Commençons

(Bonjour, mon nom est Stéphane)

Objectif

- Vue claire de la dynamique d'une page web
- Compréhension de la logique des CMS
- Quelques réflexes de « bonne pratique »
- La réponse à un maximum de questions

Code is poetry

Le code html de toutes les pages créées peut être lu, analysé, copié-collé

Web is democracy

Les navigateurs DOIVENT s'accommoder de la disparité et des erreurs du code pour afficher aux mieux les contenus

Le W3C recommande

Il ne peut imposer, mais concerte les acteurs pour éviter une explosion du vocabulaire des codes utilisés sur le net.

Alors pourquoi bien coder ?

- Une bonne structure assure une stabilité du rendu visuel
- Une bonne structure est plus facile à modifier
- Une bonne structure est mieux indexée par les moteurs de recherche
- Une bonne structure est un gage de pérennité de la page

Tout commence par un arbre

Si une page web est un ensemble organique fait de html et css, amené à la vie par javascript, la base reste une structure html sémantiquement et syntaxiquement correcte.

Et une page, c'est un arbre.

Matériel pour ce premier jour

- Le dossier de travail de ma clé USB
- Un éditeur de code (smultron, notepad++, ...)
- Firefox
- Extensions pour firefox :
    - firebug et/ou Web developer
    - Colorzilla

HTML

HyperText Markup Langage

Le HTML,
Langage de description

Le html n'est pas un code d'instruction, donc pas un « langage de programmation ».

Il livre un contenu, et un balisage qui donne des indications sémantiques sur ce contenu.

Additionnellement, il donne des indications sur la page et les normes qu'elle utilise.

```
1  <html>
2   <head>
3    <title>
4     Voici le titre
5    </title>
6   </head>
7   <body>
8       <p>
9    Une phrase avec un <a href="cible.html">hyperlien</a>.
10      </p>
11     <p>
12      Ceci est un paragraphe.
13      </p>
14  </body>
15 </html>
```

Une balise

<balise attribut="valeur">Contenu</balise>

Ex : Clic

Ok, on pourra dire aussi un TAG.

L'arbre

Les balises s'imbriquent pour former l'arbre de la page.

Ex:

```
<body>
    <p>
        Tags <em>imbriqués</em>
    </p>
</body>
```

Parents et enfants

Lorsque l'on parle des relations entre éléments à l'intérieur de l'arbre, on parle de parents et d'enfants.

Logiquement, les éléments enfants sont ceux contenus dans un élément parent.

Cette notion sera importante lors de la mise en forme visuelle.

Principales balises

<head> : définit l'entête de la page
<body> : le corps de la page, la partie visible
<h1> <h6> : différents niveaux de titres
<p> : un paragraphe
<a> : lien
<img /> : image
**** : emphase (italique)
**** : gras
** **: une liste “ non-ordonnée“

: un retour chariot

** ** et **** : deux balises “ non sémantiques“

Comportement visuel par défaut

Disons-le une première fois, car on le répétera :
Toutes les balises, hormis div et span (qui sont neutres sémantiquement comme vu), ont un comportement visuel par défaut !

Par exemple, les liens sont bleus et soulignés.

Block et inline :
Deux types de balise

Il existe deux types de balises. Aucune exception à ces types.

Les balises de type “ **block**“ se comportent comme des boîtes : elles occupent 100 % de la largeur de leur parent, prennent la hauteur nécessaire à l'affichage de leur contenu, possèdent des marges, et forcent la mise à la ligne : elles se placent sous l'élément qui les précède, et les éléments qui les suivent se trouverons eux aussi à la ligne.

Seuls les éléments de type “ block“ peuvent être positionnés.

Block et inline :
Deux types de balise

Les éléments de type “ **inline** “ se comportent comme du texte : ils occupent en largeur et hauteur l'espace nécéssaire à leur affichage, n'ont pas de marges, ne peuvent pas être positionné.

a, img, em et strong sont des balises inline.

Exemple : Voici un texte avec emphase.

Bonnes pratiques html

- Pensez sémantique
- Ce qui s'ouvre se ferme
- Codez les balises en minuscules
- Mettez les attributs entre guillemets
- Evitez le surcodage
- Commentez votre code :

  <!-- comme ceci -->

Exercice 1
Un peu de formatage sémantique

Un texte provenant de wikipedia, brut, à baliser.

Pensez sémantique, pensez à l'organisation de la page.

Allez, faites ça en 10 minutes...

Donner quelques précisions au navigateur

Si comme on l'a dit, le navigateur web est conçu pour s'accommoder de toutes les situations, il est préférable de lui donner des indications précises pour être sûr qu'il restituera correctement votre code.

- le doctype
- le namespace
- le type d'encodage des caractères

Donner quelques précisions au navigateur

Ça donne ça :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
```

Dans le cas d'un document xhtml 1.0 transitionnel avec un encodage en utf-8, bien sûr.

CSS

Cascading Style Sheets

Les CSS entrent dans la danse

Les CSS décrivent le comportement visuel des éléments de la page. Ces comportements peuvent être différents et même en conflit avec la sémantique des éléments, bien qu'il y aie généralement une relation logique entre les deux.

Les css ont été inventées pour mettre de l'ordre dans le html, et séparer structure du code (sémantique) et mise en page (visuelle).

Le W3C recommande la séparation des codes. C'est une bonne pratique même si ça n'est pas toujours possible.

Déclarer un comportement CSS : 3 endroits où le faire

On peut décrire pour le navigateur le comportement d'un élément de la page à trois endroits :

1) Dans l'élément lui même

```
<p style=“ color:red“ >Un texte en rouge</p>
```

Déclarer un comportement CSS : 3 endroits où le faire

On peut décrire pour le navigateur le comportement d'un élément de la page à trois endroits :

1) Dans l'élément lui même

```
<p style=“ color:red“ >Un texte en rouge</p>
```

Déclarer un comportement CSS : 3 endroits où le faire

On peut décrire pour le navigateur le comportement d'un élément de la page à trois endroits :

2) Dans le “head“ de la page

```
<head>
    <style type='text/css'>
        p { color:red; }
    </style>
</head>
```

Déclarer un comportement CSS : 3 endroits où le faire

On peut décrire pour le navigateur le comportement d'un élément de la page à trois endroits :

3) Dans un fichier séparé (recommandé)

```
<link href='style.css' rel='stylesheet' type='text/css' />
```

Déclarer un comportement CSS : syntaxe

Element { attribut: valeur; }

Par exemple :

```
body {
    font-family: arial, helvetica, sans-serif;
    font-size:11px;
    color: #ccc;
    }
```

Déclarer un comportement CSS : 3 façons d'atteindre les éléments

On peut sélectionner les éléments à styler de trois façons :

1) sélectionner par balise

```
p {
    color:#fc0;
    font-size:14px;
    Line-height:140%;
    }
```

Déclarer un comportement CSS : 3 façons d'atteindre les éléments

2) sélectionner par id

Html : <p id=“ principal“ >un texte</p>

Css :

```
#principal {
border:1px solid red;
}
```

Déclarer un comportement CSS : 3 façons d'atteindre les éléments

3) sélectionner par classe

Html : <p class=“ niveau1“ >un texte</p>

Css :

```
.niveau1 {
border:1px dashed yellow;
letter-spacing:0.1em;
}
```

Héritage CSS

La notion d'héritage est centrale dans le codage css

Certains comportements sont passés des parents vers les enfants.

La définition de l'attribut font-family employée pour la balise body, par exemple va affecter tous les éléments de la page.

Par contre, un margin n'affectera que l'élément pour lequel il est défini.

Sélection hiérarchique

Autre notion importante : la sélection hiérarchique

```
#principal p {
        color:red;
        }
```

Ne va affecter que les paragraphes situés dans un élément possèdant l'id “ principal“ et aucun autre paragraphe dans la page.

Priorité

Et encore une notion importante !

a { color:green; } supplante le bleu des liens
p a { color:yellow; } supplante la ligne précédente
#principal a { color:red; } supplante la ligne précédente

Si deux règles ont la même importance, la dernière est prise en compte.

Comportement visuel par défaut

Disons-le une deuxième fois :
Toutes les balises, hormis div et span (qui sont neutres sémantiquement comme vu), ont un comportement visuel par défaut !

Par exemple, les h1 jusque h6 ont des tailles par défaut, sont en gras et possèdent une marge en haut et en bas.

Ok, un cas pratique

Une page à habiller, avec une image dans le texte, une image de fond, un positionnement déjà fait.

A vous de coder les comportement visuel de tout ça.

Positionnement CSS

Le big stuff.

Le positionnement permet de modifier le comportement dans l'espace de la page des éléments.

Rappel : seuls les éléments de type “ block“ peuvent être positionnés.

Il y a 4 types de positionnement.

4 positionnements et une bidouille

On peut positionner un élément dans le flux, en relatif, en absolu, en fixe. Ajoutons le positionnement en float et on a tous les outils disponibles.

A vous de coder les comportements visuel de tout ça.

Par défaut : le flux

Un élément positionné en “static“ se place sous son élément frère - ou soeur - qui le précède dans le code et pousse vers le bas l'élément frère - ou soeur - suivant.

```
#element {
position:static;
}
```

Un peu bizarre : le relatif

Un élément positionné en “relative“ reste dans le flux, mais peut bouger de cette position sans que cela n'affecte l'endroit qu'il occupe, ni la position des éléments frère – ou soeur – qui le suivent.

```
#element {
position:relative;
left:-10px;
}
```

Le king dangereux : l'absolu

Un élément positionné en “absolute“ quitte le flux et se positionne relativement à son ancêtre positionné.
Les éléments suivants l'ignorent, ce qui peut provoquer des chevauchements foireux si on ne pense pas bien le positionnement des éléments.
Un élément en absolu adapte sa taille par rapport à son contenu.

```
#element {
position:absolute;
Left:0;
top:50px;
}
```

Le frère mal aimé : le fixe

Un élément positionné en “fixed“ quitte le flux. Il se positionne relativement à la fenêtre du navigateur.

```
#element {
position:fixed;
Left:0;
top:50px;
}
```

La bidouille : le float

Un élément recevant un comportement en float rest lié à sa position dans le flux. Les éléments inline qui le suivent vont respecter son positionnement, mais pas les éléments de type block, qui vont l'ignorer.
On peut forcer les éléments suivants à se placer sous lui en leur donnant la propriété “clear“ en left, right ou both.

```
#element {
float:right;
width:200px;
}
```