là où il aurait normalement dû se trouver.