Chapitre 6
Mise en page HTML et CSS

1. Les blocs et leur position à l'écran

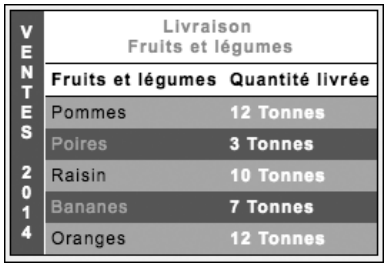
La mise en page par le code HTML est simplement le fait de positionner des blocs à l'endroit souhaité. Le sujet a été abordé au chapitre HTML - section Méthode et balises pour structurer une page et va être développé ici en combinant HTML et CSS.

1.1 Les tableaux

Ils ont longtemps été la méthode principale pour disposer le contenu d'une page. La méthode consistait à créer un tableau, et dans une cellule du tableau insérer un autre tableau, qui lui-même pouvait contenir un ou des tableaux.

C'était efficace mais il fallait s'y retrouver dans le code car tous ces tableaux imbriqués faisaient que parfois il était difficile de savoir sur quel tableau on était en train de travailler.

Repartons de l'exemple avec les fruits et légumes, vu au chapitre HTML - section Méthode et balises pour structurer une page, et voyons comment obtenir cette apparence pour notre tableau. Cet exemple peut être observé via le fichier **6_1_1_Tableaux.html**.



V E N T E S 2 0 1 4	Livraison Fruits et légumes	
	Fruits et légumes	**Quantité livrée**
	Pommes	12 Tonnes
	Poires	3 Tonnes
	Raisin	10 Tonnes
	Bananes	7 Tonnes
	Oranges	12 Tonnes

```
<table id="fruitsLegumes">
    <tr>
        <td rowspan="7" class="titreGauche">V E N T E S   2
0 1 4</td>
        <th colspan="2" class="titre">Livraison <br />Fruits et
légumes</th>
    </tr>
    ...
```

La première colonne « VENTES 2014 » utilise la propriété HTML `rowspan`. L'espace occupé par ce `<td>` sera équivalent à 7 lignes, si bien que la première colonne prendra toute la hauteur du tableau. Le texte est écrit avec des espaces entre chaque lettre, pour faire qu'à l'affichage les lettres soient les unes en dessous des autres.

Le titre « Livraison Fruits et légumes » utilise la largeur de deux colonnes grâce à la propriété `colspan`, qui regroupe deux colonnes. Le `<br />` fait que le texte s'écrit sur deux lignes.

La balise `<table>` a par défaut des marges internes et un espace entre les bordures. Si aucune valeur ne vient corriger cela, il y aura de l'espace entre les cellules, et pour les lignes de détail, ce serait assez disgracieux d'avoir un espace en arrière-plan entre « Pomme » et « 12 Tonnes ». Pour cela, il faut mettre l'espace entre les cellules à 0. Cela pourrait être écrit directement dans la balise `<table>` en écrivant `cellspacing="0"`. La méthode utilisée ici passe par le CSS.

```
#fruitsLegumes, th {
    letter-spacing: 1px;
    border:1px solid #000;
    margin: 0;
    padding: 0;
```

```
    border-spacing: 0px;
}
```

La propriété CSS `border-spacing` permet de modifier cette valeur, et elle est forcée à 0.

Les deux premières lignes du tableau, des `<th>`, avec le fond blanc, ont une bordure intérieure. La balise `<th>` (*table head*) écrit automatiquement en gras et au centre. La balise `<th>` est stylisée avec l'ID `#fruitsLegumes` et devrait avoir une bordure tout autour d'elle de 1 pixel, noire, avec un trait continu.

Pour annuler cela, un style est créé uniquement pour la balise `<th>` qui remet la bordure à 0 pour l'ensemble et en fixe une pour le bas :

```
th {
    padding: 4px;
    background-color :#fff;
    color: #000;
    border:0px solid #000;
    border-bottom: 1px solid #000;
}
```

De la même façon, le titre de gauche a une bordure blanche sur le côté droit :

```
.titreGauche {
    text-align: center;
    width: 15px;
    background-color: #1362bb;
    color:#FFF;
    font-weight: bold;
    border-right: 1px solid #FFF;
}
```

Les tableaux restent utilisables pour l'affichage de données comme dans cet exemple. Leur principal défaut est de toujours ressembler à un tableau. Pour le responsive design, il est pratique de pouvoir, uniquement avec le CSS, changer complètement l'apparence d'une page. Or, avec le tableau ce sera soit impossible soit très compliqué. C'est pour cela que les balises `<div>` ont tant de succès.

1.2 Les div et les nouvelles balises HTML5

Les div ont été abordées dans le chapitre CSS3 et des notions de débordement et de positionnement ont été vues. Avec ce qui a été abordé entre-temps, il va être possible de faire une page HTML avec une zone header (ou menu) en haut de la page qui ne bougera pas quand l'ascenseur descendra dans la page, de la même façon que le bas de page, qui pourra rester toujours en bas quelle que soit la taille de la page. Voir l'exemple : **6_1_2_lesDivs.html**.

Pour le code HTML, il y a quatre grandes parties. Elles sont incluses dans les balises `<header>`, `<nav>`, `<div>` et `<footer>`.

```
<header id='topPage'>Un TOP qui bouge</header>
<nav id='menu'>Un menu qui bouge jusqu'en haut de la page</nav>

<div id='mainPage'></div>

<footer id="basPage">Un Footer fixe.</footer>
```

Bien que le nom de ces quatre balises soit complètement différent, elles vont fonctionner de la même façon, s'afficher de la même façon et avoir les mêmes propriétés possibles au niveau CSS. En fait, leur nom va permettre d'aider à détecter leur contenu. La balise `<header>` se trouve en haut de quelque chose, ici en haut de la page. La balise `<nav>` contient les éléments prévus pour la navigation, comme un menu. Le div a déjà été vu et ne dit rien sur son contenu, et pour finir la balise `<footer>` est la zone la plus basse d'un bloc.

Le code CSS permet de positionner les éléments précisément. L'utilisation de la position `:fixed` ainsi que de `z-index` permet de faire en sorte que si le contenu de la page scrolle, le bas de page reste toujours à sa place. Et un peu de code JavaScript va permettre de faire en sorte que le menu monte jusqu'en haut de la page et s'arrête à ce moment-là.

```
html, body {
    padding: 0;
    margin: 0;
}
```

Mise à zéro des différentes marges pour la partie HTML et pour le BODY, de façon à avoir 100% de la page disponible.

```
#topPage {
    position: absolute;
    z-index: 100;
    top: 0;
    background-color: #365D88;
    width: 100%;
    height: 100px;
}
#menu {
    position: absolute;
    z-index: 100;
    top: 100px;
    background-color: #264D78;
    width: 100%;
    height: 40px;
}
#basPage {
    position: fixed;
    z-index: 100;
    bottom: 0;
    background-color: #365D88;
    width: 100%;
    height: 50px;
}
```

Les `#topPage`, `#menu` et `#basPage` ont leur valeur de z-index fixée à 100. Ils peuvent avoir la même valeur puisqu'ils ne se chevauchent jamais.

Le `#topPage` commence en étant calé en haut (`top :0;`) et le menu le suit en ayant le top à 100 pixels, ce qui correspond à la hauteur du `#topPage`.

Le `#basPage` a une position fixe en bas, puisque `bottom` est à 0 et que position est sur `fixed`.

```
#mainPage {
    position: absolute;
    z-index: 90;
    width: 100%;
    height: 2000px;
    background: rgba(248,80,50,1);
    background: -moz-linear-gradient(top, rgba(248,80,50,1) 0%,
```

```
rgba(230,198,39,1) 100%);
    background: -webkit-gradient(left top, left bottom, color-
stop(0%, rgba(248,80,50,1)), color-stop(100%, rgba(230,198,39,1)));
    background: -webkit-linear-gradient(top, rgba(248,80,50,1) 0%,
rgba(230,198,39,1) 100%);
    background: -o-linear-gradient(top, rgba(248,80,50,1) 0%,
rgba(230,198,39,1) 100%);
    background: -ms-linear-gradient(top, rgba(248,80,50,1) 0%,
rgba(230,198,39,1) 100%);
    /*background: linear-gradient(to bottom, rgba(248,80,50,1)
0%, rgba(230,198,39,1) 100%);*/
    filter:progid:DXImageTransform.Microsoft.gradient(
startColorstr='#f85032', endColorstr='#e6c627', GradientType=0 );
}
```

Le `#mainPage` correspond à la partie principale du site. Il passera derrière le menu ou sous le menu et sous les éléments fixes en règle générale.

Si ce code est exécuté, le `#menu` va bouger vers le haut sans s'arrêter lorsqu'il sera au maximum de la page. Il continuera de monter et disparaîtra de l'écran.

Il est possible d'interroger le navigateur pour savoir si l'ascenseur est en cours d'utilisation, autrement dit, si la page est en train de scroller.

Pour que le `#menu` monte et s'arrête en haut, il suffit de détecter le déplacement du `#menu`, et lorsqu'il arrive à la limite de la page, de changer sa propriété CSS `position` et de la rendre fixe de façon à ce qu'il ne bouge plus. Il faudra naturellement la remettre relative lorsque l'ascenseur redescendra pour que le `#menu` retrouve sa place à l'écran.

```
function init() {
    var element = document.getElementById('menu');
    posDepartMenu =
parseInt(window.getComputedStyle(element).getPropertyValue('top')
);

    leDoc = document.documentElement;
    leBody = document.body;
}
```

La fonction `init();` déclenchée par le `onLoad` de `<body>` va récupérer dans la variable `posDepartMenu` la position de départ du menu.

Même si la position de départ du menu est connue dans le CSS et est de 100 px, le CSS pourrait changer. La valeur pourrait ne plus être 100 et si la valeur 100 était inscrite en dur dans le code JavaScript, le CSS ne s'afficherait plus correctement. Cela permet de dissocier le CSS du JavaScript.

La valeur récupérée par `getPropertyValue('top')` sera en fait « 100px ». Les deux caractères px vont être embêtants pour effectuer le calcul, JavaScript considérant cela comme du texte. L'instruction `parseInt()` permet de convertir « 100px » en une valeur entière. Nous avons donc la valeur 100 stockée dans `posDepartMenu`.

Les deux variables `leDoc` et `leBody` sont en fait juste là pour ne pas avoir à réécrire la ligne de code complète : il suffira d'écrire `leDoc` plutôt que d'écrire `document.documentElement`. De même que pour `leStyle` ci-dessous :

```
window.onscroll = function(event) {
    var leTop = (leDoc && leDoc.scrollTop || leBody &&
leBody.scrollTop || 0);
    document.getElementById("basPage").innerHTML = leTop + " / "
+ posDepartMenu;

    var leStyle = document.getElementById('menu').style;
    if (leTop > posDepartMenu) {
        leStyle.top = '0px';
        leStyle.position = 'fixed';
    } else {
        leStyle.top = posDepartMenu + 'px';
        leStyle.position= 'relative';
    }
};
```

`window.onscroll` est l'événement qui est déclenché lorsque l'utilisateur scrolle la page. La fonction ci-dessus sera donc appelée à chaque fois que l'utilisateur fera bouger l'ascenseur.