Documents structurés pour le *web*

Yannis Delmas

Documents structurés pour le *web*

Yannis Delmas

Date de publication 2011 / 09 / 20

Table des matières

Chapitre 1. Avant-propos

1. Objectifs

Ce document présente les principaux formats de documents textuels structurés en usage sur le *web*, principalement HTML, XHTML, XML et les feuilles de style CSS. Nous nous concentrerons d'abord sur ce qu'est une page *web* standard et donnerons les concepts sous-jacents. L'objectif de cette première partie n'est pas de produire une page *web* par le code mais de pouvoir s'y repérer. Ce cours abordera ensuite de façon plus précise la structure XML, afin d'en préciser les tenants et aboutissants. Nous terminerons par une étude précise des feuilles de style CSS pour XML et (X)HTML.

Ce document n'est qu'un support de cours, il ne prétend pas être autonome. Le cours correspondant est délivré dans le cadre du mastère *Web éditorial* de l'Université de Poitiers [http://www.univ-poitiers.fr/]. Il ne couvre exactement tous les points indiqués ici. Ce document a été conçu au format DocBook [http://docbook.sourceforge.net/] pour être consulté sous forme de pages web (XHTML) ou sous forme imprimée (PDF).

2. Prérequis

Il est préférable que les compétences suivantes soient maîtrisées avant d'aborder ce cours :

- Utilisation d'un logiciel de traitement de texte, avec usage des styles pour la mise en forme,
- Navigation ordinaire sur le web et maîtrise de la notion d'adresse *web* (URL).

3. Statut de ce document

Ce document a été réalisé à destination des étudiants du master *Web éditorial* [http://sha.univ-poitiers.fr/masterweb/] de l'Université de Poitiers [http://www.univ-poitiers.fr/]. Il est mis à disposition de tous. Il est d'usage gratuit dans les cas de formations délivrées par un organisme à but non lucratif (institution gouvernementale, association). En cas de formation dans un cadre à but lucratif ou pour tout autre usage, merci de prendre contact avec l'auteur [http://yannis.delmas-rigoutsos.nom.fr] ou ses ayant-droits.

Chapitre 2. Introduction : genèse du HTML

1. Les objectifs de conception du *web*

Tim Berners-Lee invente ce qui sera le *web* au sein du CERN (Centre européen pour la recherche nucléaire), ceci n'est pas sans importance. Le CERN est un centre de recherche où collaborent des scientifiques de nombreuses nationalités organisés en équipes de recherches de taille considérable. T. Berners-Lee conçoit donc le *web* comme un *système d'information distribué hypermédia*, dans un contexte très particulier, marqué d'abord par une immense quantité d'information à partager entre un grand nombre d'acteurs. Ce contexte sera à l'origine de nombreux objectifs de ce support de publication.

Passons en revue ces objectifs. Nous en déduirons les principales caractéristiques de la définition du *web*.

1.1. Désignation universelle

Une désignation universelle des ressources est nécessaire pour se repérer dans un système d'information distribué à l'échelle de la planète.

1.2. Possibilités graphiques et multimédias

Les traitements de texte disposaient, à l'époque, de possibilités étendues de mise en forme, de mise en page et d'inclusion d'éléments non-textuels. Ceci devait être possible dans le nouveau support de publication. Ceci était d'autant plus nécessaire que les articles scientifiques utilisent abondamment ces fonctionalités.

1.3. Hypertextualité

L'information scientifique présente, par nature, à la fois une forte intrication et une large fragmentation. Elle bénéficie considérablement du croisement intertextuel. Une certaine forme d'hyper-référence existait déjà dans le système Gopher (cf. cours d'histoire). L'hypertextualité était, par ailleurs, déjà en usage dans certains textes imprimés. On voit maintenant à quel point cette fonctionnalité est au cœur même du *web*.

On dit que le web est hypermédia : il permet le multimédia et l'hypertextualité.

1.4. Interopérabilité et ouverture

L'interopérabilité est le fait, pour un service d'application (logiciels ou données) de pouvoir être utilisé dans de nombreux contextes, indépendamment du matériel ou du système d'exploitation. Ceci est essentiel dans la communauté scientifique puisque celle-ci utilise un parc très hétérogène, y compris au sein d'un même laboratoire.

Le frein habituel à l'interopérabilité est la défense d'intérêts commerciaux. Dans le cas du *web*, ce problème ne se pose pas du fait du choix d'utiliser des normes et standards ouverts, selon l'usage en vigueur au sein de la communauté scientifique.

1.5. Compatibilité ascendante *et* descendante

La compatibilité ascendante est nécessaire partout. La demander n'est pas original : il est essentiel de pouvoir continuer à utiliser un document, sans avoir à conserver d'anciens logiciels ou à remettre

constament les documents à jour. Notons quand même qu'elle n'est qu'imparfaitement assurée par les éditeurs de logiciels, même pour des applications bureautiques usuelles.

La grande nouveauté, dans le cas du *web* est d'aller plus loin et de demander une certaine compatibilité descendante : un document réalisé avec un logiciel récent doit pouvoir être lu par un logiciel plus ancien. Une stricte logique commerciale conduit la plupart des éditeurs à éviter soigneusement cette propriété. Dans le cas du *web*, conçu sans but lucratif, on n'a aucun intérêt à imposer de force un changement régulier de logiciel.

1.6. Limitation des transferts de données

L'objectif est de limiter raisonnablement le transfert de données pour une simple raison d'économie de moyens. Par contre, il ne s'agit pas, en soi, de limiter la quantité d'information en transit.

1.7. Modularité

Comme toujours, la modularité est une bonne pratique, en informatique comme ailleurs, qu'on s'attachera à mettre en œuvre.

2. Les choix pratiques effectués

La modularité, c'est séparer les tâches et ne pas s'amuser à regrouper ce qu'il n'est pas fonctionnellement nécessaire de regrouper.

Ainsi, les formats de fichiers cherchent souvent à éviter la redondance et cherchent un maximum de compacité. Ne serait-ce que pour gagner du temps de téléchargement. Ceci est parfois inutile, voire nuisible, en termes de modularité : les modems individuels se chargent déjà de compresser les paquets de données. Si l'on veut gagner de la place, il vaut mieux évacuer toute considération d'économie de bouts de chandelle des formats de fichier et se réserver la possibilité de demander une compression des données par le serveur *web* (éventuellement *via* une extension). On répartit ainsi les tâches : le format de fichier doit remplir sa tâche de bonne représentation (selon les 5 premiers points listés ci-dessus) et l'éventuel compresseur se charge de la compacité des données, *s'il y a lieu*. On pourra quand même retenir le point numéro 6 sous la forme : on veut éviter de télécharger plusieurs fois la même chose.

Second problème : le multimédia. Comment représenter les données non-textuelles ? Il n'y a guère que deux grands types de méthodes : soit on se limite à certains types de données, soit on délègue à des extensions spécialisées. Ici aussi, la bonne méthode est la modularité. Le point important est donc, du point de vue de la page *web*, de décrire correctement le format des données : pour cela on utilisera les types MIME. Chaque format de données doit être enregistré dans le catalogue MIME (point 1). Le catalogue est enregistré à l'IANA [http://www.iana.org/] sous le nom *Media Types Directory* [http://www.iana.org/assignments/media-types/].

Toujours pour ce qui concerne le point 2, même si une image apparaît dans une page, elle constitue souvent, par ailleurs, une forme documentaire autonome. Ainsi on peut demander à consulter une image indépendamment d'une page qui peut la contenir. D'autre part, il y a des images qui reviennent régulièrement dans les pages *web* d'un même site et qu'il serait absurde de recharger systématiquement : les éléments du décors, logos, règles, fonds, etc. Les données non textuelles ne seront donc pas stockées dans la page web mais dans des fichiers annexes. La page web se contentera de "dire" : je veux ici une image, on utilisera tel fichier à tel format.

Pour en finir avec le point 1, désignation de partout d'un fichier ou d'un élément à l'intérieur d'une page : voir le système des URI/URL (on y reviendra dans un autre cours).

On réalisera les compatibilités ascendante et descendante par la méthode de structuration de texte, détaillée dans ce cours, qui consiste à "attribuer" des "propriétés" à des "portions" d'un document. Si une propriété donnée est connue du logiciel de lecture, on en tient compre, sinon on l'ignore. Cette méthode permet également de régler élégamment la question de l'hypertextualité : un hyperlien ne sera qu'une propriété comme une autre des portions de document.

La question de l'ouverture du format (point 4) fut réglée en utilisant un cadre standard ouvert déjà existant : le SGML, simplifié pour l'occasion. Il sera, plus tard, remplacé par XML, nous y reviendrons.

Chapitre 3. HTML et les hypertextes

1. Le texte structuré

Le principe de la structuration de texte consiste à partir d'un texte brut, “au kilomètre”, et à attribuer à certaines de ses portions des “types” ou “qualités”. Ces “qualificatifs” peuvent aussi bien être sémantiques que d'aspect, au choix de l'auteur, comme dans les deux exemples suivants.

Exemple 3.1. structuration grammaticale

phrase[sujet[Le chat]sujet verbe[mange]verbe complément[la souris]complément .]phrase

Exemple 3.2. balisage de présentation

citation[italique[Le malade imaginaire]italique de gras[Molière]gras c'est aussi... [à la ligne] l'imaginaire malade.]citation

Un tel procédé, que l'on appelle balisage (*markup*), quand il est systématique, permet aussi bien une compatibilité ascendante que descendante, et même une certaine compatibilité avec les logiciels non prévus pour comprendre les “types” et “qualités” utilisés dans un document donné. Ainsi, si un afficheur ou un éditeur ne comprend pas l'indication « citation » dans l'exemple ci-dessus, il n'en tiendra simplement pas compte alors qu'il pourra rendre correctement l'« italique » et le « gras ».

Plusieurs systèmes de structuration de texte existaient à l'époque de la création du *web*. Le plus répandu était le RTF (*Rich Text Format*, format de texte enrichi), utilisé par les traitements de texte, notamment *Microsoft Word*, comme format d'échange. Sans y insister, nous signalerons qu'il ne répond qu'à peu des exigences ci-dessus. Le second plus répandu, largement utilisé dans la communauté scientifique, était LATeX. Son principal défaut est sa puissance : rien ne le distingue fondamentalement d'un langage de programmation. En conséquence, sa spécification et la rédaction des logiciels auraient été particulièrement ardues. Troisième cadre possible : le SGML, *Standardized Generalized Markup Language* (langage général et standardisé de balisage). Bien qu'étant, lui aussi, très complexe, il était, comme son nom l'indique, standardisé et relativement limité pour ce qui concerne les documents textuels. Autre avantage : il avait été retenu par la communauté scientifique du traitement automatique des langues et par certains grands éditeurs comme le langage standard de balisage de textes. En particulier SGML répondait à une norme de l'ISO (*International Organization for Standardization*) : ISO 8879. C'était le candidat idéal.

2. HTML et les langages de balisage

En SGML, puis en XML, qui est d'une certaine façon son successeur, les “types” et “qualités” de portions de textes sont indiqués à l'aide d'un marquage (*markup*) à l'aide de balises (*tag*). Ces balises sont ajoutées au texte proprement dit, entre chevrons. Ainsi, les exemples précédents pourraient être codés de la façon suivante (en XML).

Exemple 3.3. structuration grammaticale

```
<phrase><sujet>Le chat</sujet> <verbe>mange</verbe>
<complement>la souris</complement>.</phrase>
```

Exemple 3.4. balisage de présentation

```
<q><i>Le malade imaginaire</i> de <b>Molière</b> c'est aussi...
<br />l'imaginaire malade.</q>
```

Dans ces exemples, nous observons les trois types de balises :

- Des balises ouvrantes, par exemple « début de gras », `<b>`, où le type est indiqué entre chevrons,
- Des balises fermantes, par exemple « fin de gras », `</b>`, où ce même type est précédé d'une oblique,
- Une balise ouvrante et fermante, `<br/>`, qui indique un saut de ligne.

En XML, à chaque balise ouvrante correspond une et une seule balise fermante. La portion de document entre ces deux balises est appelée la *portée* de ce marquage. Le texte brut présent dans cette portée est appelé *contenu textuel* du marquage. En SGML, certaines balises fermantes pouvaient parfois être sous-entendues (dans certains cas bien identifiés).

Les balises qui sont à la fois ouvrantes et fermantes ont une portée vide, sans contenu textuel. Il s'agit le plus souvent de sauts de ligne ou d'images. Les normes les plus récentes (XHTML et, plus généralement, XML), prévoient qu'elles soient indiquées `<nom />`, pour éviter toute confusion avec les balises ouvrantes non fermantes. Les anciennes normes (SGML) notaient simplement `<nom>`. Il est conseillé d'utiliser systématiquement la forme complète `<nom />`.

Les balises ouvrantes peuvent comporter des attributs précisant ou qualifiant leur rôle. Par exemple, en HTML, la géométrie d'une cellule de tableau ou le fichier source d'une image :

Exemple 3.5. exemple d'attributs d'une balise

```
<img src="logo_ACME_XS.jpeg" alt="ACME SARL" />
```

Certains de ces attributs sont obligatoires : le fichier source d'une image, par exemple. D'autressont facultatifs : la fenêtre de destination d'un lien, par exemple.

Dans tous les cas, la règle d'or est que les portées des balises s'incluent ou se côtoient mais ne se chevauchent jamais. Le contenu d'une portion de document SGML, et en particulier de HTML, se présente donc comme un arbre. On peut visualiser cette structure d'une façon similaire à l'arborescence des fichiers sur un disque dur.

On le voit, ces systèmes de balisage sont des codes, des langages. On trouvera donc deux modes d'édition des documents *web* : soit de l'édition “dans le code”, soit de l'édition sous une forme proche de la forme finale, telle qu'elle sera visualisée dans les navigateurs. Ce second mode est appelé WYSIWYG : *What You See Is What You Get* (ce que vous voyez, c'est ce que vous obtiendrez).

Le texte des pages *web* peut utiliser différents alphabets, différents encodages de polices de caractères, nous y reviendrons ultérieurement. Toutefois, comme jusque récemment l'encodage des caractères était souvent anarchique d'un système d'exploitation à un autre, voire sur un même système, et comme, de plus, nombre de communications ne se faisaient qu'avec sept bits par caractère, ne retenant que les caractères du code ASCII (qui ne contient que des caractères latins sans accent), le HTML prévoit un codage des caractères spéciaux et autres lettres accentuées. Par ailleurs, il était impératif de pouvoir coder les caractères qui permettent de représenter les balises. Pour cette raison, le HTML utilise, en plus des balises, la méthode des *entités* pour coder les caractères non-ASCII. Voici quelques exemples :

Tableau 3.1. entités HTML usuelles

entité	explication	valeur
`&lt;`	*less than* (plus petit que)	<
`&gt;`	*greater than* (plus grand que)	>
`&amp;`	*ampersand* (esperluette)	&
`&laquo;`	guillemet français ouvrant	«

entité	explication	valeur
`&raquo;`	guillemet français fermant	»
`&eacute;`	e accent aigu (*acute*)	é
`&egrave;`	e accent grave	è
`&ecirc;`	e accent circonflexe	ê
`&euml;`	e tréma (*umlaut*, en allemand)	ë
`&ccedil;`	c cédille	ç
`&euro;`	euro	€
`&copy;`	*copyright*	©

Ainsi, le code contiendra « `0 < 1` » pour représenter « 0 < 1 ».

3. HTML et structuration logique

3.1. Principes

Dans toute sa généralité (qu'on n'a fait ici que suggérer), le SGML offre trop de choix, en particulier, il ne limite ni ne structure assez les balises utilisables. Pour cette raison, on définit des *types de documents*, sortes de sous-formats du SGML, précisant quelle balise peut apparaître à quel endroit d'un document. Ces types sont aussi appelés *applications de SGML*. HTML est ainsi une application de SGML qui impose, par exemple, qu'une cellule de tableau (balise `<TD>`) n'apparaisse que dans une ligne de tableau (balise `<TR>`), laquelle, elle-même, ne doit apparaître que dans un tableau (balise `<TABLE>`). Un tel type de documents, ou application, est décrit, en SGML, sous la forme d'une DTD, *Document Type Definition* (définition de type de document).

Pour les pages *web*, le type de documents aujourd'hui le plus courant est le HTML, *HyperText Markup Language* (langage de balisage pour les hypertextes). Il existe plusieurs versions successives de la DTD. La dernière, spécifiée par le W3C, est la version 4.0.1 (la version 5 est en cours de finalisation). Toutefois, il faut savoir que HTML, *stricto sensu*, n'est pas tout à fait compatible avec XML. HTML 4 a donc été réécrit sous une forme à la fois plus stricte et plus extensible appelée XHTML 1.0 (*eXtensible HyperText Markup Language*, langage extensible de balisage pour les hypertextes). Il était (alors) prévu d'abandonner progressivement le HTML en faveur du XHTML. Depuis, la doctrine du W3C a changé radicalement et il est désormais convenu de conserver et la syntaxe du HTML (plus souple, bien adaptée aux navigateurs, parfois peu regardants sur l'exactitude de l'interprétation) et la syntaxe du XHTML, plus stricte, bien adaptée aux situations très formalisées (programmation, livres électroniques). La version 5 du langage existe donc sous les formes HTML 5 et XHTML 5.

Ce que nous dirons dans ce chapitre concernant HTML s'appliquera en fait largement au XHTML (sauf indication contraire) et nous écrirons généralement nos exemples sous une forme compatible avec XML (et XHTML, donc). Notons qu'en HTML les noms des balises et de leurs attributs peuvent être écrits indifféremment en minuscule ou en majuscules. En XHTML seules les minuscules sont utilisées. Dans les deux cas, des espaces et des sauts de ligne peuvent être utilisés dans les balises entre les attributs, les chevrons et le nom, et hors des balises entre les différents mots. Leur nombre, toutefois, n'importe généralement pas : ils servent juste de séparateurs entre mots ou éléments.

3.2. Éléments blocs

Le principe de base du HTML est de structurer le document de façon logique et de ne pas se contenter d'indications de nature visuelle. Ainsi les titres de premier niveau, par exemple, sont-ils désignés comme tels (balise `<h1>`) et non seulement comme du texte gros et gras aligné à gauche. On effectue ainsi une séparation des tâches entre la structure logique et sa présentation (le rendu visuel). Nous verrons que ce rendu devra être réalisé (principalement) au moyen de feuilles de styles. Les principaux niveaux logiques de texte sont les suivants en HTML 4 et 5 :

Tableau 3.2. niveaux logiques de texte en (X)HTML

balise	signification	Remarque
`<body>`	le document entier	
`<section>`	section du document	HTML 5, généralement un titre suivi d'un contenu
`<nav>`	section de navigation	HTML 5, interface : liens vers (reste du site ou portions du document)
`<article>`	section autonome	HTML 5, article (billet, brève...) ayant un sens en soi
`<aside>`	section accessoire	HTML 5, encadré, complément
`<h1>`	titre de premier niveau	titre du document ou nom du site
`<h2>`	titre de second niveau	niveau de titre immédiatement inférieur à `<h1>`
`<h3>`	titre de 3^{e} niveau	niveau de titre immédiatement inférieur à `<h2>`
`<h`$_n$`>`	titre de n^{e} niveau (jusqu'à 6)	
`<p>`	paragraphe	

Insistons que ces éléments renvoient à la structure du contenu et non à son aspect. Ainsi, si l'on trouve le `<h1>` trop gros (dans son navigateur), il faut plutôt envisager de redéfinir l'aspect du `<h1>`, par une feuille de style, que d'utiliser `<h2>` à la place ou, pire, d'utiliser des balises de mise en forme. Ainsi, si l'on doit aller à la ligne mais tout en restant dans le même paragraphe, il ne faut pas utiliser deux balises `<p>` mais une seule, incorporant un saut de ligne, `<br />`. À l'inverse, si une portion de texte forme un tout et ne doit pas être coupée, il faut l'encadrer par une balise `<nobr>`.

En plus de ces types de base, certains autres types de blocs de texte ont été définis qui permettent de répondre à des besoins spécifiques, mais en restant toujours dans le domaine d'une structuration logique. La plupart peuvent eux-même contenir de nombreux autres types de blocs de texte.

Tableau 3.3. autres éléments blocs du (X)HTML

balise	signification
`<div>`	bloc de texte le plus général (aucun aspect prédéfini)
`<td>`	cellule de donnée d'un tableau
`<th>`	cellule de titre d'un tableau
`<li>`	item d'une liste (à puce ou numérotée)
`<dt>` / `<dd>`	alinéas d'un glossaire (termes / définitions)
`<pre>`	texte préformaté (comme dans les exemples de code de ce cours)
`<blockquote>`	paragraphe de citation (marges augmentées)
`<header>`	HTML 5, en-tête de page (bandeau fixe ou autre)
`<footer>`	HTML 5, pied de page (bandeau fixe ou autre)
`<hgroup>`	HTML 5, pour grouper un titre et un sous-titre

Dans cette liste rappelons que certains blocs ne sont pas autonomes. Les cellules de tableaux ne peuvent apparaîtres que dans des tableaux, `<table>`, sous certaines conditions. Les éléments de listes ne peuvent apparaître que dans des listes à puces, `<ul>`, ou dans des listes numérotées, `<ol>`. Précisément, les listes sont de trois types :

- Listes numérotées : balise `<ol>`, chaque point est encadré par le marquage `<li>` ;
- Listes non numérotées (comme celle-ci) : balise `<ul>`, chaque point a le marquage `<li>` ;
- Listes de définitions, glossaires, index, etc.: marquage `<dl>`, les termes ou entrées sont désignées par `<dt>`, les définitions par `<dd>`.

Ces listes peuvent être imbriquées et leur aspect peut être déterminé par les styles. Voici des exemples :

Exemple 3.6. liste numérotée

code	rendu

`<p>Pour un canard à l'orange, prévoir :</p>`
`<ol>`
`  <li>un canard</li>`
`  <li>quatre oranges</li>`
`  ...suite de la liste...`
`</ol>`
	Pour un canard à l'orange, prévoir :
	1. un canard
	2. quatre oranges

Exemple 3.7. liste de définition

code	rendu

`<p>Glossaire :</p>`
`<dl>`
`  <dt>HTML</dt>`
`  <dd>Hyper Text Mark-up Language</dd>`
`  <dt>URL</dt>`
`  <dd>Uniform Resource Locator</dd>`
`  ...suite de la liste...`
`</dl>`
	Glossaire :
	HTML
	Hyper Text Mark-up Language
	URL
	Uniform Resource Locator

Tous les exemples précédents nous montrent que même si la structure logique est associée à des attendus en terme d'aspect, ceux-ci ne sont pas impératifs. Selon les navigateurs, l'augmentation de marge des paragraphes de citation pourront être différents. Le texte préformaté sera rendu dans une fonte à chasse fixe (tous les caractères occupent la même largeur sur la ligne d'écriture) mais la norme ne précise pas laquelle.

Signalons, enfin, que le marquage `<div>` (division) est spécial : il permet de regrouper (ou d'isoler) une série de blocs de texte. Il peut ainsi servir à regrouper tout le texte relevant d'une section, comme dans l'exemple suivant. Si on lui donne une position (à l'aide d'une feuille de style, par exemple), il peut afficher n'importe quel bloc de texte à n'importe quel endroit de la fenêtre ou de la page. Dans l'exemple suivant, on utilise l'attibut `class` (classe) pour caractériser la division, on pourra, si l'on souhaite, lui attribuer un rendu à l'aide des styles. Nous reviendrons sur l'attribut `id` un peu plus loin.

Exemple 3.8. balisage de divisions pour hiérarchiser des sections

```
<div class="section" id="elephAfro">
  <h1>Les éléphants d'Afrique</h1>
  <p>Début des considérations sur les éléphants.</p>
  ...suite de la section (blocs)...
  <div class="soussection" id="elephAfroHabitat">
    <h2>Habitat</h2>
    <p>Premières considérations sur ces éléphants.</p>
    ...suite de la sous-section (blocs)...
  </div>
</div>
```

3.3. Éléments linéaires

Le contenu des blocs de textes est principalement constitué d'éléments linéaires (*inline*) : textes, images, etc. Ces éléments linéaires utilisent principalement les balises suivantes :

Tableau 3.4. principaux éléments linéaires du (X)HTML

balise	signification	rendu
`<span>`	balise neutre	aucun rendu

balise	signification	rendu
` `	saut de ligne (sans contenu textuel)	retour à la ligne
`<hr />`	règle horizontale (sans contenu)	ligne horizontale
`<img />`	image (sans contenu)	l'image désignée
`<strong>`	insister	(habituellement) en gras
`<em>`	souligner (*emphasis*)	(habituellement) en italique
`<sup>`	exposant (*superscript*)	exposant
`<sub>`	indice (*subscript*)	indice
`<code>`	code	chasse fixe
`<samp>`	exemple d'affichage (à l'écran)	chasse fixe
`<cite>`	titre d'œuvre	en général en italique

Précisons que la ligne horizontale (règle, filet) se note `<hr>` (déconseillé) ou `<hr />` (conseillé en HTML et obligatoire en XHTML). L'aspect de cette règle peut être précisé par des attributs, mais il est plus intéressant d'employer les styles pour changer l'aspect des filets : on a plus de lattitude (on peut en changer la couleur) et on peut donner d'un coup le même aspect à toutes les règles d'un document ou d'un site web. Nous reviendrons sur ce point.

4. HTML et présentation de page

Bien entendu, le balisage d'aspect n'est pas absent du HTML. Ce fut, avant les feuilles de style, dont l'usage n'apparaît qu'avec HTML 4, le seul moyen de contrôler l'aspect des pages.

Dans le cas des blocs de texte, le moyen traditionnel de donner des indications de rendu est d'utiliser des attributs d'aspect. Par exemple, il est possible de préciser dans la balise ouvrante (paragraphe, titre, cellule, etc.) un attibut `align` (alignement) valant `left`, `right`, `center` ou `justify`. Toutefois, nous recommandons fortement d'utiliser des indications de style pour tout ce qui concerne le rendu visuel (plutôt que des attributs d'aspect). Quel que soit le choix, on évitera absolument de mêler attributs d'aspect et indications de style : les effets de ces mélanges sont souvent aléatoires !

Notons que l'on trouve encore dans certaines pages web la balise `<center>` qui n'est qu'un raccourci pour `<div align="center">`. Cette pratique est à éviter.

Les balises de présentation les plus courrantes sont les suivantes.

Tableau 3.5. principales balises de présentation

balise	rendu	usage en HTML 4	usage en HTML 5
`<b>`	gras	toléré à la place de `<strong>`	relief typographique, sans insistance
`<i>`	italique	toléré à la place de `<em>`	autres usages courants de l'italique
`<tt>`	chasse fixe	toléré à la place de `<code>`, `<samp>` etc.	obsolète
`<u>`	trait en dessous	obsolète	n'existe plus
`<s>`	trait au travers	obsolète	n'existe plus

Les balises `<b>` et `<i>` peuvent être employées dans le cas de textes habituellement rendu en gras ou en italique, à condition qu'il n'existe pas de balise pour cet usage. Par ailleurs, il est fortement conseillé de compléter la balise d'un attribut `class` précisant l'usage, ou d'un attribut `lang` indiquant un changement de langue.

Exemples :

Exemple 3.9. utilisations correctes de la balise `<i>`

```
<p>Le <i class="taxonomie">Felis silvestris catus</i> mange la Souris.</p>
<p>Le <i lang="en">web</i> est un outil fantastique.</p>
<p>Le terme <i>utilisabilité</i> est défini ci-après.</p>
```

Le "trait en dessous" correspond à une mauvaise pratique puisque, selon les règles d'orthotypographie, le souligné est rendu traditionnellement par l'italique en imprimerie et par le "trait en dessous" en manuscrit. Par ailleurs le trait en dessous est réservé aux liens dans le cas du web. Les traits (dessous, au travers, dessus) ne doivent être réalisés qu'en utilisant des indication de styles.

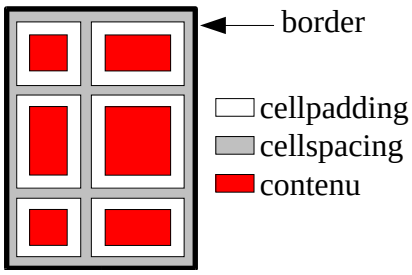
Outre ces balises simples signalons également la balise `<font>`, qui permettait de définir la police d'écriture. Cette balise est largement utilisée par certains traitements de texte quand ils exportent leurs documents sous forme de pages web. Même si elle est disponible dans les éditeurs de pages *web*, son usage est *très* fortement déconseillé (on peut la considérer comme obsolète dès HTML 4).

5. Tables HTML : entre structuration et présentation

Les mises en page ont longtemps été réalisées à l'aide de tableaux. Aujourd'hui, cependant, les feuilles de style permettent de positionner les blocs de contenu de façon précise, systématique et adaptable. Il ne faut donc plus *du tout* utiliser les tableaux pour autre chose que des données tabulées. Les tableaux doivent refléter une structuration logique du contenu, non un souhait de rendu visuel.

Un tableau est construit par `<table>...</table>`. À l'intérieur d'une table, chaque ligne est créée par `<tr>...</tr>` (*table row*). À l'intérieur de chaque ligne, les cellules sont créées par `<th>...</th>` (*table head*, cellule d'en-tête) ou par `<td>...</td>` (*table data*, cellule normale de donnée).

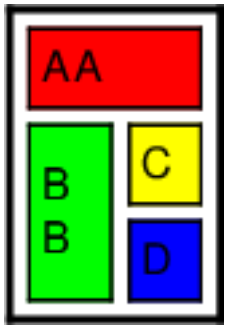
Figure 3.1. aires d'un tableau



Ces quatre balises disposent d'un certain nombre d'attributs qui permettent de les mettre en forme. On les évitera et on préférera des règles de style.

Un tableau régulier sera mis en page plus rapidement si le nombre de colonnes est spécifié comme attribut de `<table>` : `<table cols="3">`. Le HTML autorise des cellules à occuper plusieurs cases de la grille du tableau (à fusionner) : à l'intérieur de `<td>`, l'attribut `rowspan` spécifie le nombre de lignes sur lequel s'étend la cellule (par défaut, 1) ; l'attribut `colspan` spécifie le nombre de colonnes sur lequel s'étend la cellule (par défaut, 1). Voici un exemple avec ces attributs :

Exemple 3.10. alignement

code	rendu
`<table cols="2">` `  <tr>` `  <td colspan="2"> AA </td>` `  </tr>` `  <tr>` `  <td rowspan="2"> B   B </td>` `  <td> C </td>` `  </tr>` `  <tr>` `  <td> D </td>` `  </tr>` `</table>`	

Dans le cas de longues tables, existe la possibilité de préciser un en-tête et un pied de table (balises `<thead>`, `<tfoot>` et `<tbody>`) qui permettent, le cas échéant, de répartir un tableau sur plusieurs pages imprimées. Dans le cas de longues tables il est également conseillé d'employer les marqueurs `<colgroup>` et `<col>` pour indiquer des attibuts de colonnes ou groupes de colonnes (voir la documentation pour plus de précisions). Enfin, on peut ajouter une légende aux tables en utilisant la balise `<caption>` comme premier élément à l'intérieur de la `<table>`.

6. HTML et description de documents

Mais la force d'un langage de balisage est qu'il ne se limite pas à des indications de contenu. Par nature, le marquage structurel relève d'un autre plan d'information que le contenu *stricto sensu* d'un texte. Précisément : une page web, outre son contenu, contient des informations sur celui-ci, appelées méta-données.

En pratique, la structure générale d'un document (X)HTML est la suivante :

- Une ou deux lignes précisent le format de document employé, la version du (X)HTML utilisée, notamment.
- La page web est le contenu d'une balise `<html>`.
  - Celle-ci contient un en-tête, `<head>`, comportant principalement des méta-données :
    - Jeu de caractères : `<meta http-equiv="Content-Type" content="text/html; charset=`*`utf-8`*`">`,
    - Titre de la page : `<title>`*`intitulé`*`</title>`,
    - Auteur : `<meta name="Author" content="`*`nom de l'auteur`*`" />`,
    - Mots-clefs : `<meta name="Keywords" content="`*`liste, de, mots-clefs`*`" />`,
    - Feuilles de style à employer,
    - etc.
  - La page contient ensuite un corps de document, <body>, le contenu proprement dit, qui sera affiché.
    - Ce contenu est organisé en blocs contenant du textes, qui se suivent les uns les autres.

Voici un exemple en XHTML :

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Qu'est-ce qu'une page web ?</title>
    <meta name="Author" content="Yannis Delmas" />
    <meta name="Keywords" content="...mots-clefs..." />
    <link rel="stylesheet" href="cours.css" type="text/css" />
    ...suite de l'en-tête...
  </head>
  <body>
    ...contenu du document...
  </body>
</html>
```

Vu comme un langage informatique le HTML est donc d'abord un code de description de l'organisation (et de l'aspect) d'un texte. Si l'on ignorait le balisage, on trouverait simplement le texte brut du contenu de la page, “au kilomètre”.

Dans la mesure où l'on a affaire à un langage informatique, susceptible d'être observé (et édité) comme tel et non seulement sous forme de rendu (WYSIWYG), il peut être intéressant de disposer dans le code source de commentaires. Ceux-ci ne sont pas destiné à l'utilisateur, même s'il peut en prendre connaissance en regardant le code. Ces commentaires ont la forme suivante, qui n'est pas à proprement parler une balise : `<!-- commentaire -->`. Ce genre de construction peut servir à masquer temporairement une partie d'une page.

Signalons, pour être complet que le HTML peut servir également à décrire un découpage de la fenêtre du navigateur en plusieurs panneaux (*frames*). Cette ensemble de panneaux vient en remplacement de la balise `<body>` et chacun indique l'URL d'une autre page web lui servant de contenu. Cette manière de faire est désormais obsolète et doit absolument être évitée. La mise en page doit désormais être réalisée à l'aide d'une feuille de style.

7. HTML et hypermédia

Les images sont incorporées aux pages web à l'aide de la balise spécifique `<img />`, à la fois ouvrante et fermante. Ses principaux attributs sont :

- `alt` : obligatoire, équivalent textuel de l'image ;
- `src` : obligatoire, URL où trouver l'image (format JPEG, PNG ou GIF) ;
- `width` : optionnel, largeur d'affichage en pixels ou % (permet de réserver la place) ;
- `height` : optionnel, hauteur d'affichage en pixels ou % (permet de réserver la place).

Quand ni `width` ni `height` ne sont précisés, l'image sera affichée à sa taille originale (en pixels). Si l'un de ces attributs seulement est précisé, l'image sera redimensionnée en conservant ses proportions. Quand cela est possible, il est conseillé de préciser ces dimensions pour accélérer le rendu visuel de la page. Rappelons que, comme cette balise est vide, il vaut mieux l'indiquer comme ouvrante et fermante : `<img />`.

Les possibilités de position des images sont assez limitées : on ne peut placer une image “où l'on veut”. Que veut dire, d'ailleurs «où l'on veut» dans une page HTML, puisque le support final, l'écran du lecteur, reste inaccessible et divers ? Là encore, la mise en page sera réalisée à l'aide d'une feuille de style.

Signalons au passage que les règles et filets sont parfois réalisées dans les sites web professionnels à l'aide d'images spécifiques. Cette pratique est à éviter désormais, au profit de la balise `<hr />`, dont le rendu visuel sera construit à l'aide de feuilles de style.

Les pages web peuvent inclure de nombreux autres objets visuels ou multimédia, principalement à l'aide des balises `<object>` en HTML 4, à quoi s'ajoutent les balises `<video>` et `<audio>` en HTML 5.

Pour compléter l'aspect hypermédia, c'est à dire la possibilité de contenir ou de “recevoir” des liens hypertextes ou hyperliens, le HTML dispose d'un objet particulier appelé ancre. L'ancre (*anchor*) est désignée par la balise `<a>` et peut contenir n'importe quel type de contenu linéaire (texte, images, etc.). Sans indication particulière, le client entourera les images et soulignera le texte en bleu. À l'aide des styles on peut demander un rendu visuel différent. La source d'un hyperlien est l'ancre qui le désigne dans le document en cours. L'URI de destination d'un hyperlien est indiquée par l'attribut `href` (hyper-référence) de l'ancre. L'attribut optionnel `target` (cible) permet dans quelle fenêtre afficher la destination. Voici un exemple.

```
<a href="mailto:yannis.delmas@univ-poitiers.fr">Yannis Delmas</a> enseigne à
l'<a target="_blank" href="http://www.univ-poitiers.fr/">Université de Poitiers
```

L'URI peut désigner n'importe quel objet utilisable par le client. Il s'agit en général d'une page web. On parle alors d'URL. Il peut également s'agir d'un élément à l'intérieur d'une page web. La seule condition pour qu'une partie ou un élément d'une page web puisse être référencé par un lien est qu'il ait un nom. Pour donner un nom à un marquage HTML, il suffit de l'indiquer comme attribut `id`. Dans l'exemple de l'éléphant, ci-dessus, la section a pour nom `elephAfro` et la sous-section le nom `elephAfroHabitat`.

L'URI de l'élément nommé *`N`* et situé à l'intérieur du document d'URL *`U`* est : *`U`*`#`*`N`*. Si l'on veut désigner un élément à l'intérieur du document courant, l'URI devient alors simplement `#`*`N`*. Si l'exemple des éléphants est une partie du document `http://www.museum.fr/cours/animaux.html`, alors l'URI absolue de la section concernée est `http://www.museum.fr/cours/animaux.html#elephAfro`. Si l'on suit un lien qui a cette destination, le document `animaux.html` est chargé et le début de cette section est affiché en haut de la fenêtre. Si l'on désigne cette section à partir d'une table des matières située dans le même document, il suffit de donner `#elephAfro` comme hyper-référence. Si l'on désigne cette section à partir du document `http://www.museum.fr/cours/plantes.html`, il suffit de donner `animaux.html#elephAfro`.

Si l'on veut donner un nom à un ensemble de blocs dans un document, il suffit de les regrouper à l'intérieur d'une `<div>` et de donner un nom à ce marquage `<div>`. Pour regrouper des éléments linéraires (à l'intérieur d'un bloc), il est d'usage d'utiliser une ancre (balise `<a>`) quand il s'agit simplement de donner un nom pour en faire une destination d'hyperlien. Pour les autres usages, on regroupe à l'aide d'une balise `<span>`, qui joue pour les éléments linéaires le même rôle que `<div>` pour les blocs.

Parfois on souhaite simplement désigner un point dans un document. On utilise pour cela une une ancre vide :

```
<a id="chapitre1"></a>
```

Comme cette balise n'existe pas sous forme ouvrante-fermante, on doit faire suivre la forme ouvrante de la forme fermante.

Chapitre 4. XML et ses applications

1. XML vs. SGML

Le SGML et le XML se distinguent par trois aspects essentiels : la complexité, la souplesse et l'extensibilité des documents.

Le SGML est beaucoup plus complexe que le XML. À ce titre, il permet donc de faire plus de choses (en tant que langage, il est plus puissant). C'est, du moins, la théorie ; dans la pratique, la plupart des applications usuelles du SGML peuvent être effectivement réécrites à moindre frais comme des applications de XML. L'intérêt de la simplicité de ce dernier est la plus grande facilité de conception des outils informatiques qui permettent de l'utiliser, de le manipuler.

En tant que langage, le SGML est d'une écriture beaucoup plus souple que le XML. Par exemple, on ne distingue pas les balises ouvrantes-fermantes des balises ouvrantes. Autre exemple : il n'est pas toujours nécessaire de clore une balise ouvrante, la balise fermante (et donc la portée) restant donc implicite. Ceci peut être un avantage quand l'application est bien déterminée et que le fichier est produit “à la main” par un humain. Cela donne un certain confort aux concepteurs web travaillant en HTML. En revanche, cela est inutile quand le code est produit par un programme informatique. Enfin, cela peut s'avérer nuisible dans les cas où le document a vocation à être utilisé par plusieurs applications. La rigidité de XML devient, dans ce cas un avantage considérable : cela permet de concevoir des outils génériques, susceptibles d'être incorporés dans des outils plus spécifiques à telle ou telle application de XML. Par exemple, il existe de nombreux outils permettant de mettre en forme ou de réécrire des fichiers XML, de quelque application que ce soit (cf. cours ultérieur à ce sujet).

Ceci nous amène à une troisième propriété du XML : ses types de documents sont extensibles. C'est d'ailleurs ce que signifie XML : *eXtensible Markup Language* (langage extensible de balisage). Ce qualificatif désigne le fait qu'un même fichier XML peut incorporer des marquages relevant de plusieurs types de documents différents et indépendants (cf. un peu plus loin) ; par exemple : un document textuel en XHTML peut contenir des équations mathématiques écrites en MathML. Les navigateurs comprenant et XHTML et MathML afficheront correctement un tel document. D'autres logiciels pourront se contenter d'afficher (ou d'éditer) seulement une partie de tels fichiers (seulement la partie XHTML ou seulement les équations MathML, par exemple).

Une autre limitation du SGML est qu'il n'était bien adapté qu'aux caractères latin “de base” (code ASCII). Tout autre caractère devait être représenté par des entités, ce qui est rapidement un obstacle quand il s'agit de travailler avec de nombreuses langues, donc de nombreux encodages particuliers. XML résoud cette difficulté en fonctionnant nativement avec le jeu de caractères universel Unicode [http://www.unicode.org/standard/translations/french.html] (ISO 10646). Tous les logiciels qui utilisent XML sont sensés comprendre Unicode. Ce jeu de caractères comprend plusieurs blocs de caractères tels que « latin », « API », « symboles mathématiques », « arabe », « chinois »… Normalement, tous les alphabets et systèmes idéographiques en usage actuellement dans le monde y sont représentés (mais cela ne veut pas dire que tous les caractères sont affichables sur tous les ordinateurs). Le jeu de caractère Unicode comprend plusieurs milliers de caractères, il existe donc plusieurs encodages usuels de ses tables. Les deux principaux de ces encodages sont UTF-8, codant les caractères latins de base sur un octet et les autres sur plusieurs, et UTF-16 codant chaque caractère sur deux octets. En Occident, les caractères sont la plupart du temps codés par un ou deux octets en UTF-8, ce qui en fait l'encodage privilégié de cette aire géographique. L'UTF-16 est plus “économique” pour le reste du monde.

Malgrès ces différences, XML reste largement compatible avec SGML, dont il est une application (pour l'essentiel).

2. Structure des données XML

Grosso modo un document XML est un assemblage de balises similaire à un fichier HTML : un marquage de document contient une suite de marquages et de portions de texte ; chacun de ces

marquages est lui-même susceptible de contenir une suite de marquages et de portions de texte ; et ainsi de suite *ad libitum*. Comme pour un fichier (X)HTML, un fichier XML se présente donc comme un “arbre” dont les noeuds sont des marquages ou des portions de texte. Les “feuilles” sont les marquages sans contenu et les portions de texte. Comme en (X)HTML des portions entière de document peuvent apparaître sous la forme d'entités dont la forme générale est `&nom;`.

Comme en (X)HTML, à cette structure en arbre peuvent s'ajouter, dans un fichier XML, des commentaires, entre `<!--` et `-->`, destinés à un lecteur humain et ne relevant pas du contenu du document XML. Les commentaires masquent tout ce qu'ils englobent (balises ou autre). Peuvent également s'ajouter des instructions de traitement, entre `<?` et `?>`, destinées à un logiciel de traitement du fichier, par exemple un processeur tel que PHP (cf. cours ultérieur).

Ajoutons enfin qu'un fichier XML devra commencer par une ligne de prologue similaire à la suivante, qui sert à indiquer que l'on utilise XML et à préciser le jeu de caractères du contenu textuel :

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Pour l'essentiel, XML n'impose pas d'autre contrainte. N'importe quelle balise peut apparaître dans n'importe quelle autre et le texte peut apparaître à n'importe quel endroit. Un fichier qui se plie à cette structure est dit bien formé ou conforme à XML.

Notre présentation est, bien entendu, simplifiée. Nous nous contenterons ici d'ajouter que certains attributs standardisés peuvent s'appliquer à toute balise XML. Pour de plus amples précisions on se reportera aux recommandations du W3C [http://www.w3.org] ou à des ouvrages spécialisés.

L'un de ces attributs généraux de XML est `xml:lang` qui indique la langue du contenu d'un marquage, selon la façon usuelle (RFC 1766) et comme en HTML. Précisément, cette langue est généralement composée d'un code ISO de langue à deux lettres (ISO 639 : liste [http://www.loc.gov/standards/iso639-2/frenchlangn.html])[1], par exemple : `fr` pour le fançais, `en` pour l'anglais, `de` pour l'allemand… Ces codes peuvent éventuellement être suivis par un tiret et un code de ISO de pays à deux lettres (ISO 3166 : liste [http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-fr1-semic.txt])[2] indiquant une localisation linguistique, par exemple : `en-GB`, `en-US`… On peut également donner un sous-code de variant linguistique enregistré auprès de l'IANA [http://www.iana.org] (liste [http://www.iana.org/assignments/language-tags]).

Un autre groupe d'attributs, essentiel, est `xmlns:nom`, qui permet de gérer l'extensibilité. Nous y reviendrons un peu plus loin.

La structure du XML, on le voit, est extrêmement générale et, à ce titre, propre à représenter un très grand nombre de type de données, particulièrement les données à contenu textuel. Le fait que XML ne soit lié à aucun alphabet particulier (puisqu'il accepte Unicode) lui ouvre grand les portes de l'interopérabilité.

De plus, la généralité de sa structure permet d'opérer des traductions automatiques d'un type de documents XML à un autre par de simples réécritures consistant à remplacer telle balise et son contenu par telle portion de document, intégrant à un endroit convenu ledit contenu. C'est le principe des traducteurs automatiques XSL ou XSLT, abusivement appelés « feuilles de style ». Ce type de réécritures est extrêmement puissant ; il permet, par exemple, de réécrite directement des ensembles d'enregistrements de bases de données (écrits sous forme XML) en documents XHTML. Le présent document, par exemple, a été créé au format Docbook, qui est une application de XML, ce qui permet par de telles réécriture d'obtenir, à volonté, un jeu de pages XHTML, une page XHTML unique, un document PDF, un document RTF…[3]

[1] L'autorité de la norme ISO 639 est la Bibliothèque du Congrès étasunien : http://www.loc.gov/standards/iso639-2/.
[2] La référence pour la norme ISO 3166 est : http://www.iso.ch/iso/en/prods-services/iso3166ma/index.html.
[3] RTF n'est pas du XML mais a une structure suffisamment proche pour qu'un tel système de réécriture puisse en produire mécaniquement. PDF n'est pas non plus du XML mais il peut être produit mécaniquement à l'aide d'un format de description de page, FO, qui, lui, est une application de XML.

3. Applications de XML

Tels que nous les avons présentés, pour l'instant, les fichiers XML sont utiles, mais à peine plus structurés qu'une série de 0 et de 1 (l'avant-XML, si l'on peut dire). Ce qui fait un document XHTML n'est donc pas tant le fait d'être un fichier XML mais le fait de relever d'un type de documents bien précis, XHTML en l'occurrence. Comme pour le SGML un *type de document XML*, ou *application de XML*, sert à indiquer quelles sont les balises valides et quel doit être le contenu de chacune d'elles. Comme en SGML les applications de XML peuvent être définies à l'aide de DTD. En revanche, XML ne se limite pas aux DTD et accepte d'autres types de définition de types de documents, en particulier les *schémas XML*, qui permettent de mettre en œuvre plus aisément les objectifs d'extensibilité. Un fichier qui se plie complètement aux contraintes d'une telle définition, et quelques autres que nous ne détaillerons pas ici, est dit *valide*.

La référence à une DTD est ainsi indiquée (`nomÉlémentDocument` étant le nom de l'élément qui contient tout le document) :

```
<?xml version="1.0" standalone="no" encoding="jeu de caractères"?>
<!DOCTYPE nomÉlémentDocument SYSTEM "URI d'une DTD">
```

ou, mieux à l'aide d'un identificant public "formel" :

```
<?xml version="1.0" standalone="no" encoding="jeu de caractères"?>
<!DOCTYPE nomÉlémentDocument PUBLIC "identifiant formel public d'une DTD"
    "éventuellement, URI d'une DTD">
```

Dans le premier cas (`SYSTEM`), l'URI d'une DTD est donnée et le document peut être validé au regard de cette DTD. Le second cas (`PUBLIC`) correspond à des identifications définitives de DTD officielles. Dans ce cas, les logiciels peuvent se référer à un catalogue de DTD dont ils disposeraient par ailleurs et, le cas échéant, utiliser une autre URI que celle fournie pour retrouver la DTD (par exemple une URI locale). L'URI n'est alors utilisée que comme roue de secours.

L'identifiant public “formel” d'une DTD a la forme générale suivante :

```
{-|+|ISOréf}//nom du propriétaire//DTD #description//langue
```

Ne vous amusez pas à en inventer : ceux-ci ne correspondent normalement qu'à des identifications publiques standardisées et en général, vous n'aurez qu'à les recopier. Voici les principales DTD (les premières sont SGML, les suivantes XML) :

- HTML 3.2

  ```
  -//W3C//DTD HTML 3.2 Final//EN
  ```

- HTML 4.0, strict

  ```
  -//W3C//DTD HTML 4.01//EN
  http://www.w3.org/TR/html4/strict.dtd
  ```

- HTML 4.0, de transition

  ```
  -//W3C//DTD HTML 4.01 Transitional//EN
  http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd
  ```

- HTML 4.0, avec cadres (*frames*)

  ```
  -//W3C//DTD HTML 4.01 Frameset//EN
  http://www.w3.org/TR/1999/REC-html401-19991224/frameset.dtd
  ```

- XHTML 1.0, strict

```
-//W3C//DTD XHTML 1.0 Strict//EN
http://www.w3.org/TR/1999/PR-xhtml1-19991210/DTD/xhtml1-strict.dtd
```

- XHTML 1.0, de transition

```
-//W3C//DTD XHTML 1.0 Transitional//EN
http://www.w3.org/TR/1999/PR-xhtml1-19991210/DTD/xhtml1-transitional.dtd
```

- XHTML 1.0, avec cadres

```
-//W3C//DTD XHTML 1.0 Frameset//EN
http://www.w3.org/TR/1999/PR-xhtml1-19991210/DTD/xhtml1-frameset.dtd
```

- SVG 1.1

```
-//W3C//DTD SVG 1.1//EN
http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd
```

- MathML 2.0

```
-//W3C//DTD MathML 2.0//EN
http://www.w3.org/TR/MathML2/dtd/mathml2.dtd
```

4. Différences entre XHTML 1.0 et HTML 4.0

La liste de la fin de la section précédente (identifiants de DTD) nous montre que le XHTML 1.0 (et le HTML 4) existent en trois versions :

- Le *XHTML strict* est recommandé pour tous les usages courants. Certaines balises d'aspect, obsolètes, y sont limitées voire proscrites. Le XHTML ne comporte pas non plus les balises `<applet>` et `<iframe>` (cadres en-ligne), qu'on remplacera, autant que possible par des `<object>`.

- Le *XHTML de transition* (*transitional*) sert quand on a besoin d'admettre des éléments proscrits par le XHTML strict, dans une optique de transition en douceur depuis le HTML 4. Autant que possible, il vaut mieux corriger ses mésusages et utiliser le XHTML strict plutôt que le XHTML de transition.

- Le HTML traditionnel permet deux types de documents : les pages web et les divisions de fenêtres en cadres (*frames*). Ces deux types de documents sont séparés depuis HTML 4 : on utilise XHTML strict et de transition pour les pages et le *XHTML frameset* pour les découpages. Cette troisième forme est à éviter.

Cela mis à part, XHTML 1.0 n'est pas fondamentalement différent du HTML 4.0.1. Les différences résultent du passage à XML. Les principales spécificités du XHTML sont :

- L'organisation des balises est stricte : toute balise ouverte doit être fermée et deux marquages ne peuvent se chevaucher. En particulier, les éléments vides doivent impérativement utiliser une balise ouvrante et fermante (ou, éventuellement, deux balises : une ouvrante et une fermante). Ainsi, on n'écrira pas `<br>` mais `<br/>`.

- Les noms des éléments XHTML sont tous en minuscule. Comme la casse est significative, en XML, on prendra soigneusement garde à cet aspect. Ce point devient très important quand on souhaite progammer des manipulations de la structure d'une page.

- Les attributs implicites sont interdits. Ainsi, on n'écrira pas `<option selected>` mais `<option selected="selected" />`.

- Les attributs doivent impérativement être munis de guillemets, simples ou doubles. Ainsi, on n'écrira pas `<input size=3>` mais `<input size="3" />`.

- Les balises et attributs d'aspect sont, autant que possible, proscrits. On se reposera sur des feuilles de style, ou, exceptionnellement, sur l'attribut `style`. En particulier, on évitera les balises `<font>`, `<u>` et `<s>`.

- Les fragments sont identifiés par l'attribut `id` et non plus l'attribut `name`. Pour des raisons de compatibilité, on utilisera de préférence les deux. L'attribut `name` existe encore pour les éléments de formulaire (cf. cours ultérieur).

Finalement, XHTML est une bonne école de rigueur.

5. Extensibilité pour XML en général et pour XHTML en particulier

XML est extensible en ce sens qu'un fichier XML relevant d'un type donné, par exemple une page *web* XHTML, peut contenir des balises relevant d'un autre type, par exemple des formules mathématiques MathML. Ceci se fait en utilisant des *espaces de noms* (*namespace*).

On peut se représenter un espace de noms comme étant une sorte liste de noms de balises et de noms d'attributs. On indique qu'un marquage et tout son contenu relèvent d'un espace de noms en utilisant la construction suivante :

```
<espNom:balise1 espNom:attribut1="valeur" xmlns:espNom="URI de l'espace de nom"
  ...contenu...<espNom:balise2>...</espNom:balise2>...
</espNom:balise1>
```

Tous les attributs et toutes les balises incluses dans cette `<balise1>` doivent être précédés de l'identifiant choisi pour l'espace de nom suivi d'un symbole deux-points (ici `espNom:`). L'identifiant de l'espace de noms est quelconque, mais on évitera d'être imaginatif et on suivra l'usage. Voici l'exemple d'une formule mathématique simple (y=1+x) :

```
<mathml:math xmlns:mathml="http://www.w3.org/1998/Math/MathML">
  <mathml:mrow>
    <mathml:mi>y</mathml:mi><mathml:mo>=</mathml:mo>
    <mathml:mn>1</mathml:mn><mathml:mo>+</mathml:mo><mathml:mi>x</mathml:mi>
  </mathml:mrow>
</mathml:math>
```

On le voit, même pour un marquage simple, le code est assez verbeux. Une telle forme est nécessaire quand on utilise plusieurs espaces de noms simultanément. Ce n'est pas le cas général : bien souvent on peut abréger une telle écriture en omettant le nom de l'espace de noms. Voici l'exemple d'une formule plus complexe (représentée sur l'image suivante) :

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mi>y</mi><mo>=</mo>
    <mfrac> <mn>1</mn>
      <msqrt>
        <mrow><msup><mi>x</mi><mn>2</mn></msup><mo>+</mo><mn>1</mn></mrow>
      </msqrt>
    </mfrac>
  </mrow>
</math>
```

Dans certains cas, l'espace de noms se manifeste seulement par des attributs. Ainsi l'attribut `xml:lang` se lit-il : l'attribut `lang` de l'espace de noms `xml`. D'autres tels espaces peuvent être ajoutés. Par exemple, l'espace XLink permet d'ajouter pour n'importe quel élément un lien vers d'autres ressources (métadonnées, hyperlien, etc.).

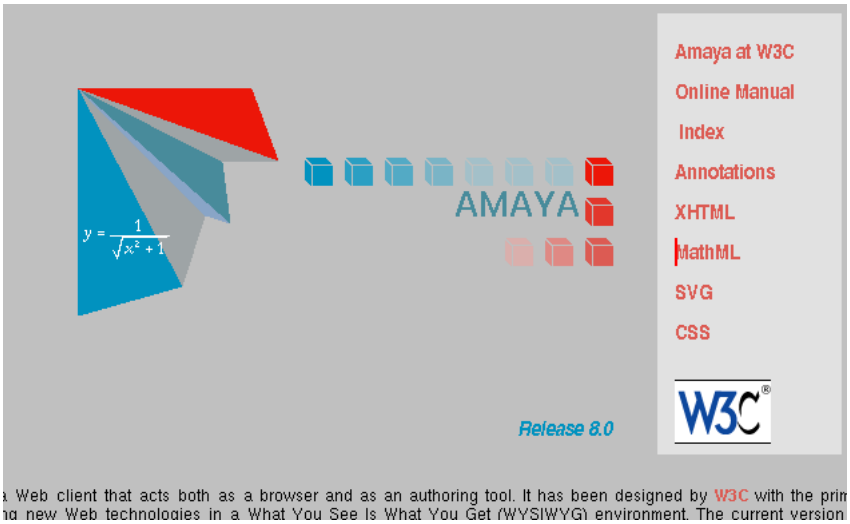
L'URI à employer pour un espace de nom est normalisée. Elle ne s'invente pas. Voici les principales :

Tableau 4.1. Principaux espaces de noms

nom	usage	URI
xml	XML	http://www.w3.org/XML/1998/namespace
xhtml	HTML : texte	http://www.w3.org/1999/xhtml
mathml	MathML : formule mathématique	http://www.w3.org/1998/Math/MathML
svg	SVG : dessin vectoriel	http://www.w3.org/2000/svg
rdf	RDF : description de resources	http://www.w3.org/1999/02/22-rdf-syntax-ns#
xlink	XLink : liens pour n'importe quel élément	http://www.w3.org/1999/xlink
atom	Atom : fil/flux de syndication	http://www.w3.org/2005/Atom

L'intérêt de ce type d'architecture est d'intégrer de façon modulaire différent types de possibilités de sens, de contenu ou de présentation dans un même document. L'exemple ci-dessous est une capture d'écran d'une version simplifiée de la page de présentation de l'éditeur web Amaya [http://www.w3.org/Amaya/]. Cet éditeur/navigateur permet de mêler dans une même page des graphismes SVG, des formules MathML et du texte XHTML.

Figure 4.1. Exemple de XHTML, SVG et MathML mêlés



6. XML, transformations et architecture trois tiers

Les documents XML ayant une structure d'arbre, ils bénéficient de la propriété suivante : si l'on remplace dans un document `<abc>defg</abc>`, où `<abc>` est une balise quelconque et `defg` un contenu quelconque, contenant ou non des balises, par un fragment de document XML intégrant le contenu `defg`, alors le résultat est toujours un arbre XML. Une telle opération, quand elle est opérée systématiquement sur un document XML, c'est à dire en remplaçant toutes les occurrences de `<abc>`, est appelée réécriture par substitution. Ces réécritures sont très simples à programmer.

Prenons un exemple de fichier initial :

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<donnees>
  <enregistrement>
```

```
    <nom>Dupond</nom>
    <prenom>François</prenom>
    <tel>2134</tel>
  </enregistrement>
  <enregistrement>
    <nom>Durand</nom>
    <prenom>Côme</prenom>
    <tel>2457</tel>
  </enregistrement>
  ...
</donnees>
```

Puis appliquons lui la transformation suivante (écriture informelle) :

```
<donnees>¤</donnees> =>  <html>
                           <head>…</head>
                           <body>
                             <table>
                               <tr><th>nom</th><th>prénom</th><th>tél.</th></tr
                               ¤
                             </table>
                           </body>
                         </html>
<enregistrement>¤</enregistrement> => <tr>¤</tr>
<nom>¤</nom>                       => <td><b>¤</b></td>
<prenom>¤</prenom>                 => <td>¤</td>
<tel>¤</tel>                       => <td>¤</td>
```

On obtient alors le document suivant :

```
<html>
  <head>…</head>
  <body>
    <table>
      <tr><th>nom</th><th>prénom</th><th>tél.</th></tr>
      <tr>
        <td><b>Dupond</b></td>
        <td>François</td>
        <td>2134</td>
      </tr>
      <tr>
        <td><b>Durand</b></td>
        <td>Côme</td>
        <td>2457</td>
      </tr>
      …
    </table>
  </body>
</html>
```

Cet exemple montre qu'il est beaucoup plus facile d'opérer une réécriture par substitution que de faire un programme d'affichage de données, HTML ou autres, quel que soit le langage utilisé, pour peu que l'on dispose d'un outil de réécriture.

Il existe un système standard de réécriture pour le XML, appelé XSL. Il permet ce type de réécritures et d'autres, plus complexes, qui permettent de tenir compte du contexte d'apparition de chaque balise. En anglais, XSL veut dire : *eXtensible Stylesheet Language* (langage de feuilles de style extensible). Il faut, bien sûr, comprendre « feuille de style » dans un sens très large : tout ce qui permet de passer d'une représentation des données à une autre (en vue d'un affichage). SGML possédait déjà un tel

système, similaire mais trop lourd : DSSSL. Ce système est particulièrement efficace pour l'affichage de données dans les architectures trois-tiers modernes. Commençons par un rappel.

Le principe des architectures trois-tiers est une évolution de la logique client-serveur :

1. Les applications les plus simples sont constitués d'un logiciel unique gérant les données et effectuant les traitements demandés par l'utilisateur au moyen d'une interface.

   Exemple : la quasi-totalité des traitements de texte actuels.

2. L'architecture client-serveur sépare l'interface de la gestion et du traitement. La gestion et le traitement sont dévolus à un logiciel serveur (de l'application) tandis que des logiciels clients servent d'interface aux utilisateurs. Ceci permet à plusieurs utilisateurs d'accéder simultanément aux mêmes données situées sur un serveur donné.

   Exemple : la plupart des progiciels de gestion de stock actuels.

3. L'architecture trois-tiers sépare l'interface, la gestion et le traitement. Les données et leur gestion sont dévolues à un logiciel serveur, généralement un serveur gestionnaire de bases de données. L'utilisateur utilise l'interface standardisée d'un logiciel client, généralement un navigateur *web*. Le traitement des données, selon les instructions de l'utilisateur, est alors effectué par un logiciel intermédiaire. Ce logiciel est client du serveur de données et serveur pour logiciel d'interface. Le logiciel intermédiaire est souvent lui-même composé de plusieurs « agents ». Cette organisation permet de décharger au maximum les serveurs de données pour les rendre plus disponibles. Les logiciels intermédiaires peuvent également souvent être dédoublés, ce qui permet d'augmenter encore la capacité de traitement.

   Exemple : les logiciels de réservation de train sur les *web*.

Quand un serveur de données trois-tiers fournit des données sous forme XML (au besoin via un traducteur), il est facile de les réécrire par XSL pour les présenter sous forme de page *web*, voire directement en XML à l'aide d'une feuille de style CSS (chapitre suivant).

Chapitre 5. Présentation par feuille de style CSS

1. Introduction

Le principe fondateur des styles est de séparer contenu et rendu visuel. La logique était peu affirmée aux débuts du HTML ; elle est ferme depuis l'institution de XML et du XHTML. Nous nous contenterons de présenter ici les feuilles de style CSS ; nous reviendrons dans un cours suivant (M2) sur les transformations XSL (XSLT, aussi appelée, un peu abusivement, “feuilles de style” XSL). Le principe des feuilles de style CSS est très simple : il s'agit d'attribuer des qualités de présentation, généralement visuelles, à un certain nombre de balises dans un certain nombre de contextes.

Actuellement, la recommandation CSS en est à sa “version” 2. La “version” 3 est en cours de validation. Dans le cas de CSS, on parle plus volontiers de « niveaux » que de « versions » parce que chaque niveau vise d'abord à compléter le précédent. La version 1 est donc appelée « CSS niveau 1 », ou « CSS 1 ». Son successeur est le « CSS niveau 2 », ou « CSS 2 », etc. Nous développerons ici surtout la partie élémentaire du CSS 2, qui correspond essentiellement à la version 1 plus les ajouts les plus usuels[1]. Les feuilles de styles commencèrent à être comprises, progressivement, par les navigateurs à partir de Netscape 4 et Internet explorer 4. Les principaux navigateurs interprêtent aujourd'hui la totalité de CSS 2 et une partie de CSS 3.[2].

L'intérêt essentiel des feuilles de style CSS est d'être cascadables # c'est à dire fonctionner en cascade. C'est d'ailleurs ce que veut dire le sigle : *Cascading StyleSheet*. Ceci signifie trois choses : 1) un document donné peut inclure plusieurs sources d'indications de style (notamment externes) ; 2) de nombreux documents peuvent utiliser une même source, ce qui permet de maintenir aisément l'homogénéité d'aspect d'un ensemble de documents ; 3) les sources sont priorisées, afin de déterminer laquelle s'applique en cas de conflit. Nous reviendrons en détail sur ces trois points.

Les indications de style peuvent se trouver en trois endroits :

- Au niveau d'une balise ouvrante particulière, pour s'appliquer à l'élément qu'elle désigne ; on parle de feuille de style en-ligne [*inline style sheet*] ;
- Au niveau d'un fichier (X)HTML, pour s'appliquer à tout ou partie du document ; on parle de feuille de style incorporée à la page [*embeded style sheet*] ;
- Sous forme de fichier CSS autonome susceptible de s'appliquer à un ensemble de documents, voire à un site web entier ; on parle alors de feuille de style externe [*external stylesheet*].

Elles forment, le cas échéant, toute une hiérarchie d'indications de style du plus local au plus global.

2. Déclarations de style, règles de style

2.1. Présentation, forme générale, déclarations en-ligne

Une déclaration [*declaration*] de style est simplement l'affectation d'une valeur, par exemple « rouge clair », à une propriété d'une portion de document, par exemple « l'arrière-plan ». Les propriété sont

[1]Pour aller plus loin, nous renvoyons tout simplement à la source la plus fiable possible : les recommandations du W3C, puisque celles-ci sont lisibles (ce qui n'est pas le cas de toutes les recommandations du W3C !). Le point d'entrée le plus pratique est généralement l'index. Version 1 : VO [http://www.w3.org/TR/REC-CSS1], VF [http://www.yoyodesign.org/doc/w3c/css1/index.html]. Version 2.1 : VO [http://www.org/TR/CSS21/]. Version 2 : VF [http://www.yoyodesign.org/doc/w3c/css2/cover.html]. On consultera également avec profit la W3C Cheat Sheet [http://www.w3.org/2009/cheatsheet/].

[2]Une excellente liste de compatibilité [http://www.quirksmode.org/css/contents.html] peut être trouvée sur le site Quirksmode.

le plus souvent de mise en page ou de mise en forme (visuelle ou auditive). La forme générale d'une telle déclaration est :

nom-propriété : *valeur-propriété*

On peut regrouper plusieurs déclarations en les délimitant par des points-virgules :

$nom\text{-}propriété_1$: $valeur_1$; $nom\text{-}propriété_2$: $valeur_2$; …

Ces déclarations peuvent être directement attachées à une balise (X)HTML au moyen de l'attribut HTML `style`. C'est la forme *en-ligne* des feuilles de style :

```
<span style="color: #0000FF; background-color: #FFCCCC">ceci sera bleu sur rose
```

L'exemple ci-dessus montre qu'il est possible d'affecter des indications de style à tout fragment de document, même s'il ne correspond pas, *a priori*, à une balise : pour cela on peut introduire une balise neutre (s'il n'y a rien de plus approprié) : `<span>` pour un fragment en-ligne, `<div>` pour une suite de blocs.

2.2. Règles de style, feuilles de style incorporées

Les déclarations en-ligne présentent un inconvénient notable : elles ne concernent qu'un point précis du document. Or, précisément, tout l'intérêt du texte structuré est dans la généralité : tous les paragraphes doivent avoir tel aspect, tous les titres tel autre, etc. On dispose, pour cela, des règles de style [*rule*/*ruleset*]. Nous allons voir ici les plus simples d'entre-elles. Nous traiterons de cas plus complexes plus loin.

Le cas le plus simple de règles de style consiste à définir ou redéfinir l'aspect d'une balise (HTML ou autre). Il prend la forme suivante :

nom_balise { *déclarations de style* }

On peut ajouter à ces règles des commentaires, placés entre `/*` et `*/`.

Ces règles sont simplement mises les unes à la suite des autres. On peut également ajouter des espaces et sauts de ligne pour améliorer la lisibilité (sans se dispenser des points-virgules). Voici un exemple issu de la version web de ce document :

```
body { background: #FFFFFF }    /* document : fond blanc */
h1   { color: #800000; font-family: sans-serif }
p    { text-align: justify }    /* paragraphes justifiés */
```

Pour être appliquées à une page, ces règles peuvent être ajoutées au fichier, à l'aide de la balise spéciale `<style>`, placée dans l'en-tête (`<head>`). On parle alors de *feuille de style incorporée* (au document) :

```
<style type="text/css">
  ...règles de style...
</style>
```

Il est également possible de définir des règles ne s'appliquant qu'à un seul élément dans la page. Pour cela, il suffit de donner un nom à ce fragment et d'employer une règle de la forme suivante :

#nom_fragment { *déclarations de style* }

Ceci permet d'éviter complètement les déclarations de style en-ligne.

2.3. Regroupements et classes CSS

Dans les exemples précédents « `h1 { color: #800000; font-family: sans-serif }` » n'était qu'une forme ramassée pour :

```
h1 { color: #800000 }
h1 { font-family: sans-serif }
```

De la même façon, il est possible de regrouper plusieurs éléments HTML ou XML qui doivent bénéficier des mêmes règles en les listant séparés par des virgules, comme dans l'exemple suivant.

```
p { text-align: justify }
h1, h2, h3, h4, h5 { color: #800000; font-family: sans-serif }
```

L'exemple précédent est donc en tout point équivalent à l'écriture étendue suivante.

```
p { text-align: justify }
h1 { color: #800000; font-family: sans-serif }
h2 { color: #800000; font-family: sans-serif }
h3 ...
```

Ces possibilités de regroupement s'appliquent également aux identifiants de fragments (`#abc`). Toutefois tenter de grouper des fragments identifiés (spécifiques) et des noms de balises (génériques) signale généralement un défaut de conception. En effet, si plusieurs partagent les mêmes caractéristiques de présentation, c'est probablement qu'ils partagent un rôle ou un usage. La bonne méthode, dans ce cas, est d'adopter une posture générique en définissant une *classe CSS* désignant ce rôle ou cet usage. Ceci se fait de la façon suivante :

```
.nom_de_la_classe { déclarations }
```

Notez bien le point qui précède immédiatement le nom de la classe. Notez également que nous déconseillons fortement que le nom de la classe comporte un caractère spécial (pas d'espace, pas d'accent…). Une telle classe s'utilise dans le corps d'un document (X)HTML à l'aide de l'attibut `class`, qui s'utilise sans point :

```
<abc class="nom_de_la_classe" ...>
```

Cet attribut n'est explicitement disponible que pour le type de documents (X)HTML. Il n'est pas défini par défaut pour les autres applications de XML[3].

Voici un exemple d'utilisation de classes CSS dans un document HTML. La classe `alinea` signale les balises (`<p>` probablement) qui représentent un bloc de texte devant marquer l'alinea (un bloc qui correspond à un début de paragraphe, du point de vue du contenu) tandis que `corpsDeTexte` signale des blocs de texte qui ne le doivent pas (un bloc correspondant à la suite d'un paragraphe, interrompu, par exemple, par un tableau ou une formule).

```
/* styles de paragraphe */
.formule               { text-align:  center  }
.corpsDeTexte, .alinea { text-align:  justify }
.alinea                { text-indent: 1.5cm   }
```

Autre exemple : on peut rendre plus lisible les tableaux en alternant des lignes plus ou moins claires ou de coloris différent[4].

```
/* feuille de style */
.lignePaire   { background-color: #E0E0E0 }
.ligneImpaire { background-color: #CCCCCC }

<!-- document HTML -->
<table>
  <tr class="entete">       <th> … </th> </tr>
  <tr class="lignePaire">   <td> … </td> </tr>
  <tr class="ligneImpaire"> <td> … </td> </tr>
```

[3] Nous verrons toutefois qu'il existe une construction CSS 2 qui généralise le principe des classes à tout document XML.
[4] Il existe une méthode plus efficace et plus générale en CSS 3 (cf. cours ultérieur).

```
  …
</table>
```

2.4. Feuilles de style externes et modularité CSS

La même logique qui conduit à regrouper des règles dans une feuille de style incorporée et dans des classes veut que l'on mette en commun des indications de style entre plusieurs pages. Ceci permet, par exemple, de disposer d'une feuille de style unique pour tout un site *web*. Ainsi, quand on doit opérer des changements d'aspect, il suffit de modifier cette unique feuille de style pour que l'aspect de toutes les pages du site (qui y font référence) soit changé.

En HTML, l'appel d'une *feuille de style externe* se fait en indiquant dans l'en-tête de la page (`<head>`) :

```
<link rel="stylesheet" type="text/css" href="nom_du_fichier_css" />
```

Le fichier CSS indiqué est simplement un fichier texte contenant une suite de règles CSS et doté de l'extension `.css`.

Un même document peut faire appel à autant de feuilles de style que nécessaire.

Imaginons maintenant qu'un site dispose de rubriques avec chacune un code couleur différent. On pourra alors mettre dans une feuille de style externe ce qu'il y a de commun à tout le site et ajouter une feuille de style supplémentaire propre à chaque rubrique :

```
<link rel="stylesheet" type="text/css" href="site.css" />
<link rel="stylesheet" type="text/css" href="rubrique-2.css" />
```

Une autre manière, plus élégante, d'obtenir le même effet est que chaque feuille de style de rubrique fasse référence elle-même à la feuille de style générale du site. Ceci permet de mettre en œuvre une modularité entre feuilles de style. Ceci se fait en incluant dans la feuille de style de rubrique la règle spéciale suivante :

```
@import url("uri de la feuille de style à importer");
```

En XML, l'appel d'une feuille de style se fait en ajoutant après le prologue XML l'instruction de traitement suivante :

```
<?xml-stylesheet type="text/css" href="nom_du_fichier_css" ?>
```

En XHTML, il est possible de combiner les deux déclarations (HTML et XML), même si ce n'est pas formellement nécessaire :

```
<?xml version="1.0" encoding="…" ?>
<?xml-stylesheet type="text/css" href="url_du_fichier_css" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <link rel="stylesheet" type="text/css" href="url_du_fichier_css" />
    …
```

Quand la feuille de style est interne, cette méthode est aussi applicable : il suffit de lui donner un nom. Voici un exemple :

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/css" href="#feuilleInterne" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    …début de l'en-tête…
    <style type="text/css" id="feuilleInterne">
```

```
…règles de style…
</style>
…
```

3. Propriétés de style usuelles

Les paragraphes ci-après, présentent les propriétés et valeurs les plus courantes. Il ne saurait être question, dans un tel cours, de viser l'exhaustivité. Pour plus de détails, on se reportera à la recommandation du W3C.

3.1. Les déclarations CSS de couleur

Le cas le plus élémentaire de déclaration de style est une affectation de couleur, couleur d'écriture ou couleur de fond. Les propriétés correspondantes sont, généralement : `color` et `background-color`.

Outre la valeur spéciale `transparent` (pour les fonds), le modèle de couleur est au standard rouge-vert-bleu. Chacune de ces composante varie entre 0 (0 %) et 255 (100 %). Comme en HTML, la forme canonique d'une valeur de couleur est #RRVVBB, chaque composante étant écrite en hexadécimal (base seize). Le tableau ci-dessous montre les principales valeurs. En ajoutant les couleurs de bases qui n'y seraient pas déjà (second tableau) et le `transparent`, toutes les couleurs composées de ces valeurs (p.ex.: `#CCFF99`, ou encore `#99CC33`, etc.) forment ce qu'il est convenu d'appeler la palette des « couleurs web » (il y en a : 1+6*6*6+8 = 225).

hexadécimal	00	33	66	99	CC	FF
décimal	0	51	102	153	204	255
proportion	0 %	20 %	40 %	60 %	80 %	100 %

Certains mots-clefs suivants peuvent servir de raccourcis, nous ne les conseillons pas, en général, dans la mesure où il faut toujours garder présent à l'esprit le fait que le rendu d'une couleur peut être très variable d'un support à l'autre,... même si la norme prévoit une correction d'affichage pour homogénéiser le rendu d'un client à l'autre. Souvenons-nous que, dans tous les cas, l'impression de couleur varie d'une culture à l'autre, même pour un support donné.

`black`	`#000000`	noir		`green`	`#008000`	vert foncé	
`gray`	`#808080`	gris foncé (50 %)		`lime`	`#0FF00`	vert vif	
`silver`	`#C0C0C0`	gris clair (75 %)		`olive`	`#808000`	marron / khaki	
`white`	`#FFFFFF`	blanc		`yellow`	`#FFFF00`	jaune franc	
`maroon`	`#800000`	pourpre		`navy`	`#000080`	outremer / bleu foncé	
`red`	`#FF0000`	rouge vif		`blue`	`#0000FF`	bleu vif	
`purple`	`#800080`	violet / mauve		`teal`	`#008080`	turquoise foncé	
`fuchsia`	`#FF00FF`	rose / fuchsia		`aqua`	`#00FFFF`	turquoise clair / bleu ciel	

D'autres écritures sont possibles, pour information :

- *#RVB* avec un seul chiffre hexadécimal par composante. Ceci équivaut au simple redoublement des chiffres : `#FC3` équivaut à `#FFCC33`.

- `rgb(rouge, vert, bleu)` où chaque composante est donnée entre 0 et 255, p.ex.: `rgb(255,204,51)`.
- `rgb(rouge, vert, bleu)` où chaque composante est donnée entre 0 % et 100 %, p.ex.: `rgb(100%,80%,20%)`.

Le niveau 3 de CSS ajoute (notamment) les écritures suivantes :

- `rgba(rouge, vert, bleu, alpha)` où chaque composante est donnée entre 0 et 255 ou entre 0 % et 100 % et l'opacité par un nombre entre 0 (opaque) et 1 (transparent).
- `hsl(teinte, saturation, luminosité)` où la teinte est un nombre entre 0 et 360 (angle en degré dans la roue chromatique), la saturation un nombre entre 0 % (noir-gris-blanc) et 100 % (saturé) et la luminosité un nombre allant de 0 % (noir) à 50 % (normal) à 100 % (blanc).
- `hsl(teinte, saturation, luminosité, alpha)` : comme le précédent, mais avec un niveau d'opacité en plus.
- `transparent` : complètement transparent.

Voici quelques exemples. On pourra utiliser le site CSS Coloratum [http://css.coloratum.com/] pour tester les couleurs et leurs écritures CSS.

```
background-color : transparent ; /* pour une cellule de tableau, par exemple */
background-color : #F0F0F0 ;     /* CSS web de ce cours: exemples de code */
background-color : white ;       /* ce n'est pas toujours la couleur par défaut
color : #800000 ;                /* CSS web de ce cours: titres */
color : black ;
```

3.2. Images et feuilles de style

Les arrière-plans peuvent également être rendus par une image. Pour cela on peut utiliser les propriétés suivantes :

- `background-color` : Il faut toujours préciser une couleur, affichée tant que (ou si) l'image n'est pas chargée par le client. Cette couleur est aussi sensée être visible à travers les parties transparentes d'une éventuelle image de fond.
- `background-image` : URL de l'image à utiliser ou `none`. Pour cela on utilise la forme : `url("adresse à utiliser")`.
- `background-repeat` : `repeat` (x et y), `repeat-x`, `repeat-y` ou `no-repeat`.
- `background-attachment` : `scroll` (collée au contenu) ou `fixed` (collée au contenant).
- `background-position` : Position de l'image par rapport au fond, généralement : `left`, `center` ou `right` suivie de `top`, `center` ou `bottom`. Il est également possible de préciser des distances (en `cm`, `px`, `%` etc. - sur les unités de mesure cf. plus loin).

Voici un exemple définissant une image de fond (soulignant une marge, p.ex.) à 24 points du bord gauche :

```
background-color: white;
background-image: url("decors/marge_ecolier.png");
background-repeat: repeat-y;
background-position: 24pt;
```

En voici un autre plaçant un logo en bas à droite d'une page et l'attachant à la fenêtre-contenant :

```
background-color: white;
background-image: url("logo.png");
```

```
background-repeat: no-repeat;
background-attachment: fixed;
background-position: 100% 100%;
```

L'écriture de ces exemples peut être condensée en utilisant une super-propriété `background`. Sa valeur est simplement une liste des valeurs des propriétés élémentaires ci-dessus, chacune étant facultative. Les exemples ci-dessus se résument ainsi :

```
background: white url("decors/marge_ecolier.png") repeat-y 24pt;
```

```
background: white url("logo.png") no-repeat fixed 100% 100%;
```

C'est pour cette raison que l'on voit parfois simplement : `background: #CCCCCC` (au lieu de `background-color`) ou encore : `background: #CCCCFF url("label.png")`.

Les images peuvent également servir, par exemple, à définir l'aspect des puces des listes (propriété `list-style-image` ou, sous forme condensée : `list-style`).

3.3. Les déclarations CSS de police de caractères

Outre les couleurs et images, l'effet le plus couramment employé est l'effet de typographie. On choisit une police de caractères, une variante, une graisse, une taille, etc. Tous ces paramètres posent problème, examinons-les un par un.

3.3.1. Choix du caractère

Tous les ordinateurs ne disposent pas des mêmes polices de caractères. C'est peu dire : même les polices de base ne correspondent pas. La même police (ou à peu près) appelée « Times » sous tel système d'exploitation est nommée « Times New Roman » sous tel autre. Qui plus est, une même police peut être affichée très différemment d'un ordinateur à l'autre, en fonction des réglages effectués par l'utilisateur. Et ceci sans parler des polices “fantaisie” dont l'audience est très limitée.

La quasi-totalité des logiciels de traitement de texte ont choisi d'évacuer purement et simplement le problème en imposant pour chaque élément textuel une et une seule police, qu'elle soit ou non disponible sur la machine de consultation. Ils évacuent ainsi tout espoir d'interopérabilité. D'autres logiciels, de mise en page principalement, disposent de tables d'équivalence entre MacOS et MSWindows et proposent à l'utilisateur de remplacer, en bloc, telle police par telle autre.

Le choix du HTML est à la fois plus souple et plus simple : il consiste à ne jamais spécifier une seule police mais toujours une liste de polices, par ordre de préférence décroissante. De plus, cette liste doit *toujours* se terminer par une famille générique : `serif` (avec empatement), `sans-serif` (sans sérif ou “baton”), `cursive` (ressemblant à du manuscrit), `fantasy` (fantaisie) ou `monospace` (à chasse fixe).

Voici un exemple tiré de la recommandation :

```
font-family: Baskerville, 'Times New Roman', "Heisi Mincho W3", Symbol, serif
```

Dans ce cas, les lettres occidentales seront tirées de la police *Baskerville* ou, à défaut, *Times*. Les caractères japonais, s'ils ne sont pas trouvés dans ces deux polices seront trouvés dans la police *Heisi*. De même, les symboles mathématiques seront trouvés dans la police *Symbol*. En dernier recours, par exemple sur un client n'ayant ni *Baskerville* ni *Times New Roman* ni etc., on utilisera la police avec empatement par défaut (définie dans les préférences du logiciel client).

Cet exemple montre, par ailleurs, que les noms de familles comportant des espaces (ou autres caractères spéciaux) doivent être placés entre guillemets (ou apostrophes).

CSS 3 ajoutera la possibilité d'utiliser une police de caractère téléchargeable à l'aide de la commande de style `@font-face` (voir la recommandation CSS fonts [http://www.w3.org/TR/css3-fonts/]). Certains dépôts, notamment celui de Google [http://www.google.com/webfonts], rendent une telle

intégration plus facile. Attention : ceci ne dispense pas d'indiquer des fontes de secours : tous les navigateurs ne mettent pas encore en œuvre cette possibilité.

3.3.2. Style d'écriture

La propriété `font-style` vaut : `normal`, `italic` ou `oblique` (pas supporté partout).

La propriété `font-variant` vaut : `normal` ou `small-caps` (pas supporté partout).

La propriété `font-weight` désigne la graisse qui varie progressivement : `100`, `200`, `300`, `400` ou `normal`, `500`, `600`, `700` ou `bold` (gras), `800`, `900` (extra-gras). Il va sans dire que sur la plupart des clients (actuels) seuls `normal` et `bold` sont rendus (et, exceptionnellement, `900`). On peut également indiquer, pour cette propriété, une valeur relative, comprise à partir du contenant : `bolder` (plus gras), `lighter` (plus maigre).

La propriété font-stretch vaut : `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `normal`, `semi-expanded`, `expanded`, `extra-expanded` ou `ultra-expanded`.

3.3.3. Taille d'écriture

La propriété CSS de taille d'écriture est nommée `font-size`. C'est généralement la propriété la plus maltraitée, les auteurs de pages web pensant généralement que leur système est configuré de la même manière que celui du lecteur, que leur navigateur est le même, la résolution également, ainsi que l'état de leurs yeux.

Pour des éléments en-ligne, nous recommandons généralement d'utiliser les valeurs relatives : `larger` (plus grand) et `smaller` (plus petit). Pour les éléments blocs, nous recommandons généralement d'utiliser les valeurs absolues : `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large` ou `xx-large`. Toutes ces valeurs se réfèrent, en dernière instances, aux paramètres du client, tels qu'ils ont été déterminés par le lecteur, en fonction de son écran, de ses habitudes, de ses goûts et de son acuité visuelle.

On peut définir des tailles relatives plus précisément en utilisant l'une des trois unités relatives suivantes (relatives à la valeur par défaut) :

- `em` : largeur d'emme « M », largeur de la plus large des lettre, ex.: `0.5em`,
- `ex` : hauteur d'ix « x », hauteur des lettres minuscules sans “jambe”, ex.: `3ex`,
- `%` : pourcentage (de la taille de police du contenant), ex.: `75%`.

Tous ces moyens de définir la taille d'écriture sont corrects parce qu'ils se réfèrent aux préférences du client. Ceci est important : si le lecteur voit mal et qu'il doit utiliser des caractères de 32 pt, on imagine mal au nom de quoi les rédacteurs de page le lui interdiraient - alors que rien ne les y contraint.

On l'aura compris, il existe d'autres moyens de définir la taille d'écriture, de façon absolue. Pour cela, on peut utiliser l'une des unités suivantes (si l'on sait ce que l'on fait) :

- `cm` : centimètre ;
- `in` : pouce (2,54 cm) ;
- `pt` : point (1/72 in) ;
- `px` : pixel - pixels d'écran, sauf pour les terminaux mobiles (cf. cours ultérieur).

3.4. Les indications CSS de formatage de blocs

Pour le formatage des blocs, nous avons évoqué plus haut les couleurs de fond et de forme. Ajoutons les propriétés suivantes définissant l'encadrement :

- `border-width` - largeur du bord : `thin` (fin), `medium` (moyen), `thick` (épais) ou une longueur (ex.: `4px`);
- `border-style` - aspect du bord : `none` (aucun), `hidden` (caché), `dotted` (pointillés), `dashed` (tirets), `solid` (ligne continue), `double` (deux lignes continues, tenant dans la largeur indiquée), `groove` (bordure enfoncée), `ridge` (bordure en relief), `inset` (bloc enfoncé), `outset` (bloc en relief) ;
- `border-color` - couleur du bord ;
- `border` : propriété rassemblant les trois précédentes, p.ex.: `border: thin dotted #999`.

On peut également particulariser chacune de ces propriétés pour les côtés du bloc en remplaçant `border` par `border-top`, `border-right`, `border-bottom` ou `border-left`. Voici en exemple la règle de style pour les éléments de code de ce cours (version web) :

```
.programlisting {
   background: #F0F0F0;    /* couleur de fond */
   border: 1px solid gray; /* bordure */
   padding: 2px;           /* marge intérieure, cf. ci-après */
   white-space: pre;       /* préserver les espaces */
}
```

Les blocs délimitent quatre espaces. Le plus intérieur est celui du contenu. Entre le contenu et la bordure on trouve la marge intérieure ou *padding*. On trouve ensuite la bordure, qui peut avoir une épaisseur non négligeable. Enfin, le bloc est encadré par une marge extérieure ou *margin*, sachant que les marges de deux blocs peuvent se recouvrir. Ces dimensions sont réglables par les propriétés suivantes :

- `width`, `height` - largeur et hauteur du contenu ;
- `padding` (et `padding-top`, `padding-right`, `padding-bottom`, `padding-left`) - épaisseur de la marge intérieure ;
- `border-width` (et `border-top-width`…) - largeur du bord (cf. ci-dessus) ;
- `margin` (et `margin-top`…) - épaisseur de la marge extérieure.

Voici quelques exemples :

```
body { margin: 2em }         /* toutes les marges à 2em */
body { margin: 1em 2em }     /* haut : 1em, droite : 2em, bas = haut, gauche =
body { margin: 1em 2em 3em } /* haut : 1em, droite : 2em, bas : 3em,  gauche =
```

L'arrangement du contenu est déterminé principalement par les deux propriétés suivantes :

- `text-align` - positionnement du texte : `left` (à gauche), `center` (centré), `right` (à droite), `justify` (justifié) ; pour les tableaux, il est aussi possible de préciser `","` (ou `"."`) pour un alignement (par colonne) sur les virgules (ou les points), mais les navigateurs ne le prennent pas encore en compte ;
- `text-indent` - retrait de la première ligne d'un bloc (en plus de la marge intérieure), p.ex.: `text-indent: 3em`.

Enfin, la position du bloc est déterminée par plusieurs propriétés sur lesquelles nous reviendrons ultérieurement, qui permettent d'établir une mise en page :

- `display` : nature de l'élément : bloc, en-ligne, bloc en-ligne, etc. ;
- `position` : mode de positionnement de l'élément par rapport à son contexte ;
- `z-index` : pour ordonner plusieurs éléments du même contexte ;

- `float`, `clear` : pour placer un élément dans la marge courante ou libérer cette marge ;
- `top`, `right`, `bottom`, `left` : pour positionner un élément précisément dans le contexte.

3.5. Autres indications CSS usuelles

Les autre attributs de style les plus usuels sont les suivants :

- `text-decoration` - "décoration" du texte : `none` (aucune), `underline` (trait en dessous), `overline` (trait au-dessus), `line-through` (trait par dessus), `blink` (clignotement, à utiliser avec circonspection) ; ceci permet de ne pas souligner un lien, par exemple : `a:link { text-decoration: none }` ;
- `list-style` - aspect d'une liste : numérotée, à puce, quelle puce, etc.

Notez, toutefois, que nous n'avons évoqué qu'une partie seulement des attributs possibles. De nombreux autres existent, qui permettent de régler plus finement tel ou tel détail.

Notez également que les attributs disponibles dépassent le cadre de CSS2 puisque certains navigateurs reconnaissent également des attributs définissant l'aspect des barres de défilement ou même indiquant que telle portion de texte est éditable dans la fenêtre-même du navigateur.

4. Règles de style complexes

4.1. Classes (CSS 1)

Nous avons déjà évoqué les classes HTML. Leur nom apparaît tel quel dans l'attribut HTML `class` et précédé d'un point dans les feuilles de style. Voici un exemple :

```
/* feuille de style */
.avecAlinea { text-indent: 2cm }

<!-- document HTML -->
<p class="avecAlinea">Nous avons déjà…</p>
```

Il est possible, toutefois, d'affiner l'usage de ces classes pour désigner expressément les paragraphes (`<p>`) de classe `avecAlinea` et les blocs de citation (`<blocquote>`), afin de les différencier. Pour cela on accole simplement le nom de balise et le sélecteur de classe dans la feuille de style. Voici un exemple :

```
/* feuille de style */
p.avecAlinea { text-indent: 2cm }         /* paragraphe de classe «avecAlinea»
blocquote.avecAlinea { text-indent: 1cm } /* citation de classe «avecAlinea» */
.avecAlinea { text-indent: 2cm }          /* n'importe quoi de classe «avecAlin

<!-- document HTML -->
<p class="avecAlinea">Nous avons déjà…</p>
<blocquote class="avecAlinea">Lorem ipsum dolor…</blocquote>
```

Il faut donc comprendre la notation `.avecAlinea` comme « n'importe quoi de classe `avecAlinea` ». Le point, dans un sélecteur CSS signifie, en particulier, « …de classe… ».

4.2. Imbrication au sens large (CSS 1)

Imaginons maintenant que nous voulions redéfinir la puce des listes d'un document. Nous pouvons poser :

```
ul { list-style-image: url("puce-liste-12x12.png") }
```

Le problème est qu'alors les listes à puce de niveau 2 ou plus (des listes apparaissant dans des listes) auront aussi cette puce, puisqu'elles utilisent également la balise `<ul>`.

Pour résoudre ce problème, la norme CSS 1 introduit la notion d'inclusion de sélecteurs. Cette inclusion se note simplement par un espace. Voici une liste d'exemples :

- `h1 strong` : un `<strong>` qui est dans un `<h1>` (si les `<h1>` sont gras, il peut être judicieux de rendre `<strong>` autrement dans ce contexte),
- `h1 .valeur` : quelquechose de classe `valeur` qui est dans un `<h1>` (si `h1` et `.valeur` utilisent une couleur proche, il peut être bon de les distinguer ainsi),
- `ul ul` : une liste `<ul>` qui est dans une autre liste `<ul>` (liste de second niveau, au moins),
- `ul ul ul` : une liste de troisième niveau (au moins),
- `em em` : une mise en valeur `<em>` qui est dans une autre (il serait judicieux d'enlever l'italique).

Voici une utilisation possible de ces exemples :

```
h1 { text-weight: bold; color: #800000; }
h1 b { text-weight: normal; }
.valeur { text-weight: bold; color: #800000; }
h1 .valeur { color: #336666; }
ul { list-style-image: url("puce-liste-12x12.png") }
ul ul { list-style-image: url("puce-liste-10x10.png") }
ul ul ul { list-style-image: url("puce-liste-8x8.png") }
em { font-style: italic }
em em { font-style: normal }
```

On prendra garde de distinguer scrupuleusement les trois écriture suivantes :

- `abc, def { xyz }` : la déclaration `xyz` s'applique à `<abc>`, d'une part, et à `<def>`, d'autre part ;
- `abc def { xyz }` : la déclaration `xyz` s'applique à toutes les `<def>` qui sont dans une `<abc>` ;
- `abc.def { xyz }` : la déclaration `xyz` s'applique à toutes les `<abc>` de classe `def`.

4.3. Pseudo-classes de liens (CSS 1, revues par CSS 2)

La balise `<a>` (en CSS 3 toutes les balises) se combine à quatre pseudo-classes qui s'utilisent ainsi :

```
a:visited { color: blue }    /* lien déjà visité */
a:link    { color: red }     /* lien pas encore visité (attention !) */
a:active  { color: lime }    /* lien actif (sélectionné au pointeur) */
a:hover   { color: fuchsia } /* CSS 2 : lien actuellement survolé */
a:focus   { border: 1px }    /* CSS 2 : le lien reçoit les événements clavier */
```

En CSS 1, seuls les trois premières sont définies. Les deux dernières ont été ajoutées à CSS 2 (mais `:hover` était en usage dès avant cette recommandation).

En CSS 1 les rapports entre `:link`/`:visited` et `:active`/`:hover` n'étaient pas clairs. Ils ont été précisé en CSS 2 : ces deux types de pseudo classes se combinent. On peut ainsi être visité et inactif, non-visité et désigné (`:hover`), visité et actif, etc. Pour une raison de prévalence que nous expliquerons plus loin, il est donc nécessaire que `:link` et `:visited` apparaissent avant `:active` et `:hover` (ou alors, il faut utiliser des combinaisons explicites, telles que `:link:hover`, par exemple).

Les pseudo-classes se combinent avec les classes comme ceci : `a.externe:visited`.

4.4. Pseudo-éléments CSS 1

CSS 1 introduit également la notion de pseudo-éléments qui représentent des balisages virtuels.

Le pseudo-élément :first-line, qui s'applique à un bloc, désigne sa première ligne. L'exemple suivant mettra en petites majuscules la première ligne de chaque paragraphe :

```
p:first-line { font-variant: small-caps }
```

Le pseudo-éléments `:first-letter`, lui, désigne la première lettre d'un élément. L'exemple suivant produira une lettrine sur tous les paragraphes (`float` permet de placer la lettrine dans la marge gauche du paragraphe - il va sans dire que l'effet est désastreux sous les paragraphes qui n'ont qu'une ligne !) :

```
p:first-letter { font-size: 200%; float: left }
```

Ces pseudo-éléments se combinent, eux-aussi, avec les autres sélecteurs. Ainsi `.avecLettrine:first-letter` désigne la première lettre de n'importe quel élément de classe `avecLettrine`.

En CSS 3 les pseudo-éléments s'écrivent avec deux foix deux points : `::first-line`, `::first-letter`. On verra d'autres pseudo-éléments plus loin.

4.5. Règles de sélection complexe (CSS 2)

CSS 2 commence par étendre la notion de classe du HTML. En effet, en CSS 1, une balise donnée ne peut indiquer qu'une classe, au plus, alors qu'en CSS 2 une balise peut relever d'autant de classes que nécessaire. Pour cela, il suffit de lister ces classes dans l'attribut `class`, en séparant leurs noms par des espaces. Voici un exemple :

```
<p class="avecAlinea cursif">Lorem ipsum…</p>
```

Dans ce cas, on peut imaginer des règles a plusieurs sélecteurs :

```
.avecAlinea { text-indent: 2cm }
.cursif { font-family: cursive }
p.avecAlinea.cursif { text-indent: 1.8cm }
```

CSS 2 ne se restreint pas au cadre (X)HTML mais a été conçu pour s'appliquer à n'importe quel document XML, qu'il relève ou non d'un type de document. Or, en général, les documents XML n'ont pas nécessairement la notion de classe et quand un type de document dispose d'une notion de classe, celle-ci n'est pas nécessairement désignée par un attribut nommé « `class` ». CSS 2 va donc travailler avec n'importe quel attribut, à l'aide des sélecteurs suivants :

- `[`*`attribut`*`]` : sélectionne n'importe quelle balise disposant de l'attribut en question ;

  p.ex.: `a[title] { text-decoration: overline }` (les `<a>` qui ont leur balise title apparaissent avec une barre au-dessus, au lieu de leur aspect par défaut) ;

- `[`*`attr`*`="`*`val`*`"]` : sélectionne n'importe quelle balise dont l'attribut nommé « *`attr`* » possède la valeur « *`val`* » ; voici un exemple tiré de la recommandation :

  ```
  dialogue[personnage="Roméo"]    { voice-family: "Lawrence Olivier", charles, m
  dialogue[personnage="Juliette"] { voice-family: "Vivien Leigh", victoria, fema
  ```

- `[`*`attribut`*`~="`*`valeur`*`"]` : sélectionne n'importe quelle balise dont l'attribut indiqué est une liste de mots séparés par des espaces et dont l'un au moins est la valeur indiquée ; ainsi `[class~="abc"]` est un synonyme de `.abc` ; ceci permet de reproduire le comportement des classes du HTML pour n'importe quel fichier XML.

Dans certains cas, il peut être utile de sélectionner un élément `<b2>` immédiatement à l'intérieur d'un élément `<b1>`, c'est à dire les cas `<b1>...<b2>...</b2>...</b1>` à l'exclusion des cas `<b1>...<x>...<b2>...</b2>...</x>...</b1>`. On dit, dans ce cas, que `<b2>` est un fils de `<b1>`. Ceci se note de la façon suivante :

```
b1 > b2 { ... } /* b2 est fils de b1 */
```

Pour sélectionner un élément `<b2>` qui suit immédiatement un élément `<b1>`, c'est à dire les cas `<b1>...</b1><b2>...</b2>`, on peut utiliser la notation suivante :

```
b1 + b2 { ... } /* b2 vient après b1 */
```

Signalons un dernier[5] ajout de CSS 2 : la possibilité d'ajouter du contenu avant ou après n'importe quel élément. On utilise pour cela les pseudo-éléments `:before` (avant) et `:after` (après), qui fonctionnent sensiblement comme `:first-letter` vu plus haut, ainsi que la propriété `content` (contenu). Voici un exemple (nous renvoyons à la recommandation pour le fonctionnement des compteurs, qui fonctionnent encore très mal) :

```
h1:before { content: "Chapitre " counter(chapno, upper-roman) ". " }
.attention:before { content: "Attention ! " }
p.attention:first-letter { color: #FFCC00 }
```

5. Conflits entre des règles

La plupart des exemples de ce chapitre nous donnent des cas concrets de conflits entre règles, que nous avons provisoirement négligés. Considérons par exemple une liste de second niveau ; au sens strict deux règles sont susceptibles de s'appliquer : « `ul {…}` » et « `ul ul {…}` ». Il faut un moyen de les départager. De même, si deux feuilles de style décrivent une propriété des balises `<p>`, laquelle dervra s'appliquer ?

La norme CSS 1, révisée par CSS 2, donne la règle suivante de résolution des conflits :

1) Trouver toutes les déclarations susceptibles de s'appliquer à tel élément donné. S'il n'y en a pas, la propriété est héritée du contenant (quand cela a un sens, c'est à dire principalement pour la mise en forme).

2) Trier par origine : les feuilles de l'auteur l'emportent sur celles de l'utilisateur, qui l'emportent sur les valeurs par défaut du logiciel client[6].

3) Trier par la *spécificité* du sélecteur, c'est à dire par le niveau de généralité de la règle. Plus un sélecteur est spécifique, c'est à dire moins la règle est générale, plus celle-ci a de force. La spécificité se calcule ainsi : le nombre d'identificateurs du sélecteur sert de rang des centaines (très spécifique), le nombre de classes[7] sert de rang des dizaines et celui de balises[8] (très général) sert de rang des unités[9]. Voici des exemples tirés de la recommandation :

```
*                 /* spécificité =   0 0 0 (aucune : absolument général) */
li {...}          /* spécificité =   0 0 1 (faible : très général) */
li:firt-line {...} /* spécificité =   0 0 2 */
ul li {...}       /* spécificité =   0 0 2 */
ul ol li {...}    /* spécificité =   0 0 3 */
li.red {...}      /* spécificité =   0 1 1 */
a:hover {...}     /* spécificité =   0 1 1 */
```

[5] Dernier pour ce qui concerne ce cours. Il y en a, en fait, d'autres, qu'il n'est pas le lieu ici de décrire (élément apparaissant immédiatement après tel autre, sélection par langue, etc.). Cf. la recommandation, si nécessaire, § 5.1.

[6] En fait un utilisateur peut imposer absolument certaines règles de style à l'aide du marqueur `!important`, que nous n'avons pas présenté ici (CSS 2).

[7] Les pseudo-classes comptent comme des classes, de la même façon que les sélections par attributs (entre crochets, cf. ci-dessus).

[8] En CSS 2.1 les pseudo-éléments comptent comme des balises.

[9] Si une règle de style est placée dans un attribut `style` (indication de style en-ligne) plutôt que dans une feuille de style (incorporée ou externe), elle a forcément priorité. Cela revient à considérer que les attributs de `style` comptent pour une spécificité de 1 000.

```
ul ol li.red {...} /* spécificité =   0 1 3 */
#x34y {...}        /* spécificité =   1 0 0 (forte : très spécifique) */
style="..."        /* spécificité = 1 0 0 0 (complètement spécifique) */
```

4) Enfin trier par ordre de définition : quand deux règles ont la même origine et la même spécificité, la dernière définie l'emporte. Si une feuille A inclut une feuille B, la feuille B est considérée comme antérieure (donc plus générale, donc moins prioritaire).

6. Présentations différentes pour supports différents (CSS 2)

La manière de prendre en compte la diversité des supports sur lesquels un même document (muni d'une feuille de style donnée) peut être rendu évolue considérablement d'un niveau à l'autre du CSS.

CSS 2 définit un certain nombre de supports possibles dont voici les principaux :

- `all` : tous,
- `braille` : afficheur tactile Braille,
- `embossed` : imprimante Braille,
- `handheld` : téléphone cellulaire ou assistant personnel (faible résolution/taille d'écran, bande passante limitée),
- `print` : imprimante et apperçu avant impression (support opaque),
- `projection` : présentations, diapositives, transparents,
- `screen` : écran d'ordinateur,
- `tv` : télévision (faible résolution, capacité de défilement limitée).

En pratique seuls `screen` et `print` sont réellement utilisés par les concepteurs de sites web. CSS 3 prolonge cette méthode en permettant de désigner des règles de style applicables à des supports de dimension inférieure (resp. supérieure) à telle ou telle valeur, d'orientation portrait (verticale) ou paysage (horizontale), couleur ou noir et blanc, etc. (cf. la recommandation [http://www.w3.org/TR/css3-mediaqueries/]). Cette méthode est très commode pour adapter une mise en page à de nombreux terminaux de consultation. Nous y reviendrons dans un cours ultérieur.

Pour indiquer que telles règles de style concernent tels supports particuliers on peut utiliser une construction spéciale `@media` à l'intérieur d'une feuille de style de la façon suivante :

```
li { list-style-image: … }   /* règle applicable à tous les supports */

@media print {               /* règles pour l'impression seule */
  body { font-size: 10pt }
  .avecAlinea { text-indent: 1.5cm }
  #menu, #pub { display: none }
}

@media screen {              /* règles pour l'écran seul */
  body { font-size: 12pt }
  .avecAlinea { text-indent: 30px }
}

@media screen, print {       /* règles pour l'écran et l'impression */
  body { line-height: 1.2 }
}
```

Toutefois les éditeurs *web* actuels (dont Dreamweaver, jusqu'à la version 8 au moins) ne reconnaissent pas ou mal cette syntaxte, qui présente, par ailleurs, des risques de confusion pour les développeurs et des problèmes de compatibilité avec certains navigateurs. Une solution consiste à créer des feuilles de style spécifiques à tel ou tel medium et d'utiliser une forme spéciale de l'instruction d'importation :

```
/* dans la feuille de style principale */
@import url("ecran.css") screen, projection, handheld, tv;
@import url("impression.css") print;
```

Troisième possibilité : lier à une page plusieurs feuilles de styles spécifiant les médias, comme suit[10].

```
<!-- dans l'en-tête HTML -->
<link rel="stylesheet" type="text/css"                href="styles/general.css"
<link rel="stylesheet" type="text/css" media="print"  href="styles/impression.c
<link rel="stylesheet" type="text/css" media="screen" href="styles/ecran.css" /
```

Pour ceux qui souhaitent soigner leur feuille de style pour l'impression, nous renvoyons à la recommandation [CSS 2.1, chap. 13], qui décrit les ajustement possibles des sauts de pages, de la gestion des lignes veuves et orphelines et de la mise en page.

On prendra garde au fait que toutes ces constructions ne fonctionnent pas forcément très correctement sur tous les navigateurs. En cas de besoin, on pourra utiliser une feuille de style adaptée aux insuffisances d'Internet Explorer en plaçant le contournement suivant dans l'en-tête HTML[11]:

```
<!--[if lte IE 6]>
  <link rel="stylesheet" type="text/css" href="styles/ie-jusqu-a-v6.css" />
<![endif]-->
```

[10]Notons que Dreamweaver MX ne fait pas la différence entre les types supports (Dreamweaver 8 le fait correctement). Pour éviter des difficultés pratiques, il est conseillé, dans ce cas, d'utiliser la fonction « masquer » dans la rubrique « feuilles de style à la conception ».
[11]Cf. le site Quirksmode [http://www.quirksmode.org/css/condcom.html] pour plus de détails.

Index

X