

Guide des Projets Web

Méthodologie pour l'écriture du code
(X)HTML

LE STUDIO

Date : Septembre 2008/Février 2009

Index

1	Introduction	5
2	Philosophie de travail	7
2.1	Le bon code est l'âme d'un site	9
2.2	Le choix du langage.....	12
3	Les règles.....	13
3.1	La déclaration du DOCTYPE	13
3.2	L'appel aux feuilles de style.....	15
3.3	Déclaration du " <i>namespace</i> "	17
3.4	Déclaration de la langue principale	18
3.5	Définition du " <i>content-type</i> "	19
3.6	Titre du document.....	21
3.7	Les règles de compatibilité XHTML.....	22
3.8	Respecter la syntaxe du langage et ses éléments	24
3.9	Utiliser les tabulations et les retours à la ligne pour un code clair	26
3.10	Ne pas "forcer" l'ajout d'espaces ou les retours à la ligne.....	28
3.11	Valider votre code	29
4	Méthodes et conventions.....	30
4.1	L'outil d'édition.....	30
4.2	Appeler toujours la feuille de styles de développement	32
4.3	Le code (X)HTML d'abord, le design CSS après (mais l'un n'exclut pas l'autre) ..	33
4.4	Respecter les règles de nommage de styles	34
4.5	Conventions des commentaires	36
4.6	Ecrire un code flexible et évolutif.....	38
4.7	Eviter de commencer l'édition (X)HTML par la page d'accueil	39
4.8	Penser en termes de "composants"	40
5	Les bonnes pratiques	46
5.1	Eviter les " <i>div-ities</i> " (la « DIVite aiguë »): Chaque balise a sa fonction, chaque contenu a un rôle	46
5.2	Bien considérer l'attribution de classes et ou d'Ids : ni trop/ni trop peu.....	47
5.3	Il ne doit pas exister de contenu "anonyme"	49
5.4	Ne pas ajouter de propriétés directement dans la page avec l'attribut "style"	50
5.5	Utiliser l'attribut "title" avec discernement	51
5.6	Savoir utiliser les listes de définitions	54
5.7	 et sont des informations sémantiques : définissez-les dans le code	57
5.8	Utiliser des lignes (<hr />) pour séparer les principaux groupes de contenu	58
5.9	Toujours utiliser la balise <h1> pour le titre du site.....	59
5.10	Le titre du site doit envoyer l'utilisateur à la page d'accueil	61
5.11	Toujours ajouter un lien caché pour sauter les menus	62
5.12	Créer vos menus avec soin.....	63
5.12.1	Un menu de rubriques (ou une liste de liens) doit toujours être structuré par des balises de liste ()	63
5.12.2	Traiter et identifier la rubrique courante.....	65
5.13	Une ligne de navigation (fil d'Ariane) n'est pas une liste.....	69
5.14	Les images.....	70
5.14.1	Donner toujours une alternative textuelle aux images.....	70
5.14.2	Donner ou pas une dimension aux images ?.....	71

5.14.3	Pour les images décoratives, préférez l'utilisation d'images de fond	72
5.15	Formulaires	73
5.15.1	Donner toujours un Id aux formulaires	73
5.15.2	Bien considérer les attributs de la balise <form>	74
5.15.3	Utiliser toujours les fieldsets pour grouper les champs de même nature....	75
5.15.4	Si possible, préférer la balise <legend> pour le titre d'un fieldset	76
5.15.5	Utiliser les labels pour associer un intitulé à son champ	77
5.15.6	Toujours grouper le label et son champ dans une balise <p>	78
5.15.7	Attribuer toujours un ID et un "name" à un champ de formulaire	79
5.15.8	Prévoir la construction de boutons liens et des boutons input	80
5.15.9	Créer une structure pour les boutons des formulaires	83
5.15.10	Coder correctement les <i>Checkboxes</i> et/ou <i>Radios</i>	85
5.15.11	Prévoir les éléments d'information fréquents dans un formulaire	90
5.16	Tableaux	92
5.16.1	Utiliser la bonne balise pour le titre d'un tableau	92
5.16.2	Ajouter un résumé des données du tableau	92
5.16.3	Utiliser la balise correcte pour les entêtes de colonne ou de ligne	93
5.16.4	Renseigner la logique de lecture du tableau	94
5.16.5	Considérer les cas de figure possible (alignement cellules, lignes alternées, lien pour les entêtes, flèche de tri, etc.)	95
6	Le codage javascript	99
6.1.1	Ne pas définir une fonction directement dans la page	102
6.1.2	Séparer le comportement du contenu	103
6.1.3	Donner toujours une alternative au javascript	103
6.1.4	Penser les fonctions pour qu'elles ne soient pas dépendantes d'un seul type d'input ou de navigateur	104
6.1.5	Utiliser la syntaxe correcte pour le passage des variables par une URL ...	105

Vos contacts chez Le Studio AW

Le Studio se tient à votre disposition pour vous apporter tous renseignements complémentaires. Vos interlocuteurs sont :

Angela RICCI

☎ 03 20 60 80 16

Auteur

✉ angela.ricci@atosorigin.com

Jean-Luc MACKE

☎ 03 20 60 79 35

Responsable Studio

✉ jean-luc.macke@atosorigin.com

Merci à Jean-Luc pour la relecture et les nombreuses corrections. Merci également à Frédéric Bruant pour la relecture et les commentaires, et à Sophie Vanraes pour sa patience à me corriger et à donner du sens à mes phrases.

Principes permanents

Au même titre que le Studio, vous êtes acteurs des projets web. Vous souhaitez participer au fait que la qualité du produit final dépende aussi de vous ?

Alors un moyen simple, avant de démarrer et tout au long du développement, c'est de se poser les questions suivantes :

1. **"Est-ce que je fais bien, et tout de suite ?"**

Nous pouvons céder à la tentation d'écrire un code rapide et pauvre avec l'idée de modifier ou corriger plus tard. Vous le savez, on le sait tous, on ne va jamais revenir sur ce qui a été fait parce qu'on n'a pas le temps, parce qu'on ne se souvient plus, etc. On doit donc faire bien et réfléchir tout de suite, tout le temps. Et, en plus, ça ne nous prend pas plus de temps.

2. **"Est-ce que je fais simple et avec assurance ?"**

Les bonnes solutions sont souvent les plus simples, dès lors qu'on maîtrise son sujet. Pour ça, n'hésitez pas à consulter les personnes capables de vous aider ou de faire une recherche approfondie sur les sujets concernés.

3. **"Est-ce que je teste, teste, teste... ?"**

Notre design, nos fonctionnalités, l'accessibilité et l'utilisabilité doivent être testés, et ça dans différents contextes (vitesse de connexion, par exemple), différents hardwares, différents navigateurs, etc.

4. **"Est-ce que je crée et je produis pour le futur ?"**

Nos efforts de création et de production doivent valoir la peine : on doit toujours concevoir, produire et développer en ayant l'évolutivité et la flexibilité de notre produit en tête.

1 Introduction

Comme nous l'avons vu dans le document d'introduction de ce guide, le propos du Studio est ici de promouvoir et de diffuser sa démarche pour assurer la qualité de nos produits lors de la production de sites web.

Quand nous parlons de qualité, nous visons surtout l'ensemble des caractéristiques inhérentes à un site qui rendent possible, **de façon efficace, économique et contrôlée**, son **évolution**, sa **maintenance** et sa **portabilité** vers d'autres médias.

Notre but est d'offrir à nos clients le meilleur produit possible, dans les plus brefs délais et en réduisant de façon importante le temps et le coût de maintenance et de suivi.

L'élément majeur qui est le point de départ de la démarche et qui joue un rôle primordial est la volonté d'écrire un code source de qualité pour nos pages web. C'est tout le propos de ce document qui propose une méthodologie pour l'écriture de ce code comme nous le verrons ci-après.

Mais avant cela, passons en revue chacune des caractéristiques qui font la qualité d'un site c'est-à-dire : l'évolution, la maintenance et la portabilité.

L'évolution ou la compatibilité future

On dit qu'un site est "évolutif" si sa compatibilité future est assurée, c'est-à-dire, s'il a été construit de façon à toujours être supporté par différents distributeurs et périphériques dans un futur même lointain.

Le respect des standards web est un des moyens de garantir la compatibilité future, car c'est sur ces standards que l'évolution du web repose.

Respecter les standards est donc un des moyens le plus sûr d'assurer à long terme l'investissement fait sur votre produit web. Si nous ne sommes pas prêts à respecter les standards avec rigueur, nos efforts seront vains.

La maintenance

La maintenance d'un site web est incontournable car elle dépend de l'évolution du site.

Pour favoriser la maintenance, un site doit être :

- **modulaire** : encore une fois, la séparation données/présentation/comportement apparaît comme un grand atout. En effet, elle permet de gérer séparément chacun de ces aspects, sans interférences ou conflits, par chacun des experts dans leur domaine respectif.
- **documenté** : ça va sans le dire : la maintenance d'un système est dix fois plus facile quand il est clairement documenté.

En effet, si un site est pensé de façon "modulaire" et correctement documenté nous pouvons effectuer rapidement un diagnostic afin d'identifier et d'isoler la source d'un problème.

La portabilité

On dit qu'un site est "portable" quand la totalité de son contenu (ses données) peut être accédé par n'importe quel système web – téléphones, bornes, navigateurs, mobiles, synthétiseurs vocaux, etc.

La séparation données/présentation/comportement permet la récupération des données "pures" d'un site et de les adapter pour un système spécifique.

La portabilité d'un site web permet une très grande économie de temps et de moyens et évite la duplication d'efforts en termes de développement et de maintenance. En effet, la "matrice" des données ainsi affranchie de toute information de présentation ou de comportement, n'a pas besoin d'être recréée pour chaque nouveau système, mais simplement adaptée pour mieux s'intégrer au média.

Les efforts qui sont dans ce cas "économisés", peuvent être consacrés pour des questions plus pointues comme l'adaptation de l'interface au média, pour une expérience utilisateur optimale.

2 Philosophie de travail

Remarque : La plupart des conventions, recommandations et bonnes pratiques listées dans ce document s'appliquent quelque soit du langage (XHTML ou HTML) choisi durant la phase de cadrage initial. Les différences entre ces deux langages seront commentées s'il y a lieu.

Au début du design CSS, nous avons tendance à toujours écrire notre code de la même manière qu'avec une mise en page en tableau, mais en remplaçant simplement les balises `<table>` par des balises `<div>`, et en leur ajoutant des classes à "tout-va" à ces balises. Ainsi, on ne faisait rien de plus qu'auparavant, si ce n'est séparer le contenu de la forme, en supprimant toutes les définitions de styles telles que les balises `` ou `<center>`, ou encore les attributs tels que `align`. Et c'était bien tout compte fait, car c'est ce qu'on était supposés faire, non?

Mais est-ce que cette "séparation" contenu/forme est suffisante ? Sommes-nous sûrs de savoir pourquoi notre façon de travailler doit changer ?

Si on a pensé que les choses seraient plus faciles, on s'est bien trompé, car non seulement on travaillait toujours de la même façon mais en plus de cela, on s'est ajouté une difficulté supplémentaire : le design CSS.

Si on devait écrire le code pour une page comme celle ci-dessous, cela se traduirait par :



```
<div class="titre">ACTEURS ET PROJETS</div>
<div class="paragraphe">U....</div>
<div class="soustitre">...</div>
<div class="paragraphe">....</div>
<div class="soustitre">...</div>
<div class="paragraphe">....</div>
<div class="soustitre">...</div>
<div class="paragraphe">...</div>
<div class="titrenoir">...</div>
<div class="soustitrebleu">Terra Numerica</div>
<div class="paragraphe">....</div>
```

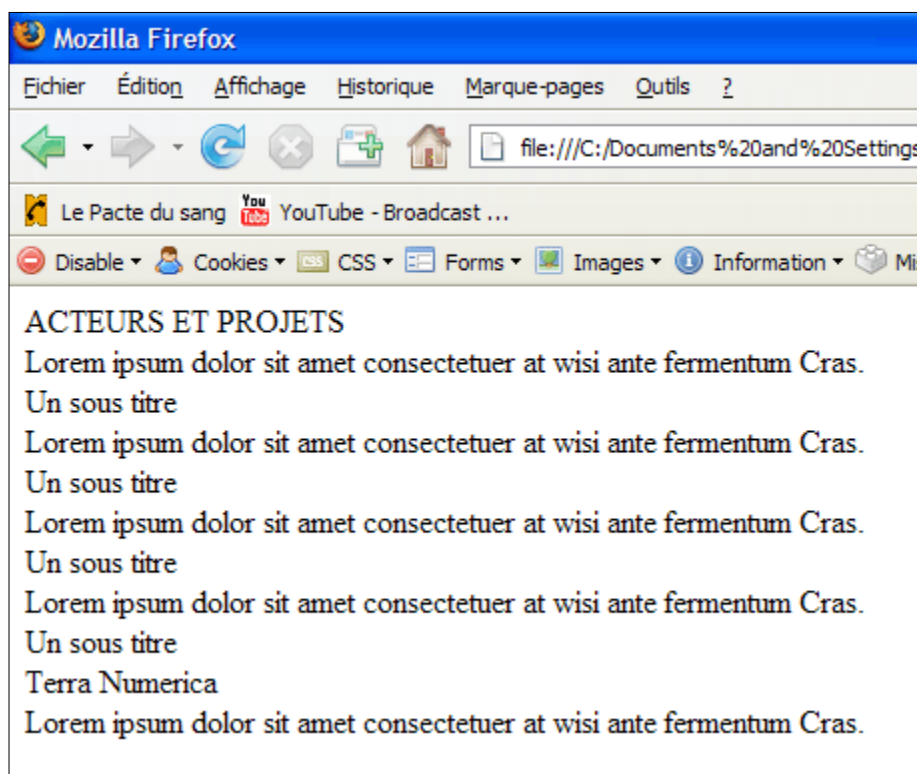
ou alors ...

```
<div class="titre">ACTEURS ET PROJETS</div>
U....
<div class="soustitre">...</div>
....
```

```
<div class="soustitre">...</div>
.....
<div class="soustitre">...</div>
.....
<div class="titrenoir">...</div>
<div class="soustitrebleu">Terra
Numerica</div>
.....
```

Ces deux exemples de code sont aussi complexes et redondants que l'était notre code avec la mise en page "tableau". Effectivement, on n'a pas d'instructions de style, mais on n'a pas de structuration hiérarchique ni de logique de contenu. Ce qui à coup sûr pose problème pour l'évolutivité du site, sa portabilité sur d'autres agents utilisateurs ou encore l'accessibilité.

Ce code, vu sans sa feuille de styles, donnait :



A part le fait que le titre principal est écrit en majuscules, on ne distingue absolument pas le rôle de chaque ligne de texte dans l'ensemble du document. Les titres et sous-titres se confondent avec les paragraphes. Lorsque l'on n'a pas les instructions de style, le texte devient difficile à lire et, plus grave encore, ça rend impossible l'interprétation du contenu par d'autres agents utilisateurs ou applications.

Du point de vue design CSS, nous avons une feuille de styles redondante, car il a fallu ajouter des définitions qui normalement n'ont aucune raison d'exister. Et le plus délicat c'est que, dans le cas du deuxième exemple de code, on n'a même pas la possibilité de donner un style

aux paragraphes de façon simple, car ils se présentent comme des contenus " anonymes " sans leurs balises.

Ce code complique la façon d'atteindre une portion de contenu spécifique par la feuille de styles, et il rend la dynamisation plus difficile. Rien n'est vraiment identifiable ou spécifique.

2.1 Le bon code est l'âme d'un site

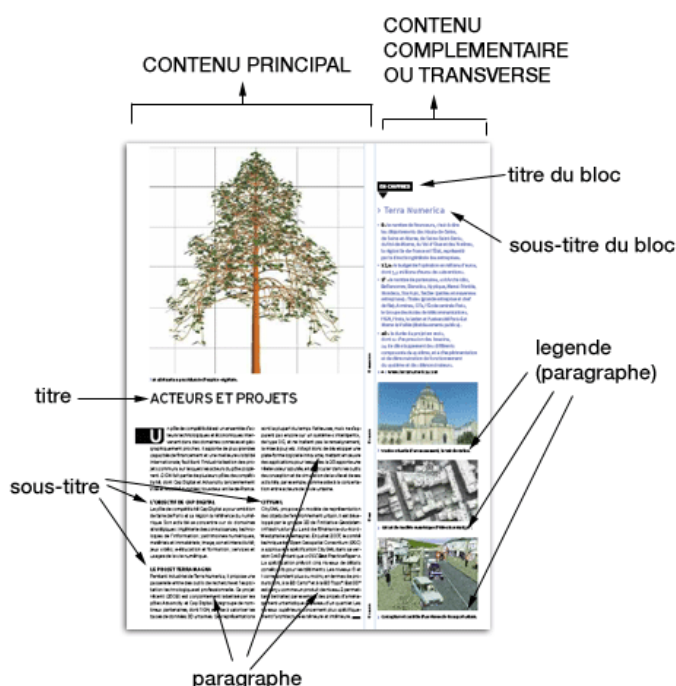
Le (X)HTML est ce qu'on appelle un " langage de balises ", c'est à dire qu'il nous permet d'attacher à chaque portion de contenu, une étiquette qui indique quel est son rôle en relation avec le contenu qui l'entoure.

Il existe une " étiquette " spécifique pour la plupart des rôles qu'une portion de contenu peut avoir dans un document : paragraphes, titres et sous-titres, listes, listes de définitions, éléments de formulaires, etc.

Un bon code (X)HTML est donc celui qui qualifie correctement le contenu selon son rôle, assurant ainsi une structure hiérarchique cohérente.

Prenons n'importe quel document dans n'importe quel support, et examinons-le attentivement. Pouvons-nous identifier sur ce document ce qui est un titre, des sous-titres, des paragraphes, etc. ?

Si on examine la page de l'exemple précédent, on peut rapidement identifier : le titre de la page, les sous-titres, les différents paragraphes, ainsi qu'une colonne de contenu transverse ou complémentaire au contenu principal situé dans la colonne plus étroite à droite.



Si on traduit cette page en utilisant les balises qui prennent en compte la nature de chaque type de contenu, son code serait :

```
<h1>Acteurs et projets</h1>
<p>U.....</p>
<h2>....</h2>
<p>....</p>
<h2>....</h2>
<p>....</p>
<h2>....</h2>
<p>....</p>

<h2>....</h2>
<h3>Terra numerica</h3>
<p>....</p>
<p>....</p>
<p>....</p>
<p><img ... /></p>
<p>...legende...</p> etc.
```

Pas une seule balise `<div>` n'a encore été ajoutée à notre code, mais si on visualise cette page sans la CSS, on a déjà une plus grande facilité de lecture, car on peut maintenant identifier les différents types de contenu :

Acteurs et projets
Lorem ipsum dolor sit amet consectetur at wisi ante fermentum Cras.
Sous-titre
Lorem ipsum dolor sit amet consectetur at wisi ante fermentum Cras.
Sous-titre
Lorem ipsum dolor sit amet consectetur at wisi ante fermentum Cras.
Sous-titre
Lorem ipsum dolor sit amet consectetur at wisi ante fermentum Cras.
Titre du bloc
Terra numerica
Lorem ipsum dolor sit amet consectetur at wisi ante fermentum Cras.
Lorem ipsum dolor sit amet consectetur at wisi ante fermentum Cras.
Lorem ipsum dolor sit amet consectetur at wisi ante fermentum Cras.

De cette façon, on a un code propre, simple et hiérarchisé.
On peut facilement atteindre le contenu, par son rôle, pour lui donner un style ou pour le manipuler par javascript, par exemple.

Ce code peut également être utilisé pour être transformé dans d'autres formats (comme en audio, par les synthétiseurs vocaux, par exemple) sans perte de cohérence.

La façon dont on écrit le code est donc **le point le plus sensible** de toute la philosophie de la séparation contenu/forme.

Pensez à votre code (X)HTML comme aux fondations de votre maison. Vous pouvez changer une fois par an le papier peint, vous pouvez peindre vos fenêtres tous les trois ans de la couleur que vous voulez, vous pouvez changer les meubles tous les mois si vous le pouvez, mais vous ne pouvez modifier les fondations que difficilement et avec des gros efforts et pas mal de dépenses.

Nous ne pouvons pas "bricoler" avec le code. Si votre code n'a pas été pensé pour être "solide", évolutif, sans faute, votre projet ne sera pas un projet réussi.

L'ensemble des recommandations que vous trouverez dans le présent document ont pour objectif de vous donner toutes les "armes" pour réussir "vos fondations".

2.2 Le choix du langage

On a tous compris : le bon code est l'âme d'un site. Mais peut-on dire si le bon code est en langage HTML ou XHTML ? La question est plus précisément: est-ce que le choix entre un langage ou un' autre peut avoir une influence sur la qualité de notre site ?

Le choix du Studio ira toujours dans le sens de la qualité.

Donc bien plus que le choix du langage de *mark-up*, l'important sera la prise en compte des caractéristiques qui font d'un produit un produit réussi.

La polémique autour du XHTML, bien que très importante et passionnante, n'a pas lieu d'être discutée ici. Néanmoins elle existe et chaque aspect de ce langage, qu'il soit positif ou négatif, a été pris en compte et réfléchi avant que le Studio le choisisse comme langage de prédilection. Bien sûr, tous les deux, HTML et XHTML, sont tous les deux des langages standards et seront amenés à évoluer avec le web.

La préoccupation du Studio se porte sur la qualité de nos livrables : indépendamment du langage choisi, notre but est de pouvoir livrer un code qui présente les caractéristiques listés précédemment (voir le chapitre 1, *Introduction*).

Les critiques émises sur le XHTML portent souvent sur les mauvaises habitudes et les mauvaises pratiques (qui évidemment peuvent être constatées aussi pour les sites en HTML4).

Aujourd'hui, nous ne pouvons pas bénéficier des réels avantages du langage XHTML – rappelez-vous, Internet Explorer ne le supporte pas – et l'exploiter comme une vraie application XML. Mais si la qualité de nos pages est assurée, nos sites seront prêts pour le futur.

Le XHTML ne doit pas être choisi d'office. Même s'il est notre choix de prédilection, le "pour" et le "contre" de son utilisation doit être pesé à chaque nouveau projet et discuté avec les équipes de développement lors du cadrage.

3 Les règles

Dans ce chapitre sont listés les points qui doivent obligatoirement être pris en compte lors de l'édition (X)HTML.

Ignorer ou mal définir un ou la totalité de ces points rend le code invalide.

3.1 La déclaration du DOCTYPE

Les racines du HTML viennent du SGML qui a toujours été un langage de spécification de balisage structurel. C'est donc du SGML qu'HTML 4 a hérité le concept du DTD ou « *Document Type Definition* » : **un document qui définit les règles que le navigateur va utiliser pour interpréter les balises HTML.**

La déclaration `<!DOCTYPE>`, **qui doit toujours apparaître en début du document** HTML ou XHTML, informe les services de validation (HTML et CSS) et les navigateurs plus récents de la DTD (*Document Type Definition*) à utiliser pour façonner le « *markup* ». Le choix du DOCTYPE détermine donc la façon dont on va écrire nos pages et dont les navigateurs vont les afficher.



ALERTE IE6 !

L'absence d'un DTD, un DTD invalide ou un DTD qui n'apparaît pas sur la toute première ligne du document déclenche le mode "quirks", qui est particulièrement nocif sous IE6

W3C a écrit un DTD pour chacun des langages de *markup* standard et ses versions. Ces déclarations DOCTYPE sont ¹ :

Pour HTML :

DTD STRICT, pour le mode standard :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">
```

¹ Il existe également un DOCTYPE pour les *framesets*. Mais comme nous déconseillons fortement l'utilisation des frames, il nous paraît normal de ne pas insister sur ce type de DOCTYPE

DTD LOOSE, pour le mode hybride (presque-standard) :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">
```

Pour XHTML :

XHTML 1.0 Strict, Transitional, Frameset

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

REFERENCES:

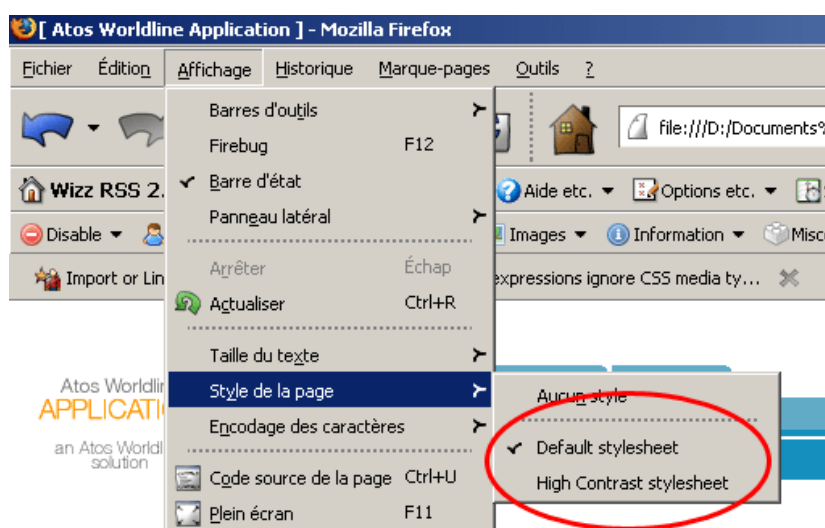
- **Pour plus de détails à propos des DOCTYPES :**
<http://www.w3.org/QA/Tips/Doctype>

3.2 L'appel aux feuilles de style

Il existe deux façons d'appeler une feuille de styles dans votre page (X)HTML : par la déclaration CSS `@import` ou par la balise `<link>`.

Les principales caractéristiques de ces deux méthodes sont :

- la balise `<link>` est une balise (X)HTML, alors que `@import` est une déclaration CSS. Ainsi, `<link>` doit apparaître toujours dans la balise `<head>` du document, alors que la déclaration `@import` peut apparaître dans le document (X)HTML et/ou dans une feuille de styles externe.
- sans la balise `<link>`, IE peut présenter un bug qui force l'affichage de la page sans les styles (le "FOUC", ou *Flash Of Unstyled Content*), et avant le chargement de la CSS, ce qui crée un effet de "clignotement" du contenu très désagréable pour l'utilisateur. Cependant, ce bug peut être corrigé par la présence d'une balise `<script>` avant l'appel à la CSS.
- IE6 et 7 présentent des problèmes de spécification des types de médias avec la déclaration `@import`. Donc cette déclaration ne peut être utilisée que pour la valeur de média par défaut "all". Pour appeler les CSS pour d'autres médias, il faut les appeler avec la balise `<link>`.
- la balise `<link>` donne la possibilité d'alterner plusieurs CSS. Elle permet de définir une CSS par défaut et une ou plusieurs CSS secondaires qui peuvent être alternées par javascript ou par des fonctionnalités de base de certains navigateurs modernes, comme Firefox.



```

. . .
<link href="cust.css" rel="stylesheet" title="Default stylesheet" type="text/css"/>
<link href="highc.css" rel="alternate stylesheet" title="High Contrast stylesheet"
type="text/css" />
. . .

```

- Comme les versions des navigateurs antérieurs aux versions 5 ne reconnaissent pas l'instruction `@import`, ces navigateurs vont juste l'ignorer et les pages seront affichées sans le rendu graphique ni la mise en page. Si le code de votre page est structuré sémantiquement et s'il est bien construit, l'affichage sans la feuille de styles ne posera pas de problème, car le contenu affiché sera sans le design, mais il sera tout à fait lisible et cohérent. C'est ce qu'on appelle la "dégradation élégante".

Jusqu'ici, la déclaration `@import` était le choix de prédilection à cause des vieux navigateurs. et la dégradation élégante était le bon compromis à faire pour la compatibilité entre navigateurs.

Aujourd'hui nous pouvons constater que ces navigateurs ont complètement disparu et le choix pour `@import` peut donc être ré-considéré.

Chacune de ces deux méthodes présentent des avantages et des inconvénients, mais elles sont toutes les deux valables et acceptées et elles présentent techniquement le même niveau de performance. Cependant, en considérant la disparition des navigateurs de 4^{ème} génération et le besoin croissant d'avoir des CSS multiples pour des différents médias, notre recommandation va aujourd'hui plutôt pour l'utilisation de la balise `<link>`, plus flexible et plus adaptée au code (X)HTML.

3.3 Déclaration du "*namespace*"

Si vous codez en XHTML, pensez toujours à ajouter l'attribut `xmlns` à la balise `<html>` pour définir le *namespace* ou "espace de nommage" de la page.

Le *namespace* est une méthode qui permet de qualifier les noms utilisés pour les éléments et les attributs en documents XML.

Déclarer le *namespace* sur nos pages web aide à éviter un conflit entre les noms des éléments de base et les noms d'éléments spécifiques définis pour une application XML.

Dans la plupart des cas, la valeur de l'attribut `namespace` pointe vers le serveur de W3C :

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
```

Utilisez toujours cette valeur par défaut. Dans le cas où une application XML spécifique nécessite un *namespace* propre, cette valeur sera modifiée lors de la dynamisation.

HTML x XHTML:

La déclaration du *namespace* est nécessaire uniquement pour le langage **XHTML**.

REFERENCES:

- **Normative Definition of XHTML 1.0 :**
<http://www.w3.org/TR/xhtml1/normative.html>

3.4 Déclaration de la langue principale

L'identification de la langue principale d'un site ou d'une application web est primordiale pour pouvoir "dialoguer" avec l'utilisateur de façon appropriée.

L'accessibilité, les outils de traduction et d'édition, la sélection des familles de polices, les scripts, ainsi que les fonctionnalités de recherche sont largement facilités par l'identification de la langue principale.

La langue principale de la page doit être déclarée dans la balise `<html>` de chaque page d'un site par l'attribut `"lang"`. Un code de deux lettres est réservé pour chaque langue (ISO639) afin de définir la valeur de cet attribut. Dans le cas de la langue française, par exemple la valeur est `"fr"` :

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
```

Pensez également à indiquer dans vos pages web, tout changement de langue. Par exemple, si dans votre texte vous utilisez un terme dans une autre langue que la langue principale du site, indiquez ce changement avec l'attribut `"lang"` :

```
<p>Si vous codez en XHTML, pensez toujours à indiquer le  
<em lang="en">namespace</em>.</p>
```

Dans l'exemple ci-dessus l'attribut `"lang"` a été appliqué à la balise ``, mais il peut être attribué à n'importe quelle autre balise.

HTML x XHTML:

En **XHTML**, la déclaration de la langue doit se faire également par l'attribut `"xml:lang"`. Si le fichier XHTML est manipulé comme un fichier HTML (servi comme `"text/html"`), cet attribut sera utilisé à chaque fois que le fichier sera traité comme un document XML, par un script ou pour la validation du code par exemple.

REFERENCES:

- Une très bonne référence, à lire :
Declaring language in XHTML and HTML (Draft)
<http://www.w3.org/International/tutorials/language-decl/en/all>
- **Codes for the Representation of Names of Languages :**
http://www.loc.gov/standards/iso639-2/php/code_list.php

3.5 Définition du "*content-type*"

La valeur de "*content-type*" d'une balise `<meta>` informe à l'agent utilisateur (un navigateur, par exemple) le type de contenu que lui sera "livré".

Deux informations importantes sont données par le "*content-type*" : le format des données (MIME), et le jeu de caractères utilisé par ces données.

Le format des données

Le format des données va dépendre du langage choisi pour votre page.

Si vous écrivez en HTML, tout est simple : vous devez définir le format de données par "`text/html`".

Si vous écrivez en XHTML là c'est plus délicat. Comme le XHTML est une application XML, la recommandation de W3C est de définir le type de contenu par "`application/xhtml+xml`". Or Internet Explorer, même à la version 7, ne supporte pas ce format de données, ce qui crée un obstacle à la recommandation.

En XHTML, nous sommes donc "obligés" de définir le format de données par "`text/html`", ce qui va obliger l'agent utilisateur quel qu'il soit, à interpréter nos pages XHTML comme étant du HTML.

Le format des données doit donc toujours être "`text/html`".

Les avantages d'écrire en XHTML pour les designers ne sont pourtant pas perdues dans ce cas **si nous suivons les recommandations de compatibilité** détaillées dans le point 3.7.

Note : Un point important à considérer est le "*content negotiation*", ou "négociation du contenu".

La négociation du contenu, définie au niveau du serveur web, permet de lui laisser le choix du format de données, en fonction de l'agent utilisateur.

Cette méthode n'est pas nouvelle mais est assez délicate et dépend de la configuration du serveur.

Le jeu de caractères

Le "*charset*" ou "jeu de caractères", est une collection de lettres et des symboles utilisés par un système d'édition. Par exemple, le jeu de caractères ASCII couvre l'ensemble des lettres et des symboles de la langue anglaise.

Le choix du jeu de caractères va définir les caractères spéciaux qui seront supportés dans vos pages.

Pour la langue française nous utilisons par défaut le "*charset*" `iso-8859-1`, qui couvre l'ensemble des lettres et caractères spéciaux nécessaires pour les langues latines. Néanmoins, pour plusieurs raisons qui sortent du champ de ce document, nous recommandons l'utilisation du jeu de caractères Unicode "`utf-8`", et cela indépendamment que vous codiez

en HTML ou en XHTML.

HTML x XHTML:

En **XHTML** utilisez toujours le jeu de caractères "utf-8", car le langage XML supporte uniquement Unicode.

REFERENCES:

- **Character sets & encodings in XHTML, HTML and CSS :**
<http://www.w3.org/International/tutorials/tutorial-char-enc/>

3.6 Titre du document

Le titre du document doit être correctement renseigné dans la balise `<title>`, dans le `<head>` du document.

La valeur de `<title>` doit toujours contenir le nom du site et le chemin logique vers le document. Par exemple, si nous nous trouvons sur la page "*Rando*", de la rubrique "*Loisirs*" du site "*Les Régions de France*", la valeur de `<title>` doit être :

```
<head>
. . .
<title>[ Les Régions de France : Loisirs : Rando ]</title>
. . .
</head>
```

C'est toujours une bonne idée de présenter le chemin du document entre crochets "[]", pour permettre une meilleure séparation entre la valeur du titre et le nom du navigateur :



3.7 Les règles de compatibilité XHTML

Note : Ce point est important uniquement si vous codez en XHTML.

Contexte

Internet Explorer ne permet pas de livrer nos pages XHTML en tant qu'application XML. Nous devons donc laisser les navigateurs traiter nos pages comme si elles étaient écrites en HTML en déterminant le *"content-type"* comme `text/html`.

Si nous avons choisi le XHTML pour l'écriture de notre code nous devons suivre un certain nombre de règles qui vont permettre d'assurer la rétrocompatibilité de nos pages.

Ces règles sont très importantes et demandent une attention spéciale de la part de l'auteur du code.

Ce qu'il faut faire

1. Préférez toujours le jeu de caractères Unicode (utf-8) qui est le seul système d'encodage accepté pour le XML
2. Pensez toujours à ajouter un espace avant les caractères de fermeture d'une **balise simple** (`</>`). Par exemple, `
`, `<hr />`, ``, etc. Une balise simple est celle qui ne présente pas de contenu ajouté (comme `
`) ou celle qui a un contenu intrinsèque (comme `<input>` et ``).
3. Ajouter toujours la balise de fermeture aux balises "pleines" (celles qui présentent du contenu ajouté). Par exemple, écrivez `<p></p>`, et non `<p />`, ou `<div></div>` et non `<div />`.
4. Les déclarations de styles et fonctions javascript doivent toujours être définies dans des fichiers externes.
5. Evitez d'ajouter des **espaces multiples** ou retour à la ligne pour les valeurs des attributs. Par exemple, ne faites pas `<p class="introduction article">...</p>`.
6. N'utilisez plus l'élément `"isindex"` il est déprécié en faveur de la balise `<input>`.
7. Lors de la définition d'une langue, utilisez toujours les attributs `"lang"` et `"xml:lang"` ensemble.
8. N'utilisez jamais le caractère `"&"` pour le passage de variables par une URL. Ce caractère doit être remplacé par son entité de référence `"&"`. La façon correcte serait donc :

```
<a ref="http://www.test.fr?nom=DUPONT&amp;prenom=MARTIN">XXXXX</a>
```

NOTE: Ce point est également vrai si vous codez en HTML.

9. Pour l'édition CSS (plus de détails sur "*Méthodologie pour le design CSS*"):
 - a. utilisez uniquement des noms des balises et attributs en minuscule;
 - b. l'élément `<tbody>` doit apparaître explicitement dans la page, si référencé par la CSS.

3.8 Respecter la syntaxe du langage et ses éléments

Contexte

Aucune des recommandations ou bonnes pratiques ne servirait à quelque chose si la syntaxe de notre langage n'est pas maîtrisée.

La syntaxe du langage ainsi que la valeur sémantique de ses éléments doivent être prises en compte lors de l'écriture de notre code pour qu'il soit cohérent et surtout valide.

Un langage de "*mark-up*" est formé d'un ensemble de balises. Chaque balise traduit un élément sémantique qui va, à son tour, définir la nature de cet élément.

Par exemple, la balise `<p>` traduit l'élément "paragraphe", qui est un élément destiné à contenir uniquement un ensemble de mots et de lettres (les éléments de type "*inline*", comme les chaînes de caractères).

La balise `<div>` quant à elle, est comme une boîte, un container (un élément du type "*block*"). Elle n'a pas de valeur sémantique en soit mais sa fonction est de grouper ou de séparer des éléments.

Après cela, comment envisager d'insérer une balise `<div>` dans une balise `<p>` ? Comment la structure visuelle ci-dessous peut-elle vous paraître "correcte" ?

Format :	11 cm x 8 cm pliée
Echelle :	1:25 000
Nombre de titres de la collection :	1 779
Délais de livraison :	sous 15 jours les

Ces cartes topographiques d'une très grande précision contiennent tous les détails existant sur le terrain : voies de communication jusqu'au moindre sentier, constructions jusqu'au hangar, bois, arbre isolé, rivière, source... Sans oublier la représentation du relief par des courbes de niveau.

Au moment d'écrire votre code vous devez être attentif à toujours garder cette cohérence de relation entre les éléments père/fils issue certes d'une logique sémantique, mais avant tout, qui dépend du DOCTYPE choisi.

Un DOCTYPE transitionnel sera beaucoup moins rigide quant à la cohérence de relation entre les éléments, alors qu'un DOCTYPE strict s'approche le plus possible d'une logique sémantique.

Ce qu'il faut faire

- **Connaître la syntaxe**

Prenez en compte les spécificités de chaque langage, et respectez strictement sa syntaxe (fermeture de balises, attribution de valeurs d'attributs, sensibilité de la casse, etc).

Pour connaître les spécificités de syntaxe de HTML4.1 ou du XHTML1.0, utilisez les liens de référence.

- **Utilisez la logique et les références du DOCTYPE choisi**

Normalement, il suffit d'utiliser la logique du Studio pour faire ces choix et pour éviter de créer de mauvaises relations entre les éléments (est-ce que je mettrais un tableau dans un élément d'une liste ?...).

L'astuce consiste à ne pas insérer un élément *block* dans un élément *inline*.

(Consulter le document "*Guide des projets Web: Méthodologie de travail pour l'édition CSS*", pour plus de détails sur les types d'éléments)

- **Validez votre code**

La validation du code va indiquer toute erreur, anomalie ou imbrication non-valide des balises (voir point 3.11).

♦ **REFERENCES:**

- Le **W3Schools** est un très bon site de référence pour les langages HTML, XHTML, CSS ou encore Javascript (<http://www.w3schools.com/>)
- Le **DTD Mapper** mappe un DOCTYPE donné et vous donne une liste des tous ses éléments valides, avec leurs attributs et ses possibles éléments père/fils. C'est une excellente référence pour savoir quel élément peut aller où... (<http://www.themaninblue.com/experiment/DTDMapper/>)
- **Outil XHTML 1.1 – Hiérarchie** : qui décrit les différentes balises en précisant pour chacune la liste autorisée des éléments parents et enfants (<http://giminik.developpez.com/xhtml/>)

3.9 Utiliser les tabulations et les retours à la ligne pour un code clair

Contexte

Le code original créé par Le Studio, est la source première d'un projet Web. C'est sur nos gabarits que tout le travail de design et de développement va reposer. En d'autres mots, nos gabarits sont la matrice de nos sites ou applications web et doivent donc présenter une rigueur d'écriture afin qu'ils soient facilement compris et dynamisés.

Un code qui n'a pas une cohérence de structure visuelle créée par des retours à la ligne et des tabulations est difficile à lire et ses éléments sont très difficiles à identifier :



Dans l'exemple ci-dessous, on peut voir la différence entre un code bien structuré visuellement et un qui ne l'est pas. Dans l'exemple de droite on arrive à identifier les différents blocs de contenu ainsi que la hiérarchie des éléments, créée par la tabulation.

Ce qu'il faut faire

- Gardez une balise par ligne.
- Dans le cas d'un contenu trop long, ajouter des retours pour chaque balise interne:

```
<p id="pBdCrumbs">
  <a href="index.html">Accueil</a> &gt;
  <a href="liste_des_demandes.html">Liste des demandes</a> &gt;
  Modification d'une demande</p>
```

- Pensez à séparer les principaux blocs d'information (*header*, *footer*, colonnes, formulaires, etc) par une ligne blanche.
- Utilisez la tabulation pour indiquer le niveau de profondeur de chaque balise.
- Ne séparez pas la balise de fermeture d'un lien (``) de son contenu :

Correcte :

```
<p><a href="index.html">Accueil</a></p>
```

Pas correcte :

```
<p>  
  <a href="index.html">Accueil  
  </a></p>
```

3.10 Ne pas "forcer" l'ajout d'espaces ou les retours à la ligne

Contexte

Nous avons l'habitude d'ajouter les fameux espaces ` ` ou encore d'ajouter des paragraphes vides (`<p>
</p>`) pour augmenter la marge entre deux blocs de texte.

Bien sûr, avant on ne se souciait pas de la séparation contenu/forme et la méthode la plus rapide était toujours la meilleure.

Aujourd'hui par contre, nous savons que nous pouvons contrôler les espaces (marges, retours à la ligne, *padding*s) par la CSS et que ça nous assure non seulement un code propre mais aussi une homogénéité d'affichage des éléments dans toutes les pages.

Ce qu'il faut faire

Il suffit bien sûr de ne pas forcer l'espacement par des balises vides.

Les retours à la ligne (`
`) par contre, peuvent être insérés directement sur les pages mais uniquement dans des cas où cela se justifie d'un point de vue structurel, c'est-à-dire, si sans la feuille de styles, le retour à la ligne se justifie.

3.11 Valider votre code

Contexte

Nos pages ne seront correctes que si nous validons notre code (X)HTML.

La validation du code permet d'identifier des erreurs simples de syntaxe, ainsi que des choix de structuration qui ne respectent pas le DOCTYPE choisi (comme l'ajout d'une balise `<div>` dans une balise `` par exemple).

Une page ne peut pas être considérée comme "finalisée" tant qu'elle n'a pas été validée.

Ce qu'il faut faire

Validez chaque page avec le validateur de W3C avant de passer à une nouvelle page. Cela vous évitera de répercuter des erreurs sur d'autres pages.

Pendant le codage vous pouvez utiliser le plug-in **HTML Validator**, sur Firefox, pour contrôler la qualité du code.

REFERENCES:

- **La validateur W3C :**
<http://validator.w3.org/>
- **HTML Validator, un *plug-in* pour Firefox :**
<http://users.skynet.be/mgueury/mozilla/>
(cette extension de Firefox permet de valider un code au fur et à mesure qu'on crée une page, mais il ne doit pas remplacer le validateur de W3C)

4 Méthodes et conventions

4.1 L'outil d'édition

Recommandé	Equivalents	Avantages
Editeur ASCII avec des fonctionnalités d'aide à l'édition (coloration syntaxique, tabulation, etc)	PSPAD, ULTRAEDIT, APTANA, Context	Un éditeur ASCII convient parfaitement. Il est tout ce qu'on a besoin pour bien écrire notre code (X)HTML, et il est le seul moyen d'assurer un contrôle sur la structure de contenu et d'éviter d'être prolix

Note à propos des éditeurs

Parce que le langage **XHTML** est une application **XML**, et que XML ne supporte que le jeu de caractères Unicode (UTF), UTF est donc le jeu de caractères ASCII privilégié pour le XHTML. Attention, certains éditeurs n'enregistrent pas automatiquement les documents avec ce " charset ". Avant de choisir un éditeur qui vous convient, soyez sûr qu'il peut facilement enregistrer vos fichiers avec le jeu de caractères Unicode.

Note à propos de DreamWeaver

DreamWeaver a été pendant longtemps un outil inestimable pour " l'écriture " d'une page web. Sa fonctionnalité WYSIWYG (*What You See Is What You Get* : ce que vous voyez c'est ce que vous aurez) permettait de créer des pages web assez complexes sans avoir besoin de voir ou même de connaître la syntaxe HTML. Aujourd'hui, si nous voulons créer des pages web évolutives, flexibles et portables, cette " insouciance " n'est plus possible.

La philosophie de la séparation du contenu de la forme demande un exercice intellectuel beaucoup plus ardu et intéressant que le simple " assemblage " WYSIWYG d'éléments de textes ou graphiques. Nous ne pouvons plus nous fier à ce que DW nous montre (et souvent ce qu'il nous montre est très loin de la réalité) ni à son code les yeux fermés ni à la manière dont il va enchaîner les balises ou structurer notre page.

Cela ne veut pas dire qu'on ne doit plus utiliser ce puissant logiciel. Cela signifie seulement qu'aujourd'hui sa fonction WYSIWYG ne nous sert plus à rien.

Si l'interface et l'environnement de DreamWeaver vous plaisent, vous pouvez bien sûr continuer à l'utiliser, mais assurez vous que les recommandations d'édition soient respectées.

- **Si vous utilisez DreamWeaver** : nous recommandons de rester uniquement dans le mode " code " et de ne pas manipuler des éléments via l'interface WYSIWYG, pour éviter que DreamWeaver n'ajoute des balises ou des éléments non souhaités.

Important : pour la vérification du rendu graphique, seuls les navigateurs listés dans le document "*Méthodologie de travail pour le design CSS*" peuvent être utilisés comme source fiable.

Remarque : Un jeu de "*templates*" est disponible dans le répertoire "*templates*", fourni avec ce document. Ils peuvent être un bon point de départ pour l'édition des pages car ils contiennent déjà une structure de pages ainsi que toutes les informations de document recommandées.

4.2 Appeler toujours la feuille de styles de développement

Contexte

La maintenance et l'évolution d'un site dépend souvent de plusieurs personnes, par fois même réparties dans plusieurs équipes différentes.

Même si nos pages et feuilles de styles aient été très bien commentées et réfléchies, les modifications ou adaptations du design CSS sont pratiquement inévitables. Il va toujours exister des cas de figures qui n'ont pas été pris en compte lors du design CSS, ou encore des demandes de modification de dernière minute.

Ces modifications ou ajouts doivent être faits, par fois, par l'équipe de développement, par des personnes avec différents niveau de connaissance du design CSS.

Si des modifications sont faites directement dans la feuille de styles principale par quelqu'un qui ne maîtrise pas assez le design CSS, elles peuvent avoir d'impacts importants sur l'affichage des pages.

Ce qu'il faut faire

Appelez toujours une CSS qui sera exclusive pour l'ajout de nouveaux styles ou pour la modification de styles existants par l'équipe de développement. De cette façon, l'intégrité des livrables originaux sera assurée, et le diagnostic des problèmes sera plus rapide.

Cette feuille de styles doit être la dernière à être appelée, pour que ces définitions aient priorité sur celles définies dans la feuille de styles principale :

```
. . .
<title>[ Application Centrale de Traitement ]</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">@import url(styles/main.css);</style>
<style type="text/css">@import url(styles/dev.css);</style>
. . .
```


4.3 Le code (X)HTML d'abord, le design CSS après (mais l'un n'exclut pas l'autre)

Contexte

Pour que le designer ou l'intégrateur puisse contrôler l'affichage des éléments dans la page, il doit donner au contenu une structure logique et écrire un code propre. En s'affranchissant de la présentation graphique, la réflexion pour le codage devient plus précise et objective.

Ce qu'il faut faire

Essayez le plus possible, **d'écrire vos "codes composants" avant de passer au design CSS**. Cela est primordial si vous n'avez pas beaucoup d'expérience avec le (X)HTML et le design CSS. Vous serez ainsi en mesure de vous concentrer uniquement sur la structure du contenu, en évitant les redondances.

Dans le cas contraire, veillez à **toujours tester la page sans la CSS**. Ainsi vous pourrez être sûrs que le contenu sera bien structuré et qu'il gardera une cohérence et une lisibilité optimales.

Ecrire le code avant de s'occuper du design CSS ne veut pas dire que nous devons l'oublier complètement. L'ajout des classes et d'Ids font partie de la "zone d'intersection" entre le code et le design (voir point n° 5.2) : les balises donnent la nature du contenu, les classes et les Ids les classifient ou leur donnent une identité. Le classement et l'identification d'éléments ne fonctionnent pas uniquement pour le design, mais également pour la manipulation du contenu (par javascript, par exemple).

Au fur et à mesure que vous gagnez en expérience, vous vous sentirez plus à l'aide pour travailler simultanément le code et le design CSS.

4.4 Respecter les règles de nommage de styles

Contexte

Comme expliqué dans le point précédent, au moment où l'on écrit notre page (X)HTML, il faut déjà anticiper le classement et l'identification des éléments en ajoutant des classes et Ids. C'est donc principalement à ce moment que l'on doit réfléchir aux noms de styles.

De nombreuses personnes de spécialités différentes sont à même de travailler avec un code et/ou une feuille de styles faits par une tierce personne. Ainsi, même si le code est proprement commenté, la façon dont on nomme les styles peut prêter à confusion, créer des difficultés pour les mises à jour ou pire encore, créer des bugs d'affichage.

Ce qu'il faut faire

Lors de l'édition (X) HTML, suivez ces simples règles :

- Choisissez toujours des **noms sémantiques** et non dépendants de la forme (éviter des noms comme "*paragraphrouge*")
- N'utilisez **jamais de caractères spéciaux** (comme _ ; : \$ * , ? – etc.) ou des **espaces** pour nommer les styles
- Ne **commencez jamais** le nom d'une classe ou Id **par un chiffre ou par un caractère spécial**
- Considérez, là où c'est possible, l'utilisation des classes déterminées par les *microformats* existants
- Préférez, sauf précisions contraires pour des projets spécifiques, les noms de styles et les commentaires **en anglais** (après tout on est dans une multinationale, et nos projets le sont souvent aussi)
- Pour les blocs de contenu les plus courants, préférez les noms définis ci-dessous (basés sur les nouvelles balises HTML5) :

Bloc	Nom de style
Container principal (toujours un <div>)	container
Container des colonnes principales (toujours un <div>)	wrapper
Haut de page (toujours un <div>)	header

Corps de la page - colonne principale (toujours un <div>)	main
Navigation	nav
Colonne de contenu transverse (toujours un <div>)	aside
Pied de page (toujours un <div>)	footer
Menu d'outils (contact, plan du site, etc.)	tools
Fil d'Ariane (toujours un <p>)	breadcrumbs
Logo/Nom du site	Préférez toujours la balise <i>h1</i> , sans spécifier de classe ou d'ID

REFERENCES:

- **A propos des *Microformats* :**
<http://microformats.org/about>
- **A propos des noms de classes/balises sémantiques :**
<http://tantek.com/presentations/2004etech/realworldsemanticspres.html> et
<http://microformats.org/wiki/semantic-class-names>

4.5 Conventions des commentaires

Contexte

L'ajout de commentaires sur le fichier (X)HTML permet de clarifier la lecture et de faciliter la compréhension du code.

Ce qu'il faut faire

IMPORTANT : N'oubliez pas que les spécificités de votre code ainsi que de votre CSS, doivent être dûment documentés (voir le chapitre "*Documentation*", du document "*Méthodologie de cadrage des projets Web*"). Les commentaires contextuels dans le code serviront donc uniquement à se repérer, pas à expliquer le code.

Pour aider dans la lecture du code :

- Ajoutez toujours un commentaire à l'ouverture d'un **bloc composant** et un commentaire de fermeture :

```
<!-- HEADER / -->
<div id="header">
    <h1>Nom du site</h1>
    . . .
</div>
<!-- /HEADER -->
```

Ajoutez un "/" à la fin de l'explication du bloc pour indiquer son ouverture et un "/" avant l'explication du bloc pour indiquer sa fermeture.

- Pensez à ajouter un commentaire simple à coté de la fermeture des principaux containers pour indiquer à quel bloc cette fermeture fait référence :

```
<div id="container">
    <div id="header">. . .</div>
    . . .
    <div id="wrapper">
        . . .
    </div><!-- /wrapper -->
    . . .
</div><!-- /container -->
```

En effet, comme la fermeture des principaux containers d'un fichier se situe souvent en fin de fichier et très loin de leur ouverture, on se retrouve souvent face à une séquence de fermetures de balises `<div>` que l'on ne peut pas associer facilement à un container spécifique.

Indiquer à quel container la fermeture du `<div>` appartient aide à la compréhension et le découpage du code.



ALERTE IE6 !

Les commentaires HTML peuvent créer le célèbre "bug" des "caractères fantômes" (la duplication des derniers caractères d'un fichier (X)HTML en fin de page).

L'apparition de ces caractères fantômes est très aléatoire mais très fréquente. La seule façon d'éviter cela c'est de trouver et de supprimer le commentaire qui crée le "bug" (souvent on finit par tous les enlever).

Nous vous recommandons de faire un point avec l'équipe de développement pour vérifier avec elle si elle peut supprimer les commentaires une fois la page interprétée ou, si elle veut garder les commentaires, d'utiliser la syntaxe de commentaires conditionnels de Microsoft, qui fort heureusement ne cause pas ce "bug".

Voir document "*Méthodologie de cadrage de projets web*".

4.6 Ecrire un code flexible et évolutif

Contexte

C'est encore un point où le codage (X)HTML et le design CSS se croisent.

Nous avons souvent tendance à tester notre design CSS avec un exemple de contenu figé, tel qu'il a été codé lors de l'édition (X)HTML. Or lors de la qualification du site, on peut découvrir que ce design est "cassé" avec un nouveau contenu de taille différente. Ou alors on découvre que l'on n'a pas pris en compte certains états ou contextes qui peuvent influencer sur le contenu, comme par exemple :

- l'indication de la rubrique courante;
- les différents espacements entre un titre et le contenu qui suit;
- les déséquilibres d'alignement avec des passages à la ligne inattendus...

Ce qu'il faut faire

Il faut surtout nous assurer que l'exemple de contenu que nous utilisons pour notre édition (X)HTML soit aussi exhaustif que possible.

L'exhaustivité du contenu va dépendre bien sûr de son contexte :

- Dans le cas **d'un menu de rubriques** par exemple : pensez à ajouter tous les états et cas possibles comme : des intitulés très longs ou très courts; une rubrique courante cliquable et identifiée par un Id; une rubrique courante non-cliquable et identifiée par un Id; un nombre de niveaux important, etc.
- Dans le cas d'un **contenu éditorial** par exemple : ajoutez les 6 niveaux de titres (`<h1>`); différents exemples de paragraphes et de liens; diversifiez au maximum la séquence d'éléments (sous-titre suivi par un titre ou par un paragraphe, etc); ajoutez des images de tailles différentes ou enfin ajoutez des listes et des listes de définition (`<dl>`), etc.

Même si ce travail peut sembler fastidieux il offre une garantie contre toute mauvaise surprise.

4.7 Eviter de commencer l'édition (X)HTML par la page d'accueil

Contexte

La page d'accueil est souvent atypique et peut présenter des différences structurelles importantes. Si vous commencez l'écriture (X)HTML par la page d'accueil, vous risquez d'effectuer un travail redondant **et de devoir revenir souvent sur des décisions prises, principalement en ce qui concerne la structure des pages.**

Ce qu'il faut faire

Côté codage (X)HTML, un des points les plus importants lorsqu'on commence la production d'un site Web, c'est l'identification et la définition des blocs composants (point n°4.8). Commencer l'édition par la page d'accueil est donc déconseillé, justement parce que la page d'accueil peut présenter des blocs différents des autres pages.

Nous vous recommandons d'avoir une très bonne visibilité des différents gabarits et éléments des pages internes et de commencer par elles.

4.8 Penser en termes de "composants"

Contexte

Pour que notre CSS soit la plus optimisée et la plus efficace possible, nous devons nous efforcer de penser en termes de "composants". Par "composants" nous entendons les blocs de contenu caractéristiques et identifiables tels qu'un bloc de mise en avant d'un produit, une liste de brèves, etc., qui apparaissent régulièrement sur les pages du site (hors éléments de navigation, *header* et *footer*).

Si nous avons la possibilité d'avoir une vision globale de tous les blocs composants d'un site (soit au travers des maquettes de design, soit au travers d'un *storyboard*), l'écriture (X)HTML ainsi que le design CSS, seront largement facilités et optimisés.

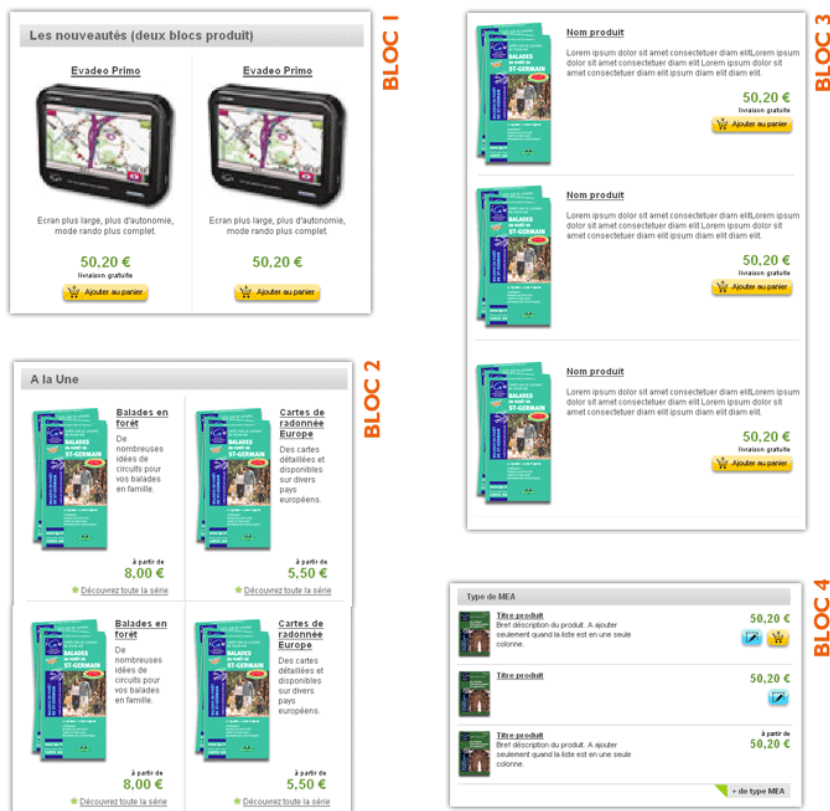
Ce qu'il faut faire

1. 1^{ère} étape : étude de la maquette et identification des différents blocs composants

Dans cette première étape, nous devons procéder à une analyse des *storyboards* et/ou des maquettes fournies comme point de départ à l'écriture de notre code.

Cette analyse va permettre non seulement de définir la structure de nos gabarits de page (haut et bas de page, nombre de colonnes, largeur des colonnes, etc.), mais également d'identifier les différents "blocs" d'informations qui vont la composer.

Dans l'exemple qui suit nous avons identifié 4 blocs différents, chacun présentant un produit ou une liste de produits :



2. 2^{ème} étape : groupement des blocs composants en fonction de leurs différences et/ou de leurs similitudes

Une fois les différents blocs identifiés, nous pouvons lister les caractéristiques de chacun d'entre eux, pour pouvoir "croiser" leurs différences ou leurs similitudes.

A ce stade, nous devons apporter une attention toute aussi importante au design qu'au contenu, car les caractéristiques de style seront également déterminantes pour l'optimisation de notre code.

En prenant l'exemple ci-dessus, nous pouvons lister les caractéristiques suivantes (les similitudes sont présentées sur fond gris et les différences en rouge) :

BLOC1	BLOC2	BLOC3	BLOC4
100% de la largeur de la colonne	100% de la largeur de la colonne	100% de la largeur de la colonne	100% de la largeur de la colonne
Structuré en 2 colonnes	Structuré en 2 colonnes	Structuré en 1 colonne	Structuré en 1 ou 2 colonnes
Titre du bloc : Gris	Titre du bloc : Gris	Pas de titre	Titre du bloc : Gris

sur image de fond dégradée grise	sur image de fond dégradée grise		sur image de fond dégradée grise
Contenu: 2 blocs produits avec Nom produit, image produit centrée , description, informations de prix	Contenu: 2 ou plus blocs produits avec Nom produit, image produit à gauche, description, informations de prix	Contenu: 2 ou plus blocs produits avec Nom produit, image produit à gauche, description, informations de prix	Contenu: 2 ou plus blocs produits avec Nom produit, image produit à gauche, description, informations de prix

En examinant le tableau on remarque que, même si les 4 blocs paraissent assez **différents graphiquement**, ils ont beaucoup de caractéristiques identiques.

Le plus important dans ce cas, c'est que **le contenu est identique pour tous** : nom du produit, image, description et informations de prix. Et pour notre édition (X)HTML, c'est bien de contenu dont il s'agit.

Néanmoins comme signalé précédemment, n'oublions pas que le design est également important pour définir les blocs composants. En effet, même si notre but est de structurer le contenu de nos blocs, les différences de design peuvent demander des modifications sur cette structure.

Pour mieux comprendre, prenons l'exemple de nos 4 blocs : ils ont effectivement exactement le même contenu. Nous pouvons donc utiliser un même code (donc, un même bloc composant) pour ces quatre blocs, comportant les informations de nom du produit, image du produit, description, prix, mentions spéciales et bouton d'ajout au panier :

```
<div>
  <h4><a href="#">Nom du produit</a></h4>
  <p><a href="#" title="Pour plus de detail">
    </a></p>
  <p>Ceci est le paragraphe de description du produit</p>
  <div>
    <p>50,20 &euro; <span>livraison gratuite</span></p>
    <p><a href="#">Ajouter au panier</a></p>
  </div>
</div>
```

Affiché dans un navigateur on obtiendrait :



Une fois que le contenu a été structuré logiquement, nous devons revenir à l'analyse du design pour pouvoir déterminer quelles portions doivent être "étiquetées" pour le design CSS.

Notez que sur notre code, nous avons déjà groupé certaines informations avec la balise `<div>`. Une première balise `<div>` regroupe l'ensemble des informations et une deuxième regroupe les informations de prix ainsi que le bouton d'ajout au panier (informations qui, on le sait par l'observation des maquettes, vont être toujours manipulées ensemble).

Ces deux groupes doivent donc être classés selon leurs rôles, pour qu'on puisse plus tard les distinguer les uns des autres :

```
<div class="prodBlock">
  <h4><a href="#">Nom du produit</a></h4>
  <p><a href="#" title="Pour plus de detail">
    </a></p>
  <p>Ceci est le paragraphe de description du produit</p>
  <div class="priceBlock">
    <p>50,20 € <span>livraison gratuite</span></p>
    <p>
      <a href="#">Ajouter au panier</a></p>
    </div>
  </div>
```

Pour le premier `<div>` qui regroupe l'ensemble des informations du produit, nous donnons la classe **"prodBlock"** pour "bloc produit". Et pour les informations de prix, nous l'attribuons la classe **"priceBlock"** pour "bloc prix".

De la même façon, nous allons classer les paragraphes, car chacun d'entre eux a un rôle très précis :

```
<div class="prodBlock">
  <h4><a href="#">Nom du produit</a></h4>
  <p class="prodImg">
    <a href="#" title="Pour plus de detail">
      </a></p>
  <p class="description">
    Ceci est le paragraphe de description du produit</p>
```

```
<div class="priceBlock">
  <p class="price">50,20 &euro; <span>livraison gratuite</span></p>
  <p class="addButton">
    <a href="#">Ajouter au panier</a></p>
</div>
</div>
```

Maintenant notre contenu est structuré de façon logique et chaque information est classée selon son rôle.

Bien sûr, plus tard lors du design CSS on aura peut être besoin de "peaufiner" ce classement, mais nous avons déjà beaucoup anticipé le travail.

Au fur et à mesure qu'on gagne en expérience, notre code gagne en "prévoyance". Par exemple si on examine les maquettes à nouveau, on va rapidement se rendre compte que le bouton d'ajout au panier a des coins arrondis. Connaissant bien le design CSS, on sait qu'on ne peut pas avoir ce type de design avec un simple lien. On peut donc d'ores et déjà, ajouter une balise `` au contenu du lien car on sait que cette balise nous donnera plus de flexibilité pour designer notre bouton :

```
<a href="#"><span>Ajouter au panier</span></a></p>
```

On a notre bloc produit mais on sait qu'il peut être affiché dans 4 contextes différents selon qu'il apparaît comme une mise en avant, un item d'une liste de produits, une vignette, etc.

Pour sa "mise en contexte", nous allons donner à notre bloc produit un "bloc-père" et c'est ce bloc-père qui va déterminer les propriétés d'affichage uniques à chaque contexte.

Il ne faut pas oublier que s'il existe une différence graphique entre les blocs, c'est parce qu'il y a une raison : dans notre exemple nous avons différentes façons d'afficher les produits en fonction de leurs mises en avant. Nous avons des mises en avant pour les promotions, pour les offres spéciales, pour les nouveaux produits, pour les produits les plus vendus, etc. C'est donc d'après la " fonction " des différents blocs qu'on va nommer les blocs-pères et non pas selon leurs différences d'affichage.

Pour plus de détails sur la façon de nommer les styles, voir point 4.4.

3. 3^{ème} étape : séparation des composants dans un fichier XHTML à part, commenté

Nos blocs composants vont bien sûr faire partie d'une page.

Pour le design CSS il est très important d'avoir nos composants "en situation" pour pouvoir vérifier leur comportement avec le contenu qui les entoure, et de corriger les éventuels problèmes de positionnement et d'affichage.

D'un autre côté nos livrables seront beaucoup plus exploitables si nous séparons chaque composant principal dans un fichier (X)HTML à part. Le code de ce fichier, exclusif au

composant, peut être compris et traité plus facilement par l'équipe de développement car il ne se retrouve pas "noyé" au milieu d'autre code.

Les commentaires nécessaires pourront ainsi être ajoutés de façon plus claire et détaillée.

Nous devons donc créer un fichier (X)HTML pour chaque bloc composant comme par exemple, le haut de page, le bas de page, les blocs produits, les blocs informatifs, etc. L'idéal étant de définir en avance avec l'équipe de développement les blocs qui seront traités séparément (pour plus de détails, voir document "*Méthodologie de cadrage des projets Web*").

Nommez ces fichiers avec le préfix "tpl_" (pour "*template*") pour indiquer qu'ils ne font pas partie des pages assemblées, par exemple : "*tpl_blocProduit.html*".

5 Les bonnes pratiques

5.1 Eviter les " *div-ities* " (la « DIVite aiguë »): Chaque balise a sa fonction, chaque contenu a un rôle

Contexte

Nous devons penser notre contenu en termes de sémantique et pas en termes de présentation. Cependant la balise `<div>` n'a pas une valeur sémantique intrinsèque, c'est-à-dire que cette balise ne va pas donner un rôle à son contenu.

La fonction de la balise `<div>` c'est de créer des groupes pour assembler les contenus qui seront manipulés ensemble, ou pour séparer un contenu de son contexte. Elle a donc une fonction **uniquement structurelle** et c'est avec elle que nous formerons nos "briques" logiques pour construire nos pages.

Nous appelons "*divities*" (qui pourrait être traduit par "divite aiguë", en français) l'usage abusif ou déformé de la balise `<div>`, c'est-à-dire, quand elle est utilisée à la place des balises de valeur sémantique (un `<div>` ne peut pas remplacer un `<h1>`) ou quand nous l'ajoutons à nos pages inutilement.

Ce qu'il faut faire

Utilisez les balises adaptées pour chaque type de contenu et ajoutez des balises `<div>` pour grouper les contenus qui doivent être manipulés ensemble.

Si vous avez un doute concernant le type de balise à utiliser, n'hésitez pas à vérifier le rendu avec votre navigateur sans la feuille de styles. Si le contenu vous paraît structuré et clair, ça veut dire que vous avez fait le bon choix.

Ne cédez pas trop vite non plus à la tentation d'ajouter des balises `<div>` uniquement pour pouvoir ajouter un élément pour le design CSS. Réfléchissez bien à la façon dont le site pourrait évoluer et dans quels cas ce serait acceptable d'avoir un groupement de contenu supplémentaire.

5.2 Bien considérer l'attribution de classes et ou d'Ids : ni trop/ni trop peu

Contexte

Avant tout les classes et les Ids sont utilisés pour classer et/ou identifier le contenu d'une page. Lors de l'écriture (X)HTML, nous devons anticiper le besoin en termes de classement/identification de contenu pour le design CSS et également pour les fonctions javascript.

Ce qu'il faut faire

L'attribution des classes et d'Ids doit être une action bien réfléchie : vous devez toujours vous demander si, d'un point de vue évolutivité et flexibilité, ils sont nécessaires ou non.

Par exemple, il ne sert à rien de donner à tous les éléments d'une même liste la classe "*item*", si tous ces éléments **partagent les mêmes fonctions sémantiques et les mêmes caractéristiques graphiques**. De plus donner la même classe à tous les enfants d'un même père est complètement inutile parce que, de toute façon, ils ne sont pas plus facilement identifiables entre eux.

Si toutes les balises `` d'une liste ont la classe "*item*" par exemple, elles vont partager les mêmes propriétés de style exactement de la même façon que s'elles n'avaient aucune classe attribuée.

D'un autre côté il est toujours intéressant d'identifier par un Id un élément unique dans la page (le nom de l'utilisateur, par exemple), même si du point de vue design il ne se démarque pas des autres éléments. En faisant ainsi, on crée une page plus flexible et plus évolutive : les développeurs pourront manipuler plus facilement cet élément. Et plus encore pour l'évolution du site, cet élément pourra se démarquer visuellement si nécessaire.

Les quelques pistes listées ci-dessous peuvent vous aider à faire le bon choix en ce qui concerne la classification et l'identification de contenu :

- Les classes sont utilisées **pour les exceptions** : si au moment d'écrire votre code, vous savez que ce paragraphe, liste ou n'importe quelle autre portion de contenu doit se **comporter** de façon différente de celle définie par défaut, n'hésitez pas à lui attribuer une classe. Assurez-vous néanmoins, de garder une cohérence avec d'autres classes déjà définies.
- Les classes peuvent être utilisées en **combinaison avec d'autres classes** : si vous avez des blocs très similaires, mais qui ont une propriété unique telle que la couleur de fond par exemple, utilisez une classe pour grouper les propriétés communes et une classe pour chacune des différentes couleurs. En faisant ainsi vous économiserez des définitions, mais surtout assurerez une cohérence entre les blocs similaires. Bien sûr, par-dessus tout évitez de donner des noms de classes comme " rouge " :

```
<div class="window alert">...</div> /* rouge */  
<div class="window">...</div> /* normal */  
<div class="window perso">...</div> /* une autre couleur */
```

- N'hésitez pas à donner un Id aux éléments **uniques (structurants et/ou dynamiques)** ou qui doivent être **manipulés par javascript** : les blocs de structuration de page (containers, haut et bas de page, colonnes, etc.), les éléments de navigation, etc.



ALERTE IE6 !

Combiner un Id avec une classe ou alors deux classes spécifiques, est une procédure assez valable et flexible, mais IE6 n'arrive pas à interpréter correctement leur déclaration. Ainsi, tant qu'IE6 sera un navigateur utilisé, évitez de créer un code pour lequel le design CSS va dépendre des déclarations du type :

```
#wrapper.main { . . . }  
.blockInfo.actus { . . . }
```


5.3 Il ne doit pas exister de contenu "anonyme"

Contexte

Un contenu qui n'a pas sa balise est un contenu inatteignable et donc incontrôlable.

Prenons l'exemple du code ci-dessous :

```
...  
  
<body>  
  <h1>Mon site</h1>  
  <img src= "imgs/logo.gif" alt= " " />  
  <p>Lorem ipsum sit amet.</p>  
  <a href= "doc2.html ">page suivante</a>  
</body>  
</html>
```

Nous avons deux informations qui n'ont pas de rôle précis : l'image "logo.gif" et le lien vers la page suivante. Dans ce cas, nous laissons au navigateur le soin de décider son mode d'affichage pour ces informations.

Ce qu'il faut faire

Ne jamais laissez une portion de contenu sans une balise qui lui donne un rôle précis.

Dans le cas de l'exemple précédent, il nous suffit de baliser les informations "anonymes" avec un paragraphe :

```
...  
  
<body>  
  <h1>Mon site</h1>  
  <p><img src= "imgs/logo.gif" alt= " " /></p>  
  <p>Lorem ipsum sit amet.</p>  
  <p><a href= "doc2.html ">page suivante</a></p>  
</body>  
</html>
```

5.4 Ne pas ajouter de propriétés directement dans la page avec l'attribut "style"

Contexte

L'attribut `style` permet d'ajouter directement dans une page (X)HTML des déclarations CSS :

```
<p style="color: #C00;">Ceci est un paragraphe rouge.</p>
```

Cet attribut est très utile pour faire des tests de style lors du design CSS *.

Même si `style` est un attribut tout à fait valide du point de vue des spécifications, nous vous recommandons fortement de ne pas l'utiliser. En effet, il ajoute des propriétés de styles qui sont prioritaires par rapport à tout autre définition faite par ailleurs (dans une CSS, par exemple). Ainsi, les mises à jour et les modifications de styles faites dans la(es) CSS seront inutilisées, annulant ainsi la philosophie de séparation contenu/forme.

* Préférez Firebug pour ça. Voir "*Méthodologie de travail pour l'édition CSS*", chapitre "Débogage".

Ce qu'il faut faire

Ne jamais laisser l'attribut `style` présent sur le code original de vos pages.

5.5 Utiliser l'attribut " title " avec discernement

Contexte

Internet Explorer nous a donnée la très mauvaise habitude d'afficher la valeur de nos attributs "alt" dans un bulle d'aide lorsque l'on passait le curseur sur une image. Mais comme d'habitude, IE se trompait.

En affichant une bulle d'aide pour une image, IE nous incitait à fournir systématiquement à l'utilisateur une information redondante :



```
<p>  
  <a href="index.html">  
    </a>  
</p>
```

La bulle d'aide dans ce cas, ne dit rien qui puisse intéresser l'utilisateur car il voit bien l'image. Donc répéter la même information n'a aucune valeur ajoutée pour lui.

Disons que, conscients de cette redondance nous avons décidé de modifier la valeur de l'attribut "alt" par quelque chose de plus intéressant pour l'utilisateur :



```
<p>  
  <a href="index.html">  
    </a>  
</p>
```

Effectivement, ça peut être une idée intéressante d'indiquer à l'utilisateur que la fonction du lien c'est de l'envoyer vers la page d'accueil. Mais d'un autre côté, si un utilisateur ne peut pas voir les images, pour une raison quelconque, il va perdre une information très importante pour lui : le titre du site. Au lieu de cela il aura un gros titre que lui dit :



Comme on peut le voir, l'utilisateur a gagné ici une information (la fonction du lien) mais il en a perdu une autre (l'information donnée par l'image). Et nous, nous avons toujours une bulle d'aide redondante.

Finalement, on pourrait être tenté de modifier la valeur de "alt" par quelque chose comme "**W3Cschools.com – Retour à la page d'accueil**". Mais là on n'est plus seulement redondant, on ajoute un problème supplémentaire en donnant une telle longueur à la valeur du "alt" : cet attribut doit fournir une alternative textuelle **courte** à l'image et, même si le W3C ne précise pas ce qu'est un texte court celui-là n'en est certainement pas un.

Si on réfléchit sur la fonction de l'attribut "alt" on comprend mieux la raison de toute cette frustration : l'attribut "alt" doit fournir une brève description de l'image, mais il n'est pas supposé nous montrer une bulle d'aide ou expliquer la fonction d'un lien (car on parle bien de l'image ici, non ?).

Alors, oublions cet attribut et concentrons nous sur ce que nous intéresse vraiment : expliquer un lien et plus précisément, expliquer la fonction d'un lien image. Il doit forcément y avoir un attribut plus adapté, n'est-ce pas ?

Ce qu'il faut faire

Comme on l'a expliqué avant, la frustration de ne pas pouvoir atteindre totalement nos objectifs vient du fait qu'IE6 nous a induit en erreur en affichant, quand il ne fallait pas, une bulle d'aide avec la valeur de l'attribut "alt" : ce n'est pas un comportement standard.

Plus grave encore, il existait déjà un attribut fait pour ça : l'attribut "title".

La fonction de l'attribut "title" est exactement celle de fournir des explications complémentaires à propos d'un élément de la page par le biais d'une bulle d'aide.

L'attribut "title" est extrêmement utile et flexible : il peut être appliqué pour la quasi-totalité des balises (exception faite des balises <base>, <head>, <title>, <html>, <meta>, <param>, <script> et <style>) et il peut décrire ou préciser une action, indiquer une fonction, etc.

Cet attribut est optionnel mais dans les cas des liens, il est **fortement recommandé**. Si le lien est une image son utilisation est **indispensable**.

Prenons notre exemple précédent :

- nous avons une image pour afficher le logo de notre site et cette image est un lien qui pointe vers la page d'accueil;
- l'image du logo a déjà son alternative textuelle donnée par l'attribut "alt" avec une valeur exactement égale au texte de l'image;

- nous souhaitons expliquer à l'utilisateur où le lien du logo va l'amener (ce n'est pas sur le site "W3Cschoools.com" car on y est déjà...) mais on sait déjà que l'attribut "alt" ne peut pas nous aider.

La solution est très simple avec l'attribut "title" :



Images **ON**



Images **OFF**

```
<p>
  <a href="index.html" title="Retour à la page d'accueil">
    </a>
</p>
```

Notez que "title" est appliqué à la balise <a> et pas à la balise . Parce que cet attribut explique le lien et pas l'image, il doit être appliqué au lien.

A chaque fois qu'une image aura la fonction de lien, pensez à ajouter l'attribut "title" à la balise <a>.

5.6 Savoir utiliser les listes de définitions

Contexte

Comme nous l'avons déjà vu dans le chapitre "**Introduction**", la manière de structurer logiquement le contenu de nos pages est extrêmement importante et décisive pour l'évolutivité, la portabilité et l'accessibilité du site.

Dans ce souci de structuration logique, les listes de définitions nous fournissent une aide précieuse pour structurer des contenus qui, à première vue, ne paraissent pas facilement "structurables".

La balise <dl>

Une liste de définition est formée par un titre de liste – un terme - et un ou plusieurs items – descriptions du terme. Par exemple, pour l'affichage d'une adresse, la structuration du contenu par des listes de définition serait tout à fait adaptée :

ACME Industries
24, Rue des tortues
59000 – LILLE
FRANCE

Comme une liste hiérarchique, créée avec les balises et , les listes de définitions sont créées avec la balise <dl> plus une balise de titre de liste, <dt>, et une balise d'item de liste <dd>.

Pour notre adresse, notre code serait :

```
<dl>
  <dt>ACME Industries</dt>
  <dd>24, Rue des tortues</dd>
  <dd>59000 – LILLE</dd>
  <dd>FRANCE</dd>
</dl>
```

Et son affichage par défaut serait :

ACME Industries
24, Rue des tortues
59000 – LILLE
FRANCE

Notez que contrairement à une liste hiérarchique formée par les balises `` ou ``, les listes de définitions ne présentent pas de notion de hiérarchie, mais elles ont plutôt une relation terme/description.

Dans une même liste de définitions, nous pouvons avoir plusieurs termes, suivis par une ou plusieurs descriptions :

```
<dl>
  <dt>Siège ACME Industries</dt>
  <dd>24, Rue des tortues</dd>
  <dd>59000 - LILLE</dd>
  <dd>FRANCE</dd>
  <dt>ACME Industries Paris</dt>
  <dd>150, Avenue Franklin</dd>
  <dd>75000 - PARIS</dd>
  <dd>FRANCE</dd>
</dl>
```

Siège ACME Industries
24, Rue des tortues
59000 – LILLE
FRANCE
ACME Industries Paris
150, Avenue Franklin
75000- Paris
FRANCE

Les listes de définitions nous donnent une grande flexibilité pour le design CSS également. Dans l'exemple ci-dessous, nous avons le contenu suivant à structurer :



En examinant la maquette, nous avons donc une série d'informations à propos d'un élément précis (dans notre cas "Placé 1"). Cette série d'informations étant suivie par une action, symbolisée par l'icône à droite.

On serait rapidement tenté de construire un tableau pour ce contenu et même si dans ce cas précis, l'utilisation d'un tableau pourrait se justifier, une liste de définitions paraît beaucoup plus adaptée et beaucoup plus simple et facile à manipuler pour le design CSS. Le résultat est souvent assez impressionnant :



```
<dl>
  <dt>
    
    <strong>Placé 1</strong>
  </dt>
  <dd>12:08:59 [ 15/10/2008 ]</dd>
  <dd><strong>Durée :</strong> 00:08:50</dd>
  <dd><strong>Regroupement :</strong> 12345678987456</dd>
  <dd class="alertAc">
    <a href="#" title="Libérer main">
      </a>
    </dd>
</dl>
```

Ce qu'il faut faire

N'oubliez jamais que les listes de définitions existent et qu'elles peuvent être une solution intéressante pour la structuration du contenu.

N'oubliez pas : à défaut d'écrire toute votre page avant de passer à la phase de design CSS, regardez souvent le résultat sans la feuille de styles. Vous allez vous rendre compte à quel point le contenu va vous paraître logique et structuré.

5.7 et sont des informations sémantiques : définissez-les dans le code

Contexte

Aujourd'hui les balises de style comme ``, `<i>`, `<u>`, `<align>`, etc., ne sont plus acceptées parce qu'elles portent une information de style directement liée à la balise.

Ces balises donnent au contenu une étiquette de présentation (gras, italique, souligné, etc.) complètement dépourvue de valeur sémantique.

Les balises `` (pour "*fort*") et `` (pour "*mettre l'accent sur*"), elles, ont un rôle uniquement sémantique : la mise en avant ou l'accentuation d'une portion de contenu. La façon dont ces mises en avant sont présentées importe peu, même si par défaut, les navigateurs web les affichent respectivement en gras et en italique.

Comme le rôle de ces balises est sémantique, les différents agents utilisateurs vont les interpréter à leur façon. Par exemple, un synthétiseur vocal va donner plus d'accent au contenu balisé avec `` et "dira" dans un volume plus fort le contenu balisé par ``.

Ce qu'il faut faire

Les informations sémantiques doivent bien sûr être présentes dans les pages. Ainsi, quand nous devons mettre en avant ou mettre un accent sur une portion de notre contenu, nous devons ajouter ces balises directement sur nos pages.

La façon dont ces mises en accent vont être présentées visuellement sera ensuite traitée par la feuille de styles.

5.8 Utiliser des lignes (<hr />) pour séparer les principaux groupes de contenu

Contexte

Tester les pages sans le design CSS vous permet de mieux appréhender la structuration et la hiérarchisation du contenu. Les balises `<hr />`, cachées par la feuille de styles, permettent de grouper les contenus de nature différente, facilitant ainsi la lecture de la page par des agents utilisateurs non-graphiques.

Ce qu'il faut faire

Consultez la page sans sa feuille de styles et identifiez les groupes de contenu principaux de la page. Insérez une balise `<hr />` pour séparer les contenus que vous estimez nécessaires pour une meilleure lecture de la page.

Par exemple, c'est toujours une bonne idée d'ajouter une ligne pour séparer le pied de page du corps de la page.

Il ne faut pas non plus abuser de cette méthode : soyez judicieux.



ALERTE IE6 !

Dans certains cas, IE6 peut présenter des bugs aléatoires avec l'ajout des balises `<hr />`. Ca ne veut pas dire qu'il faut les éviter, mais ayez ça en tête lorsque vous vous trouvez face à des problèmes d'affichage peu usuels sous IE6.

5.9 Toujours utiliser la balise <h1> pour le titre du site

Contexte

Quand nous parlons de structuration de contenu, il s'agit de donner à notre contenu une logique et une hiérarchie qui reposent sur sa relation avec les autres éléments qui l'entourent.

Ainsi, il est nécessaire de bien comprendre le rôle de chaque type de contenu afin de l'identifier correctement.

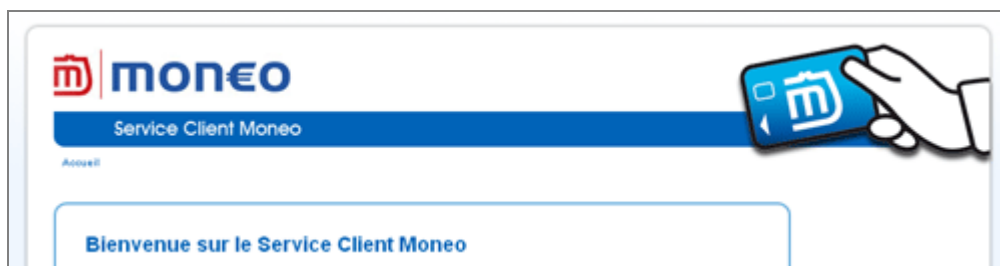
Si on prend un livre, effectivement on ne peut pas dire que le titre du livre apparaît sur chaque page comme titre principal, mais il apparaît toujours discrètement en haut de page. Or, une page d'un livre est physiquement liée aux autres pages et à la couverture. Le livre et ces pages sont manipulés ensemble. On a donc moins besoin de rappeler au lecteur la référence de ce qu'il lit.

Dans un site web, les pages sont bien sûr liées entre elles par des liens hypertexte, mais ce lien est transparent. Nous pouvons arriver sur la page d'un site, sans même être passés par sa page d'accueil (ou sa "couverture"). Le besoin ici d'indiquer à chaque page le niveau premier d'identification de notre document est beaucoup plus important.

Ce qu'il faut faire

Le titre du site est le titre de niveau 1, ou titre principal et il doit donc être étiqueté comme tel avec la balise <h1>.

Attention à ne pas confondre le logo de l'entreprise avec le titre du site :



Dans l'exemple ci-dessus, même si graphiquement le logo "Moneo" est plus en évidence, ce n'est pas lui le titre du site, mais "Service Client Moneo" (il existe déjà un site avec le titre "Moneo" : www.moneo.fr).

Une fois définie, la balise <h1> ne doit plus être utilisée dans la page et tous les autres titres qui suivent doivent respecter une logique hiérarchique partant du niveau 2.

Dans ce sens, **le titre de la page** doit toujours être de niveau 2 (<h2>), les sous-titres de niveau 3, etc.

Les titres d'un **contenu "transverse"** sont souvent de niveau 3 s'ils **complètent ou s'ils enrichissent le contenu principal**. Ils doivent apparaître comme étant de niveau 2 dans le cas contraire.

5.10 Le titre du site doit envoyer l'utilisateur à la page d'accueil

Contexte

A n'importe quel moment de sa visite, l'utilisateur d'un site doit pouvoir revenir à la page d'accueil. Soit pour revoir des informations qui l'intéressent, soit pour avoir un nouvel aperçu des informations qu'il peut y trouver, soit pour retrouver ses repères dans la navigation du site.

Ainsi, le titre du site doit pouvoir envoyer l'utilisateur à tout moment à la page d'accueil, même si un autre lien de navigation sur la page permet également de le faire.

Ce qu'il faut faire

Dans les gabarits des pages internes, qu'il y ait ou non, dans la navigation, un lien vers la page d'accueil, prévoyez toujours d'ajouter un lien à la balise `<h1>` qui pointe vers la page d'accueil. Indiquez la destination de ce lien par l'attribut `title`, que ce lien contienne ou non une image:

```
<h1>  
  <a href="index.html" title="Retourner à la page d'accueil">  
    </a>  
</h1>
```

5.11 Toujours ajouter un lien caché pour sauter les menus

Contexte

Un synthétiseur vocal va lire une page web du début à la fin en s'arrêtant seulement par une commande utilisateur.

Comme l'utilisateur n'a pas une commande qui lui permet simplement de "sauter" la lecture d'un bloc de contenu, à chaque nouvelle page, le synthétiseur vocal va répéter les mêmes informations trouvées en début de page et notamment le(s) menu(s) de navigation. Ce qui peut être très ennuyant et très long.

L'ajout de "points de contrôle" dans la page améliore le confort de l'utilisateur d'un synthétiseur vocal (ou même d'un navigateur textuel) car ils lui permettent de sauter la lecture des informations répétitives.

Ce qu'il faut faire

Un "point de contrôle" n'est rien de plus qu'un lien (caché ou pas pour les navigateurs ordinaires) qui pointe vers là où se trouve l'information spécifique de la page :

```
<p class="skip"><a href="#wrapper">Sauter le menu</a></p>
<ul id="ulMainNav">
  <li><a href="#">Rubrique 1</a></li>
  <li><a href="#">Rubrique 2</a></li>
  <li><a href="#">Rubrique 3</a></li>
</ul>
```

Dans l'exemple ci-dessus, le lien "Sauter le menu" permet d'aller directement vers le corps de la page, structuré dans le bloc identifié par l'Id "wrapper".

La classe "skip" est définie dans la CSS pour "cacher" ce lien des navigateurs ordinaires (ceux qui peuvent interpréter la CSS), par contre il sera lu par un synthétiseur vocal ou un navigateur textuel.

Conseil

N'ajoutez pas des ancres sur la page pour les points de contrôle. Utilisez toujours comme pointeur les Ids des blocs structurants comme par exemple #container, #header, etc..

5.12 Créer vos menus avec soin

5.12.1 Un menu de rubriques (ou une liste de liens) doit toujours être structuré par des balises de liste ()

Contexte

Un menu n'est rien d'autre qu'une liste d'options qui peuvent, elles aussi, avoir de "sous-options". On a donc ici une relation hiérarchique de valeurs.

Pour être bien structuré sémantiquement, un menu de rubriques doit donc pouvoir présenter ces deux principales caractéristiques : la notion de **liste** et de **hiérarchie**.

Ce qu'il faut faire

Le choix sémantique évident pour structurer correctement le code de votre menu sera la balise (ou , pour les listes numérotées).

Ces balises permettent justement de lister des rubriques et de les hiérarchiser. Par exemple, le code ci-dessous peut être utilisé pour créer le menu du site "Boutique IGN" :

```
<ul id="ulMainNav">
  <li><a href="#" title="Retour à l'accueil">Accueil</a></li>
  <li><a href="#">Cartes & guides</a>
    <ul class="dropDown">
      <li><a href="#">Rando France</a></li>
      <li><a href="#">Rando Etranger</a></li>
      <li><a href="#">Tourisme France</a></li>
      <li><a href="#">Tourisme Etranger</a></li>
      <li><a href="#">Atlas et cartes routi&egrave;res</a></li>
      <li><a href="#">Aviation / g&eacute;ologie</a></li>
      <li><a href="#">Cartes anciennes</a></li>
    </ul></li>
  <li><a href="#">GPS / Pocket PC</a>
    <ul class="dropDown">
      <li><a href="#">Multi usages</a></li>
      <li><a href="#">Randonn&eacute;e</a></li>
      <li><a href="#">Air / Mer / G&eacute;ologie</a></li>
      <li><a href="#">Pocket PC</a></li>
      <li><a href="#">Accessoires GPS</a></li>
    </ul></li>
  . . .
</ul>
```



L'image ci-dessus présente une copie d'écran d'une partie du menu du site "IGN Loisirs", avec un sous-menu déroulant déployé (rubrique "Cartes & Guides").

A travers la feuille de styles, nous avons pu transformer complètement l'apparence de notre menu qui garde, même sans la CSS, la structure sémantique souhaitée (image à droite).

- ♦ [Accueil](#)
- ♦ [Cartes & guides](#)
 - ◊ [Rando France](#)
 - ◊ [Rando Etranger](#)
 - ◊ [Tourisme France](#)
 - ◊ [Tourisme Etranger](#)
 - ◊ [Atlas et cartes routières](#)
 - ◊ [Aviation / géologie](#)
 - ◊ [Cartes anciennes](#)
- ♦ [GPS / Pocket PC](#)
 - ◊ [Multi usages](#)
 - ◊ [Randonnée](#)
 - ◊ [Air / Mer / Géologie](#)
 - ◊ [Pocket PC](#)
 - ◊ [Accessoires GPS](#)
- ♦ [CD / DVD](#)
 - ◊ [Randonnée](#)
 - ◊ [Photos](#)
 - ◊ [Air / Mer / Géologie](#)
- ♦ [Photos](#)
- ♦ [Reliefs, posters, globes](#)
 - ◊ [Cartes en relief](#)
 - ◊ [Posters](#)
 - ◊ [Cartes anciennes](#)
 - ◊ [Globes](#)
- ♦ [Carte à la carte](#)

5.12.2 Traiter et identifier la rubrique courante

Contexte

Lors de la **phase de design**, nous pensons au traitement visuel de la "mise en avant" de la rubrique courante.

Lors de la **phase de codage des menus**, nous devons considérer également la façon selon laquelle la rubrique courante va être traitée sémantiquement et fonctionnellement :

- pour être toujours **identifiée** comme rubrique courante **sans la feuille de styles** et...
- ... pour déterminer dans quel contexte elle sera **cliquable** ou non.

Ce qu'il faut faire

Pour identifier la rubrique courante

Pour identifier la rubrique courante, nous devons toujours **lui donner un ID**.

En effet, l'attribution d'un ID va permettre au design CSS d'avoir plus de contrôle sur la rubrique courante dans ses différents contextes, ainsi que d'accentuer le caractère unique de cet élément dans la page.

Comme toute rubrique d'un menu se trouve dans une balise `` (voir point 5.12.1), et comme la rubrique courante peut ou non être cliquable en fonction du contexte dans lequel elle se trouve (nous verrons pourquoi tout à l'heure), préférez l'attribution de l'Id à la balise `` :

```
<ul id="ulMainNav">
  <li><a href="#" title="Retour à l'accueil">Accueil</a></li>
  <li id="secOn"><a href="#">Cartes & guides</a>
    <ul class="dropDown">
      <li><a href="#">Rando France</a></li>
    </ul>
  </li>
  . . .
</ul>
```



Comme montré par l'illustration ci-dessus, notre id "secOn" va déterminer une couleur différente pour la rubrique courante, mais on n'a rien qui puisse l'identifier si la CSS est désactivée ou si l'utilisateur n'a pas, ou a désactivé, les informations de couleur de son navigateur.

- ♦ [Accueil](#)
- ♦ [Cartes & guides](#)
 - ♦ [Rando France](#)
 - ♦ [Rando Etranger](#)
 - ♦ [Tourisme France](#)
 - ♦ [Tourisme Etranger](#)
 - ♦ [Atlas et cartes routières](#)
 - ♦ [Aviation / géologie](#)

Dans ce cas, nous devons ajouter une information supplémentaire pour indiquer à l'utilisateur quelle est la rubrique courante. L'utilisation de l'attribut `title` est une bonne méthode :

```
<ul id="ulMainNav">
  <li><a href="#" title="Retour à l'accueil">Accueil</a></li>
  <li id="secOn" title="Rubrique courante">
    <a href="#">Cartes & guides</a>
    <ul class="dropDown">
      <li><a href="#">Rando France</a></li>
      . . .
    </ul>
  </li>
</ul>
```

Cependant, l'attribut `title` doit être attribué à la balise `` plutôt qu'au lien de la rubrique à fin de ne pas générer de confusion avec ce qui pourrait être l'explication du lien. En effet, ici l'attribut `title` explique le contenu de la balise `` et pas l'action du lien.

Evidemment, dans le cas où **la rubrique courante n'est pas cliquable** (ne présente pas de lien), la différenciation se fait naturellement. Mais là encore il est intéressant d'ajouter l'attribut `title` pour renforcer son identification.

Dans les deux cas (avec ou sans lien), pensez toujours à ajouter la balise `` à la rubrique courante pour renforcer visuellement et sémantiquement son état et ça indépendamment si l'affichage de la rubrique est définie en gras ou pas dans la CSS :

Avec lien :

```
<ul id="ulMainNav">
  <li><a href="#" title="Retour à l'accueil">Accueil</a></li>
  <li id="secOn" title="Rubrique courante">
    <strong><a href="#">Cartes & guides</a></strong>
    <ul class="dropDown">
      <li><a href="#">Rando France</a></li>
    </ul>
  </li>
  . . .
```

Sans lien :

```
<ul id="ulMainNav">
  <li><a href="#" title="Retour à l'accueil">Accueil</a></li>
  <li id="secOn" title="Rubrique courante">
    <strong>Cartes & guides</strong>
    <ul class="dropDown">
      <li><a href="#">Rando France</a></li>
    </ul>
  </li>
  . . .
```

Pour déterminer dans quel contexte elle sera cliquable ou non

Une rubrique **doit présenter un lien** seulement dans deux cas :

- quand elle **n'est pas** la rubrique courante;
- quand elle est la rubrique courante, mais que l'utilisateur se trouve dans **une de ses sous-rubriques** (et pas dans la page d'accueil de la rubrique).

Dans ce deuxième cas, la rubrique courante doit encore être identifiée ainsi que sa sous-rubrique courante, mais le lien doit être disponible pour permettre à l'utilisateur de "remonter" d'un niveau de navigation :



L'exemple ci-dessus montre que nous sommes sur la page "Rando France" qui est une sous-rubrique de la rubrique de niveau 1 "Cartes & Guides". Dans ce cas, la rubrique de niveau 1 devient cliquable pour permettre à l'utilisateur de revenir à l'accueil de cette rubrique. La rubrique de niveau 2 en revanche, ne peut pas être cliquable car c'est là que l'utilisateur se trouve.

La même logique doit être suivie pour tous les sous-niveaux de rubrique.

Comme déjà précisé, dans le cas où la rubrique n'est pas cliquable, **remplacez le lien par la balise **. Cela va à la fois renforcer la différence avec les autres rubriques du menu et, d'un autre côté, permettre au design CSS de garder la même cohérence graphique, c'est-à-dire que les propriétés données par la balise <a> seront transmises par la balise . Pour plus de détails, consultez le document "Guides des projets Web : Méthodologie pour d'édition CSS".

A retenir

Lors du codage, pensez à ajouter tous les cas de figure (**tous les états des rubriques**) pour couvrir toutes les éventualités côté code et pour faciliter le design CSS.

5.13 Une ligne de navigation (fil d'Ariane) n'est pas une liste

Contexte

Comme expliqué au point 5.12.1, un menu ne peut pas être mieux structuré sémantiquement que par des balises de liste. Néanmoins, une ligne de navigation ou fil d'Ariane, n'est pas un menu car **il n'a pas une logique de niveaux hiérarchiques**, mais plutôt **une logique séquentielle**.

Structurer le fil d'Ariane par des balises de liste, comme montre l'exemple ci-dessous, n'est donc pas correct :

```
<ul id="bdCrumbs">
  <li><a href="#">Accueil</a> &gt;</li>
  <li><a href="#">Ma rubrique</a> &gt;</li>
  <li>Ma sous-rubrique</li>
</ul>
```

- Accueil >
- Ma rubrique >
- Ma sous-rubrique

Ce qu'il faut faire

Structurez toujours votre fil d'Ariane avec une balise <p> :

```
<p id="bdCrumbs">
  <a href="#">Accueil</a> &gt;
  <a href="#">Ma rubrique</a> &gt;
  Ma sous-rubrique</p>
```

Accueil > Ma rubrique > Ma sous-rubrique

5.14 Les images

5.14.1 Donner toujours une alternative textuelle aux images

Contexte

Nos pages doivent être portables et accessibles :

- **portables** car elles doivent pouvoir être lues par n'importe quel média capable de comprendre le langage (X)HTML (navigateurs, navigateurs textuels, synthétiseurs vocaux, tv, mobile, bornes, etc.).
- **accessibles** car elles doivent pouvoir être lues indépendamment des paramètres et contraintes que l'utilisateur peut présenter.

Dans ce souci de portabilité et d'accessibilité, la possibilité ou non d'afficher les images est une donnée non contrôlable : nous ne pouvons pas prévoir (et encore moins l'empêcher) si un utilisateur va "lire" nos pages avec un synthétiseur vocal ou s'il a simplement désactivé l'affichage des images (par exemple parce que là où il se trouve la connexion est trop lente pour se permettre le luxe de charger les images).

Donc, pour être sûr que tous vont accéder à la totalité des informations sur nos pages en ce qui concerne les images, nous devons simplement nous assurer de leur donner des alternatives textuelles.

Ce qu'il faut faire

L'attribut "alt" nous permet de donner à la balise une valeur qui **explique** ou qui **remplace** l'information donnée par l'image :



```
<p></p>
```

L'exemple ci-dessus nous montre que si pour une raison quelconque, l'utilisateur ne peut pas "voir" l'image, il peut tout de même connaître l'information qu'elle contient ou veut faire passer.

Les images purement décoratives (qui ne véhiculent pas d'information en soit) doivent elles aussi présenter l'attribut "alt", mais dans ce cas sa valeur doit être nulle :



5.14.2 Donner ou pas une dimension aux images ?

Contexte

Rappelons nous, notre code (X)HTML est une matrice : plus solide et flexible il est, moins on devra le retoucher ou le modifier dans le temps.

Or les informations de largeur et de hauteur d'images ne sont rien de plus que des informations de style qui peuvent avoir besoin d'être actualisées quand les images sont remplacées.

L'ajout des attributs de largeur et de hauteur est toute à fait valide, même pour des DOCTYPE strictes, mais on gardera une plus grande flexibilité de mise à jour de nos pages si ces attributs ne sont pas prédéfinis.

D'une autre côté, ne pas spécifier les dimensions des images peut ralentir le chargement d'une page, car les navigateurs vont devoir examiner chaque image et calculer l'espace pour les afficher avant d'afficher le contenu qui suit.

Si les attributs "width" et "height" sont définis dans la balise , le navigateur va pouvoir réserver l'espace pour l'image avant même de la charger.

Ce qu'il faut faire

On est ici dans une vraie impasse. Et si le doute persiste cela veut dire qu'on ne peut pas statuer entre un choix ou un autre.

Cependant, n'oublions pas que les dimensions peuvent être définies dans la feuille de styles plutôt que dans nos fichiers (X)HTML, et ça c'est une ressource importante à prendre en compte.

Ainsi, autant que possible **laissez les définitions de dimension des images dans les feuilles de styles**. Cependant, évitez de créer de nouvelles classes pour autant (voir les recommandations pour le traitement des images dans "*Méthodologie de travail pour le design CSS*").

5.14.3 Pour les images décoratives, préférez l'utilisation d'images de fond

Contexte

Sur une page web, nous rencontrons un grand nombre d'images qui sont utilisées pour le design tels que les coins arrondis, les dégradées, les textures, etc.

Ces images "décoratives" n'ont pas de valeur informative, c'est-à-dire qu'elles ne véhiculent pas d'information mais elles sont une partie importante d'un ensemble dédié à créer l'identité et l'ambiance du site.

Et quand on parle d'ambiance on parle de style, de présentation, de forme, mais pas de **contenu**. Ainsi, ces images n'ont pas vocation à être appelées directement par la page, mais à être définies dans la feuille de styles.

Ce qu'il faut faire

De préférence et autant que possible affichez les images d'une page comme images de fond définies dans la feuille de styles, au lieu de les appeler directement dans le code (X)HTML.

La seule exception est donnée aux images qui véhiculent une information comme par exemple le logo du site, ou les textes sous forme d'image qui pour des raisons d'accessibilité, doivent être évitées au maximum.

NOTE: Si le design de votre site demande l'utilisation de familles de polices spécifiques pour les titres par exemple, utilisez plutôt une technique de remplacement d'image (voir les recommandations pour le traitement des images dans "*Méthodologie de travail pour le design CSS*").

5.15 Formulaires

L'écriture d'un formulaire est non seulement un travail long mais surtout très délicat. Les enjeux d'accessibilité y sont plus sensibles et **la qualité du code** est d'autant plus indispensable.

Savoir bien écrire un formulaire n'est pas vraiment difficile, mais on doit surtout s'assurer de bien prendre en compte tous les cas de figure possibles. Notre code doit être sans erreurs.

Le design avec ses différents éléments va forcément influencer le mode de construction du formulaire.

5.15.1 Donner toujours un Id aux formulaires

Contexte

Les valeurs par défaut des balises sont définies lors du design CSS y compris celles des éléments de formulaire. Toutefois, nous trouvons souvent des différences spécifiques sur chaque formulaire, comme la largeur de la colonne des intitulés de champs par exemple.

L'attribution d'un Id unique à un formulaire ne permet pas seulement de gérer ses exceptions dans la feuille de styles cela fournit également au développement un moyen indépendant de l'atteindre.

Ce qu'il faut faire

A chaque nouveau formulaire, attribuez un Id unique à la balise `<form>` en suivant les règles de nommage présentées par le point 4.4.

HTML x XHTML:

N'attribuez pas d'attribut `name` à la balise `<form>` si vous codez en **XHTML Strict**, sous peine d'invalider votre code.

Dans le cas contraire, vous pouvez attribuer l'Id et l'attribut `name` à condition que les deux aient la même valeur.

5.15.2 Bien considérer les attributs de la balise <form>

Contexte

Les attributs de la balise <form> dépendent beaucoup du DOCTYPE choisi.

Le tableau ci-dessous vous donne un rappel des trois principaux attributs et dans quel contexte ils peuvent/doivent être utilisés :

Attribut	DOCTYPE *	CONTEXTE
method	Tous	Optionnel (valeur par défaut : "get")
action	XHTML 1.0 Strict	obligatoire
	XHTML 1.0 Transitionnel	obligatoire
	HTML 4 Strict	obligatoire
	HTML 4 Loose	optionnel
name	XHTML 1.0 Strict	Interdit
	XHTML 1.0 Transitionnel	Interdit
	HTML 4 Strict	Possible
	HTML 4 Loose	Possible

* Les DOCTYPEs du type "frameset" ne sont délibérément pas pris en compte en cohérence avec notre recommandation d'abolir les *framesets*

Ce qu'il faut faire

Devant ces quelques différences entre les DOCTYPEs, néanmoins importantes, nous préconisons la déclaration constante des attributs pour satisfaire tous les cas de figure.

Donc **indépendamment du DOCTYPE choisi**, nous vous recommandons de :

- **Toujours déclarer l'attribut `action`** : comme il est obligatoire pour 3 des 4 DOCTYPEs utilisés par le Studio, prenez l'habitude de toujours le déclarer avec une valeur vide.
- **Toujours déclarer l'attribut `method`** : même s'il est optionnel, déclarer cet attribut permet de modifier la valeur "get" par défaut qui n'est plus autant utilisée

qu'auparavant. Donc renseigner cet attribut avec la valeur "post" facilite le travail de dynamisation dans la plupart des cas.

- **Ne jamais déclarer l'attribut name :** il est interdit pour le langage XHTML (DOCTYPE strict ou pas). Le mieux est de ne jamais le déclarer pour ne pas être induit en erreur. Dans le cas où l'équipe de développement a besoin de cet attribut, c'est à elle de l'ajouter et en toute connaissance de cause. Bien sûr, ce détail doit être prévu lors du cadrage du projet.
- **Toujours attribuer un id :** l'attribution d'un Id permet d'identifier le formulaire pour le design CSS et pour le javascript en remplacement de l'attribut name.

```
<form id="frExe" method="post" action="">
. . .
</form>
```

5.15.3 Utiliser toujours les fieldsets pour grouper les champs de même nature

Contexte

Dans un formulaire, la balise <fieldset> est dédiée au groupement des champs qui partagent la même nature d'information. Par exemple, nous pouvons grouper toutes les données d'adresse dans un même *fieldset*, ce qui permet de créer une unité entre ces champs, tout en les séparant des champs d'autres types de données.

Dans l'exemple ci-dessous, nous avons deux groupes de champs différents qui traitent de données différentes (adresse de facturation et adresse de livraison) :

The image shows a web form with two distinct sections, each enclosed in a fieldset. The first fieldset is titled 'Votre adresse de facturation' and contains five input fields: 'N° et libellé voie', 'Complément d'adresse', 'Code Postal', 'Commune', and a dropdown menu for 'Pays' set to 'France'. The second fieldset is titled 'Votre adresse de livraison' and contains a checkbox labeled 'Utiliser le même adresse que l'adresse de facturation' followed by the same five input fields as the first fieldset. Each input field has a small green question mark icon to its right.

Le fait de grouper les champs de même nature va rendre la structure du formulaire plus simple et évidente, facilitant ainsi sa compréhension et son remplissage.

L'utilisation des *fieldsets* facilite aussi la manipulation isolée des groupes de champs par javascript, comme dans le cas des champs dynamiques par exemple, qui sont affichés ou non en fonction des choix de l'utilisateur.

Ce qu'il faut faire

Avant l'écriture d'un formulaire, faites un schéma des données qu'il présente et groupez logiquement les données de même nature. Ainsi, lors du codage, vous pouvez facilement ajouter vos balises `<fieldset>` pour créer les différents groupes :

```
<form method="post" action="">
  <fieldset>
    <h4>Votre adresse de facturation</h4>
    . . . (Les champs) . . .
  </fieldset>

  <fieldset>
    <h4>Votre adresse de livraison</h4>
    . . . (Les champs) . . .
  </fieldset>
</form>
```

Si nécessaire, vous pouvez également créer des sous-groupes de champs, car une balise `<fieldset>` peut contenir d'autres *fieldsets*.

Attention, un *fieldset* peut apparaître uniquement dans une balise `<form>`.

5.15.4 Si possible, préférer la balise `<legend>` pour le titre d'un *fieldset*

Contexte

La balise `<legend>` définit un titre pour un groupe de champs ou *fieldset* :

La balise `<legend>` doit être la balise à choisir pour le titre d'un *fieldset*, mais aujourd'hui elle présente des limites trop importantes pour le design CSS. En effet, son affichage ne peut pas être très différent de son affichage par défaut (image ci-dessus).

Premièrement, cette balise souffre de la grande différence d'interprétation d'affichage par les navigateurs et deuxièmement elle n'a pas les qualités d'un élément bloc ordinaire. De ce fait, nous sommes souvent contraints pour respecter le design, d'utiliser une balise du type `<h>` à la place de `<legend>`.

L'utilisation de balises `<h>` dans ce cas n'est pas une erreur en soit (ce sont des balises de titres après tout), mais ça nous force à abandonner un élément qui a toute sa légitimité.

Ce qu'il faut faire

Vous devez avant tout, **étudier le design du formulaire proposé** pour pouvoir considérer l'utilisation ou non de la balise `<legend>`, tout en sachant qu'il est pratiquement impossible de donner à cette balise une largeur en pourcentage (même si elle est "floatée"), de la positionner, ou d'avoir un même rendu entre les navigateurs standards et IE6.

Si son utilisation n'est pas possible, utilisez une balise `<h>` en considérant les niveaux des titres déjà déclarés dans la page. Dans un cas "classique", si le nom du site est `<h1>`, le nom de la page `<h2>` et si le titre du formulaire est `<h3>` (s'il y a lieu), nous devons utiliser, bien sûr, les balises `<h4>` pour les titres des *fieldsets*.

Assurez-vous surtout de garder une cohérence entre les différents formulaires. Si dans une page les titres des *fieldsets* sont des balises `<h4>`, utilisez toujours cette balise pour toutes les pages de formulaire.

Pour les formulaires spécifiques comme le bloc d'identification (identification/mot de passe) ou le formulaire de recherche simple, si des titres existent, ils doivent respecter la hiérarchie de la page.

5.15.5 Utiliser les labels pour associer un intitulé à son champ

Contexte

La balise `<label>` existe pour définir l'intitulé d'un champ de formulaire et pour créer une association entre l'intitulé et son champ.

L'utilisation de cette balise est très importante car :

- **elle facilite le remplissage d'un formulaire par l'activation des intitulés** (si l'utilisateur clique sur un intitulé, le curseur se positionne dans le champ auquel il est associé). De cette façon, l'utilisateur peut cocher une case en cliquant sur la case à cocher comme sur son intitulé par exemple

- **elle facilite la compréhension du formulaire pour les utilisateurs de synthétiseurs vocaux**
- **elle facilite la mise en page des formulaires par le design CSS** (les balises `<label>` offrent une plus grande flexibilité pour aligner les intitulés et les champs de formulaire sans avoir besoin d'utiliser des balises de tableau)

Ce qu'il faut faire

Pour que l'intitulé soit associé à son champ nous devons utiliser l'attribut `"for"` avec la valeur de l'Id attribué au champ :

```
<p><label for="ipName">Nom :</label> <input type="text" id="ipName" /></p>
```

Ceci implique, bien évidemment l'attribution d'un id unique à chaque champ de formulaire, ce qui n'est pas de tout une contrainte mais une aide supplémentaire au design CSS et au javascript. En effet, chaque champ est ainsi manipulable aussi bien de façon générique que de façon individuelle.

Consultez le document "*Méthodologie de travail pour le design CSS*" pour plus d'information.

5.15.6 Toujours grouper le label et son champ dans une balise `<p>`

Contexte

L'intitulé d'un champ de formulaire et son champ forment un groupe unique, où chacun des ses deux éléments est sémantiquement identifié par les balises `<label>` et `<input>`. Néanmoins, ces deux balises elles-mêmes, ne sont que des portions de contenus spécifiques (des chaînes de caractères, ou des éléments *inline*) et n'ont donc pas de fonction structurale, comme les titres ou les paragraphes.

Ce qu'il faut faire

Nous devons toujours donner au groupe label/input une balise structurante qui va les positionner ou les cadrer dans la page.

Pour la plupart des cas, la balise la plus adaptée est la balise `<p>`.

Cette structuration en paragraphes va nous donner également une plus grande souplesse pour le design CSS, comme la possibilité de définir les marges entre les champs, l'alignement en relation au *fieldset*, etc.

HTML x XHTML:

Attention : le langage **HTML4** n'accepte pas des balises `<p>` dans un *fieldset* ! Ca peut paraître complètement incongru mais votre page ne sera pas validée si vous avez une quelconque balise `<p>` dans un *fieldset*.

Ceci est l'un des principaux handicaps que le langage HTML présente pour la structuration du contenu et pour le design CSS.

En **XHTML**, à l'inverse, votre **page ne sera pas validée** si vous ne groupez pas le label et son input dans une balise `<p>`.

5.15.7 Attribuer toujours un ID et un "name" à un champ de formulaire

Contexte

L'attribution d'un Id à un champ de formulaire est toujours nécessaire pour pouvoir l'associer à son intitulé, mais pas seulement. Une fois identifié par un Id unique, le champ peut être manipulé individuellement par javascript et il peut acquérir des propriétés uniques, comme la largeur.

Même si l'attribution de l'Id à un champ ouvre toutes les possibilités d'un accès unique, il est encore peu utilisé par les *frameworks* pour l'interprétation d'un formulaire. Donc, la présence de l'attribut "name" est toujours nécessaire.

Ce qu'il faut faire

Pour les balises `<input>` du type "text", attribuer toujours un ID et un "name", avec la même valeur :

```
<input type="text" id="ipName" name="ipName" value="" />
```

Contrairement aux balises `<input>` de type "text", l'attribut "name" des balises `<input>` du type "checkbox" ou "radio" va définir le groupe auquel la case à cocher appartient. Donc, sa valeur ne peut pas être la même que la valeur de l'ID :

```
<input type="radio" id="regY" name="register" value="yes" /> Oui  
<input type="radio" id="regN" name="register" value="no" /> Non
```

De cette façon, nous pouvons identifier le groupe de cases à cocher **"register"** et savoir que celles-ci peuvent avoir la valeur "yes" ou "no".

5.15.8 Prévoir la construction de boutons liens et des boutons input

Contexte

Les boutons de formulaire peuvent être "écrits" de trois façons différentes :

- ♦ avec la balise `<input>`
- ♦ avec la balise `<a>`
- ♦ avec la balise `<button>`

Le choix des balises qui vont appliquer les actions d'un formulaire va dépendre du script qui sera utilisé lors de la dynamisation des pages. Néanmoins, chacune de ces manières présente ses avantages et ses inconvénients, tant du point de vue de la sémantique que du point de vue du design CSS.

La balise `<input>`

La **balise `<input>`** peut déterminer des boutons d'annulation ou de validation en fonction de la valeur de l'attribut `type` ("submit" ou "reset") et elle est toujours identifiée et affichée comme un bouton, quel que soit l'agent utilisateur.

Elle a comme avantage une prise en compte correcte pour l'accessibilité, la relative facilité de modification de ses propriétés par le design CSS, ainsi qu'une prise en compte naturelle par le formulaire de ses actions de validation ou d'annulation.

La balise `<input>` peut encore être facilement remplacée par des images, sans diminuer l'accessibilité, à condition bien sûr, que l'attribut `"alt"` soit correctement renseigné :

```
<input type="image" src="imgs/btValid.gif" alt="Valider" />
```


La balise <a>

Remplacer les balises `<input>` par des liens simples n'offre pas plus de flexibilité ou de ressources, mais ils sont fréquemment utilisés pour le développement en raison des *frameworks* ou scripts qui dépendent des actions passées par des balises `<a>`.

Contrairement à la balise `<input>`, les liens ne sont pas pris en compte naturellement par un formulaire. Ces actions sont donc entièrement dépendantes des attributs d'événements comme l'attribut `onclick`.

Bien évidemment, les liens nous donnent une grande flexibilité pour le design de nos boutons de formulaires mais en même temps ils ne sont pas sémantiquement exacts dans le contexte d'une action de formulaire. Ils nécessitent donc une classe CSS spécifique pour chaque type d'action (validation, annulation, ou autre) ainsi que l'attribution d'une fonction différente à l'événement `onclick` pour chaque type d'action.

La balise <button>, la solution miracle sauf pour IE

Alors que les liens ont un rôle un peu atypique pour les actions de formulaire, et que la balise `<input>` présente quelques restrictions de design, la balise `<button>` semble réunir les points forts de ces deux premières balises.

La balise `<button>` est apparue avec la version 4.0 du langage HTML pour faciliter la customisation du design des boutons de formulaire. En effet, comme cette balise peut contenir la plupart des éléments (X)HTML, les possibilités graphiques sont illimitées :

```
<button type="submit" value="boutonOk">
  <p> Ceci est un bouton</p>
</button>
```



Ok... Les exemples ne sont pas de très bon goût mais le but ici est de démontrer que la balise `<button>` nous donne toute la liberté de design, toujours en gardant une cohérence sémantique.

Tout comme la balise `<input>`, la balise `<button>` dépend de l'attribut `type` pour définir son rôle dans un formulaire :

- ◆ **type="submit"** : c'est la valeur par défaut de cet attribut pour la plupart des navigateurs (sauf pour, oui vous avez deviné... IE6). Si le bouton est du type "submit" il a la fonction de validation du formulaire
- ◆ **type="reset"** : fonction d'annulation
- ◆ **type="button"** : fonction neutre

Les balises du type "submit" ou "reset" sont naturellement interprétées par le formulaire, alors que le type "button" n'a pas d'action par défaut et va donc dépendre des attributs d'événement.

La balise `<button>` nous ouvre donc une grande palette de possibilités : la possibilité de garder la sémantique des éléments d'un formulaire de façon parfaitement accessible ainsi que des possibilités graphiques illimitées.

On en a rêvé, et on l'a eu : une balise parfaite, parfaitement intégrée tant du côté sémantique que du côté dynamisation.

Néanmoins, je suis désolée de vous décevoir en vous ramenant à la cruelle réalité : **IE6 existe** et il est encore à ce jour un des navigateurs les plus utilisés. Et IE6 va de nouveau anéantir nos espoirs, car il ne permet tout simplement pas d'utiliser la balise `<button>`.

Les bugs et le comportement non-standard de cette balise sous IE6 (voir sous IE7) sont nombreux et parfois incontournables, ce qui rend son utilisation impossible. Nous pouvons citer comme exemple :

- ◆ **La valeur par défaut non-standard** : d'après les recommandations de W3C la valeur par défaut de l'attribut `title` doit être "submit". Or, sous IE6 sa valeur par défaut est "button". Ce petit détail n'est pas très bloquant, car il nous suffit de toujours déclarer explicitement la valeur de l'attribut `title`. D'une autre côté, ce détail va demander une attention supplémentaire lors de la dynamisation des pages.
- ◆ **Le comportement non-standard** : au clic, un bouton créé avec une balise `<button>` doit se comporter exactement comme la balise `<input>`, c'est-à-dire, il doit renvoyer la valeur définie dans son attribut `value`. Avec IE6 ça n'arrive pas... Pour des raisons mystérieuses, IE6 va renvoyer la chaîne de caractères qui se trouve entre l'ouverture et la fermeture de la balise `<button>` ("Ceci est un bouton", dans notre exemple précédent). Ce problème est un peu plus grave que le premier bien sûr, et il invalide complètement l'utilisation de cette balise si on a besoin de récupérer la valeur du bouton.
- ◆ **Le bug** : Sous IE6 et sous IE7, quand un formulaire présente plus d'une balise `<button>` de valeur "submit", les valeurs de toutes ces balises vont être renvoyées en même temps et ceci indépendamment des clics effectués sur les boutons. Dans le cas où tous les boutons ont la même valeur pour l'attribut `name`, le contenu de la première balise bouton sera renvoyé; et la valeur renvoyée continue à surprendre, en dépendant du contexte. Il faut dire qu'on peut quand même se passer du fait d'avoir plusieurs boutons du type "submit" dans la plupart des cas, mais avec un comportement si changeant, nous devons à chaque nouveau formulaire prendre tellement de précautions et faire tellement de tests que cette balise devient inutilisable.

REFERENCES:

- **Référence de la balise <button> pour Internet Explorer :**
[http://msdn.microsoft.com/en-us/library/ms535211\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms535211(VS.85).aspx)
- **Un intéressant article à propos de l'utilisation de la balise <button> :**
<http://particletree.com/features/rediscovering-the-button-element/>

Ce qu'il faut faire

Nous avons pris la peine de vous la présenter parce qu'on garde toujours un grand espoir dans le futur du web ... Mais, oui vous avez compris, nous ne pouvons pas profiter des grands avantages que cette balise nous propose. Du moins pas sans avoir testé les différents comportements sous Internet Explorer.

L'option privilégiée sera donc les **boutons sous forme de balises <input>**. Néanmoins, à défaut d'une très claire décision lors du cadrage de projet pour une option ou pour une autre, **prévoir également la construction des boutons sous forme de liens**, en prévision du développement.

Ainsi lors du design CSS, le designer ou l'intégrateur pourra définir les styles nécessaires à ces deux types de boutons, laissant le choix quant à l'utilisation de l'un ou de l'autre aux équipes de développement.

5.15.9 Créer une structure pour les boutons des formulaires

Contexte

Comme pour les autres éléments d'un formulaire, la façon de structurer ses boutons est importante pour garder la cohérence et la logique de remplissage.

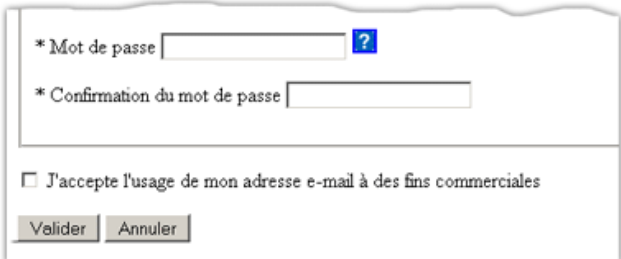
Prévoir la présentation et la structuration des boutons permettra une plus grande flexibilité pour le design CSS et facilitera la création d'un formulaire accessible.

Ce qu'il faut faire

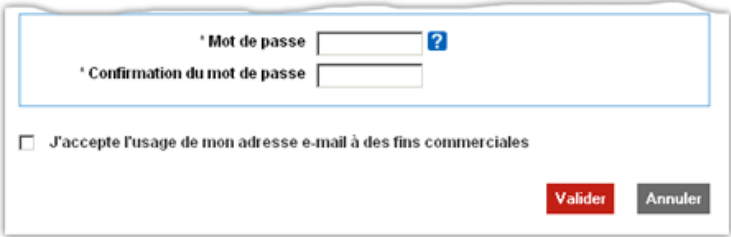
Pour les boutons <input> il vous suffit de les grouper dans un même paragraphe, identifié par une classe, parce qu'ils seront toujours affichés comme des boutons avec ou sans la prise en compte d'une CSS :

```
<p class="btsBar">
  <input type="submit" class="btValid" value="Valider">
  <input type="reset" class="btReset" value="Annuler">
</p>
```

sans la CSS →



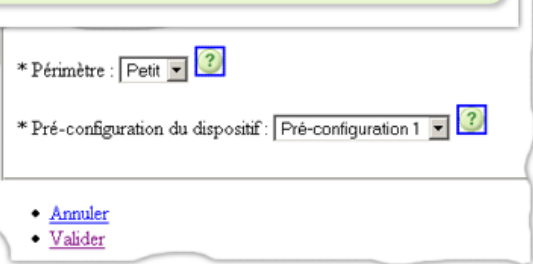
← **avec la CSS**



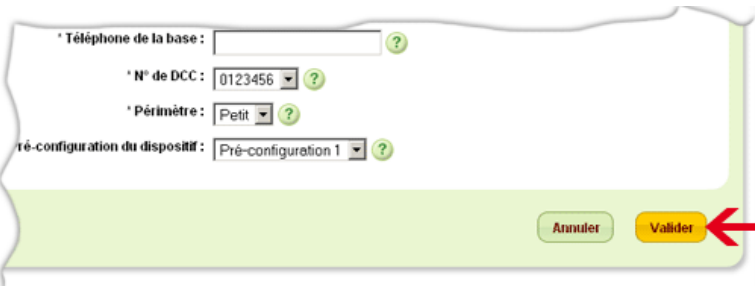
Dans le cas des liens, la meilleure option est de les structurer sous forme de liste, elle aussi identifiée par une classe. Ainsi, les différents liens seront proprement séparés :

```
<ul class="formBts">
  <li><a href="index.html" class="resetBt">
    <span>Annuler</span></a></li>
  <li><a href="mise_en_service.html" class="validBt">
    <span>Valider</span></a></li>
</ul>
```

sans la CSS →



← **avec la CSS**



Si les liens étaient construits dans un paragraphe, le résultat serait beaucoup moins structuré sans la CSS :

* N° de DCC : 0123456 ?

* Périmètre : Petit ?

* Pré-configuration du dispositif : Pré-configuration 1 ?

[Annuler](#) [Valider](#)

5.15.10 Coder correctement les *Checkboxes* et/ou *Radios*

De par leur structure de données unique, les éléments de formulaire de type "checkbox" et "radio" sont les plus délicats à coder et sont ceux qui demandent une plus grande flexibilité de construction et de design.

Les cases à cocher créent à elles seules la plupart des exceptions structurelles dans un formulaire et ces exceptions doivent être prévues lors de l'écriture du code, pour éviter des surprises pour le design CSS.

C'est avec les radios/checkboxes que le design est généralement plus contraignant également. Autant, lister des paragraphes avec leurs labels et leurs champs est une tâche simple, autant la structure des cases à cocher est particulière. Et leur construction peut devenir vraiment délicate si le design l'impose :

Vos informations personnelles

Je veux : ☐ danser la samba
☐ aller au Hawaï
☐ remplir ce super formulaire

Situation Civil ☐ marié ☐ divorcé ☐ célibataire

* Nom ⚠ Ce champ est obligatoire

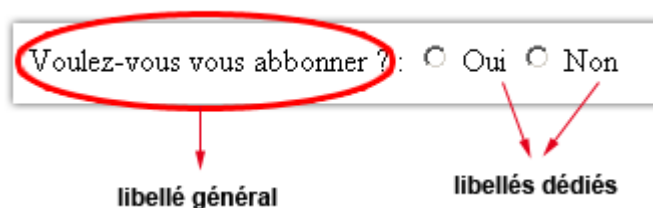
Du : au :

Tous les :

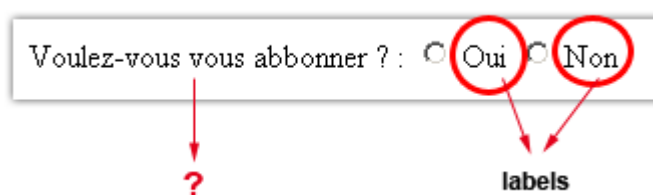
☐ lundi ☐ mardi ☐ mercredi
☐ jeudi ☐ vendredi ☐ samedi ☐ dimanche

Si on examine les exemples ci-dessus, on réalise rapidement que la même structure (X)HTML n'est pas possible pour tous les cas. Mais indépendamment des constructions que le design peut imposer, nous ne devons jamais perdre de vue la logique structurale et l'accessibilité du contenu.

Le premier point à considérer est le rôle des balises `<label>`. Dans le cas des cases à cocher, nous pouvons avoir un libellé général pour deux ou plusieurs cases à cocher, qui définit leur sujet, plus un libellé dédié pour chacune des ces cases :



Or, nous savons que la balise `<label>` doit être directement associée à sa balise `<input>`. Donc contrairement aux balises `<input>` de type texte, la balise `<label>` ne peut pas contenir le libellé général, mais doit contenir l'intitulé de chacune des cases à cocher :



Dans ce cas, deux questions s'imposent à nous : comment le libellé général va-t-il être associé aux cases à cocher auxquelles il fait référence ? et quelle balise est donc la plus correcte pour le libellé général ?

Une première solution va répondre à ces deux questions : en regroupant le libellé général avec les cases à cocher et ses libellés respectifs dans un même paragraphe, nous créons une liaison naturelle entre les différents éléments et nous donnons, par la même occasion, une structure sémantique au libellé général :

```
<p>Voulez-vous vous abonner ? :  
  <input type="radio" id="regY" name="register" value="yes" />  
  <label for="regY">Oui</label>  
  <input type="radio" id="regN" name="register" value="no" />  
  <label for="regN">Non</label></p>
```

C'est assez simple, non ? Une construction tout à fait correcte du point de vue sémantique, syntaxique et de l'accessibilité. Cependant, nous ne devons pas oublier le design CSS. Avec cette construction, nous n'avons pas beaucoup de possibilités de design. Notre libellé principal n'est pas identifié et il ne peut donc pas être manipulé par la CSS. Comment donner à cet intitulé, donc, les mêmes propriétés que les intitulés des autres champs ?

Nous ne sommes pas encore à la phase du design CSS, mais nous savons par anticipation que notre code est insuffisant pour obtenir les résultats graphiques souhaités.

Voyons le **design proposé** pour identifier les propriétés probables des intitulés de notre formulaire :

Vos informations personnelles

Voulez-vous vous abonner ? ☐ oui ☐ non

*** Nom**  Ce champ est obligatoire

*** Prénom**

*** E-mail**

Confirmez votre E-mail


- ils sont gras;
- ils sont alignés à droite;
- ils ont un certain espacement qui les séparent de leurs champs;
- ils sont positionnés dans la colonne de gauche, pendant que les champs sont alignés dans la colonne de droite.

Or nous savons bien qu'avec notre code nous serons incapables de reproduire le design proposé. Si on appliquait la feuille de styles à notre code, le rendu graphique serait plutôt :

Vos informations personnelles

Voulez-vous vous abonner ? ☐ ☐

oui non

*** Nom**  Ce champ est obligatoire

*** Prénom**

*** E-mail**

Confirmez votre E-mail

```
<form id="frCoords" method="post" action="">
  <fieldset>
    <h4>Vos informations personnelles</h4>
    <p>Voulez-vous vous abonner ? :
      <input type="radio" id="regY" name="register" value="yes" />
      <label for="regY">Oui</label>
      <input type="radio" id="regN" name="register" value="no" />
      <label for="regN">Non</label></p>
    <p><label for="ipPNom">* Prénom</label>
      <input type="text" id="ipPNom" name="ipPNom" /></p>
    <p><label for="ipEmail">* E-mail</label>
      <input type="text" id="ipEmail" name="ipEmail" /></p>
```

```
<p><label for="ipEmailC">Confirmez votre E-mail</label>
  <input type="text" id="ipEmailC" name="ipEmailC" /></p>
</fieldset>
</form>
```

Bizarre, non ? Mais nous n'avons pas besoin de trop réfléchir pour comprendre la raison de ce rendu chaotique : nous avons plusieurs balises `<label>` dans un même paragraphe, ces balises ne se trouvent pas là où on les attendait, mais elles héritent quand même des propriétés par défaut définies dans la feuille de styles.

La première chose à faire dans ce cas c'est "extraire" le libellé général du reste du paragraphe pour pouvoir le manipuler indépendamment. Et assurément, nous savons déjà que la balise `` nous donne justement la possibilité d'"extraire" ou d'identifier une chaîne de caractères dans un paragraphe. Dans ce cas aussi, elle s'applique parfaitement :

```
<p><span>Voulez-vous vous abonner ? :</span>
  <input type="radio" id="regY" name="register" value="yes" />
  <label for="regY">Oui</label>
  <input type="radio" id="regN" name="register" value="no" />
  <label for="regN">Non</label></p>
```

Mais nous savons aussi que le simple étiquetage avec la balise `` ne nous suffit pas car on ne peut toujours pas le différencier des autres balises `` qui peuvent apparaître dans la page.

Nous devons donc lui donner une classe ou un ID. Si on choisit d'attribuer un ID à ce ``, on sera forcément bloqué pour les autres cas similaires qui peuvent apparaître dans ce même formulaire. La bonne solution serait donc de lui attribuer une classe pour qu'elle puisse être utilisée plusieurs fois :

```
<p><span class="lbLike">Voulez-vous vous abonner ? :</span>
  <input type="radio" id="regY" name="register" value="yes" />
  <label for="regY">Oui</label>
  <input type="radio" id="regN" name="register" value="no" />
  <label for="regN">Non</label></p>
```

Nous avons choisi ici une classe de nom "**lbLike**" qui va nous indiquer que cette balise est "comme un label". De cette façon lors du design CSS, nous pourrions définir cette classe avec exactement les mêmes propriétés par défaut que la balise `<label>`, pour nous assurer que le rendu graphique sera identique dans les deux cas :

Vos informations personnelles

Voulez-vous vous abonner ? ☒ oui ☐ non

* Nom ⚠ Ce champ est obligatoire

* Prénom

* E-mail

Confirmez votre E-mail

Réfléchissons maintenant un peu aux balises `<label>` de notre construction. Comparons le design proposé pour ces labels avec les autres labels du formulaire.

Les intitulés de nos cases à cocher sont complètement différents des autres intitulés :

- ils ne sont pas gras;
- ils ne sont pas positionnés dans la colonne de gauche mais à droite des cases à cocher;
- ils ont un espacement à gauche entre eux et leurs cases à cocher.

Encore une fois nous devons étiqueter ces labels d'exception, pour plus tard pouvoir leur donner des définitions qui vont nous permettre d'obtenir le rendu graphique souhaité :

```
<p><span class="lbLike">Voulez-vous vous abonner ? :</span>
<input type="radio" id="regY" name="register" value="yes" />
<label for="regY" class="lbCheck">Oui</label>
<input type="radio" id="regN" name="register" value="no" />
<label for="regN" class="lbCheck">Non</label></p>
```

La construction avec des fieldsets

Une autre option de construction pour les cases à cocher est de les structurer dans un `<fieldset>` à part, en utilisant la balise `<legend>` pour le libellé général :

```
<fieldset class="checks">
  <legend>Voulez-vous vous abonner ? :</legend>
  <p><input type="radio" id="regY" name="register" value="yes" />
    <label for="regY" class="lbRadio">Oui</label>
    <input type="radio" id="regN" name="register" value="no" />
    <label for="regN" class="lbRadio">Non</label></p>
</fieldset>
```

Cette construction est d'ailleurs beaucoup plus élégante et donc préférable à la première. Malheureusement, à cause des problèmes déjà mentionnés avec la balise `<legend>`, cette option est très restrictive du point de vue design CSS et il nous serait impossible d'obtenir le rendu graphique ci-dessus.

5.15.11 Prévoir les éléments d'information fréquents dans un formulaire

Contexte

Comme nous le savons, un formulaire web est une zone d'une page web avec laquelle l'utilisateur peut interagir. Cette interaction peut être traduite par un dialogue : l'utilisateur "dit" quelque chose au site en entrant une valeur dans un ou plusieurs champs et en cliquant sur un bouton d'"envoi" et le site lui "répond".

La facilité avec laquelle ce dialogue va s'établir dépendra de son niveau de clarté, ainsi que du niveau et des possibilités de réponse du site.

Pour un bon niveau de clarté de contenu, un formulaire doit conjuguer la clarté des intitulés avec un ensemble d'éléments d'information qui vont guider ou aider l'utilisateur comme :

- **indication des champs obligatoires** : un des premiers éléments à prendre en compte est la mention des champs obligatoires. Il ne vous suffit pas d'ajouter un "*" devant l'intitulé du champ, mais il faut également penser à expliquer cet astérisque.
- **les commentaires de champs** : ces commentaires peuvent apparaître comme une explication de la donnée ou une aide au contexte;
- **les indications de type de données demandées** : elles vont montrer la "syntaxe" attendue pour un champ;
- **les icônes d'aide** : pour les champs qui demandent des données peu courantes, une explication plus détaillée va être nécessaire. Dans ce cas, nous devons ajouter une icône d'aide qui va permettre d'ouvrir un texte plus conséquent, sous forme de *tooltip*, *popup*, ou simplement dans une nouvelle page.
- **les calendriers** : pour les champs de date, une icône qui permet d'ouvrir un *tooltip* avec un calendrier pour le choix d'une date.
- **les informations d'erreur** : quand l'utilisateur commet une erreur de remplissage de son formulaire, le système doit l'avertir de l'erreur et si possible, indiquer où se trouve cette erreur. Ainsi, un ensemble d'informations d'erreur doivent être prévues. La façon dont les erreurs sont traitées par un formulaire peut varier, mais on doit au moins prévoir les informations d'erreur suivantes :
 - une zone de message d'erreur général
 - une indication du champ qui présente l'erreur
 - les messages d'erreur contextuels

Ce qu'il faut faire

Normalement, tous les éléments d'information d'un formulaire doivent être prévus lors de la phase de design et nous devons les inclure dans notre construction.

Quand vous "construisez" un formulaire, pensez à ajouter tous ces éléments :

The diagram illustrates a registration form with various annotations for error handling and required fields:

- Message d'erreur général**: Points to a general error message box at the top of the form.
- Indication des champs obligatoires**: Points to a label indicating required fields.
- champs obligatoires**: Points to a specific required field label.
- Message d'erreur contextuel**: Points to a contextual error message for a specific field.
- Commentaires du champ**: Points to a comment for a specific field.
- Champ qui présente l'erreur**: Points to a field that is currently displaying an error.
- icônes d'aide ou calendrier**: Points to icons for help or a calendar.

The form itself includes sections for "Vos informations personnelles" (Name, Email, Confirmation) and "Vos identifiants" (Username, Password, Confirmation). It also features checkboxes for newsletter and partner information, and buttons for "CONTINUER >" and "ANNULER".

5.16 Tableaux

Dans la philosophie de la séparation du contenu et de la forme, le tableau n'a bien sûr plus de fonction de " grille " de mise en page. Cependant, les balises de tableau sont toujours des balises tout à fait valides pour la mise en forme des données tabulaires.

Parce qu'utilisés longtemps pour la mise en page, les balises et attributs propres aux tableaux sont méconnus et souvent mal utilisés.

Dans les prochains points, nous allons essayer de mieux connaître ces balises et attributs pour la construction de tableaux bien structurés et parfaitement accessibles.

5.16.1 Utiliser la bonne balise pour le titre d'un tableau

Contexte

Nous avons souvent besoin d'identifier nos tableaux par un titre et on peut avoir le réflexe naturel d'utiliser une balise <h>, alors qu'il existe une balise spécifique pour les titres des données tabulaires.

Ce qu'il faut faire

Utilisez toujours la balise <caption> pour le titre d'un tableau. Cette balise doit toujours apparaître dans la balise <table> :

```
<table ...>
  <caption>Les frères Marx</caption>
  ...
</table>
```

5.16.2 Ajouter un résumé des données du tableau

Contexte

La lecture des données d'un tableau par les synthétiseurs vocaux est un processus compliqué et fastidieux qui peut facilement dérouter l'utilisateur.

En expliquant à l'utilisateur le contenu du tableau qu'il a devant lui, il va pouvoir considérer l'intérêt ou non de le consulter. Et ça c'est déjà un grand confort et un gain de temps, surtout si on lui donne la possibilité de "sauter" la lecture du tableau avec un lien caché (voir point n° 5.11).

Ce qu'il faut faire

L'attribut "summary" de la balise <table> peut être utilisé pour décrire rapidement le contenu et les types de données d'un tableau :

```
<table summary= "Ce tableau présente les dates de naissance et de  
décès des frères Marx, ainsi que les principales caractéristiques  
de chacun">
```

5.16.3 Utiliser la balise correcte pour les entêtes de colonne ou de ligne

Contexte

Un tableau peut avoir ses entêtes structurés horizontalement en faisant référence aux colonnes des données, ou alors verticalement en faisant référence aux lignes du tableau.

Dans tous les cas, l'important c'est que les en-têtes soient facilement identifiés comme tels tant du point de vue visuel que du point de vue structurel, pour faciliter la compréhension du tableau "lu" par un synthétiseur vocal.

Ce qu'il faut faire

Les entêtes d'un tableau doivent toujours être définis par la balise <th> (pour "table header"), quelle que soit leur orientation (verticale ou horizontale).

Dans l'exemple suivant les cellules en gris clair sont des entêtes :

Les frères Marx			
	Né en	Décédé en	Caractéristiques
Grouxo	1890	1977	Moustache, lunettes et cigare
Harpo	1888	1964	Muet
Zeppo	1901	1979	"Humain"
Chico	1887	1961	Chapeau et accent italien

5.16.4 Renseigner la logique de lecture du tableau

Contexte

Le plus délicat pour l'utilisateur d'un synthétiseur vocal quand il "écoute" un tableau, c'est de pouvoir "situer" une donnée et de savoir à quel en-tête elle fait référence. Ce point est primordial pour la compréhension du contenu du tableau.

Dans l'exemple de tableau précédent, quand l'utilisateur écoute par exemple, l'année "1977", il doit savoir à quel frère Marx cette date fait référence et si le tableau parle de la date de naissance ou de décès.

Comme un entête de tableau peut avoir une orientation horizontale ou verticale, nous devons indiquer cette orientation pour chaque entête si on veut que l'utilisateur puisse comprendre la logique structurelle du tableau.

Ce qu'il faut faire

Pour un tableau de données simples il nous suffit d'indiquer l'orientation des entêtes avec l'attribut "scope" :

```
<table summary="Ce tableau présente les dates de naissance et de
la mort des frères Marx, ainsi que les principales
caractéristiques de chacun">
<caption>Les frères Marx</caption>
<tr>
  <td></td>
  <th scope="col">Né le</th>
  <th scope="col">Mort le</th>
  <th scope="col">Caractéristiques</th>
</tr>
<tr>
  <th scope="row">Grouxo</th>
  <td>1890</td>
  <td>1977</td>
  <td>Moustache, lunettes et cigare</td>
</tr>
. . .
</table>
```

Dans l'exemple ci-dessus, pour les entêtes de la première ligne nous indiquons avec l'attribut "scope", que l'orientation est verticale, c'est-à-dire que les entêtes font référence aux colonnes et non aux lignes. Dans ce cas la valeur de "scope" doit être "col".

Pour les entêtes des lignes suivantes, comme ils font référence à la ligne sur laquelle ils se trouvent, la valeur de "scope" doit être "row".

5.16.5 Considérer les cas de figure possible (alignement cellules, lignes alternées, lien pour les entêtes, flèche de tri, etc.)

Comme pour les formulaires, les tableaux présentent une série d'éléments et de ressources visuels qui sont souvent utilisés pour faciliter la compréhension et la lecture des données.

L'alignement de cellules

Selon les recommandations d'utilisabilité du Studio relatives aux tableaux², l'alignement des colonnes d'un tableau doit se faire préférentiellement à gauche :

"Si les textes sont centrés ou alignés à droite, le début de ces textes ne sera jamais situé au même endroit (les uns sous les autres), puisqu'il est fort possible que les textes n'aient pas la même longueur. Cela est donc très fatigant pour les yeux, de devoir « chercher » le début du texte à chaque fois qu'il passe d'une ligne à l'autre."

L'exception est faite pour :

- les colonnes qui affichent des données monétaires : l'alignement devra se faire à droite
- pour les colonnes qui affichent des textes courts (maximum 20 caractères) ou des données de la même longueur (même nombre de caractères) : l'alignement devra être centré

L'alignement par défaut des cellules des tableaux doit donc être à gauche, mais nous devons prévoir deux classes pour les cellules qui affichent les données d'exception.

Comme toujours, essayez de définir de noms de classe sémantiques, comme "moneyData".

Les lignes alternées

Pour faciliter la lecture des données d'un tableau on adopte souvent un effet "pyjama", où une ligne sur deux du tableau change de couleur.

² Vous pouvez télécharger l'ensemble des recommandations d'utilisabilité du Studio sur <http://canalweb.atosworldline.com/FR/RUB/5069165/Charte-Application-Web.htm>

Pour ça, il suffit de prévoir dans votre code une classe qui doit être attribuée à une balise `<tr>` sur deux. De cette manière lors du design CSS, nous pourrons définir la couleur de fond pour toutes les balises `<td>` qui sont filles des ces `<tr>` :

```
<table summary="Ce tableau présente les dates de naissance et de
la mort des frères Marx, ainsi que les principales
caractéristiques de chacun">
<caption>Les frères Marx</caption>
<tr>
  <td></td>
  <th scope="col">Né le</th>
  <th scope="col">Mort le</th>
  <th scope="col">Caractéristiques</th>
</tr>
<tr class="alternate">
  <th scope="row">Grouxo</th>
  <td>1890</td>
  <td>1977</td>
  <td>Moustache, lunettes et cigare</td>
</tr>
<tr>
  <th scope="row">Grouxo</th>
  <td>1890</td>
  <td>1977</td>
  <td>Moustache, lunettes et cigare</td>
</tr>
. . .
</table>
```

Le tri des colonnes

Une des ressources importante à prévoir dans un tableau c'est le tri des colonnes.

Le tri suppose que les titres des colonnes "triabiles" soient bien sûr des liens, pour permettre la sélection du tri. Prévoir donc les balises de liens aux intitulés de colonne pour pouvoir plus tard définir le design de ces liens :

```
<th scope="col">
  <a href="#" title="Trier par cette colonne">Né le</a>
</th>
```


Remarquez que le lien de tri de l'intitulé présente un attribut "title" qui explique l'action du lien. Cet attribut doit toujours être présent pour expliquer la fonction du lien à l'utilisateur, car l'emploi du lien peut ne pas être clair pour tous. De plus, l'attribut "title" va nous aider à indiquer quelle est la colonne de tri actuelle.

La façon dont le tri des colonnes se fait doit, de préférence, suivre les principes suivants :

- les libellés des colonnes à trier sont cliquables;
- la colonne de tri par défaut, ou celle choisie par l'utilisateur, doit indiquer l'orientation du tri (ascendante ou descendante), généralement sous la forme d'une flèche qui pointe vers le haut ou vers le bas;
- par défaut, la première fois que l'utilisateur clique sur un intitulé pour trier une colonne, le tri se fera de façon ascendante (flèche vers le bas);
- si l'utilisateur veut inverser le tri de cette colonne, il lui suffit de cliquer une fois de plus sur son intitulé.

Liste des règles assignées					
Libellé	Type d'absence	Période de validité	Jour(s)	Heure de début	Heure de fin
Assignment quotidienne	Absence obligatoire	01/12/2008 au 01/12/2009	Toute la semaine	08h30	18h30

Ces principes posés, il faut que notre code prenne en compte le comportement attendu. Il nous faut donc prévoir les liens et les attributs "title" pour expliquer leurs actions. Mais aussi deux classes que l'on va attribuer à la cellule de l'intitulé pour "marquer" la colonne de tri active et son orientation :

```
<tr>
  <th scope="col" class="sortD">
    <a href="#" title="Tri descendant - Cliquer pour inverser le
tri">Libellé</a></th>
    <th scope="col">
      <a href="#" title="Trier par cette colonne">Type d'absence</a></th>
    <th scope="col">
      <a href="#" title="Trier par cette colonne">Période</a></th>
    <th scope="col">
      <a href="#" title="Trier par cette colonne">Jour(s)</a></th>
    <th scope="col">Heure de début</th>
    <th scope="col">Heure de fin</th>
</tr>
```

Dans l'exemple ci-dessous nous avons définie que le tri se fait pour la première colonne et que son orientation est descendante, en attribuant la classe "**sortD**".

Nous pouvons vérifier également dans l'exemple que la valeur de l'attribut "title" dans ce cas est "*Tri descendant – Cliquer pour inverser le tri*" qui indique à l'utilisateur quelle est l'orientation de tri et que la fonction de ce lien c'est d'inverser cette orientation.

Un élément qui n'apparaît pas dans notre exemple c'est justement l'image de la flèche. En effet, cette image sera appelée par la feuille de styles dans la définition des classes "**sortD**" et "**sortA**", comme une image de fond pour le lien de l'intitulé.

Notez que le choix d'afficher la flèche comme image de fond ou comme image présente directement dans notre page (X)HTML importe peu.

Comme cette image a une fonction informative (elle indique l'orientation du tri) et pas "décorative", elle a toute la légitimité pour apparaître sur la page.

La façon dont on va afficher la flèche va dépendre en grand partie du design, mais toutes les deux options sont bonnes. N'oubliez pas, bien sûr, de donner une alternative textuelle à l'image :

```
<tr>
  <th scope="col" class="sortD">
    <a href="#" title="Cliquer pour inverser le tri">Libellé
    </a></th>
    . . .
</tr>
```

Notez que nous avons modifié la valeur de "title" également : l'information d'orientation du tri est déjà donnée par l'image.

REFERENCES:

- **Designing Tables for Usability :**
<http://aenui.com/user-interface/designing-tables-for-usability/>
- **Creating Accessible Tables :**
<http://www.webaim.org/techniques/tables/data.php>

6 Le codage javascript

La position du Studio concernant le développement javascript est de s'impliquer uniquement dans la "manipulation d'interface" (par exemple, des actions comme plier/déplier, manipulation de menus déroulants, actions d'ouverture de popup, etc.).

Notre champ d'action se limite à la "promotion" et surtout au conseil pour le développement de javascript **standard** (DOM / EcmaScript), **accessible** et **non-intrusif**.

Dans ce chapitre, nous verrons brièvement les points les plus importants à tenir compte lors de la conception de nos scripts.

Le javascript "discret"

Nous ne connaissons pas d'interface web qui n'a pas un petit bout, même minime, de javascript. Dès qu'on parle du web le javascript est très utile et incontournable, à n'importe quel niveau d'application. C'est pour ça qu'il faut qu'on commence à le considérer pour sa vraie valeur et à lui donner un peu plus d'attention.

Cependant l'attention qu'on porte au javascript ne doit pas se limiter au langage lui-même, mais aussi aux bonnes pratiques, aux questions d'accessibilité et oui évidemment à la question de la **séparation contenu/comportement**.

Vous vous rappelez combien il est important de séparer le contenu de la forme en ce qui concerne la présentation de style et la mise en page, mais avez-vous songé que le comportement de la page (ou des portions de la page) n'a rien à voir non plus avec le contenu?

Les fonctions javascript ne sont pas du tout la même chose que le contenu, et pourtant on continue d'ajouter les définitions de fonction dans la page HTML, ainsi que de la polluer avec des "onclicks" et des "onsubmits"....

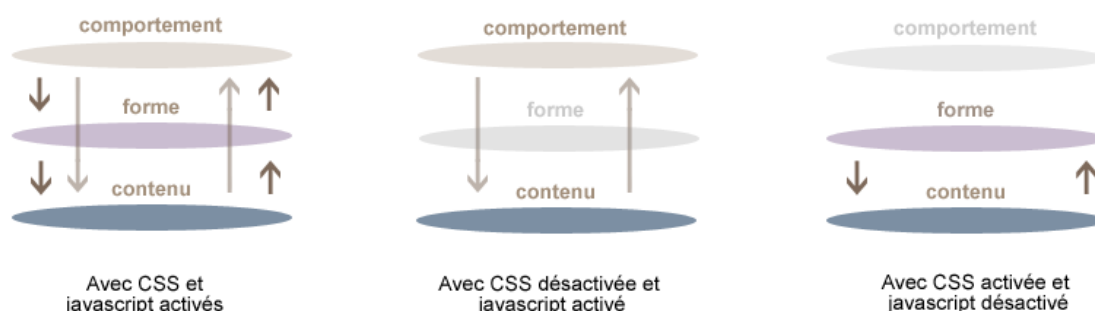
La solution pour ça c'est le javascript dit "**non-intrusif**" (pour "*unobtrusive*" en anglais, qui veut dire "**discret**").

La discrétion est accessible

Notre javascript discret profite de tous les avantages des standards du web et de la séparation contenu/forme pour donner à nos pages une grande liberté de développement et une importante flexibilité pour la gestion et la mise à jour des scripts.

Plus que le résultat ou la performance, c'est la méthode de création des scripts non-intrusifs qui donne une grande flexibilité de développement et, encore plus important, un plus grande contrôle sur leur accessibilité.

De la même façon qu'avec la séparation du contenu de sa forme, le fait d'affranchir la page et ses éléments des méthodes et des fonctions javascript nous laisse libre de tester et de modifier nos scripts tout en gardant l'intégrité du code (X)HTML original, sans poser d'obstacles à l'accès au contenu :



Dans le premier cas, si l'utilisateur accède au web via un navigateur récent et avec toutes les options activées, le contenu des pages visitées peut recevoir les "instructions" de styles, de mise en page, ainsi que de comportement (qu'est-ce qui arrive quand je clique sur ce bouton ?). De cette façon, il n'a pas seulement accès à la totalité du contenu de la page, mais également à un niveau riche d'expérience utilisateur.

Dans le second cas, tout ce qui est relatif à la "forme" de la page (mise en page, styles, etc) disparaît, mais le contenu et le comportement sont intacts : l'utilisateur peut toujours consulter la totalité du contenu de la page ainsi qu'exécuter toutes les actions disponibles. L'interaction qui peut exister entre les scripts et la feuille de styles s'annule simplement, sans qu'il y ait de perte de fonctionnalité.

Dans le dernier cas, tout ce qui est relatif au comportement de la page disparaît, mais le contenu et sa forme sont intacts : la feuille de styles continue à mettre en avant la structure et l'ambiance de la page et son contenu est toujours disponible.

Bien sûr, on doit bien comprendre que ce n'est pas l'utilisation pure et simple de la technique non-intrusive qui nous permet d'avoir ce contrôle sur l'accès au contenu, mais plutôt la philosophie du "non-intrusif" qui nous mène à considérer l'intégrité de chacune des tranches qui composent une page (contenu, forme, comportement).

Un exemple pratique : supposons qu'on souhaite ajouter un comportement spécifique à un lien. Notre code (X)HTML d'origine est :

```
<a href="mapage.html">ceci est un lien</a>
```

Vite fait (et mal fait), on remplacerait simplement le nom de la page par l'appel à la fonction js :

```
<a href="javascript: mafonction();">ceci est un lien</a>
```

Ok, on est déjà passé par cette phase et compris que ce "raccourci" peut être mal interprété par certains navigateurs et que c'est loin d'être très élégant ou flexible comme approche. On va donc ajouter la méthode "onclick" comme suit :

```
<a href="#" onclick="javascript: mafonction();">ceci est un lien</a>
```

Ca peut paraître plus "adapté" ou correct mais ce n'est ni l'un ni l'autre : du point de vue de la sémantique, un lien qui nous envoie nulle part est incorrect et ne nous sert à rien. D'un autre côté, du point de vue de l'accessibilité, l'utilisateur qui n'a pas de javascript est dans une impasse. Il peut cliquer autant de fois qu'il veut sur ce lien, rien ne va se passer.

Evidemment on peut quand même penser à lui lors de l'écriture de notre fonction afin de pouvoir l'envoyer quelque part quand même (pour qu'il ait envie de revenir sur notre site éventuellement un jour...). On laissera donc la bonne valeur de l'attribut href :

```
<a href="mapage.html" onclick="javascript: mafonction(); return false;">ceci est un lien</a>
```

C'est déjà beaucoup mieux : même sans javascript, il peut aller sur "mapage.html" et même s'il perd en confort ou en facilité, il a accès au contenu. Mais notre beau code tout propre et tout autosuffisant présente maintenant un "intrus" : le contrôle d'événement "onclick". Est-ce bien nécessaire ? Non, parce qu'on est libre de ne pas mélanger la "tranche contenu" de notre site avec le javascript non-intrusif.

Tout ce dont on a besoin c'est de pouvoir identifier ce lien-là parmi les autres dans notre code (c'est bien probablement déjà prévu par le design et donc par le code) :

```
<a href="mapage.html" class="specialLink">ceci est un lien</a>
```

On peut donc écrire notre fonction (dans un fichier js externe !), pour qu'elle aille chercher tous les liens qui ont la classe "specialLink" et pour qu'elle applique la fonction souhaitée quand ces liens seront cliqués.

Je sais ce que vous êtes en train de penser à cet instant : "Bof, on a quand même été obligé d'ajouter une classe uniquement pour ça !...." Mais non ! Car il ne faut pas oublier que, si ce lien-là déclenche une action spécifique, ça veut dire qu'il est différent et s'il est différent par son action, il sera sûrement différent par son apparence ou par sa localisation dans la page. On peut donc se dire que certainement le designer aura pris ça en compte ainsi que celui qui a écrit le code (X)HTML, et que ce lien aura une classe spécifique ou id unique lors de l'intégration des pages.

Ca peut ne pas être toujours le cas, je l'admets. Mais l'ajout d'une classe ou d'un id reste sémantiquement correct.

L'exemple est simple, mais il montre bien la philosophie du non-intrusif. Si cette méthodologie était appliquée on commencerait tout de suite par cette dernière ligne d'exemple, c'est-à-dire par vérifier si notre lien a une classe ou un id qui nous permet de l'atteindre puis d'ajouter une classe/id si ce n'est pas le cas et écrire notre fonction en conséquence. Sans s'en rendre compte, on a déjà fait un grand pas vers l'accessibilité.

REFERENCES:

- **Très bon didacticiel pour le javascript non-intrusif :**
<http://onlinetools.org/articles/unobtrusivejavascript/chapter1.html>
- **Excellent article à propos de jQuery et codage Javascript : "jQuery and Javascript Coding : Examples and Best practices" :**
<http://www.smashingmagazine.com/2008/09/16/jquery-examples-and-best-practices/>
- **Les sept règles du javascript non-intrusif :**
<http://icant.co.uk/articles/seven-rules-of-unobtrusive-javascript/>
- **Javascript non-intrusif – règles à suivre :**
<http://ajaxian.com/archives/unobtrusive-javascript-rules-to-work-by>

6.1.1 Ne pas définir une fonction directement dans la page

Contexte

Pour que la séparation contenu/comportement soit effective, nous devons trouver dans nos pages (X)HTML uniquement du contenu (X)HTML, et rien d'autre.

Si nos fonctions javascript sont définies dans notre page, nous risquons une maintenance et un débogage délicats, sans mentionner une éventuelle duplication inutile de ces définitions.

Imaginez une même fonction utilisée dans deux pages ou plus. Les octets ajoutés à une page vont être inutilement dupliqués pour la deuxième page. Et lors d'un débogage, nous allons devoir dupliquer les efforts de corrections.

Créer un fichier javascript à part, permet non seulement de centraliser de façon contrôlée nos scripts, mais aussi de les concevoir et les commenter correctement, pour une performance efficace.

La dispersion de fonctions javascript dans plusieurs pages est contreproductive et demande une maintenance compliquée.

Ce qu'il faut faire

Toute fonction javascript doit être définie dans un fichier externe, appelé avec la balise `<script>` :

```
<script src="js/general.js" type="text/javascript"></script>
```

Notez que l'attribut "type" est obligatoire, contrairement à l'attribut "language", qui est dépréciée.

6.1.2 Séparer le comportement du contenu

Contexte

Comme nous avons vu précédemment, plus on affranchit notre contenu web de son comportement, plus facilement on va le contrôler, le maintenir et le faire évoluer.

Ce qu'il faut faire

Utilisez autant que possible la méthode non-intrusive de script.

6.1.3 Donner toujours une alternative au javascript

Contexte

En soit, le javascript n'est pas un blocage pour l'accessibilité. Il a une place tout à fait légitime dans un site que se dit accessible, c'est-à-dire, le fait qu'il soit présent sur une page web ne veut pas dire que cette page est inaccessible.

Comme dit Christian Heilmann, dans son billet "*The seven rules of Unobtrusive Javascript*": le javascript doit être appliqué comme un "plus" pour l'expérience utilisateur, mais jamais en ajoutant une contrainte à ceux qui ne peuvent pas – ou qui ne veulent pas – le supporter.

Nous pouvons utiliser le javascript dans deux contextes différents :

- **pour créer des effets ou des repères visuels "cosmétiques"** comme, par exemple, des effets de "rollover" sur une image : dans ce contexte, l'absence de javascript ne met pas en cause l'accès au contenu. L'utilisateur qui n'a pas de javascript, ou qui a l'option désactivée, peut lire et comprendre le contenu de la

page, mais il n'a pas l'effet visuel de survol. Dans ce cas, l'absence d'une alternative au script n'est pas bloquante.

- **pour déclencher une action ou pour le traitement des données :** dans ce contexte, l'absence de javascript est une contrainte importante pour l'utilisateur, et il lui faut obligatoirement une alternative.

Ce qu'il faut faire

Il n'existe pas de recette miracle pour créer des alternatives à nos scripts. Nous devons étudier, au cas par cas, quelles sont les meilleures solutions.

Dans tous les cas, pensez à toujours tester vos pages avec le javascript désactivé pour évaluer les impacts et pour identifier plus facilement les aspects bloquants. C'est seulement ainsi que vous allez pouvoir traiter correctement chaque alternative.

6.1.4 Penser les fonctions pour qu'elles ne soient pas dépendantes d'un seul type d'input ou de navigateur

Contexte

Quand nous parlons de javascript, nous devons penser l'accessibilité à deux niveaux :

- l'accès au contenu et aux fonctionnalités traités par le script;
- l'accès au script lui-même.

Par "accès au script lui-même" nous voulons dire "le niveau d'accès que le script offre aux utilisateurs". Par exemple, si un javascript a été conçu et testé uniquement avec IE6, les utilisateurs des navigateurs plus récents vont sûrement rencontrer des problèmes, invalidant l'accès à ce script.

De la même façon, si notre script dépend uniquement des contrôles par la souris, l'utilisateur qui navigue par clavier ne va pas avoir accès à ces contrôles.

Ce qu'il faut faire

- Les scripts doivent être conçus de façon standard, en respectant les recommandations de W3C;
- Les scripts doivent être conçus de façon à pouvoir détecter les idiosyncrasies de chaque navigateur, pour pouvoir gérer leurs limitations;
- Les scripts doivent être conçus en utilisant des attributs d'événement indépendants du type d'input;

- Les scripts doivent être testés sur les navigateurs les plus utilisés;
- Les contrôles des scripts doivent être testés avec accès par la souris ou par le clavier.

6.1.5 Utiliser la syntaxe correcte pour le passage des variables par une URL

Contexte

Dans les langages SGML et XML, le caractère "&" est utilisé dans la syntaxe d'une "référence d'entité". Une "référence d'entité" est le code qui remplace un caractère spécial : `´` pour le caractère spécial "é", par exemple.

Depuis toujours, les agents utilisateurs ignorent l'utilisation incorrecte de "&" dans les documents **HTML**. Cependant, un document écrit en **XHTML** ne validera pas si ce caractère est utilisé ailleurs que dans les références d'entité.

Or, nous passons souvent des variables par une Url, en utilisant des "&" pour les grouper, sans nous soucier si notre code XHTML va être invalidé.

Ce qu'il faut faire

Ce problème peut être facilement évité si nous remplaçons le caractère "&" justement par sa référence d'entité : `&` :

```
<a href="http://www.monsite.fr/js/monscript.js?nom=Cruz&amp;prenom=Jose">  
Mon lien</a>
```

REFERENCES POUR LE JAVASCRIPT ACCESSIBLE :

- **Très bonne référence à propos de l'accessibilité avec javascript**
WebAIM : Creating Accessible Javascript :
<http://www.webaim.org/techniques/javascript/>
- **Excellente source pour les tests de compatibilité entre navigateurs et très bonne référence javascript : QuirksMode de Peter Paul Koch :**
<http://www.quirksmode.org/>



www.atosorigin.com