Best-practices HTML5 & css3

Retour aux fondamentaux html5 & css3

1. Rappel sur les balises de mise en forme et structurante en html5 vs balises générique html. Exemple non sémantique :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Balises sémantiques html</title>
</head>
<body>
    <h1>Structure générique</h1>
    <div id="wrapper">
        <p>Contenu</p>
    </div>
</body>
</html>
```

Ce code sera ignoré par les bots d'un moteur de recherche sauf le titre puisque celui-ci se trouve dans l'entête de premier niveau. La balise « div » qui a un rôle d'enveloppe du contenu ne sera pas interprété par les dispositifs spéciaux pour les non-voyants et le paragraphe sera masqué. Idem pour le SEO le contenu sera difficilement indexé par les moteurs de recherche voir à 60% et l'idéale serait 80%. Aujourd'hui pour améliorer la visibilité de la page dans les moteurs il faut ajouter dans les « meta » la balise **<meta name="viewport">** cela permet d'ajouter des média queries et rend la page visible dans les smarts phone et tablettes.
Exemple d'amélioration du code :

```
<body>
    <h1>Structure générique</h1>
    <div id="wrapper" role="main">
        <p>Contenu</p>
    </div>
</body>
```

Avec l'ajout d'un attribut « role » dans la balise div spécifié par la WAI (Web Accessibility Initiative), la structure devient plus optimale car le contenu sera accessible aux non-voyants.
Exemple sémantique :

```html
<!DOCTYPE html>
<html lang="FR-fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Balises sémantiques html</title>
</head>
<body>
    <h1>Structure générique</h1>
    <main>
        <p>Contenu</p>
    </main>
</body>
</html>
```

Voici ci-dessus le code optimal à la fois pour les machines et les humains.
La balise main remplace la balise div et il n'y a pas besoin d'ajouter un « role » car ces balises sont interprétées par les dispositifs spéciaux. Cidessous le rapport d'optimisation de la page :

Puisque les bonnes pratiques sont appliquées on n'a pas à se soucier de la visibilité du contenu.

Best Practice : appliquons les bonnes pratiques

2. Avec la version 3 du web tout doit avoir un sens et tout doit être accessible par tout. Dans ces nouvelles pratiques il n'y plus de place au hasard. La décentralisation de l'information nous impose une certaine rigueur.
   Ci-dessous les cinq point à respecter :

- Performance
- Progressive web app
- Accessibilité
- Bonne pratiques SEO

1. Pour la performance la page doit correspondre aux normes W3C : « doctype » exclusivement html5 aucun mixage avec des versions antérieures est possible.

2. Progressive Web App : sortir des sentiers battus et créer des sites qui peuvent grâces au service worker se transformer en application web pour les mobiles. Le service worker impose 10 règles de réalisation pour

les sites « progressive web app »

3. Accessibilité de l'information pour les humains et les machines.

4. Bonne pratique pour la structuration et l'optimisation du code html sémantique.

5. SEO optimisation des pages pour un référencement naturel grâce à la sémantique html 5 à la hiérarchisation du contenu. Il est conseillé également d'utiliser les micros données du schema.org : « itemtype et itemprop ». Respect d'un vocabulaire spécifique à un sujet.

Les balises structurantes HTML5

Header, main, section, footer

Pour commencer à structurer un document dans la partie body il est conseillé d'appliquer les quatre balises ci-dessus. Comme dans la construction d'un bâtiment ces balises servent de fondation pour la structure de notre page.
La type de display pour ces balises et de type block et les bloques sont des grands containeurs qui ont la capacité de contenir d'autre petite structure comme les h1, p, listes et formulaires.

Quelque propriété display (block, inline, item-list et table)

1. Display block pour les grands containers
2. Display inline pour les petits containers dédié à la partie texte

Tableau explicatif des éléments html sémantiques

Balises	Sens sémantiques
<p>	L'élément p définit **un paragraphe. Balise de type bloc**
les balises <h1> à <h6>	En général, h1 contient le titre le plus important de la page et donc du contenu, puis viennent les sous-titres h2, h3, etc. Après, on respecte ou non, mais ce qui compte, c'est de garder une certaine hiérarchie. Balise de type bloc
et	L'élément ul, avec son compagnon li, définit une **liste non ordonnée**. On peut l'utiliser par exemple pour une liste de liens dans un menu, ou une liste de courses... Balise de type item-list
et	L'élément ol accompagné de li définit une **liste ordonnée**. On peut s'en servir pour un sommaire ou une liste de commentaires... Balise de type item-list
<dl>, <dt> et <dd>	L'élément dl définit une **liste de définitions**. De ce fait, dl sera toujours utilisé avec dt, qui définira le terme, et dd qui contiendra la définition de ce terme. Balise de type item-list
<table>, <tr>, <th>, <td> et <caption>	Les éléments table, tr, th et td définissent un **tableau** et doivent donc contenir des données tabulaires, comme un forum et pas autre chose !
	L'élément strong définit une **forte insistance** sur un ou des mots. Ces mots prennent donc un autre sens sémantique que du texte normal : on dit qu'ils sont *emphasés*, ou plus communément *mis en évidence*. Bien pour le SEO. Le mot s'affiche en gras. Balise de type *inline*

D'autres type inline

	L'élément em définit une **insistance moyenne** sur un ou des mots. Ces mots sont emphasés, mais dans une moindre mesure qu'avec l'élément strong. Bein pour le SEO. Le mot s'affiche en italic. Balise de type *inline*
<ins>	L'élément ins définit une **insertion** de mots depuis la création du document. Ainsi, si un jour vous rédigez un article d'actualité et qu'une dépêche vient d'arriver, vous pourrez l'annoncer dans votre article grâce à l'élément ins ! L'élément est bien évidemment contenu dans une balise <p> au même titre que strong et em. Bien pour le SEO. Balise de type *inline*
<del>	L'élément del « *delete* », définit un ou des mots (toujours imbriqués dans une balise <p> **qui ont été supprimés** du document pendant son évolution. Balise de type *inline*
<pre>	L'élément pre désigne un texte qui est déjà **préformaté**, c'est-à-dire que le texte est écrit comme il doit apparaître : le nombre d'espaces blancs est respecté, et les sauts de ligne peuvent être désactivés. Balise de type bloc

<q>	L'élément q définit une **courte citation** dans un paragraphe. Balise de type *inline*
<blockquote>	L'élément blockquote est très proche de q, mais cette fois blockquote définit une **longue citation** qui peut être découpée en plusieurs paragraphes, c'est-à-dire qu'il contiendra toujours une balise <p> pour définir le texte : c'est une balise de type bloc.
<cite>	Nous restons encore dans le domaine des citations avec l'élément cite, puisqu'il désigne effectivement l'**auteur** d'une citation ou une **référence** vers une autre source. Balise de type *inline*
<abbr>	L'élément abbr sert à définir une forme abrégée, c'est-à-dire une entité qui est écrite de façon réduite. Exemple : « www » ou « http » <abbr>www</abbr>. Balise de type *inline*
<sup> et <sub>	Les éléments sup et sub servent respectivement à **mettre en exposant et en indice**. Balise de type en ligne
<code>	l'élément code indique un fragment de **code informatique**, rien de bien compliqué. Balise de type *inline*
<data>	L'élément qui peut contenir des données dans l'attribut value. Type inline
<article>	L'élément article comme son nom l'indique est sensé contenir des articles. On peut trouver ce type de balise dans des sites de press. Type bloc

Best Practice : La css3

Une fois que la structure d'une page html est faite il nous reste qu'habiller tous cela par la css. La css a pour rôle de rendre la page plus attractive et compréhensible visuellement. Adoptons les bonnes pratiques.
Pour bien structurer une css il faut avoir à l'esprit une vision claire de notre modèle html. Chaque navigateur applique sa css dans les éléments html par défaut : une mise à zéro de ces propriétés s'impose.

Le reset css

Pour tous Webdesigners cela devient un casse-tête car il faut traduire la charte graphique et maquette visuelle réalisé sur Photoshop en css. Cela nous renvoi aux bonnes pratiques html. En effet un manque de connaissances en balises html nous décourage rapidement. Il faut savoir que chaque balise html surtout les types bloc contiennent des propriétés par défaut. Pour le savoir il suffit de se servir de la console du navigateur et inspecter chaque élément. Rapidement on se rend compte qu'un paragraphe prend toute la largeur de la fenêtre et contient des marges haut et bas de 16px. Pour un élément h1 c'est 21px. La police de caractère affichée par défaut est du serif de couleur noir.
Pour remettre tout cela à zéro par le biais de notre css il faut réinitialiser ces propriétés voici un exemple de reset css :

```
body{
    font: 16px sans-serif;
    margin: 0;
}
h1,
h2,
h3,
p,
ul,
ol,
blockquote
{
    font-weight: 100;
    margin:0;
}
a{
    text-decoration: none;
}
```

Dans cet exemple toutes les balises susceptibles de nous causer des problèmes au niveau de leur propriété ont subi une modification de leurs valeurs. Pour que cela soit appliqué en cascade à un certain nombre d'éléments, la syntaxe affichée permet une économie de ligne. Inutile de déclarer cela pour chaque élément sur 7 lignes, la css nous permet de prendre un raccourci et grâce à chaque élément séparé par une virgule on peut attribuer à tous la même valeur.

Se rappeler que pour une bonne notation **CSS** vous devez cibler :

Elément	Propriété	Valeur
html	css	css

Comment peut-on attribuer des valeurs différentes css à une liste ?

Dans une problématique plus avancé d'attribution de valeurs différentes à un ensemble d'éléments « frères » de type list-item en css, la pratique la plus connu et celle d'attribuer des ID ou class à chaque élément de liste. La id est un attribut unique dans la page il ne peut pas être utilisé pour d'autres éléments ou sinon il faut en déclarer d'autre avec un nom différent. La class contrairement à la ID permet d'être utilisé plusieurs fois dans la page et dans un ensemble d'éléments. Dans une bonne connaissance du (DOM) on peut se passer de ces pratiques ou au moins ne pas trop en abuser. Le but c'est de savoir où cet élément se trouve dans la page pour lui appliquer des propriétés css souhaitées.

Première exemple classique d'attribut d'une couleur différente à deux éléments de listes :

```
<ul>
<li id="first"><data value="100">HTML</data></li>
<li><data value="90">XML</data></li>
<li><data value="80">jSon</data></li>
<li id="last"><data value="70">Css</data></li>
</ul>
```

Ci-dessus on décide pour ne pas être embêté d'ajouter deux attributs ID pour appliquer deux couleurs différentes au 1er élément et au dernier élément. A mon sens ce n'est pas la meilleure solution car les id peuvent avoir un rôle complétement différent de celui qu'on lui donne ici dans ce code. Pour un Webdesigner peut être satisfaisant mais pour la gestion des attributs cela peut devenir un casse-tête pour l'optimisation du code. Voici la css appliquées à ces deux attributs :

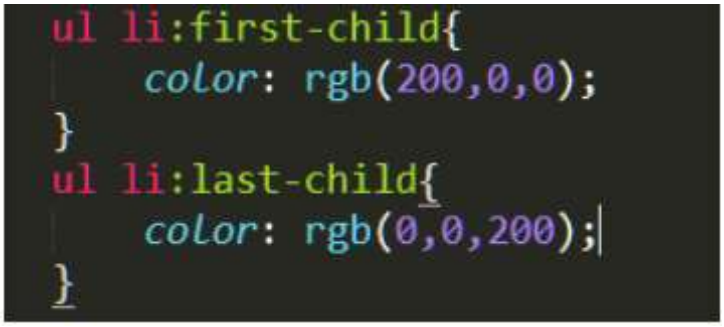
En effet cela semble clair mais vu sur cet angle j'ai du mal à savoir par la css où ces éléments appliquent leur valeur ?

Faisons plus simple compliquons les choses

Modifions le html :

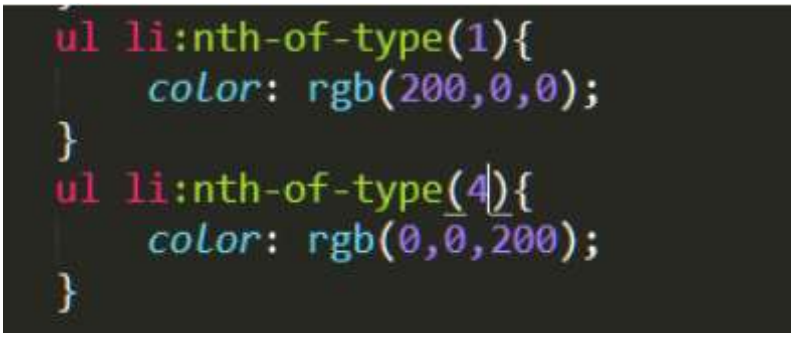
Dans le code ci-dessus les id ont été supprimé mais comment puis-je distinguer le 1er élément du dernier ?

Regardons la css :



Ci-dessus on indique à la css que le premier élément li de sa liste a une couleur rouge. Par la même méthode nous indiquons que le dernier est de couleur bleu. Ici on a une visibilité plus claire par rapport à ces propriétés car on sait déjà où ces couleurs ont été appliquées.

Une autre solution possible est la suivante :



```
ul li:nth-of-type(1){
    color: rgb(200,0,0);
}
ul li:nth-of-type(4){
    color: rgb(0,0,200);
}
```

L'énième enfant de types 1 ou 2. Par cette méthode on cible l'index des éléments et par ordre d'emplacement on attribue la couleur.
Rien de compliqué cela fait partie des bonnes pratiques de travail.