GILDAS P.
CREATIVE + CODE

Web Designer
Game Designer
Développeur web

Tel. 06 31 15 13 04
Site. www.gildasp.fr
Email. contact@gildasp.fr

Université
Rennes 2

HTML5 / CSS3

Placer des blocs et autres rappels

Par défaut la plupart des éléments (`div`, `p`, `section`, `article`, etc) sont en `display:block`.
En conséquence ces blocs font toute la largeur disponible, et prennent la hauteur de ce qu'ils contiennent.

Les images sont en `display:inline` et ne créent pas de retour à la ligne, pas plus qu'ils ne prennent toute la largeur de la page – on peut donc insérer facilement de petites images au sein d'un texte.

"Ecouler" des blocs de gauche à droite

`display:inline-block`

La **solution facile**, pour une mosaïque ou un menu horizontal.

Problème : une petite marge inaltérable autour des éléments (et on n'y peut rien, les éléments en `display:inline` ont le même problème).

Par défaut, les élements `inline-block` vont s'aligner par le bas les uns par rapport autres.
Pour changer ça, il suffit d'utiliser :
`vertical-align: top;` pour qu'ils s'alignent entre eux par le haut
`vertical-align: middle;` pour qu'ils s'alignent entre eux pas leur milieu

`vertical-align` ne sert pas du tout à centrer un élément verticalement dans un bloc, mais à gérer l'alignement des blocs `inline` (comme les images) et des blocs `inline-block` entre eux.

`float:left`
(et sa variante `float:right`)

La **solution la plus propre**, pour une mise en page au pixel près.

Un élément `float:left` perd sa largeur par défaut (il ne prend plus toute la largeur dispo, il fait alors la taille de ce qu'il contient).
On peut très simplement lui redonner une largeur maîtrisée avec `width` (en px, %, vw, vh, etc).

Problème : une série d'éléments `float:left` emporte avec elle le bloc suivant.

Solution :
- côté HTML, placer une `<div class="clear"></div>` après le dernier élément en `float:left` (au même niveau).
- Côté css, la classe associée : `.clear { clear:both; }`

La `<div class="clear"></div>` assure un retour à la ligne et un retour à l'écoulement normal de la page.

Centrer un élément `display:block`

Pour centrer une `div` ou autre, il suffit de lui mettre une marge automatique à gauche et à droite :
`margin:0 auto;` ou `margin-left:auto; margin-right:auto;`

Il faut que le bloc ait une largeur attribuée avec `width` (en px, %, vw ou autre), sinon le bloc prend toute la largeur (et on n'a pas besoin de le centrer).

Centrer une série d'éléments en `display:inline-block`

Pour centrer un ensemble d'éléments `inline-block` il suffit d'utiliser `text-align:center` sur le parent des blocs en question.

`text-align:center` agit sur les éléments enfants, et non sur le bloc ciblé/stylé.

Exemple, pour centrer horizontalement les articles d'une section (et que le contenu des articles reste aligné à gauche) :

```
section { text-align:center; }
section article {
     display:inline-block;
     text-align:left; /* s'applique aux contenus des articles */
}
```

Centrer verticalement un élément

Il y a plusieurs façon de procéder, aucune n'étant réellement satisfaisante... centrer verticalement en css c'est compliqué !

Voici un site interactif entièrement dédié au sujet : http://howtocenterincss.com/
(ce qu'ils appellent "C*ontent*" c'est l'élément à centrer, et "C*ontainer*" c'est le parent)

Voici une solution pas trop complexe et plutôt généraliste, qui s'applique à l'élément parent (rien à indiquer au niveau de l'élément qu'on veut centrer) :

```
.lepapa {
     display:flex;
     justify-content:center; /* pour le centrage horizontal */
     align-items:center; /* pour le centrage vertical */
}
```

Il s'agit d'une utilisation basique des Flexbox qui donne de bons résultats.
Plus d'infos sur Flexbox : https://css-tricks.com/snippets/css/a-guide-to-flexbox/

`display:flex` n'est pas forcément bien supporté sur les navigateurs un peu anciens, mais s'il s'agit juste de centrer verticalement un élément ce n'est pas un gros problème.

Mise en page élastique / responsive

Les principes de bases d'une mise en page élastique et responsive :

- **un conteneur à largeur variable**
  par ex :
  `width:90%; max-width:1200px;` (ici avec une largeur max)
  ou :
  `width: 50vw; height:100vh;` (un bloc qui prendra la moitié de l'écran et toute la hauteur)

- **des images élastiques** (toutes, ou juste certaines images)
  `img { width:100%; }`
  Dès lors les images vont s'ajuster à la largeur de leur parent; en appliquant un `width` au parent (en px, %, vw, etc) on dimentionne aussi l'image.

Points de rupture / mise en page conditionnelle

Les media queries permettent d'introduire des cas particuliers dans votre style, par exemple pour modifier des caractéristiques de mise en page sur mobile.

Exemple, pour changer le comportement du menu quand la fenêtre d'affichage fait moins de 600px de large :

```
@media screen and (max-width:600px){
	...
}
```

Le style css entre { et } ne sera "actif" que si la fenêtre fait moins de 600px.
On peut avoir autant de points de rupture de la mise en page qu'on le souhaite, il suffit de créer de nouveaux blocs @media.

Unités responsive

Les `vw` et `vh` sont particulièrement adaptées aux mises en page élastiques qui prennent toute la fenêtre.

Si votre mise en page ne prend pas toute la fenêtre, privilégiez les %, associés à des `max-width` par exemple. Les % sont moins faciles à utiliser que les `vw` et `vh`, car ils sont en proportion du bloc parent, et non de la fenêtre.

Ajuster/couper une image

Vous pouvez facilement ajuster l'affichage d'une image grâce à `object-fit`.
Il faut d'abord que l'image ait un format, c'est à dire une largeur et une hauteur, avec `width` et `height` (en px, %, vw, etc).

A ce stade l'image va être affichée déformée – pour corriger l'affichage :
`object-fit: cover;` pour rogner/couper dans l'image
`object-fit: contain;` pour afficher l'image complète avec des marges automatiques si besoin

Vous pouvez compléter `object-fit:contain` avec une couleur de fond sous l'image, pour matérialiser les marges : `background:red;` (par ex).

Images de fond

Côté CSS vous pouvez ajouter des images de fond à n'importe quel élément HTML.
Pour ça : `background-image: url('../img/monimage.jpg');`

Le chemin vers l'image est à indiquer depuis l'endroit où est le fichier CSS.
Si votre feuille de style est dans le dossier css/, il faut ajouter `'../'` au début de l'url pour remonter d'un niveau vers la racine du site, avant de cibler le dossier img/.

Par défaut l'image est affichée à son format réel (sa taille en pixels), et répété pour occuper tout l'espace disponible.

Vous pouvez ajuster le comportement de votre image de fond :

`background-position: center center;`	centrer l'image horizontalement et verticalement
`background-position: bottom right;`	image en bas à droite
`background-repeat: no-repeat;`	l'image ne se répète pas
`background-repeat: repeat-x;`	l'image se répète horizontalement seulement
`background-size: 300px 100px;`	l'image s'affiche en 300 x 600 px
`background-size: 100% 100%;`	l'image s'étire et prend tout l'espace
`background-size: 100%;`	l'image s'ajuste en largeur et garde son ratio en hauteur
`background-size: cover;`	l'image est coupée sans laisser de vide
`background-size: contain;`	l'image s'affiche en entier avec du vide autour
`background-attachment: fixed;`	l'image ne suit pas la page lors du scroll (très utilisé sur les templates Wordpress)

Il y a d'autres options, que vous retrouverez sur cette page (à la fin de l'article) :
https://css-tricks.com/almanac/properties/b/background-image/

Toutes ces options peuvent être compactées en une seule ligne :

```
background: url('../img/monimage.jpg') center center no-repeat 100%;
```

Positionnements fixed / absolute / relative

Par défaut tous les éléments HTML sont en `position:static`.
Les positionnement `absolute` et `fixed` vont sortir l'élément du flux normal de la page et permettre de les placer avec des coordonnées précises.

Positionnement

Quand vous utilisez `position:absolute` ou `position:fixed`, il faut impérativement compléter le positionnement avec `left` ou `right` ET `top` ou `bottom` (en px, vw, vh, em ou rem).
Ce n'est pas une bonne idée de placer des blocs `fixed` en %, car dans ce contexte les % seront en proportion de l'élément lui-même, et non en proportion de la fenêtre...

Largeur

Un élément absolute/fixed perd sa largeur par défaut (il ne prend plus toute la largeur dispo, il fait alors la taille de ce qu'il contient).
On peut très simplement lui redonner une largeur maîtrisée avec `width` (en px, %, vw, etc)

Hauteur

Un élément `absolute/fixed` fait la hauteur de ce qu'il contient (comme n'importe quelle div par défaut).
Pour contrôler sa hauteur, utilisez `height` comme pour n'importe quel autre élément (en px, vh, em, rem).

Utiliser des % pour la hauteur est délicat, il faudra que l'élément parent ait lui-même une hauteur pour que ça serve à quelque chose... privilégiez les `vh` quand c'est possible !

Exemple, pour un bloc qui prendra la moitié droite de l'écran et toute la hauteur :

```
#monbloc {
     position: fixed;
     right:0; top:0;
     width:50vw;
     height:100vh;
}
```

Profondeur

En cas de superposition, vous pouvez gérer la profondeur des éléments absolute/fixed avec `z-index`.
Z-index n'est pris en compte que sur des éléments positionnés, c'est à dire en fixed/absolute ou en relative (pratique quand on veut garder une mise en page normale).

Exemple, pour une superposition calque2 dessus, calque1 au milieu, et le contenu "normal" en-dessous :

```
#calque2 { position:fixed; z-index:2; }
#calque1 { position:absolute; z-index:1; }
#normal { position:relative; z-index:0; }
```

Imbrications

Un élément `absolute` dans un élément `absolute/fixed/relative` va se placer par rapport à son parent, et non par rapport à la fenêtre.

C'est très pratique pour créer des "univers de poche", un bloc parent `relative` qui sert de référence à ses enfants `absolute`...

Et le parent positionné en `relative` garde sa place normale dans le flux de la page.

Exemple, pour une pastille qui dépasse un peu dans un coin d'un article :

```
article { position:relative; }
article .pastille { position:absolute; right:-10px; top:-10px; }
```

Zone scrollable / gérer ce qui dépasse d'un bloc

La propriété `overflow` vous permet de décider comment gérer ce qui dépasse d'un bloc.

Par défaut un élément html fait la hauteur de ce qu'il contient.
Par défaut rien ne peut donc dépasser verticalement d'une div, pour que quelque chose dépasse il faut donc donner une hauteur à votre élément (en px, %, vh, etc); sans quoi la propriété `overflow` n'a aucun effet.
Si l'élément à une hauteur plus petite que ce qu'il contient, alors overflow prend effet.

- `overflow: auto;` à privilégier, affiche une barre de scroll si nécessaire seulement
- `overflow: scroll;` pour forcer l'affichage d'une barre de scroll tout le temps
- `overflow: hidden;` pour masquer ce qui dépasse (parfois pratique)
- `overflow: visible;` le comportement par défaut, ce qui dépasse s'affiche à l'extérieur

A noter : on peut indiquer qu'on veut une barre de scroll verticale uniquement (ou horizontale uniquement), grâce à `overflow-x` et `overflow-y`.

Exemple, pour avoir un scroll vertical mais pas horizontal :

```
#monbloc {
     width:250px; height:300px;
     overflow-x: hidden;
     overflow-y: auto;
}
```

Sélecteurs CSS

`section {}`	va sélectionner tous les `<section></section>`
`#presentation {}`	va sélectionner `<div id="presentation"></div>`
`.texte {}`	va sélectionner `<div class="texte"></div>` et tous les autres éléments qui ont la classe *texte*
`section h1 {}`	va sélectionner les `<h1></h1>` qui sont dans un `<section>`
`section.truc {}`	va sélectionner les `<section class="truc"></section>`, mais pas les autres sections
`h1, h2, h3, p {}`	va sélectioner les balises `h1` ET les `h2` ET les `h3` ET les `p`

Unités de mesure

En css on peut utiliser différentes unités de mesure suivant la nécessité (ou un choix subjectif).

px	en pixels
%	en proportion de l'élément parent – suivant les cas ça marche assez mal pour les hauteurs de bloc...
em	en proportion de la font-size spécifiée pour le body problème : 2em dans un bloc lui-même en 3em, ça fera en fait 6 fois la font-size du body (et c'est pénible)
rem	en proportion de la font-size spécifiée pour le body, où qu'on soit dans le document
vw	un pourcentage de la largeur de la fenêtre, où qu'on soit dans le document
vh	un pourcentage de la hauteur de la fenêtre, où qu'on soit dans le document

Margin / Padding

Il y a 2 syntaxes possibles pour les `margin` (idem pour les `padding`).

La version détaillée :

```
margin-top:10px;
margin-right:20px;
margin-bottom:50px;
margin-left:0;
```

La même chose en version compacte :

```
margin:10px 20px 50px 0;
```

Ca se passe dans le sens des aiguilles d'une montre : haut droite bas gauche.

```
margin:0 20px;
```

c'est la même chose que

```
margin:0 20px 0 20px;
```

Pour aller plus loin

Il existe encore d'autres façons de placer vos blocs.

`Grid layout` est un système de mise en place de colonnes intéressant :
https://css-tricks.com/snippets/css/complete-guide-grid/
https://la-cascade.io/css-grid-layout-guide-complet/

`Flexbox` permet de créer des mises en page à partir de blocs et de grilles élastiques :
https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Les transformations CSS3 sont utiles pour faire des animations, mais permettent aussi de déplacer des éléments sans bouleverser la mise en page.
C'est comme un autre mode de placement, qui n'interfère pas avec le flux normal de la page, ni avec les placements absolute/fixed/relative :
https://www.alsacreations.com/article/lire/1418-css3-transformations-2d.html

Pour en savoir plus sur la compatibilité des balises CSS3 (ou de flex et grid) avec les navigateurs qui ne seraient pas à jour :
https://caniuse.com/

GILDAS P.
CREATIVE + CODE

Web Designer
Game Designer
Développeur web

Tel. 06 31 15 13 04
Site. www.gildasp.fr
Email. contact@gildasp.fr