Conception de sites Web

ENSICAEN
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS DE CAEN
& CENTRE DE RECHERCHE

Luc Brun

Table des matières

Chapitre 1

Introduction

Un site Web distribue des fichiers sous différents formats à travers le réseau Internet en utilisant le protocole HTTP. Le format HTML est un format de fichier permettant de mettre en forme des documents. Il convient dès à présent de bien distinguer :

- Internet, le réseau mondial utilisé pour de nombreuses applications *dont* la navigation Web.
- Le protocole HTTP utilisé par les navigateurs pour communiquer avec les serveurs de sites Web.
- Le format HTML qui est un langage permettant de décrire le contenu de pages Web.

La description du protocole Internet ne fait pas l'objet de ce cours. Nous allons toutefois présenter brièvement le protocole HTTP de façon à donner une petite idée de la mécanique permettant de faire fonctionner les navigateurs.

1.1 Le protocole HTTP

Le protocole HTTP (HyperText Transfer Protocol) est le protocole le plus utilisé sur Internet depuis 1990. La version 0.9 était uniquement destinée à transférer des données sur Internet (en particulier des pages Web écrites en HTML. La version 1.0 du protocole (la plus utilisée) permet désormais de transférer des messages avec des en-têtes décrivant le contenu du message en utilisant un codage de type MIME.

Le but du protocole HTTP est de permettre un transfert de fichiers (essentiellement au format HTML) localisé grâce à une chaîne de caractères appelée URL entre un navigateur (le client) et un serveur Web (appelé d'ailleurs httpd sur les machines UNIX)

1.1.1 Communication entre navigateur et serveur

La communication entre le navigateur et le serveur se fait en deux temps :

- Le navigateur effectue une requête HTTP
- Le serveur traite la requête puis envoie une réponse HTTP.

En réalité la communication s'effectue en plus de temps si on considère le traitement de la requête par le serveur.

1.1.2 Requête HTTP

Une requête HTTP est un ensemble de lignes envoyées au serveur par le navigateur. Elle comprend :

- une ligne de requête : c'est une ligne précisant le type de document demandé, la méthode qui doit être appliquée, et la version du protocole utilisée. La ligne comprend trois éléments :
  1. La méthode,
  2. L'URL,
  3. La version du protocole utilisé par le client (généralement HTTP/1.0).
- Les champs d'en-tête de la requête : il s'agit d'un ensemble de lignes facultatives permettant de donner des informations supplémentaires sur la requête et/ou le client (Navigateur, système d'exploitation,...).
- Le corps de la requête : C'est un ensemble de lignes optionnel devant être séparé des lignes précédentes par une ligne vide et permettant par exemple un envoi de données par une commande POST lors de l'envoi de données au serveur par un formulaire

Voici donc un exemple de requête HTTP :

```
GET http://www.commentcamarche.net HTTP/1.0

Accept : text/html

If-Modified-Since : Saturday, 15-January-2000 14:37:11 GMT

User-Agent : Mozilla/4.0 (compatible; MSIE 5.0; Red Hat 9.0)
```

La liste des commandes est la suivante :

Commande	Description
GET	Requête de la ressource située à l'URL spécifié
HEAD	Requête de l'entête de la ressource située à l'URL spécifié
POST	Envoi de données au programme située à l'URL spécifié
PUT	Envoi de données à l'URL spécifié
DELETE	Suppression de la ressource située à l'URL spécifié

1.1.3 Réponse HTTP

Une réponse HTTP est un ensemble de lignes envoyées au navigateur par le serveur. Elle comprend :

1. une ligne de statut : c'est une ligne précisant la version du protocole utilisé et l'état du traitement de la requête à l'aide d'un code et d'un texte explicatif. La ligne comprend trois éléments :
   (a) La version du protocole utilisé
   (b) Le code de statut
   (c) La signification du code.
2. Les champs d'en-tête de la réponse : il s'agit d'un ensemble de lignes facultatives permettant de donner des informations supplémentaires sur la réponse et/ou le serveur. Chacune de ces lignes est composée d'un nom qualifiant le type d'en-tête, suivi de deux points (:) et de la valeur de l'en-tête.
3. Le corps de la réponse : Il contient le document demandé.

Une réponse HTTP a donc la syntaxe suivante :

```
VERSION-HTTP CODE EXPLICATION<crlf>

EN-TETE : Valeur<crlf>

.

.

EN-TETE : Valeur<crlf>

Ligne vide<crlf>

CORPS DE LA REPONSE
```

Voici donc un exemple de réponse HTTP :

```
HTTP/1.0 200 OK

Date : Sat, 15 Jan 2000 14:37:12 GMT

Server : Apache/2.0

Content-Type : text/html

Content-Lentgh : 1245

Last-Modified : Fri, 14 Jan 2000 08:25:13 GMT
```

La ligne `Content-type :` spécifie le format du document qui sera envoyé. Le serveur peut donc envoyer d'autres documents que des fichiers HTML. Plus précisément, la valeur qui suit le mot clé `Content-type :` spécifie un type MIME (Multipurpose Internet Mail Extensions).

1.1.4 Les types MIME

Le type MIME est un standard qui a été proposé par les laboratoires Bell Communications en 1991 afin d'étendre les possibilités du courrier électronique (mail), c'est-à-dire de permettre d'insérer des documents (images, sons, texte, ...) dans un courrier.

Depuis, le type MIME est utilisé d'une part pour typer les documents attachés à un courrier mais aussi pour typer les documents transférés par le protocole HTTP. Ainsi lors d'une transaction entre un serveur Web et un navigateur internet, le serveur Web envoie en premier lieu le type MIME du fichier envoyé au navigateur, afin que ce dernier puisse savoir de quelle manière afficher le document.

Un type MIME est constitué de la manière suivante :

```
Content-type: type_mime_principal/sous_type_mime
```

Une image GIF a par exemple le type MIME suivant :

```
Content-type: image/gif
```

Les principaux types MIME

application/octet-stream	Fichiers binaires non interprétés	bin
application/pdf	Fichiers Adobe Acrobat	dwg
application/postscript	Fichiers PostScript ai,eps,	ps
application/rtf	Format de texte enrichi	rtf
application/x-csh	Script C-Shell (UNIX)	dwg
application/x-dvi	Fichiers texte dvi	dvi
application/x-latex	Fichiers Latex	latex
application/x-sh	Script Bourne Shell	dwg
application/x-tcl	Script Tcl	tcl
application/x-tex	fichiers Tex	tex
application/x-texinfo	Fichiers eMacs texinfo,	texi
application/x-gtar	Tar GNU	gtar
application/x-shar	Archives Shell	shar
application/x-tar	Fichiers compressés tar	tar
application/x-ustar	Fichiers compressés tar Posix	man
application/zip	Fichiers compressés ZIP	man
audio/basic	Fichiers audio basiques au,	snd
audio/x-wav	Fichiers audio Wave	wav
image/gif	Images gif	man
image/jpeg	Images Jpeg jpg,jpeg,	jpe
image/tiff	Images Tiff tiff,	tif
image/x-portable-anymap	Fichiers Anymap PBM	pnm
image/x-portable-bitmap	Fichiers Bitmap PBM	pbm
image/x-portable-graymap	Fichiers Graymap PBM	pgm
image/x-portable-pixmap	Fichiers Pixmap PBM	ppm
image/x-rgb	Image RGB	rgb
image/x-xbitmap	Images Bitmap X	xbm
image/x-xpixmap	Images Pixmap X	xpm
multipart/x-zip	Fichiers archive zip	zip
multipart/x-gzip	Fichiers archive GNU zip gz,	gzip
text/html	Fichiers HTML htm,	html
text/plain	Fichiers texte sans mise en forme txt,g,h,c,cc,hh,m,	f90
text/richtext	Fichiers texte enrichis	rtx
video/mpeg	Vidéos MPEG mpeg,mpg,	mpe
video/quicktime	Vidéos QuickTime qt,	mov
video/msvideo	Vidéos Microsoft Windows	avi
video/x	-sgi-movie Vidéos MoviePlayer	movie

1.2 Gestion des droits - Répertoires

Nous avons vu dans les sections précédentes que les navigateurs accédent au contenu d'un site grâce à des échanges avec un serveur Web en utilisant le protocole HTTP. Les pages personnelles des utilisateurs sont généralement accessibles à travers des url du type :

```
http://www.ismra.fr/~login
```

Ces pages sont physiquement stockées dans le répertoire d'accueil de l'utilisateur sous `public_html`. Par défaut, si seul le login de l'utilisateur ou un nom de répertoire est spécifié, le serveur charge un fichier nommé `index.html` ou `index.php` si index.html n'existe pas.

Il est important de comprendre que le serveur tourne en tant que deamon et accède à vos page comme un utilisateur «normal». Il faut donc que votre répertoire public_html ainsi que l'ensemble des sous répertoires et fichiers que vous désirez publier sur le Web soient accessibles en lecture pour l'ensemble des utilisateurs. Ceci est réalisé grâce aux commandes chmod.

```
chmod a+rx repertoire (ou cgi)
chmod a+r fichier
```

Si (et uniquement si) l'ensemble des sous répertoires de public_html est destiné à être publié sous le Web, le changement de droits peut être automatisé à l'aide de la commande find :

```
find ~/public_html -type d -exec chmod a+rx {} \;
find ~/public_html -type f -exec chmod a+r {} \;
```

Ces deux commandes parcourent l'arborescence et changent récursivement les droits.

Chapitre 2

Les format HTML/XHTML

H.T.M.L. (HyperText Markup Langage) est un format d'écriture de document du type SGML(Standard Generalized Markup Language). Ce langage est défini par le W3C (World Wide Web consortium), organisme indépendant chargé de la normalisation et de la recherche sur la technologie Web.

En d'autres termes, HTML est un ensemble (réduit) de balises (ou styles ou "tags") utilisés pour définir les différents composants d'un document Web.

Les différentes balises de HTML utilisent une syntaxe SGML, on entre dans un environnement en le citant borné par les caractères < et >, on le quitte en le nommant précédé d'un caractère /. Certaines balises ne définissent pas d'environnement (retour à la ligne, trait horizontal...), dans ce cas le nom de la balise est encadré par des <, > et terminé par un / (e.g. < *br/* >).

Les documents HTML sont des textes ASCII ponctués par des séquences <balise>et</balise>

Comme dans les exemples suivants :

```
<title>Exemple de document HTML</title>
<hr/>
<h1>Important !</h1>
<hr/>

<table><tr><td>Bonjour</td></tr></table>
```

Différentes versions :

HTML 1.0 : C'est la toute première version

HTML 3.2 : C'est la version utilisée actuellement (quelques "vieux" navigateurs ne supportent pas encore toutes ses fonctionnalités)

HTML 4.0 (Décembre 97) : n'est pas encore implémenté sur la plupart des navigateurs

2.1 Structure d'un document

Un document HTML se décompose en trois sections principales :

1. Une entête spécifiant la version de HTML utilisée.
2. La section head qui code des informations générales sur le document (titre, auteur, mots clés...).
3. la section body qui code le contenu proprement dit du document.

2.1.1 Version de HTML

La version du format HTML utilisée est généralement fournie en tête de document par la commande $<!DOCTYPE>$. Par exemple, on peut spécifier le format HTML 4.O par l'entête suivante au document HTML :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
```

Notons que pour assurer la transition vers HTML 4.0, le w3C (organisme régulateur) a prévu différentes spécification correspondant à une conformité plus ou moins stricte à la norme HTML 4.0. On a notamment les deux spécifications suivantes :

le HTML 4.01 strict : Conformité stricte à la norme :

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

le HTML 4.01 de transition : Comprend la nouvelle norme et de nombreuses autres conventions maintenant abandonnées :

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

Notons toutefois que la balise doctype peut être omise. Il est toutefois recommandé d'indiquer au navigateur avec qu'elle version de HTML on compte travailler.

2.1.2 La section head

La section head contient des informations générales sur le document (titre, auteur, mots clés...) qui sont utilisées par les moteurs de recherche. Plus précisément, on peut trouver dans la section head les balises suivantes :

Le titre d'une page représente le texte généralement affiché dans la barre supérieure du navigateur. Ce titre est spécifié par les balises $< title >\ldots< /title >$

Le lien link spécifie un lien vers une feuille de style (voir Chapitre 3)

Les méta données : principalement utilisées par les moteurs. Ces balises se présentent sous la forme :

```
<meta name="nom attribut" content="valeur"/>
```

Les principaux noms de balise sont les suivants :

title : titre du document,

description : description du document,

keywords : mots clés,

author : nom de l'auteur

Notons également l'existence de la balise :

`<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">`

permettant de spécifier le jeu de caractères utilisés dans le document.

Une section head se présente donc de la façon suivante :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Un exemple de section head</title>
<link rel="stylesheet" type="text/css" href="mon_css.css"/>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<meta name="keywords" content="html,head"/>
<meta name="author" content="Luc Brun"/>
<meta name="description" content="description de la section head
dans le cadre du cours sur les sites Web dynamiques a l ENSICAEN"/>
</head>
<body>
</body>
</html>
```

2.1.3 La section body

La section body décrit le contenu d'un document. Les anciennes versions de HTML utilisaient de nombreux attributs de la balise body pour spécifier la police de caractères, la couleur du fond, la couleur des liens. . .. Actuellement, le w3c recommande de spécifier ces attributs à l'aide d'une feuille de style. Les seuls attributs restants vraiment utiles sont donc à présent :

onload : action à exécuter lors du chargement de la page,

onunload : action à exécuter lors du déchargement de la page (passage à une autre feuille)

Ces actions sont généralement spécifiées à l'aide de code Javascript (Section 4).

2.2 Formatage de textes

Le texte d'une page HTML se tape en séquence dans la section body. Notons toutefois que si l'on ne spécifie pas la police utilisée, on est limité aux caractères ASCII. Il convient alors d'utiliser des séquences de substitution (Table 2.1). Bien

192	&Agrave ;	À
198	&AElig ;	Æ
199	&Ccedil ;	Ç
200	&Egrave ;	È
201	&Eacute ;	É
214	&Ouml ;	Ö
216	&Oslash ;	Ø
224	&agrave ;	à
225	&aacute ;	á
226	&acirc ;	â
227	&atilde ;	ã
228	&auml ;	ä
229	&aring ;	å
230	&aelig ;	æ
231	&ccedil ;	ç
232	&egrave ;	è
233	&eacute ;	é
234	&ecirc ;	ê
235	&euml ;	ë
236	&igrave ;	ì
237	&iacute ;	í
238	&icirc ;	î
239	&iuml ;	ï
241	&ntilde ;	ñ
242	&ograve ;	ò
243	&oacute ;	ó
244	&ocirc ;	ô
245	&otilde ;	õ
246	&ouml ;	ö
248	&oslash ;	ø
249	&ugrave ;	ù
250	&uacute ;	ú
251	&ucirc ;	û
252	&uuml ;	ü
253	&yacute ;	ý

TAB. 2.1 – Liste des principales séquences de caractères utilisées pour coder les lettres accentuées

que plus lourdes à utiliser, ces séquences sont recommandées pour une meilleure portabilité.

Les retours chariots ne sont pas pris en compte dans le code HTML et doivent être codés explicitement à l'aide de balise `<br/>`. Un exemple de page Web ressemble donc à :

```
<body>
Ceci est ma premi&egrave;re page web<br/>

J'utilise pour cela un &eacute;diteur de texte quelconque</br>
</body>
```

2.2.1 Les paragraphes

Un paragraphe est une partie d'un document HTML encadré par les balises <p> et </p>. Un paragraphe contient usuellement du texte. Un paragraphe peut être aligné de façon spécifique en utilisant l'attribut align qui peut prendre les valeurs {left,center,right} permettant respectivement d'aligner à gauche, de centrer et d'aligner à droite. Notons que la valeur de l'attribut align peut également être spécifiée dans une feuille de style ce qui est généralement préférable (voir section 2.10). Un exemple de page HTML utilisant des paragraphes se présente donc comme suit :

```
<body>
<p align="right">
Le 9 juillet 2004
</p>
<p align="left">
Brun Luc<br/>
ENSICAEN<br/>
</p>
<p align="center">
&agrave;</p>
<p align="right">
Mr. Jean Dupont
</p>
</body>
```

le 9 juillet 2004

Brun Luc
ENSICAEN

à

Mr. Jean Dupont

2.2.2 Les divisions

Il est parfois utile d'appliquer un même alignement ou un même style à un ensemble d'objets (comme des paragraphes). Pour cela, la norme HTML nous fournie la balise `<div>` qui admet l'attribut align avec comme valeurs {left,center,right} (comme pour les paragraphes (Section 2.2.1)). L'utilisation de div permet donc de regrouper un ensemble de paragraphes. Cette propriété sera très utile lorsque nous examinerons les feuilles de style (Chapitre 3). Notons également que l'usage de l'attribut align n'est pas recommandé pour la balise `<div>` (Section 2.10).

```
<body>
<div align="right">
<p> Ceci est un premier paragraphe</p>
<p> et un second</p>
<p>tous align&eacute;s
&agrave; droite
</p>
</div>
</body>
```

Ceci est un premier paragraphe
et un second
tous alignés à droite.

2.2.3 Les lignes horizontales

La commande `<hr/>` permet d'afficher des lignes horizontales ressemblant à ceci :

Les attributs de la commande `<hr/>` sont :

align	:	alignement de la ligne : valeurs {left,center,right}
width	:	largeur de la ligne (par défaut la largeur de la fenêtre)
size	:	épaisseur de la ligne
noshade	:	ne pas appliquer d'ombrage sur la ligne (pas de valeur)

2.2.4 Modifications de polices

HTML à prévu plusieurs balises pour modifier l'apparence d'un texte. On trouve notamment :

`<b></b>` : positionne une texte en **gras**
`<i></i>` : positionne une texte en *italiques*
`<big></big>` : positionne un texte dans une police de taille supérieure.
`<small></small>` : positionne un texte dans une police de taille inférieure.
`<u></u>` : permet de souligner une portion de texte

Notons également que les commandes `<strong></strong>` et `<em></em>` correspondent à deux niveaux logiques de mise en évidence du texte. La commande `<strong></strong>` correspond généralement à du gras tandis que la commande `<em></em>` correspond à de l'italique. Toutefois ce choix peut varier en fonction du navigateur. on indique simplement par ces commandes deux niveaux de mise en évidence du texte.

La commande `<font></font>` permet d'expliciter la police à utiliser, sa taille, sa couleur…. Toutefois l'usage de cette commande n'est plus recommandé. On préfère actuellement spécifier ce genre de renseignements dans les feuilles de styles (Chapitre 3).

2.2.5 Les titres

La norme HTML prévoie 6 niveaux de titres codés à l'aide des balise `<hx></hx>` avec $x \in \{1, \ldots, 6\}$. Un titre peut être aligné à droite, à gauche ou centré avec l'attribut align.

2.3 Les listes

La norme HTML propose des listes numérotées `<ol></ol>` (ordered list) et non numérotées `<ul></ul>` (unordered list). Les différents éléments d'une liste sont encadrés par les balises `<li></li>`. Une liste ressemble donc à :

```
<body>
<ul>
<li> Un premier item,</li>
<li> et un deuxi{\`e}me.</li>
</li>
</body>
```

■ Un premier item,
■ et un deuxième.

Différents types de numérotation sont possibles pour les listes numérotées grâce à l'attribut `type` de `<ol>` (Table 2.2).

On peut également utiliser différent types de listes non ordonnées grâce à l'attribut `type` de `<ul>` (Table 2.3). Notons également que l'on peut également imposer un type de puce directement dans la balise `<li>`. Toutefois ceci peut rompre l'homogénéité des listes et n'est donc pas recommandé.

Notons également l'existence de listes de descriptions. Celles ci sont définies par les balises `<dl></dl>` (description list). Chaque élément d'une telle liste est

A :	(A,B,C. . .)
a :	(a,b,c,. . .)
1 :	(1,2,3,. . .)
i :	chiffres Romains minuscule (i,ii,iii,iv. . .)
I :	chiffres Romains majuscules (I,II,III,IV. . .)

Tab. 2.2 – Les différentes styles de listes numérotées en HTML

disc	puce noire (•)
circle	cercle (○)
square	carré (■)

Tab. 2.3 – Les différents styles de listes non numérotées en HTML

défini par deux balises.

1. L'identifiant à décrire. Celui-ci est encadré par les balises `<dt></dt>`.
2. Le descriptif de l'identifiant encadré par les balises `<dd></dd>`.

Un exemple de liste de description est fourni ci-après :

```
<dl>
<dt> HTML</dt>
<dd> Langage a balise permettant de decrire
le contenu de pages Web.</dd>
<dt> CSS</dt>
<dd> Feuille de style permettant de controler
la mise en forme d'une page HTML</dd>
</dl>
```

2.4 Les tableaux

Un tableau HTML se déclare à l'aide de la balise `<table></table>` qui contient des balises `<tr></tr>` et `<td></td>` codant respectivement les lignes et les cellules de la table. Un tableau basique s'écrit donc comme suit :

```
<body>
<table>
<tr>
 <td>Un</td><td>Deux</td><td>Trois</td>
</tr>
<tr>
 <td>4</td><td>5</td><td>6</td>
</tr>
</table>
</body>
```

Un	Deux	Trois
4	5	6

L'aspect et le positionnement d'une table est paramétré à l'aide de nombreux attributs (Tables 2.4 et 2.5).

attribut	valeur	unité
border	largeur de la bordure du tableau	pixel
cellpadding	distance entre le contenu de la cellule et le bord de celle-ci.	pixel
cellspacing	distances entre les pixels	pixel
width	largeur du tableau	pixel ou % de la largeur de la fenêtre
height	hauteur du tableau	pixel ou % de la largeur de la fenêtre
bgcolor	couleur du fond	
align	alignement du tableau	
background	image de fond	

Tab. 2.4 – Attributs de la balise `<table></table>` en HTML

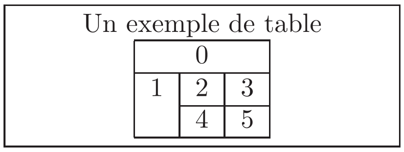
attribut	valeur	unité
align	alignement horizontal du contenu d'une cellule ou d'une ligne. Valeurs (left,center,right)	
valign	alignement vertical d'une cellule ou d'une ligne. Valeurs (top,middle,bottom)	
bgcolor	couleur de fond d'une cellule ou d'une ligne	
background	image d'arrière plan	
colspan	nombre de colonnes sur lesquelles s'étend une cellule	
rowspan	nombre de lignes sur lesquelles s'étend une cellule	
width	largeur de cellule	pixel
height	hauteur de cellule	pixel
nowrap	interdit le retour à la ligne dans une cellule	

Tab. 2.5 – Attributs des balises `<tr></tr>` et `<td></td>` en HTML

Notons enfin l'existence de la sous balise `<caption></caption>` qui permet de placer un texte explicatif sous une table. La balise `<caption>` doit être placée dans la balise `<table>`.

L'exemple représenté sur la Figure 2.1 illustre l'effet de quelques uns de ces attributs.

```
<body>
<table border="1" bgcolor="#FFFFFF" align="center">
<caption> Un exemple de table</caption>
<tr>
<td colspan="3" align="center">
0</td>
</tr>
<tr>
<td rowspan="3" valign="top">
1</td>
</tr>
<tr><td>2</td><td>3</td></tr>
<tr><td>4</td><td>5</td></tr>
</table>
</body>
```



Un exemple de table

<table>
<tr><td colspan="3" align="center">0</td></tr>
<tr><td rowspan="2" valign="top">1</td><td>2</td><td>3</td></tr>
<tr><td>4</td><td>5</td></tr>
</table>

FIG. 2.1 – Illustration des attributs d'une table en HTML

2.5 Les liens

Un lien HTML est défini par la balise `<a></a>` qui comporte un seul attribut `href` dont la valeur indique l'URL sur laquelle va pointer le lien. Par exemple la commande :

```
<body>
<a href="http://www.ismra.fr/"> ENSICAEN</a>
```

crée un lien sur le mot ENSICAEN qui sera généralement affiché en bleu et souligné. Un clic sur ce mot dirigera le navigateur vers le site désigné par l'URL (en l'occurrence celui de l'ENSICAEN). Les couleurs généralement utilisées pour les liens sont :

le bleu pour les liens non encore sélectionnés,

le mauve pour les liens visités et

le rouge pour les liens actifs.

Ces couleurs peuvent être changées par les attributs (`link`, `vlink` et `alink`) de la balise body ou dans une feuille de style attachée au document (solution recommandée).

On peut également faire un lien sur une adresse émail grâce à une syntaxe particulière de l'attribut `href`. La ligne suivante :

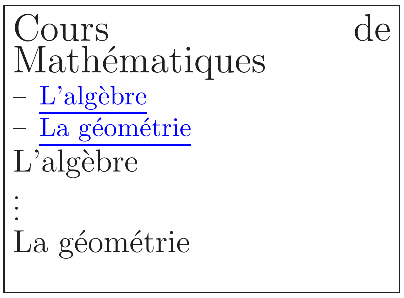
```
<body>
<a href="mailto:toto@titi.com"> cliquez ici </a> pour m'envoyer un m&egrave;l.
</body>
```

affiche le message «cliquez ici pour m'envoyer un mèl» et un clic sur la zone soulignée entraîne l'ouverture de votre gestionnaire de courrier avec le destinataire déjà rempli.

2.5.1 Les ancres

Lorsqu’un document est assez long, il est possible de positionner des ancres le long de celui-ci à l’aide de l’attribut `name` de la balise `<a></a>`. On peut alors faire des liens vers ces ancres en utilisant l’attribut `href`. L’exemple suivant illustre le fonctionnement de ce type de balises :

```
<body>
<h1> Cours de math&eacute;matiques</h1>
<ul>
<li> <a href="#chap1">
L'alg&egrave;bre</li>
<li> <a href="#chap2">
La g&eacute;om&eacute;trie</li>
</ul>
<h2> L'alg&egrave;bre
</h2>
<a name="chap1"/>
<h2> La g&eacute;om&eacute;trie
</h2>
<a name="chap2"/>
</body>
```



Cours de Mathématiques

– L’algèbre

– La géométrie

L’algèbre

⋮

La géométrie

On peut également faire référence à une ancre d’un fichier externe à l’aide de la syntaxe :

`<a href="url#ancre">texte</a>.`

Par exemple :

`<a href="toto.html#intro"> Introduction</a>`

crée un lien vers l’ancre `intro` du fichier `toto.html`.

Notons enfin que l’url indiquée par l’attribut `href` peut se présenter de deux façon :

Une URL absolue : il s’agit d’une URL indiquant de façon exhaustive le nom du serveur et le l’emplacement du fichier :

```
<a href="http://mon_site.com/site_de_toto/francais/index.html">
mon site</a>
```

Une URL relative : qui n’indique qu’un déplacement par rapport au fichier courant :

```
<a href="../english/index.html"> English version</a>
```

De façon générale il faudra toujours privilégier des URL relatives pour faciliter la probabilité du site lors d’un changement d’hébergeur ou plus simplement un changement de répertoire sur un même serveur.

2.6 Les images

L'insertion d'une image dans une page HTML se fait à l'aide de la balise `<img></img>` dont le principal attribut est `src` qui indique le fichier source de l'image à l'aide d'une URL absolue ou relative. Les formats acceptés sont gif et jpeg. Les autres attributs sont représentés sur la Table 2.6.

attribut	valeur	unité
width	largeur de l'image	pixel
height	hauteur de l'image	pixel
align	alignement de l'image par rapport à la fenêtre et le texte. Valeurs : (left,right,top,middle,bottom)	
vspace	marge verticale de l'image	pixel
hspace	marge horizontale de l'image	pixel
border	épaisseur du cadre autour de l'image	pixel
alt	texte affiché pendant le chargement de l'image	

Tab. 2.6 – Liste des attributs de la balise `<img></img>` en HTML

Si l'attribut align a une des deux valeurs `left` ou `right`, celle-ci code l'alignement de l'image par rapport à la fenêtre. En revanche, si la valeur de align correspond à `top, middle` ou `bottom`, cette valeur code l'alignement du texte par rapport à l'image. L'exemple représenté sur la Figure 2.2 affiche le logo de l'EnsiCaen avec un texte centré verticalement par rapport à l'image et un retrait horizontal de 20 pixels.

```
<body>
<img src="ensicaen.jpg"
width=2cm align="middle"
hspace="20">
Logo de L'EnsiCaen
</body>
```

Fig. 2.2 – Illustration des attributs de la balise `<img></img>` en HTML.

2.6.1 Les images clicables

Il est possible de définir des zones clicables dans une image en y associant une map définie par la balise `<map></map>`. On utilise pour cela l'attribut `usemap` de la balise `<img>`.

Une map est composée d'un ensemble de formes codées par des balises `<area></area>` . Chaque forme (area) comporte des attributs représentés sur la Table 2.7.

Les coordonnées associées aux différents types de formes sont les suivantes :

attribut	valeur
shape	type de la zone. Valeurs (rect,circle,polygon)
coords	Liste de coordonnées dans l'image définissant la zone
href	Lien associé à la zone
alt	texte de l'info bulle

TAB. 2.7 – Attributs de la balise `<area></area>` en HTML.

- Pour les rectangles (rect) : les coordonnées du coin en haut à gauche(x_1, y_1) et du coin en bas à droite (x_2, y_2),
- Pour les cercles (circle) : les coordonnées du centre (x, y) et le rayon (R).
- Pour les polygones la liste (x_i, y_i) des points du polygone.

Par exemple le rectangle, le cercle et le triangle de la Figure 2.3 seront codés par :

```
<map name="exemple">
<area shape="rect" coords="x1,y1,x2,y2" href="ex1.html"/>
<area shape="circle" coords="x,y,R" href="ex2.html"/>
<area shape="polygon" coords="x1,y1,x2,y2,x3,y3" href="ex3.html"/>
</map>
```

Un exemple plus concret d'image clicable est fournit Figure 2.4.

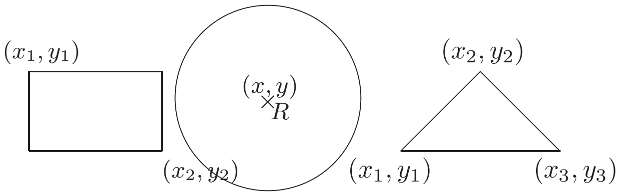


FIG. 2.3 – Un rectangle, un cercle et un triangle définissant les zones d'une map.

2.7 Les données multimédia

La balise `<a></a>` (Section 2.5) permet d'établir des liens sur n'importe quel type de document. Ces documents sont alors catalogués par leurs type MIME (Section 1.1.4) et lors d'une sélection du lien le navigateur cherchera à lancer l'application adaptée au type MIME du document.

Il est toutefois possible d'intégrer du sons ou des vidéos dans des pages Web à l'aide de la balise `<embed></embed>` (disponible uniquement sur les navigateurs compatible avec Mozilla (galeon, firefox...)). Ses attributs sont représentés sur la table 2.8.

```
<p>
Cliquez sur les petits cercles
blancs pour obtenir une image.
</p>
<map name="img_ext">
  <area shape="circle"
  coords="10,44,15"
  href="IMAGES/pict0241.jpg">
  <area shape="circle"
   coords="231,38,15"
   href="IMAGES/pict0242.jpg">
  <area shape="circle"
   coords="406,143,15"
   href="IMAGES/pict0243.jpg">
  <area shape="circle"
   coords="403,44,15"
   href="IMAGES/pict0244.jpg">
</map>
<img src="IMAGES/vue_dessus.jpg"
 usemap="#img_ext"/>
```

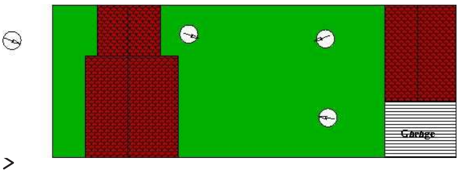


Fig. 2.4 – Un exemple réel d'image cliquable.

attribut	valeur
src	url du fichier à lire
autostart	démarrage automatique de la lecture du document lors du chargement de la page. Valeurs (true,false).
hidden	affichage d'un petit panneau pour l'application de lecture
loop	nombre de fois où l'on boucle sur la lecture

Tab. 2.8 – Liste des attributs de la balise `<embed></embed>` en HTML

Le code suivant lance automatique la lecture d'un morceau de Bellini au chargement de la page :

```
<embed src="bellini.wav" width="145" height="60"
 autostart="true" hidden="false"/>
```

Une manière plus portable d'intégrer des documents multimédia est la balise `<object>` . Dont les attributs sont représentés sur la table 2.9. Notons que tous ces attributs ne sont pas nécessaires simultanément. Un exemple simple d'utilisation de la balise `<object>` pour jouer un morceau mp3 est donné ci dessous :

```
<object data="bellini.mp3" type="application/mpeg"></object>
```

classid	uri	identificateur de l'implémentation de l'objet
codebase	uri	chemin de base pour résoudre les différents URI
codetype	content-type	type mime correspondant au classid (section 1.1.4)
data	uri	url des données
type	content-type	type mime du champs data
archive	uri-list	liste d'uri d'archives utilisables par l'objet
declare	booléen	si vrai, l'objet n'est pas immédiatement instancié
standby	text	message à afficher pendant le chargement

Tab. 2.9 – Liste des attributs de la balise `<object></object>`.

Notons toutefois que le chargement automatique de fichiers multimédia peut ralentir considérablement le chargement de votre page et donc dissuader d'éventuels lecteurs d'attendre la fin du chargement. Cette pratique n'est donc pas conseillée.

2.8 Les Cadres

Intuitivement un cadre correspond à un tableau qui partitionne la fenêtre du navigateur en affichant une page Web dans chaque cellule du tableau. Une décomposition de la fenêtre en cadres s'effectue grâce à la balise `<frameset></frameset>` dont les attributs sont représentés sur la Table 2.10.

attribut	valeur	unité
rows	hauteur des lignes	pixels ou pourcentage
cols	largeur des colonnes	pixels ou pourcentage
frameborder	bordure 3D. Valeurs (0 ou 1)	
framespacing	distance entre les cadres	pixels
bordercolor	couleur de la bordure	

Tab. 2.10 – Liste des attributs de la balise `<frameset></frameset>` en HTML.

Chaque cadre à l'intérieur des balises `<frameset></frameset>` est défini à l'aide des balises `<frame></frame>` qui possèdent également un certain nombre d'attributs (Table 2.11).

attribut	valeur	unité
src	url de la page à afficher dans le cadre	
name	nom unique pour le cadre	
margenwidth	taille des marges gauches et droites du cadre	pixel
margenheight	taille des marges haut et bas	pixel
scrolling	défilement possible ou pas. Valeurs (yes, no, auto)	
frameborder	affichage ou pas de la bordure du cadre. Valeurs (0 ou 1)	
bordercolor	couleur de la bordure	
noresize	taille du cadre non modifiable. Pas de valeur	

Tab. 2.11 – Liste des attributs de la balise `<frame></frame>` en HTML

L'exemple suivant découpe la fenêtre du navigateur en deux cadres. Un des deux cadres a 200 pixels de large, le cadre suivant occupant la place restante :

```
<body>
<frameset cols="200,*">
<frame src="menu.html" name="menu"/>
<frame src="main.html" name="principal"/>
</frameset>
</body>
```

L'utilisation de l'attribut `name` des cadres est essentiel pour une navigation correcte dans le site. En effet, une fois que l'on a défini des cadres il faut préciser pour chaque lien le cadre dans lequel il doit s'afficher. On utilise pour cela l'attribut `target` de la balise `<a></a>`. Un lien dans le fichier `menu.html` désirant s'afficher dans le cadre principal aura par exemple le code suivant :

```
<a href="loisirs.html" target="principal"> Mes loisirs</a>
```

Trois valeurs spéciales et particulièrement intéressantes de l'attribut `target` sont `_top`, `_self` et `_blank`. Ces valeurs ouvrent respectivement le lien dans :

- la fenêtre courante sans les cadres (`_top`),
- dans le même cadre que celui contenant le lien (`_self`) et
- dans une nouvelle fenêtre (`_blank`).

Les cadres tombent actuellement en désuétude dans la mesure où :

1. Un résultat équivalent peut être obtenu avec des feuilles de styles,
2. l'utilisation de cadre oblige à nommer tous les cadres et a indiquer dans chaque lien le cadre où il doit s'afficher. Ceci a tendance à «geler» l'architecture d'un site Web.

2.9 Les Formulaires

Les formulaires permettent à un utilisateur de rentrer des informations qui seront généralement traitées par le serveur en utilisant des CGI (Chapitre 5) ou l'interpréteur PHP.

La balise utilisée pour définir un formulaire est la balise `<form></form>` dont les attributs sont décrits Table 2.12.

action	:	url de l'application devant recevoir les données du formulaire.
method	:	méthode utilisée pour passer les paramètres du formulaire à l'application. Les valeurs possibles sont `get` ou `post`(voir Chapitre 5)
enctype	:	spécifie le type d'encodage du contenu de la requête. Cet attribut s'applique uniquement à la méthode post et est rarement utilisée.

Tab. 2.12 – Liste des attributs de la balise form

En soit, un formulaire n'est qu'une coquille vide destinée à recevoir d'autre balises qui vont fournir différents mécanismes pour spécifier les paramètres. Nous allons décrire ces balises ainsi que leurs paramètres dans les sections suivantes. Notez qu'une page Web décrivant également ces balises est normalement à votre disposition.

2.9.1 La balise input

Les balises input peuvent avoir des rôles et des aspects très différents suivant leurs type (spécifié par l'attribut `type`. Les différentes valeurs que peut prendre l'attribut type sont fournies Table 2.13.

text	:	valeur par défaut. zone de texte à remplir.
password	:	idem que texte mais les caractères tapés sont remplacés par des *
image	:	permet de récupérer les coordonnées (x,y) d'un clic dans l'image
checkbox	:	permet de sélectionner différents items à la souris.
radio	:	même principe que checkbox mais un seul item peut être sélectionné.
submit	:	bouton utilisé pour envoyer les données du formulaire à l'application
reset	:	réinitialise le formulaire

Tab. 2.13 – Les différentes valeurs de l'attribut `type` dans les balises `<input></input>`

Les autres attributs de la balise `input` sont décrits Table 2.14

type	:	type d'input (voir Table 2.13)
name	:	nom symbolique (de variable) du champ input. Obligatoire pour toutes les balises `input` sauf `submit` et `reset`
value	:	sert à définir un contenu par défaut pour les balises de type text ou password, la valeur de variable (`name`) pour les boutons radio et checkbox et le texte du bouton pour les boutons (`submit` et `reset`)
checked	:	pour les radio ou checkbox, indique que le bouton est sélectionné par défaut.
size	:	nombre de caractères de la zone d'entrée (input et password)
maxlength	:	nombre max. de caractères de la zone (input et password)

Tab. 2.14 – Les différents attributs de la balise `<input></input>`

2.9.2 La balise select

La balise `<select></select>` visualise un menu à option. Elle permet donc de choisir un ou plusieurs éléments parmi ceux proposés. Sa syntaxe est la suivante :

```
<select name="un_menu">
        <option> Premier choix.</option>
        <option> Deuxi{\`e}me choix.</option>
</select>
```

Les attributs de la balise `select` sont fournis Table 2.15. Notez que la balise `<option></option>` peut prendre le paramètre selected qui signifie que cette option est sélectionnée par défaut.

name	:	nom de la balise
size	:	si size vaut 1 ou n'est pas précisé, la balise `select` se présente comme un menu. Sinon, elle se présente comme une liste déroulante en affichant `size` éléments à la fois.
multiple	:	permet de sélectionner plusieurs éléments dans la liste (en maintenant la touche contrôle enfoncée)

Tab. 2.15 – Liste des attributs de la balise `select`

2.9.3 La balise textarea

La balise `<textarea></textarea>` permet de saisir plusieurs lignes de texte. Ces attributs sont fournis Table 2.16.

name	:	nom de balise
rows	:	nombre de lignes de la textarea
cols	:	nombre de colonnes

Tab. 2.16 – Liste des attributs de la balise `<textarea></textarea>`.

2.10 Balises et attributs à éviter

Nous allons voir dans le chapitre 3 que le W3C à prévu une autre norme à coté du langage HTML pour mettre en forme les documents. En conséquences toutes les balises HTML concernant l'aspect d'un document (bold, italique...) sont découragées et seront de moins en moins supportées. Les éléments du language HTML à éviter comportent des balises (Table 2.17) et des attributs (Table 2.18). Notons que des attributs apparaissant uniquement dans des balises dépréciées sont de facto à éviter.

basefont	définition des fonts par défaut
center	aligne de façon centré ses sous balises
dir	même effet que ul
font	changement local de font
isindex	une input de type texte.
menu	même effet que ul
s	texte barré
strike	texte barré
u	texte souligné

Tab. 2.17 – Liste des balises à éviter (dépréciées).

2.11 Les Règles d'or d'un Webmaster

Pour finir et en guise de conclusion sur ce chapitre, voici les 5 règles d'or du codeur HTML selon Raphael Goetter.

Principe 1 : un code propre.

Observer les règles du langage HTML strictement et rigoureusement, de façon à éviter les erreurs. Ces règles ne sont en général pas nombreuses et simples à appliquer (fermer les balises, écrire en minuscules,...) En respectant ces règles de langage, vous assurerez la pérenité de vos pages et commencerez dès lors à profiter des avantages induits par l'application des standards Web. Avantages : Avoir un code propre est la base principale de tout projet HTML en conformité avec les normes actuelles et futures.

Principe 2 : un code compréhensible.

attribut	balise	fonction
align	CAPTION	alignement de la légende par rapport à la table
align	APPLET, IFRAME, IMG, INPUT, OBJECT	alignement vertical ou horizontal
align	LEGEND	relativement au champ
align	TABLE	position de la table dans la fenêtre
align	HR (left — center — right)	alignement du hr
align	DIV, H1, H2, H3, H4, H5, H6, P (left — center — right — justify)	alignement de texte
alink	BODY	couleur d'un lien sélectionné
background	BODY	texture du fond du document
bgcolor	TABLE	couleur de fond des cellules
bgcolor	TR	couleur de fond des lignes
bgcolor	TD, TH	couleur de fond d'une cellule
bgcolor	BODY	couleur du fond du document
border	IMG, OBJECT	largeur du bord indiquant le lien
clear	BR (left — all — right — none)	controle de flux de texte
color	BASEFONT, FONT	couleur du texte
compact	DIR, DL, MENU, OL, UL (compact)	réduit l'espace inter-item
face	BASEFONT, FONT	liste de noms de fonts séparés par des virgules
height	TD, TH	hauteur des cellules
language	SCRIPT	langage de script prédéfini
link	BODY	couleur des liens
noshade	HR (noshade)	pas de dégradé sur hr
nowrap	TD, TH (nowrap)	suppression passage à la ligne
prompt	ISINDEX	contenu du message
size	HR	taille de la barre
size	BASEFONT	taille de base des fonts
start	OL	numéro de départ pour les listes ol
text	BODY	couleur du texte du document
type	LI	style d'un item
type	OL	style de numérotation
type	UL	style des items non numérotés
value	LI	réinitialise les numéro dans la liste
version	HTML	constante
vlink	BODY	couleur des liens visités
width	HR	largeur barre horizontale
width	TD, TH	largeur de cellule
width	PRE	largeur du champ pre.

Tab. 2.18 – Liste des attributs de balise à éviter.

Il faut toujours comprendre le code HTML produit. Cela implique évidemment une connaissance minimale des balises même (surtout) si on utilise un éditeur WYSIWYG qui génère du code automatique. «Faire des plats cuisinés, c'est bien. Mais c'est toujours mieux d'apprendre à faire la cuisine, même si on ne veux pas devenir un grand chef.»

Pré-requis : Un code compréhensible découle évidemment d'un code propre. Le 1er Principe est donc à observer en priorité.

Avantages : Un code compréhensible est utile pour vous (compréhension de votre code dans 6 mois, facilitation des mises à jour), mais aussi pour les autres (demande d'aides, poursuite du projet par d'autres, projet en commun, ...).

Principe 3 : un code simple.

Toujours produire un code HTML le plus simple et le plus direct possible. Éviter l'enchevêtrement et l'imbrication de balises superflues. Voir à ce sujet les articles sur OpenWeb et sur Alsacréations.

Pré-requis : Alléger la page passe aussi et surtout par la séparation entre le contenu (HTML) et la forme (CSS). Une bonne connaissance des feuilles de styles est donc nécessaire.

Avantages : Un code simple et direct est un complément d'un code compréhensible. Cela va permettre un allègement du poids de la page, du site entier (mise en cache de la feuille CSS par exemple) et un effet bénéfique sur l'utilisation de la bande passante.

Principe 4 : un code pertinent.

Toujours utiliser les bonnes balises au bon endroit. Éviter par exemple l'emploi de balise de paragraphe (P) pour définir un titre de page, alors qu'il existe une balise de titre (Hx). Pourquoi ? Tout bêtement pour mieux comprendre votre code (principe 2), faciliter votre propre mise à jour (principe 3), mais aussi pour permettre l'accessibilité de votre site aux handicapés qui utilisent des lecteurs braille par exemple (principe 5). L'utilisation des balises à bon escient se nomme la Sémantique. Vous trouverez une bonne illustration de la sémantique sur cette introduction au XHTML.

Pré-requis : Utiliser toujours la bonne balise au bon endroit nécessite une parfaite connaissance du langage HTML et de toutes les balises en vigueur, ainsi que leur comportement par défaut (bloc, en-ligne, marges par défaut,...).

Avantages : Avoir un code pertinent et une structure propre va grandement faciliter son accessibilité et sa diffusion sur le web : tous les navigateurs (même braille ou moteurs de recherche comme Google) auront la bonne interprétation de votre document. Ils comprendront que le titre est un titre et non un paragraphe amélioré, que le menu est un menu, etc...

Principe 5 : un code accessible.

Cette règle regroupe l'accessibilité (utilisabilité du site par les personnes souffrant de handicaps divers) et l'Interopérabilité (compatibilité du site sur tous les navigateurs et toutes les plateformes). Internet est fait pour être utilisable par tous. Les vieilles guerres entre les navigateurs ont faussé ce principe originel. A nous de rectifier le tir et de rendre nos pages accessibles à tous en évitant les codes propriétaires (balise MARQUEE,...), les balises obsolètes (FONT, ...) qui sont nombreuses. Pour faciliter cette démarche, il existe des Validateurs de code (x)HTML et des Validateurs d'accessibilité. A vous de les utiliser à loisir.

Pré-requis : Un code pertinent est sémantique est une priorité à atteindre avant d'en espérer faire un code Accessible et inéropérable.

Avantages : Tout simplement vérifier la définition du web telle qu'elle a été donnée à sa création : un média utilisable par tous quels que soit la plateforme et l'individu qui l'utilisent.

Principe général :

ne jamais imposer une manière de surfer. Ne pas fixer la taille de police. N'oublions pas que certains utilisateurs malvoyants ont besoin d'augmenter la taille de police. Il est vivement déconseillé d'imposer une manière de surfer de toute façon. Ne pas ouvrir de nouvelle fenêtre ou de popup sans prévenir l'utilisateur. Ne pas imposer de la musique, un affichage plein écran, un plugin, etc...

Chapitre 3

Les CSS

3.1 Introduction : De la nécessité des CSS

Une connaissance superficielle de HTML suffit à de nombreuses personnes désirant construire quelques pages Web pour un site personnel. Toutefois, la construction d'un site professionnel (ou comprenant plus d'une dizaine de pages) nécessite la distinction entre trois notions souvent confondues :

1. *Le contenu* : Ceci correspond à l'ensemble des données qui seront diffusées sur le site. Celles ci peuvent être stockées dans des fichiers, des bases de données ou sous tout autre format.
2. *La structuration* de l'information. Cette tache est dans le cadre des sites Webs usuellement dévolue au langage HTML qui défini qu'elles informations doivent être placées dans un titre, un tableau, une liste...
3. *La présentation* de l'information. Cette tache consiste à définir les tailles de polices, l'alignements..., tout ce qui concerne la présentation de l'information indépendamment de son contenu et de la structure du document.

Le langage HTML présente un danger à ce niveau là (tout au moins pour les débutants) dans la mesure où il permet à la fois de structurer un document et de fixer sa présentation. Cette approche est nuisible pour tout un ensemble de raisons. En effet, un code HTML gérant à la fois la structuration et la présentation de l'information est généralement :

1. plus lourd (en terme de lignes de code),
2. moins lisible,
3. moins homogène,
4. moins maintenable,
5. moins flexible.

Un site Web fixant la présentation des pages dans le code HTML oblige le client à rapatrier simultanément la présentation et la structuration de chaque page du site Web. Ce qui impose des temps de chargements plus *lourds*. Si les règles de présentations sont rassemblées dans un fichier, celui-ci ne sera lu qu'une fois pour tout le site.

Fixer la présentation dans le code HTML implique l'ajout de nombreuses balises. La présence de ces balises noie les balises de structure et ne permettent pas à un lecteur potentiel du code HTML de comprendre aisément la structure de la page. Celle-ci est donc moins *lisible*.

De plus supposons qu'un concepteur de page Web décide qu'un certain type d'information (par exemple les noms de section) doit être en police Vernada, 14pt et de couleur bleu. Si la présentation est incluse dans le code il y a de fortes chances que ce concepteur oublie un des attributs (Vernada, 14pt ou bleu) pour quelques sections. Ceci nui à l'*homogénéité* de la présentation.

Reprenons l'exemple du Vernada, 14pt, bleu pour les noms de sections (paragraphe précédent) et supposons que la couleur bleu des sections ne plaise pas au client pour lequel a été fait le site. Il faut alors que le concepteur du site Web passe en revue l'ensemble code HTML pour changer le bleu en la couleur préférée du client. Il faudra bien entendu faire attention à ne pas changer la couleur bleu d'autre partie du texte qui ne correspondent pas à des sections. Le code est donc clairement non *maintenable* à une grande échelle.

Enfin, externaliser la présentation permet de changer facilement celle-ci sans changer la structure du document. Le site Web est dans ce cas beaucoup plus *flexible*.

Les CSS permettent d'externaliser la présentation d'une page Web et donc de s'affranchir des problèmes précédemment mentionnés. Notons pour terminer cette section, qu'il faut généralement se méfier des outils WYSIWYG permettant une génération automatique de code HTML. En effet, ces outils utilisent généralement peu ou pas les CSS. Ces applications devraient donc être réservées au pages Web ne devant servir qu'un temps très limité. Malheureusement, en informatique comme ailleurs le temporaire devient souvent définitif...

3.2 Structure des règles CSS

Supposons que nous voulions que les titres de premier niveau `<h1></h1>` apparaissent en rouge. Une telle règle s'écrie en CSS :

```
h1
{
        color: red;
}
```

Cette règle comprend plusieurs parties (Figure 3.1) :

- Le sélecteur de la règle (ici `h1`),
- la déclaration qui elle même se décompose en :
  - une propriété à laquelle on affecte
  - une valeur.

Une propriété se termine toujours par un deux points « :», alors que la valeur se termine toujours par un point virgule « ;».

$$\underbrace{\text{h1}}_{\text{sélecteur}}\ \{\ \underbrace{\overbrace{\text{color}}^{\text{propriété}} : \overbrace{\text{red}}^{\text{valeur}}}_{\text{déclaration}};\}$$

FIG. 3.1 – Structure d'une règle

3.2.1 Les Sélecteurs

Il existe plusieurs façon de spécifier le domaine d'application d'une règle à travers les sélecteurs.

Le sélecteur simple

Un sélecteur simple se réduit à un nom de balise et s'applique donc à toute balise portant le même nom que le sélecteur. Par exemple le code suivant :

```
h1 {
    text-align: center;
    font-family: verdana;
    font-weight: bold;
    color: red;
}
```

indique que les titres de niveaux 1 doivent être centrés, en police Vernada bold et en rouge.

Notez l'existence du sélecteur universel '*' qui désigne n'importe qu'elle balise.

Les sélecteurs à évènements

Il est également possible de spécifier des mises en formes qui ne se produisent que lors de certains évènements. Par exemple le code suivant :

```
a:hover {color: red; }
```

met le texte d'un lien en rouge lorsque la souris passe dessus.

Notons qu'il est possible de combiner des sÃ©lÃ©cteurs Ã évènements. Par exemple le code suivant :

```
input:focus:hover {background: red; }
```

met le fond d'une input box en rouge si elle à été sélectionnée (le curseur est dedans) et si la sourie est dessus.

Notons que ces sélecteurs à évènements ainsi que `:first-child` (voir plus loin) et `:lang` sont appellés des pseudo-classes par le W3C. Les principaux évènements sont représentés sur la Table 3.1.

évènement	description
:focus	passage de la souris sur un texinput ou textarea d'un formulaire
:hover	passage de la souris sur un lien
:link	lien pas encore visité
:visited	lien cliqué
:active	pour un lien : actif entre la pression et le relachement de la sourie

Tab. 3.1 – Principaux évènements d'un sélecteur dans un fichier CSS

Les pseudo éléments

Ces sélecteurs permettent de modifier l'apparence de zones non délimitées par des balises. Par exemple les deux pseudo balises `span :before` et `span :after` désignent des contenus à placer respectivement avant et après une balise `span`. Par exemple :

```
span:before
{
    content:url(img.jpg);
}
span:after
{
   content: "Fin";
}
```

Affiche une image et le mot Fin après chaque balise `span`.

Les sélecteurs `:first-line` et `:first-letter` permettent de désigner respectivement la première ligne et la première lettre d'une balise de type bloc. La liste des pseudo éléments est récapitulée dans la table 3.2.

Les sélecteurs avec attributs

On peut également sélectionner des balises à l'aide de leurs attributs ou de la valeur de ces attributs. Le sélecteur `[att]` sélectionne toute les balises possédant l'attribut `att` tandis que le sélecteur `[att=val]` sélectionne toute les balises dont l'attribut `att` vaut `val`. On peut également effectuer des sélections plus fines à l'aide de l'attribut `[att =val]` qui sélectionne les balises possédant un attribut `att` défini comme une phrase dont au moins un des mot est égal à `val`. Le sélecteur `[att|=val]` sélectionne les balises possédant un attribut `att` défini comme une suite de mots séparés par des '-' et commençant par `val`.

Dans l'exemple suivant :

évènement	description
B :before	contenu à insérer avant une balise B
B :after	contenu à insérer après une balise B
B :first-line	mise en forme de la premiÃ¨re ligne de B
B :first-letter	mise en forme de la premiÃ¨re lettre de B

Tab. 3.2 – principaux pseudo éléments

```
h1 {color: black;}
h1[title] { color: blue; }
h1[title=titre] {color: red};
h1[title˜=titre] {color: yellow;}
h1[title|=tire] {color: pink;}
<h1> toto</h1>
<h1 title="titre">titi</h1>
<h1 title="le titre"> tutu</h1>
</h1 title="titre-principal">tata</h1>
```

toto sera en noir, titi en bleu, tutu en jaune et tata en rose. La liste des sélecteurs avec attributs est résumée dans la table 3.3.

évènement	description
[att]	balises contenant l'attribut att.
[att=val]	balises dont l'attribut att vaut val.
[att˜=val]	balises dont l'attribut att est une phrase contenant le mot val.
[att|=val]	balises dont l'attribut att est une suite de mot séparés par des '-' et commençant par val.

Tab. 3.3 – Sélecteurs avec attributs

Les sélecteurs contextuels

Il est également possible de spécifier une mise en forme pour une balise située dans une autre balise. Par exemple, le code :

```
p a { color: blue; }
```

indique que tous les liens situés dans dans un paragraphe doivent être en bleu. Notons que cette dernière relation reste vrai quelque soit le degré d'inclusion de la balise a. Par exemple dans le code HTML suivant :

```
<p>
<a href="toto.html"> toto</a>
<span> <a href="titi.html"> titi</a>
</p>
```

le séleteur `p a` s'appliquera aux deux balises. On dit dans ce cas que le lien dans la balise `span` est un descendant du paragraphe mais n'est pas un de ses fils. On peut préciser un lien direct par la notation

```
p>a { color: red; }
```

Dans ce cas, en reprenant le code HTML vu au dessus seule la première balise `a` sera capturée par le sélecteur `p>a`. On peut encore préciser les choses en indiquant les propriété à appliquer à la première balise à l'aide du sélecteur `:first-child`. Par exemple, dans le code suivant :

```
p>a:first-child { color: red; }
p>a { color: blue; }
p a { color: green; }
<p>
<a href="toto.html"> toto</a>
<a href="tutu.html"> tutu</a>
<span> <a href="titi.html"> titi</a></span>
</p>
```

toto sera en rouge, tutu en bleu et titi en vert.

Il est enfin possible de dÃ©signer la succession de deux balises à l'aide du sélecteur `B+C`. Par exemple :

```
span+a { color: red; }
a+a    { color: blue; }
<p>
<span> <a href="titi.html"> titi</a></span>
<a href="toto.html"> toto</a>
<a href="tata.html"> tata</a>
</p>
```

produira un toto rouge et un tata bleu. La liste des sélecteurs contextuels est fournie dans la table 3.4.

Les groupement de sélecteurs

Lorsque une même mise en forme s'applique à plusieurs descripteurs, on peut regrouper ceux-ci en les séparant par des virgules. Le code suivant

```
h1, h2, h3 {
    font-weight: bold;
}
```

indique que les titres de niveaux 1 à 3 doivent être en gras.

évènement	description
B C	s'applique si C est un descendant de B.
B>C	s'applique si C est un descendant direct de B.
B :first-child	s'applique si B est le premier fils de son père
B+C	s'applique si B et C sont concécutifs.

Tab. 3.4 – Sélecteurs contextuels

Les sélecteurs de classes

Une possibilité intéressantes des feuilles de style et de pouvoir associer une classe à l'aide de l'attribut `class` aux balises HTML. L'utilisation de l'attribut `class` permet de spécifier dans le code HTML la signification de la balise. Par exemple les règles suivantes :

```
p.intro
{
text-align: justify;
font-size: small;
}
h2.section
{
text-align: justify;
font-size: large;
}
span.texte
{
font-size: medium;
}
```

permet d'associer différents alignements et différentes tailles de textes à des partie *sémantiquement* différentes d'une page Web. Cette feuille de style pourrait s'utiliser comme suit :

```
<body>
<p class="intro">
Ceci est un petit r&eacute;sum&eacute; de la page
</p>
<h2 class="section"> Mon premier chapitre</span>
<p class="texte"> Le contenu de mon chapitre</p>
</body>
```

Sélecteurs d'identifiant

Il est également possible de spécifier une mise en forme pour une balise bien précise. Ceci est réalisé grâce à l'attribut `id` qui permet d'identifier une balise. Par exemple la règle :

```
#titrepage
{
text-align: center;
font-size: large;
}
```

permet de spécifier la présentation du titre d'une page. Le code HTML utilisant cette feuille style est le suivant :

```
<body>
<h1 id="titrepage"> Le titre de ma page</h1>
```

Combinaisons de sélecteurs

Notons enfin que les différents sélecteurs que nous venons de voir peuvent se combiner. Par exemple :

```
.menu li, p, #toto { font-size: large; }
```

permet de spécifier que les balises li dans une balise de classe menu, les balises p et une balise d'identifiant toto devront être affichées en font large.

3.3 Association CSS - Page HTML

Il existe plusieurs façon d'associer une page HTML avec une feuille de style :

3.3.1 Feuille internes

On peut spécifier l'ensemble des règles à appliquer aux balises d'une page HTML grâce à la balise `<style></style>` que l'on doit placer à l'intérieur des balises `<head></head>`. L'exemple suivant spécifie les propriétés des paragraphes d'une page Web.

```
<html>
<head>
<style type="text/css">
<!--
p { text-align: justify; }
--!>
</style>
</head>
```

```
<body>
<p> mon paragraphe justifi{\'e}</p>
</body>
</html>
```

Notez l'utilisation des tags `< !--, -- !>` qui correspondent aux balises de commentaire des anciens navigateurs. Ces navigateurs incapables de lire des règles CSS sautent ce qui pour eux est un bloc de commentaire. En revanche les nouveaux navigateurs ignorent ces balises et interprètent les règles CSS.

3.3.2 Feuilles externes

Dans ce cas la feuille de style est située dans un fichier séparé et incluse dans la page Web à l'aide de la balise `<link></link>` qui doit être placée dans la section `<head></head>`. Le code suivant :

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="ma_feuille.css"/>
</head>
<body>
</body>
</html>
```

associe la feuille de style `ma_feuille.css` à une page HTML.

Notons que l'on peut également utiliser la balise `<style></style>` de la façon suivante :

```
<style>
@import url("ma_feuille.css")
</style>
```

toutefois cette méthode ne fonctionne pas avec les vieux navigateurs.

L'inclusion de feuilles de styles externes est sans doute une des méthode les plus utilisée. C'est également la méthode généralement recommandée. Le grand avantage de cette méthode est de permettre l'inclusion d'une même feuille de style dans plusieurs fichiers HTML ce qui garanti une meilleure homogénéité du site.

3.3.3 Feuille locale

L'insertion locale de style utilise l'attribut `style` pour spécifier les propriétés d'une balise. Par exemple le code suivant :

```
<html>
<head>
<title>Insertion locale</title>
</head>
```

```
<body>
<p style="text-align: justify; color: red;"> Mon paragraphe</p>
</body>
</html>
```

défini un paragraphe écrit en rouge et à l'alignement justifié.

La spécification de règles avec des feuilles internes ou externes est normalement préférable aux feuilles de styles locales. Ce type d'insertion peut toutefois être utile si l'on désire rajouter une propriété à une règle. Par exemple le code suivant :

```
<html>
<head>
<style type="text/css">
ul li
{
    font-weight: bold;
}
</style></head><body>
<ul>
    <li> un premier item</li>
    <li style="color: red"> un item important</li>
</ul>
</body></html>
```

écrira le second item en rouge - gras.

Notez toutefois, que la spécification de propriétés pour une balise HTML en particulier peut également être effectuée en utilisant l'attribut `id` qui permet d'identifier de façon unique une balise pour le fichier CSS.

3.3.4 Définition par l'utilisateur

L'utilisateur peut également fixer sur son navigateur des règles de style pour l'affichage par défaut des balises HTML. La spécification de ces feuilles de style varie d'un navigateur à l'autre. Galeon permet par exemple de charger directement une feuille de style qui sera appliquée automatiquement à toute page HTML chargée par le navigateur.

3.4 Cascade et Héritage

3.4.1 Cascade

Nous avons vu dans la Section 3.3 plusieurs façons d'associer des feuilles de styles à des pages HTML. Dans l'hypothèse où plusieurs de ces mécanismes sont simultanément utilisés, que se passe t'il en cas de conflit ? La règle générale pour gérer les conflits est *d'appliquer la dernière règle lue.* Donc :

- les feuilles de styles de l'utilisateur ont toujours la priorité,

- S'il existe des règles locales, elles ont la priorité sur les règles internes et externes.
- En cas de conflit entre des règles internes et externes, la dernière règle lue sera appliquée (cela dépend donc de l'ordre des balises),
- Si dans une même feuille de style différentes règles entrent en conflit, la dernière règle lue a la priorité.

3.4.2 Héritage

Pour bien comprendre la notion d'héritage, il faut se représenter un document HTML comme un arbre. La Figure 3.2 représente un code HTML et son arbre associé. Disons pour simplifier, que l'arbre se déduit de l'inclusion des balises. Un propriété intéressante d'un arbre CSS est qu'une propriété définie pour un noeud de l'arbre s'applique à tous ses noeuds enfants. Ainsi une règle définie pour le noeud `<body>` s'applique à tous les noeuds contenus dans le body, ce qui évite de les redéfinir. On peut bien sur s'affranchir de ce mécanisme en redéfinissant des propriétés dans les sous noeuds.

```
<html>
<head>
<title> ma page</title>
</head>
<body>
<h1> Ma page </h1>
<div class="menu">
<ul>
<li> un item</li>
</ul>
</div>
</body>
</html>
```

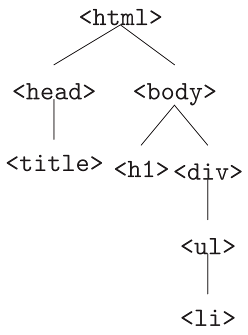


Fig. 3.2 – Un code HTML et l'arbre associé

3.5 Les textes

Il existe un certain nombre de propriétés susceptibles de s'appliquer aux textes et nous en avons déjà parcourus certaines. Les principales sont :

font-family : nom de police (Arial, Vernada…)
font-size : taille de police
font-style : *italic*, *oblique*
font-variant : SMALL-CAP
font-weight : **bold**,bolder,ligher,100, 900
font : raccourcis pour font-weight size font-familly
color : couleur
text-decoration : underline,overline,line-through(~~barré~~)

Il est inutile de faire ici une énumération de toutes les polices. Sachez simplement que les polices que vous allez spécifier doivent *a priori* être disponibles sur tous les systèmes (Windows, Unix(s), Linux, Mac…). Une telle disponibilité étant généralement impossible à garantir, il est recommandé de spécifier plusieurs polices. Le navigateur les essaieras séquentiellement et prendra la première police spécifié dont il dispose. Par exemple la règle suivante :

```
body { font-familly: vernada, arial; }
```

demande au navigateur de sélectionner en priorité la police Vernada, si celle-ci n'est pas disponible, la police arial sera sélectionnée. Si aucune des polices spécifiées n'est disponible, la police par défaut sera sélectionnée.

Dans la mesure où vous ne pouvez pas garantir la police qui sera effectivement sélectionnée, le choix d'une police doit représenter simplement un plus pour le site Web. En d'autres termes, toute l'esthétique de votre site ne doit pas reposer sur le choix d'une police alambiquée.

3.5.1 Taille des polices

Les feuilles de styles CSS nous permettent de spécifier la taille d'une police de façon absolue ou relative.

La spécification de taille de façon absolue utilise les unités représentées sur la Table 3.5.

mm	millimètre
cm	centimètres
in	inches
pc	pica (1pica=12pt)
pt	points
px	pixel

TAB. 3.5 – Unités absolues utilisées dans les fichiers CSS

La règle

```
p { font-size: 12pt; }
```

indique par exemple que nous voulons des paragraphes de taille 12pt.

Nous pouvons également indiquer une taille par rapport à la taille de la police par défaut du navigateur. On utilise pour cela les valeurs représentées sur la Table 3.6.

xx-large	large
x-large	large
large	large
medium	normal
small	small
x-small	+ small
xx-small	encore + small

Tab. 3.6 – Unités relatives pour la taille des polices dans les fichiers CSS

L'inconvénient de cette solution est que l'on ne connaît pas la police par défaut des navigateurs. On a donc un faible contrôle sur le résultat final. Un autre possibilité est de spécifier la taille par rapport à une taille précédemment définie (Table 3.7).

em	1em taille de la police précédente (ex font-size : .5em ;)
%	pourcentage de la taille de la police précédente (ex font-size : 50% ;)

Tab. 3.7 – Unités relative utilisées dans les fichiers CSS

3.5.2 Les couleurs

Les CSS vous permettent de spécifier 16 couleurs par leur nom et d'en construire de nouvelles. Les 16 couleurs disponibles sont représentées sur la Table 3.8.

aqua	bleu ciel	**navy**	bleu marine
black	noir	**olive**	vert olive
blue	bleu	**purple**	pourpre
fuchsia	fuchsia	**red**	rouge
gray	gris	**silver**	argent
green	vert	**teal**	vert sarcelle
lime	vert citron(limon)	**white**	blanc
maroon	marron/rouge foncé	**yellow**	jaune

Tab. 3.8 – Liste des noms de couleurs

L'on peut également spécifier une couleur en spécifiant directement les 3 coordonnées RVB. Chaque composante RVB est écrite en hexadécimal et ses valeurs varient de 0 à f(=16). Un triplet couleur s'écrit alors par `#rvb`. Un bleu marine peut par exemple être d'écrit par le code `#03a`.

3.5.3 L'alignement de textes

Les CSS proposent plusieurs propriétés pour aligner des blocs de textes. Les principales sont fournies sur la table 3.9.

text-indent	Indentation de paragraphe. La valeur de cette propriété est une longueur ou un pourcentage
margin-left	marge à gauche
margin-right	marge à droite
margin-top	marge du haut
margin-bottom	marge du bas
margin	toutes les marges
text-align	alignement du texte Valeurs (justify, align,left,right et none)

Tab. 3.9 – Propriétés d'alignement du texte

Notez que les CSS permettent de justifier des textes (`text-align :justify ;`) avec un retrait en début de paragraphe (`text-indent : 1cm`). On obtient donc un aspect visuel similaire à celui des traitement de textes ou de logiciels plus évolués comme LaTeX.

3.6 Les boîtes

Une façon d'assimiler le langage HTML est de comprendre toute balise comme une boîte. Il existe 2 types de base pour les boîtes : en ligne (inline) et paragraphe (block). Les boîtes en ligne sont générées par des balises telles que `<b></b>`, `<i></i>` ou `<span></span>`. Les boîtes de type paragraphe, par des balises telles que `<div></div>`, `<p></p>` ou `<table></table>`. Les boîtes de type en ligne sont disposées horizontalement les unes par rapport aux autres, alignées sur leur bord inférieur, tandis que les boîtes de type bloc sont disposées verticalement les unes par rapport aux autres, alignées sur leur bord gauche.

Il est possible de modifier le type d'une boîte avec la propriété display : display :inline ou display :block. Notez que la valeur none (display :none) permet de ne pas afficher la boîte.

Dans la suite de cette section nous ne nous intéressons qu'aux deux blocs conteneurs principaux à savoir `<p></p>` et `<div></div>`.

3.6.1 Formatage des boîtes

Le formatage d'une boîte se définit à l'aide des trois propriétés `border`, `padding` et `margin` (Figure 3.3).

- La propriété `margin` définie comme précédemment le positionnement de la boite par rapport à la boite englobante.

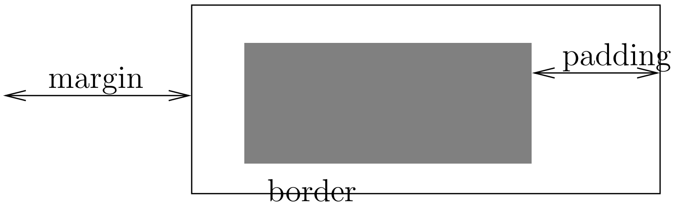


Fig. 3.3 – Formatage de boîtes

- La propriété `padding` spécifie les distances entre le bord de la boîte et son contenu. On peut spécifier soit une seule distance, soit 4 distances pour les 4 bords (en partant du haut et en tournant dans le sens des aiguilles d'une montre). La règle :
  `padding: 1mm 4mm 1mm 4mm`
  spécifie une marge de 1mm en haut et en bas et de 4mm à droite et à gauche.
- La propriété `border` définie la bordure de la boîte. Cette bordure a une épaisseur (width) un style (style) et une couleur (color) respectivement codées par les propriétés :

  border-width : épaisseur de la bordure. On peut mettre soit une seule valeur soit 4 valeurs pour les 4 bords (top, right,bottom, left),

  border-style : peut prendre les valeurs solid (–), dotted (···), dashed (- -) ainsi que double, groove, ridge, inset et outset.

  border-color : couleur de la bordure

  Les trois propriétés peuvent être spécifiées en une fois en utilisant `border width style color`. Par exemple la règle :

  ```
  p
  {
  border: 1mm solid red;
  }
  ```
  spécifie une bordure rouge en trait continu de 1mm d'épaisseur.

3.6.2 Positionnement des boîtes

Le positionnement d'une boîte s'effectue à l'aide de la propriété `position` éventuellement combinée avec les propriétés `left`, `top`, `right` et `bottom` qui spécifient le décalage de la boîte par rapport à un de ces bords.

Les différentes valeurs de position sont `static`, `absolute`, `fixed` et `relative`.

- La valeur `static` est la valeur par défaut conduisant au placement normal des boîtes.
- La valeur `absolute` permet de spécifier la position d'un bloc dans la fenêtre du navigateur.
- La valeur `fixed` est identique à la valeur `absolute` sauf que le bloc restera toujours à la même place même en cas de scrolling de la fenêtre. Ceci

peut par exemple permettre d'afficher toujours un menu en haut de page. Notons toutefois que cette valeur de `position` n'est pas supportée par Internet Explorer.
- La valeur `relative` permet de modifier la position d'un bloc par rapport à la place qu'il aurait occupé «normalement» avec la valeur `static`. Ceci est utile pour positionner des boîtes les unes par rapport aux autres.

L'exemple de la Figure 3.4 illustre une des fonctionnalités du positionnement absolu. La page est découpée en 3 parties avec un titre codant la partie du haut et prenant au maximum 10% de la hauteur de la fenêtre. Les 90% restant sont découpées suivant la largeur en 2 colonnes :
- la première colonne occupe 25% de la fenêtre et code les menus,
- la colonne restante occupe 70% de la fenêtre et code le contenu de la page.

3.6.3 Boîtes flottantes

Nous avons vu dans la Section 3.6.2 comment positionner une boîte soit vis à vis de sa position normale (valeur `relative`) soit vis à vis du coin en haut à gauche de la fenêtre du navigateur (positions `fixed` ou `absolute`).

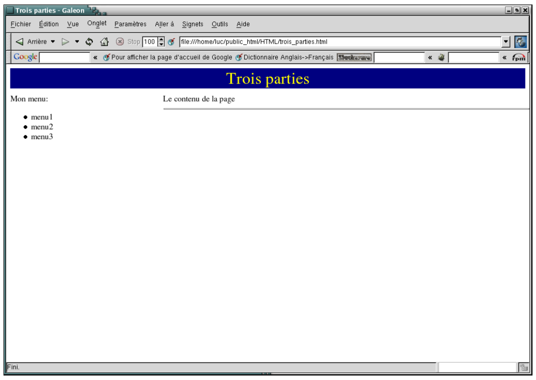
Un autre mécanisme de positionnement est fourni par les boîtes flottantes. Une boîte flottante est retirée du flux HTML normal et est placée le plus à droite (`float : right`) ou le plus à gauche (`float : left`) possible dans son conteneur. Le contenu suivant cette boîte flottante s'écoule le long de celle-ci, dans l'espace laissé libre.

Les règles suivantes permettent de créer la même page que sur Figure 3.4 mais sans utiliser la propriété `position`.

```
.titre
{
font-size: xx-large;
text-align: center;
background: navy;
color: yellow;
}
.menu
{
margin-top: 5mm;
float: left;
width: 25%;
}
.content
{
margin-top: 5mm;
margin-left: 25%
}
```

```
<html>
  <head>
    <title>Trois parties</title>
    <style>
      .titre
      {
      font-size: xx-large;
      text-align: center;
      background: navy;
      color: yellow;
      }
      .menu
      {
      position: absolute;
      top: 10%;
      width: 25%;
      }
      .content
      {
      position: absolute;
      top: 10%;
      left: 30%;
      width: 70%;
      }
    </style>
  </head>

  <body>
    <div class="titre">Trois parties</div>
    <div class="menu"> Mon menu:
      <ul>
        <li>menu1</li>
        <li>menu2</li>
        <li>menu3</li>
      </ul>
    </div>
    <div class="content">
      Le contenu de la page
    <hr/>
    </div>
  </body>
</html>
```



file:///home/luc/public_html/HTML/trois_parties.html

Trois parties

Mon menu:

- menu1
- menu2
- menu3

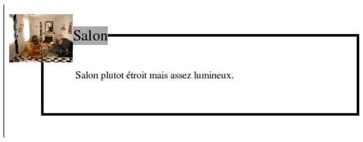
Le contenu de la page

FIG. 3.4 – Positionnement absolu de boîtes.

3.6.4 Gestion de la profondeur

Nous avons vu que la propriété `position` (combinée avec `top`,`left`,`bottom` ou `right`) permet de modifier la position des boîtes. Il est donc possible de superposer des boîtes. Cette superposition peut être utilisée pour créer des effets graphiques en gérant la profondeur des différents objets superposés. On utilise pour cela la propriété `z-index` qui permet de spécifier l'altitude d'une boîte. En cas de superposition seule la boîte de plus haute altitude sera visible. Les valeurs de `z-index` sont des valeurs entières. Notons qu'une valeur négative fait passer une boîte sous la fenêtre du navigateur et la rend donc invisible.

A titre d'exemple, la page représentée sur la Figure 3.5 superpose un cadre de boîte avec une image et une autre boîte.



Salon

Salon plutot étroit mais assez lumineux.

Fig. 3.5 – Gestion des profondeurs pour la superposition

3.7 Les listes

Les listes sont contrôlées à travers trois propriétés essentielles :

1. `list-style-type` spécifie le type de liste que l'on compte manipuler. Les principaux types sont représentés sur la Table 3.10.

Pour `ul`	
disc	●
circle	○
square	■

Pour `ol`	
decimal	1,2,3
decimal-leading-ezro	01,02,03
lower-roman	i,ii,iii,. . .
upper-roman	I,II,III. . .
lower-alpha	a,b,c. . .
upper-alpha	A,B,C. . .

Tab. 3.10 – Valeurs de `list-style-type`

2. `list-style-image` permet de spécifier une image à la place des puces prédéfinies. La règle suivante :

   ```
   ul { list-style-image: url(mon_item.jpg); list-style-type: circle;}
   ```

   indique que je désire utiliser `mon_item.jpg` comme puce pour ma liste. Si le navigateur est incapable de la charger, il utilisera des disques.
3. `list-style-position` prend deux valeurs (`inside` et `outside`) selon que les puces doivent «pénétrer» ou non dans le paragraphe en vis à vis.

Chapitre 4

Le langage Javascript

4.1 Introduction

Avant de décrire précisément le langage Javascript, il convient sans doute de préciser ses domaines d'applications ainsi que ses principales qualités et défauts. Les principaux défauts du langage Javascript sont :

1. Il est exécuté sur le navigateur du client,
2. de nombreuses propriétés du langage varient en fonction du navigateur utilisé.

Sa principale qualité est :

- Il est exécuté sur le navigateur du client.

Le fait que les programmes Javascripts soient exécutés par le navigateur du client apparaît à la fois comme une qualité et un défaut. Ceci est moins paradoxal qu'il peut sembler au premier abord. En effet tout traitement effectué sur le navigateur du client soulagera d'autant le serveur de pages HTML. Il est par exemple inutile d'envoyer des données au serveur Web si l'on peut détecter par avance l'incohérence ou l'inexactitude de celles ci. Cette propriété de Javascript est très souvent utilisée pour vérifier l'exactitude des données saisies par l'utilisateur.

Inversement, un programme tournant sur la machine d'un client ne peut accéder qu'aux données disponibles sur celui-ci. Il est par exemple exclu d'interroger des bases de données distantes par des programmes Javascript (ceci n'est pas rigoureusement impossible, mais peu pratique, difficile à réaliser et quasiment jamais utilisé en pratique).

Le second défaut de Javascript est essentiellement due à Internet explorer qui a développé une architecture de classes différente de celle utilisée par Sun et Netscape. Le résultat de ce choix stratégique est que la plupart des programmes Javascript doivent être souvent développés en deux ou trois versions en fonction du navigateur ce qui rend le développement extrêmement pénible.

En conclusion, Javascript est comme son nom l'indique un *petit* langage de script souvent utilisé pour effectuer de petites taches permettant de soulager le serveur. Javascript est également utilisé pour créer de petits effets visuels (menus déroulants, bannière défilante…). Notons enfin que Javascript peut également être utilisé pour crypter des mots de passe (ou des textes) sur le client. De cette façon, les informations sont transmises de façon cryptées au serveur ce qui permet de se garantir d'un piratage de la ligne (le pirate ne peut intercepter que des mots passe cryptés).

4.2 Déclaration de variables

Le langage Javascript peut se voir comme un mélange de C et de Java sans toutefois avoir toutes les fonctionnalités de ces deux langages. Le JavaScript est donc relativement facile à apprendre pour toute personne possédant des bases dans l'un de ces deux langages. Il possède toutefois un certain nombre de spécificités que nous allons parcourir dans les sections qui suivent.

En Javascript, vous disposez d'une liberté quasi totale en ce qui concerne le choix des noms de vos variables. Cependant :

- Chaque variable doit commencer par une lettre comprise entre a et z ou A et Z.
- Les caractères spéciaux, comme le μ ou les caractères accentués sont interdits (les espaces ne sont pas acceptés)
- Comme tous langages, des mots réservés existent (éléments JavaScript) donc attention quand même !

Il est important de savoir que JavaScript respecte la casse (différence majuscule/minuscule). Ainsi, Nombre et nombre sont deux variables différentes.

Le Javascript étant très souple au niveau de l'écriture, la déclaration des variables peut se faire de deux façons :

- soit de façon explicite, en faisant précéder la variable du mot clé *var* qui permet d'indiquer de façon rigoureuse qu'il s'agit d'une variable : `var chaine= "bonjour"`
- soit de façon implicite, en écrivant directement le nom de la variable suivie du caractère = et de la valeur à affecter : `chaine= "bonjour"`

Le navigateur détermine seul qu'il s'agit d'une déclaration de variable. Même si une déclaration implicite est tout à fait reconnue par le navigateur, il est tout de même plus rigoureux de déclarer les variables de façon explicite avec le mot var...

Lorsqu'une variable est déclarée sans le mot clé var, c'est-à-dire de façon implicite, elle est accessible de partout dans le script (n'importe quelle fonction du script peut faire appel à cette variable). On parle alors de variable globale

Lorsque l'on déclare de façon explicite une variable en javascript (en précédant sa déclaration avec le mot var), sa portée dépend de l'endroit où elle est déclarée.

Une variable déclarée au début du script, c'est-à-dire avant toute fonction sera globale, on pourra alors les utiliser à partir de n'importe quel endroit dans le script. Une variable déclarée par le mot clé var dans une fonction aura une portée limitée à cette seule fonction, c'est-à-dire qu'elle est inutilisable ailleurs, on parle alors de variable locale.

Pour les types de variables complexes, comme les tables ou les objets, il faut les déclarer avec l'opérateur new. Cet opérateur est notamment utilisé pour créer des tableaux.

4.3 Tests et boucles

4.3.1 Les tests

Les instructions conditionnelles en Javascript sont au nombre de deux :

L'instruction if : permet un test conditionnel. Les variantes du if sont :

- `if (condition) {instruction}`,
- `if (condition) {instruction} else {instruction}`
- `if (condition) {instruction} else if (condition) {instruction}`...

Un exemple de d'instruction `if` est fourni ci après :

```
if("e" == "d")
{
   document.write("Non... - ");
}
else
   if("a" == "a")
        document.write("Oui... - "); // Renvoie true
   else
   {
        document.write("Oui... - "); // Renvoie true
   }
```

Notez l'absence d'accolades lorsqu'une des branches du if ne comporte qu'une seule instruction ainsi que les commentaires codés par des `//`.

L'instruction switch : cette instruction de branchement multiple se retrouve dans beaucoup de langages. Elle permet de comparer une variable à une série de constantes et d'effectuer une série instructions en fonction du résultat du test. Le bloc d'instruction étiqueté par `default` est exécuté si aucun autre test n'est vrai.

```
var toto = "titi";
switch (toto) {
   case 'tata':
      document.write("Perdu toto vaut"+toto);
      break;
   case 12:
      document.write("Perdu toto n est pas un chiffre");
      break;
   case 'titi':
      document.write("Gagn&eacute;");
      break;
   default:
```

```
            document.write("Pas de chance rejouez");
            break;
     }
```

4.3.2 Les boucles

Les boucles toujours de façon très classique se décomposent en trois jeux d'instruction.

1. `for(init ;test ;incr)`
2. `while(condition){instruction}`
3. `do {instruction} while(condition) ;`

La principale différence entre les boucles `while` et `do-while` est que la séquence d'instruction est exécutée au moins une fois dans le cas d'une boucle `do-while`. Voici trois boucles effectuant le même traitement :

```
var i;
for(i=0;i<10;i++)
    document.write("i vaut : "+i);
i=0;
while(i <10)
{
   document.write("i vaut : "+i);
   i++;
}
i=0;
do
{
   document.write("i vaut : "+i);
   i++;
}
while(i<10)
```

On utilisera de préférence l'instruction `for` quand le nombre d'itérations est connu avant la boucle.

Les ruptures de boucles peuvent être effectuées à l'aide des instructions représentées sur la Table 4.1. Bien utilisées, ces instructions peuvent clarifier le code et le rendre plus efficace. Elles peuvent malheureusement le rendre également totalement illisible. C'est donc une question de nuance et d'expérience.

4.3.3 Les tableaux

Les tableaux JavaScript sont décrits par la classe Array. L'allocation d'un tableau se fait donc de la façon suivante :

```
var tab_vide = new Array(10); // tableau de dim 10
var tab_plein= new Array("item1","item2","item3");
```

return	:	sort de la boucle et de la fonction. S'utilise par exemple pour rechercher une valeur dans un tableau
break	:	sort de la boucle (branchement après l'accolade fermante)
continue	:	passe à l'itération suivante (branchement juste avant l'accolade fermante)

TAB. 4.1 – Ruptures de boucles en JavaScript

On peut bien entendu créer des tableaux multidimensionnels en créant des tableaux de tableaux (Figure 4.1). Notez l'ulisation du champ `length` dans ce dernier exemple. Ce champ code la taille du tableau.

```
var mat=new Array(5);
var i,j;
for(i=0;i<mat.length;i++)
{
   mat[i]=new Array(5);
   for(j=0;j<mat[i].length;j++)
       mat[i][j]=i*j;
}
```

(a) Le code JavaScript

0	0	0	0	0
0	1	2	3	4
0	2	4	6	8
0	3	6	9	12
0	4	8	12	16

(b) le tableau mat

FIG. 4.1 – Création de tableaux de tableaux en JavaScript

Javascript permet également de créer des tableaux associatifs. On peut par exemple créer le tableau suivant :

```
var email=new Array(10);
email["jean"]="jean@free.fr";
email["paul"]="paul@wanadoo.fr";
```

Les principales méthodes de l'objet Array sont données dans la Table 4.2. Ces méthodes sont illustrées sur un exemple Figure 4.2.

4.4 Fonctions

La déclaration d'une fonction en JavaScript s'effectue à l'aide du mot clé `function`.

```
function aire(largeur,hauteur)
{
    return largeur*hauteur;
}
```

Plusieurs remarques s'imposent :

Concat(tab1, tab2[, tab3, ...])	permet de concaténer plusieurs tableaux.
join(tableau) ou Tableau.join()	renvoie une chaîne de caractères contenant tous les éléments du tableau.
pop(tableau) ou Tableau.pop()	supprime le dernier élément du tableau et retourne sa valeur.
Tableau.push(valeur1[, valeur2, ...])	ajoute un ou plusieurs éléments au tableau.
Tableau.reverse()	inverse l'ordre des éléments du tableau.
Tableau.shift()	supprime le premier élément du tableau.
Tableau.slice()	renvoie un tableau contenant une partie (extraction) des éléments d'un tableau.
Tableau.splice(i,nb,[val1,])	retire nb éléments du tableau à partir de la position i. Renvoi les éléments supprimés. Les éléments val_i s'ils sont fournis sont substitués aux éléments remplacés
Tableau.sort()	trie les éléments d'un tableau.
Tableau.unshift(val1[, val2, ...])	ajoute val1,val2…en tête de tableau
Tableau.toString()	concatène les objets de l'Array en une String.
Tableau.valueOf	retourne tout simplement la valeur de l'objet Array auquel elle fait référence.

Tab. 4.2 – Les principales méthodes de la classe tableau en JavaScript

```
var tab=new Array(10);
var i;
for(i=0;i<tab.length;i++)
   tab[i]=i;
with(document)
{
 write(tab.toString()+"<br/>");
 tab.shift();
 write(tab.toString()+"<br/>");
 unshift(0);
 write(tab.toString()+"<br/>");
 tab.push(10);
 write(tab.toString()+"<br/>");
 tab.pop();
 write(tab.toString()+"<br/>");
 tab.reverse();
 write(tab.toString()+"<br/>");
 tab.sort();
 write(tab.toString()+"<br/>");
 write(tab.splice(0,3,’a’,’b’,’c’)+"<br/>");
 write(tab.toString()+"<br/>");
}
```

(a) code

méthode	résultat
	0,1,2,3,4,5,6,7,8,9
shift :	1,2,3,4,5,6,7,8,9
unshift	0,1,2,3,4,5,6,7,8,9
push	0,1,2,3,4,5,6,7,8,9,10
pop	0,1,2,3,4,5,6,7,8,9
reverse	9,8,7,6,5,4,3,2,1,0
sort	0,1,2,3,4,5,6,7,8,9
splice	0,1,2
splice	a,b,c,3,4,5,6,7,8,9(res)

(b) Résultat

Fig. 4.2 – Illustrations des méthodes de manipulation de tableaux

- Les paramètres sont passés par valeur (pas de modification des paramètres).
- Une fonction sans return est une procédure.

4.5 Classes

Les définitions de classes en Javascript sont basées sur les fonctions et l'opérateur new. L'instruction :

```
var toto=new ma_fonction()
```

crée un nouvel objet `toto` en considérant la fonction `ma_fonction` comme le constructeur de l'objet. Les champs de l'objet sont déclarés implicitement et initialisés ou utilisés par la syntaxe `this.champ`. L'exemple suivant crée un objet rectangle en spécifiant sa largeur et sa hauteur.

```
function Rectangle(x,y)
{
    this.largeur=x;
    this.hauteur=y;
}
var mon_rect=new Rectangle(10,10);
```

On peut ajouter des méthodes à une classe de deux façons :

- Soit en spécifiant ces méthodes dans le constructeur. L'exemple suivant crée un objet rectangle avec un calcul d'aire.

  ```
  function aire()
  {
      return this.largeur*this.hauteur;
  }
  function Rectangle(x,y)
  {
      this.largeur=x;
      this.hauteur=y;
      this.aire=aire;
  }
  var mon_rect=new Rectangle(10,10);
  document.write("Aire du rectangle : "+mon_rect.aire());
  ```

- Soit en l'ajoutant dynamiquement au prototype de la classe à l'aide de la propriété `prototype`. L'exemple suivant ajoute la méthode perimètre à la classe Rectangle.

  ```
  function perimetre()
  {
          return 2*(this.largeur+this.hauteur);
  }
  var mon_rect=new Rectangle(10,10);
  ```

```
Rectangle.prototype.perimetre=perimetre;
document.write("Aire : "+mon_rect.aire());
document.write("Perimetre  : "+mon_rect.permimetre());
```

Notez que l'on peut de la même façon ajouter dynamiquement des champs à une classe.

Si on doit fréquemment accéder aux champs ou aux méthodes d'un même objet, l'on peut utiliser l'instruction `with` qui permet d'utiliser implicitement l'objet. Les instructions précédentes pourraient donc s'écrire :

```
var mon_rect=new Rectangle(10,10);
Rectangle.prototype.perimetre=perimetre;
with (mon_rect)
{
    document.write("Aire : "+aire());
    document.write("Perimetre  : "+permimetre());
}
```

4.6 Classes et objets prédéfinis

Les classes et objets prédéfinis par JavaScript sont essentiellement au nombre de quatre `Math`, `String`, `img` et `Date`.

4.6.1 L'objet Math

Les fonctions mathématiques usuelles doivent être préfixées par le nom de l'objet Math, duquel elles dépendent. Ce sont les "méthodes" de calcul de l'objet Math. Par exemple :

- Math.PI désigne une propriété de l'objet Math : le nombre PI
- Math.sin(1.50) appelle une méthode de l'objet Math et calcule sin (1.50), l'angle étant exprimé en radians.

Les principales méthodes sont décrites Table 4.3.

racine carré	:	Math.sqrt()
log, exponentielle	:	Math.log() Math.exp()
valeur absolue	:	Math.abs()
fonction trigonométriques	:	Math.cos (), Math.sin() , Math.tan()
entier immédiatement inférieur / supérieur	:	Math.floor(), Math.ceil()
puissance	:	Math.pow(base, exposant)
opérateurs max, min	:	Math.max() , Math.min()
nombre aléatoire dans $[0, 1[$	:	Math.random()
entier le plus proche	:	Math.round()

Tab. 4.3 – Principales méthodes de l'objet Math en JavaScript

4.6.2 La classe String

Un objet de la classe String peut se déclarer implicitement ou explicitement :

```
var toto="toto";
var titi=new String("titi");
```

La concaténation de chaîne s'effectue avec l'opérateur `+`. Notez que JavaScript s'occupe implicitement de la conversion des différents types en String. Ainsi l'instruction `toto="J'ai étudié JavaScript pendant "+4+" heures"` donnera la chaîne de caractères : «J'ai étudié JavaScript pendant 4 heures» où l'entier 4 est converti implicitement en String.

Il est également possible d'effectuer des opérations de conversion inverses à l'aide des fonctions représentées sur la Table 4.4. Une illustration de ces conversion est fournie sur la Figure 4.3.

eval()	:	évalue numériquement une expression arithmétique fournie sous forme de chaîne de caractères.
parseInt()	:	donne un nombre entier résultant de la conversion (si possible) d'une chaîne de caractères. Si la conversion n'est pas possible, la valeur renvoyée est 0.
parseFloat()	:	donne un nombre décimal de la même façon.
toString()	:	convertit l'objet (nombre généralement) en une chaîne représentant le nombre écrit dans la base indiquée.

Tab. 4.4 – Conversion de chaînes en nombres en JavaScript

```
var a = 5+ "007";
var b = 5+ parseInt("007");
var c = 2+ parseFloat("1.1416");
var d=255;
var e="2*Math.cos(3.14)";
with (document)
{
write(" a,b,c = ", a,b,c);
write(d+ " en hexa :"+ d.toString(16));
write(e+"="+eval(e));
}
```

(a) code

```
a,b,c = 5007, 12, 3.1416
255 en hexa : ff
2*Math.cos(3.14)=-1.9999974
```

(b) résultat

Fig. 4.3 – Illustration de conversions de chaînes en nombres en JavaScript

JavaScript fourni également tout un jeu d'instructions pour manipuler des chaînes de caractères. Les principales méthodes sont représentées sur la Table 4.5.

toUpperCase()	:	met une chaîne en majuscules
substring(d, l)	:	extrait une partie de chaîne, à partir du caractère de position d+1, jusqu'à l.
charAt(n)	:	donne le n^{e} caractère de la chaîne
indexOf(s,[pos])	:	renvoi la 1re position du caractère de chaîne égal au 1ercaractère de s à partir de la position pos dans chaîne (paramètre facultatif).
split(séparateur)	:	renvoi un tableau de chaînes de caractères séparées par le séparteur dans la chaîne.

TAB. 4.5 – Principales méthodes de manipulation de String en JavaScript

4.6.3 La classe Image

JavaScript propose à partir de la version 1.2 une classe image. Contrairement aux balises `<img></img>` (Section 2.6), les objets image de JavaScript peuvent être préchargés et stockés en local sur le cache de l'utilisateur.

Une image se crée à l'aide du constructeur `Image`. Le principal champ d'une image est son fichier source `src`.

```
var mon_img=new Image();
mon_img.src='Images/mon_image.gif';
```

Les autres champs d'une image sont les attributs de la balise `<img></img>` (name, src, lowsrc, height, width, border, hspace, vspace...). La Figure 4.4 montre un code Javascript qui change l'image chaque fois que la souris rentre ou sort de celle-ci. Ce type d'effet est appelé un RollOver.

4.6.4 La Classe Date

Comme son nom le laisse supposer, la classe Date permet d'obtenir des informations sur l'année, le mois, le jour et l'heure correspondant à une date. Un objet date se crée par l'instruction `new Date`. Les différents paramètres de ce constructeur sont représentés sur la Table 4.6. On utilise ensuite les méthodes de l'objet (Table 4.7) pour obtenir les différentes informations sur la date courante. Notez la méthode `getTime()` très utile pour convertir des dates, soustraire ou ajouter deux dates,.... Une illustration de ces méthodes est fournie sur la Figure 4.5.

4.7 Insertion de code JavaScript

Il existe trois manières d'insérer du code JavaScript dans une page HTML :

1. Insertion pour exécution différée
2. Insertion à l'intérieur d'une balise HTML
3. Insertion pour exécution directe

```
<html>
<head>
<script langage="text/javascript">

var tab_image = new Array(5);
for (var i=0; i< tab_image.length;i++)
{
        tab_image[i]= new Image();
        tab_image[i].src='images/img'+i+'.jpg';
}
var n=0;

function change()
{
   if(isNaN(n))
      n=0;
   n= ++n;
   if (n == tab_image.length) n=0;
   window.document.image.src=tab_image[n].src;
}
</script>
<style type="text/css">
#rollover
{
   width: 50;
   height: 50;
}
</style>
</head>

<body>
<h4>Un exemple de RollOver</h4>

<img name="image" id="rollover"
     onmouseover="JavaScript:change()"
     onmouseout="JavaScript:change()"
     src="images/img1.jpg"/>
</body>
</html>
```

Fig. 4.4 – Illustration du fonctionnement de la classe Image en JavaScript

Date()	renvoie un objet codant la date courante
Date(”jour, mois date année heures :minutes :secondes”)	le paramètre est une chaîne de caractères suivant scrupuleusement la notation ci-contre
Date(année, mois, jour)	les paramètres sont trois entiers séparés par des virgules. Les paramètres omis sont mis à zéro par défaut
Date(année, mois, jour, heures, minutes, secondes[, millisecondes])	les paramètres sont six entiers séparés par des virgules. Les paramètres omis sont mis à zéro par défaut

Tab. 4.6 – Paramètres du constructeur de la classe Date en JavaScript

```
<html>
<head>
<title>Affichage de la date</title>
<script type="text/javascript">
var jours=new Array("Lundi", "Mardi","Mercredi",..);
var mois=new Array("Janvier","F{\'e}vrier",..);
function aff_date()
{
      var ma_date=new Date();
      var Jour=jours[ma_date.getDay()];
      var Mois=mois[ma_date.getMonth()];
      var annee=ma_date.getFullYear();
      return Jour+" "+ma_date.getDay()+" "+Mois+" "+annee;
}
</script></head>
  <body>
    <script type="text/javascript">
      document.write("Nous sommes le "+aff_date());
    </script>
  </body></html>
```

Fig. 4.5 – Une utilisation des méthodes de la classe Date. Le texte affiché est par exemple «Nous sommes le Jeudi 3 Juillet 2004»

méthode	description	valeur de retour
getDate()	valeur du jour du mois	entier (entre 1 et 31) correspondant au jour du mois
getDay()	valeur du jour de la semaine	entier qui correspond au jour de la semaine (0 : dimanche,1 Lundi...)
getFullYear()	valeur de l'année sur 4 chiffres	entier qui correspond à l'année (XXXX) : 2004
getHours()	valeur de l'heure	entier (entre 0 et 23)
getMilliseconds()	nombre de millisecondes	entier (entre 0 et 999)
getMinutes()	valeur des minutes	entier (entre 0 et 59)
getMonth()	numéro du mois	entier (entre 0(Janvier) et 11(Décembre))
getSeconds()	nombre de secondes	entier (entre 0 et 59)
getTime()	nombre de millisecondes depuis le 1er janvier 1970	entier
getTimezoneOffset()	différence entre l'heure locale et l'heure GMT (Greenwich Mean Time)	entier représentant le nombre de minutes de décalage
getYear()	valeur de l'année sur 2 chiffres	entier XX)

Tab. 4.7 – Méthodes de l'objet Date

Dans tous les cas, l'insertion s'effectue grâce à la balise `<script></script>` dont les attributs sont décrits dans la Table 4.8.

charset	:	Indique le jeu de caractères utilisés par la ressource appelée par le lien.
defer	:	Indique au navigateur qu'il doit afficher le contenu normal de la page pendant son chargement sans attendre que le script génère un éventuel contenu.
src	:	Indique l'adresse d'un script externe au document.
type	:	Définit le type de script.

Tab. 4.8 – Liste des attributs de la balise `<script></script>`

Bien que l'insertion directe soit souvent utilisée pour l'apprentissage de JavaScript, la principale utilisation de ce langage est de réagir à des évènements. Ce type d'action est généralement géré à l'aide de fonctions définies en exécution différées et appelées à l'intérieur de balises HTML.

4.7.1 Exécution différée

On l'appelle exécution différée, car le code est d'abord lu par le navigateur, stocké en mémoire, pour ne s'exécuter que sur demande expresse. Le code JavaScript est placé dans la section `head`. Dans ce cas, le code s'exécutera seulement lors d'un événement généré par intervention de l'utilisateur ou lors d'un appel de fonction dans un autre script défini dans la section `body`.

```
<html>
    <head>
    <title> </title>
    <script language="JavaScript" type="text/javascript">
    <--
..
    -->
    </script>
    </head>
<body>html, code Javascript</body></html>
```

On peut noter dans le script précédent la présence simultanée des attributs `language` et `type` qui indiquent la même chose à savoir que le script est écrit en JavaScript. Cette redondance s'explique par des raisons historiques : Jusqu'à la norme HTML 4.0, les pages utilisaient l'attribut `language`. Cet attribut a été remplacé par l'attribut `type` à partir de la version 4.0. Pour être compatible avec tous les navigateurs, le plus simple est de mettre les deux balises.

Un autre élément remarquable est la présence des anciennes balises de commentaires qui permettent aux vieux navigateurs ne comprenant pas les balises `<script></script>` d'ignorer le contenu de celles-ci. Nous avons vu le même type d'astuce dans le cadre de la balise `<style></style>` pour les feuilles de style (page 42).

Lorsque le code placé dans section `<head></head>` est réutilisable pour d'autres pages HTML, on peut le définir dans un fichier séparé est l'inclure grâce à l'attribut `src`. L'insertion du code se fait alors comme suit :

```
<html>
    <head>
    <title> </title>
    <script src="javascript/ma_page.js"
            language="JavaScript" type="text/javascript"/>
    </head>
<body>html, code JavaScript </body></html>
```

4.7.2 Réaction aux évènements

Certaines balises HTML acceptent de réagir à des évènements, soit provoqués par l'intervention de l'utilisateur, soit provoqués par une action du navigateur lui même, comme le chargement de la page HTML (événement onLoad) ou la fermeture de la page HTML en cours (événement onUnload). Dans ce cas, le code JavaScript peut être aussi inséré directement au niveau de la balise en question.

Le squelette de la page HTML est alors :

```
<html>

    <head>
    <title> .......... </title>

    <script language="JavaScript">
     <--
    (cet emplacement peut {\^e}tre vide,
    mais il est indispensable, car il
    indique au navigateur qu'il va devoir
    interpr{\'e}ter du code JavaScript.)
   -->
    </script>

    </head>

<body>
<balise html evenement="javascript:place du code {\'e}v{\'e}nement"></balise>
</body></html>
```

La liste des évènements suceptibles est fournie dans la Table 4.9. Notez que nous avons déjà rencontré une application des évènements `onMouseOver` et `onMouseOut` dans l'exemple de la Figure 4.4.

Évènement	decription	objets concernés
onAbort	l'utilisateur interrompt le chargement de l'image	image
onBlur	perte de focus (clic sur un autre élément)	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, window
onChange	modification du contenu d'un champs de données	FileUpload, Select, Submit, Text, TextArea
onClick	clique sur l'élément associé à l'événement.	Button, document, Checkbox, Link, Radio, Reset, Select, Submit
onDblclick	double-clique sur l'élément (un lien hypertexte ou un élément de formulaire)(Javascript 1.2)	document, Link
onDragdrop	glisser-déposer sur la fenêtre du navigateur (Javascript 1.2)	window
onError	une erreur apparaît durant le chargement de la page (Javascript 1.1.)	Image, window
onFocus	sélection de l'élément	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, window
onKeypress	maintient d'une touche du clavier enfoncée (Javascript 1.2)	document, Image, Link, TextArea
onLoad	chargement de la page en cours	Image, Layer, window
onMouseOver	la souris au-dessus d'un élément	Area, Layer, Link
onMouseOut	la souris quitte un élément (Javascript 1.1)	Layer, Link
onReset	ré-initialisation d'un formulaire à l'aide du bouton Reset	form
onResize	redimensionnement de la fenêtre du navigateur	window
onSelect	sélection d'un texte (ou une partie d'un texte) dans un champ de type "text" ou "textarea	text, Textarea
onSubmit	clique sur le bouton de soumission d'un formulaire	Form
onUnload	quitte la page en cours	Window

Tab. 4.9 – Liste des évènements JavaScript

4.7.3 Exécution directe

On l'appelle exécution directe, car le code s'exécute automatiquement lors du chargement de la page HTML dans le navigateur. Le code JavaScript est placé dans le corps même de la page HTML, entre les balises `<body></body>`. Dans ce cas, le code s'exécutera automatiquement lors du chargement de la page HTML dans le navigateur du client, avant même que le contenu de la page HTML ne s'affiche à l'écran. Le squelette de la page HTML est alors :

```
<html>
    <head>    <title>mon titre </title>   </head>
<body>
<script language="JavaScript" type="text/javascript">
<--
..
-->
</script>
</body>
</html>
```

4.8 Hierarchie des objets prédéfinis

Le navigateur crée une hiérarchie d'objets disponibles à partir de Javascript. Par exemple, la fenêtre du navigateur est connue sous le nom window. La page HTML affichée dans le navigateur est connue sous le nom document. Il s'agit de l'objet le plus utilisé quand l'on fait des scripts Javascript du côté client. Une liste des principaux objets est fournie sur la Figure 4.6.

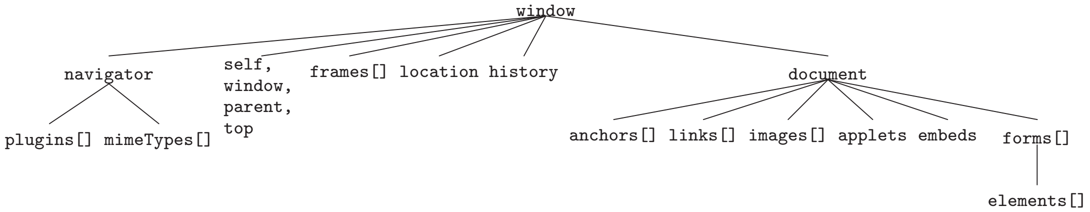


Fig. 4.6 – hierarchie des objets prédéfinis en JavaScript

Les principales propriétés de ces objets sont les suivantes :

navigator : propriétés concernant le navigateur utilisé.
principale propriétés : appCodeName, appName, appVersion, cookieEnabled, platform, userAgent

plugins : liste des plugins installés sur le navigateur

mimeTypes : types mimes supportés par la navigateur (voir Section 1.1.4)

window : objet racine de la hiérarchie. Il y a aussi un objet window pour chaque cadre sur une page multi-cadres.

Quelques propriétés et méthodes : location, status, open(), close(), alert(), confirm(), prompt(), blur(), focus(), setInterval(), setTimeout().

Le champ status correspond au message généralement affiché en bas à gauche du navigateur. Certaines personnes «jouent» avec ce champs pour faire défiler du texte dans cette zone.

La propriété setTimeout() est souvent utilisée pour rediriger le navigateur vers un autre site au bout d'un temps donné. Le code suivant redirige le navigateur au bout de 5 secondes.

```
setTimeout("location.href=’destination.html’",5);
```

on peut également obtenir le même effet avec la balise suivante dans le section `head` :

```
<meta http-equiv="refresh" content="5;url=destination.html"/> .
```

Les fonctions `alert`, `confirm` et `prompt` sont également très utilisées pour communiquer avec l'utilisateur. Le rôle de chacune de ces fonctions est détaillé Table 4.10.

document : objet qui gère les propriétés du document. Il n'y a qu'un seul objet document par page web.

Quelques propriétés et méthodes : cookie, referrer, lastModified, bgColor, fgColor.

location : propriétés concernant l'URL correspondant au document.

Quelques propriétés et méthodes : href(voir window), reload(), replace()

history : propriétés concernant les URL visitées précédemment par le client.

Principale méthode : go(-n) revient de n étapes en arrière dans l'historique.

alert(message)	:	affiche un message
bool confirm(message)	:	affiche un message et renvoie vrai ou faux selon que l'utilisateur ait cliqué sur OK ou Cancel
String prompt(message)	:	permet à l'utilisateur de rentrer des données. Celles-ci sont renvoyées par la fonction

TAB. 4.10 – Fonctions d'ouverture de boîte de dialogue.

4.8.1 Hierarchie des objets dans les formulaires

La liste des formulaires du document est contenu dans le tableau `window.document.forms[])` (Figure 4.6). On peut accéder à un formulaire particulier, par exemple :

```
<form name="main">
<input type="text" name="telephone"/>
</form>
```

de trois façons :

1. `document.forms["main"]`,
2. `document.forms[0]` en supposant que main soit le premier formulaire (méthode déconseillée)
3. `document.main`,

L'accès à un élément d'un formulaire se fait ensuite grâce au tableau elements. On a à nouveau trois solutions :

1. `document.forms["main"].elements["telephone"]`,
2. `document.forms["main"].elements[0]` non conseillé
3. `document.forms["main"].telephone`.

On peut ensuite modifier dynamiquement les propriétés des différents objets du formulaire en réagissant aux évènements `onclick`, `onfocus`...(Voir Table 4.9). Notons que la gestion des formulaires constitue un des gros domaines d'application de Javascript. En effet, la gestion des évènements permet de rendre les formulaires beaucoup plus réactifs en interpellant l'utilisateur dès qu'une erreur est commise ou en montrant/cachant dynamiquement des informations en fonction des valeurs déjà remplies par l'utilisateur.

La Figure 4.7 illustre ce fonctionnement. On propose à l'utilisateur de donner soit son adresse soit son numéro de téléphone. Si l'utilisateur préfère donner son téléphone, il valide le champ téléphone, on y donne alors le focus et on invalide le champ adresse. L'opération inverse est effectué si l'utilisateur choisit de donner son adresse et on invalide les deux champs si l'utilisateur ne veut pas être contacté.

4.9 Petits «trucs» de programmation JavaScript

4.9.1 Reconnaître le navigateur

En raison des nombreuses incompatibilités entre Internet Explorer et les navigateurs dérivés de Netscape (Mozilla, Galeon, Epiphany...), il est souvent indispensable de savoir reconnaître le navigateur du visiteur pour exécuter des scripts différents en fonctions de la tâche à accomplir et des spécificités des navigateurs.

Pour reconnaître les navigateurs qui respectent la norme officielle HTML, il suffit de tester si la fonction getElementById est reconnue :

```
if (document.getElementById) { // Navigateur qui supporte DHTML
officiel }
```

Ici, si la fonction existe, le test est vrai.

De la même manière, pour tester Netscape de version 4.X, on utilise ce code :

```
  if (document.layers) {
    // Navigateur Netscape 4.X
  }
```

```
<html>
  <head>
    <title>formulaire</title>
    <script type="text/javascript">
      function tel()
      {
      document.form.telephone.disabled="";
      document.form.telephone.focus();
      document.form.adresse.disabled="true";
      return true;
      }
      function adr()
      {
      document.form.adresse.disabled="";
      document.form.adresse.focus();
      document.form.telephone.disabled="true";
      return true;
      }
      function none()
      {
      document.form.adresse.disabled="true";
      document.form.telephone.disabled="true";
      return true;
      }
    </script>
  </head>

  <body>
    <h1>Contact</h1>
    <form name="form">
      <input type="radio"  value="adr" name="mode"
             onclick="return adr()" />lettre<br/>
      <input type="radio" value="tel"
             name="mode" onclick="return tel()"/>t&eacute;l&eacute;phone<br/>
      <input type="radio" value="none"
             name="mode" onclick="return none()" checked/>pas de contact<br/>
      Num&eacute;ro :<input type="text" name="telephone" disabled="true"><br/>
      Adresse :<input type="textarea" name="adresse" disabled="true">
    </form>
  </body>
</html>
```

Fig. 4.7 – Réactivité des formulaires : un exemple

Ici l'objet layers est propre à Netscape 4.X. Ce test est vrai uniquement avec ce navigateur.

Pour reconnaître Internet Explorer 4, on utilise ce test :

```
if ((document.all)&&(!document.getElementById)) {
  // Navigateur IE 4 uniquement
}
```

Ici, all est un objet propre à Internet Explorer. Pour être sûr de ne pas détecter IE5 aussi, on vérifie en plus que la fonction getElementById n'est pas supportée.

Il peut être aussi utile de reconnaître les navigateurs qui ne supportent aucune des versions du DHTML. Le plus simple est d'écrire :

```
if ((!document.layers)&&(!document.all)&&(!document.getElementById)) {
    // Navigateur trop ancien !
  }
```

4.10 Accéder aux éléments

4.10.1 Retrouver un élément par son Id

Du fait de l'incompatibilité entre les différents navigateurs, l'accès aux différents éléments d'une page Web à partir de l'arborescence des objets est souvent une tâche assez délicate.

Une autre solution consiste à utiliser attribut `id` disponible dans pratiquement toutes les balises. De fait, tous les navigateurs récents (Netscape 6.0 et Internet Explorer 5.0) implémentent la méthode `document.getElementById(id)` permettant de retrouver un objet à partir de son champ id. La fonction suivante étend cette fonctionnalité aux versions plus anciennes des navigateurs.

```
function getobj(id)
{
   if(document.layers)
       return document.id; // Netscape 4.x
   if(document.getElementById)
       return document.getElementById(id) // Netscape 6.x IE 5.x
   if(document.all)
       return id; // IE 4.x
   alert("Unable to retreive object from Id");
}
```

De même, l'accès au style d'un élément est assez variable en fonction du navigateur. La fonction suivante permet de s'affranchir de ce problème :

```
function getstyle(id)
{
    var obj ;
```

```
    if (document.layers)
        return  document.id;
    if (document.getElementById)
        return document.getElementById(id).style;
    if (document.all)
        return id.style;
}
```

4.10.2 Retrouver un élément par son nom

La méhode `document.getElementsByName();` renvoie l'élement identifié à partir de sa balise name. L'exemple suivant stocke dans la variable up_divs l'objet qui code la balise de nom up.

```
<div name="up">200</div>
<div name="up">145</div>
<div name="down">146</div>
<div name="other">178</div>

up_divs = document.getElementsByName("up");
```

4.10.3 Retrouver un élément par son type

La méthode suivante `document.getElementByTagName` renvoie un tableau contenant toutes les balises dont le type est spécifié en paramètre. L'instruction :

```
tables = document.getElementsByTagName("table");
```

renvoie par exemple une liste de toutes les tables présentent dans le document.

4.10.4 Retrouver l'ensemble des éléments d'une classe

Il n'existe actuellement pas de fonctions permettant de retrouver l'ensemble des balises d'une classe. Cette fonctionnalité peeut toutefois être définie par l'utilisateur à l'aide d'une fonction parcourant l'ensemble des balises et testant pour chacune le nom de sa classe.

```
function getElementByClass(classname)
{
  var elt=new Array();
  var alltags=document.all;
  if(! alltags)
   alltags=document.getElementsByTagName("*");
  for (i=0; i<alltags.length; i++)
    if (alltags[i].className==classname)
      elt.unshift(alltags[i]);
  return elt;
}
```

4.10.5 Père d'un élément dans la hiérarchie

Nous donnons enfin une fonction permettant de retrouver le père d'un objet dans la hiérarchie.

```
function getparent(obj)
{
    var parent ;

    if (obj.parentNode)
            return  obj.parentNode ;
    if (obj.parentElement)
              return  obj.parentElement ;
}
```

Chapitre 5

Les CGI

Le terme CGI signifie *Common Gateway Interface* (interface commune d'accès). Le terme CGI ne recouvre donc pas un type d'application particulier (et encore moins le langage dans lequel est écrite cette application) mais un protocole de communication entre le serveur et l'application. En effet, lors d'une requête du client, le serveur va appeler l'application en y transmettant les paramètres suivant un protocole précis. L'application doit retourner un résultat dont le formatage est spécifié par le protocole.

5.1 Récupération des paramètres

L'application CGI peut récupérer les informations transmises par le serveur de trois façons différentes :

1. Lecture dans le tableau d'arguments.

   Dans ce cas, les paramètres sont passés en argument du programme CGI et sont donc récupérés dans (argv,argc) en C ou C++ et dans $* en shell. Ce type de passage d'arguments est assez rare. Il se produit par exemple sur le lien suivant :

   ```
   <a href="toto.cgi?Il etait une fois"> cliquez</a>
   ```

   Dans ce cas, nous aurons par exemple en shell $1=il, $2=etait, $3=une et $4=fois.
2. Lecture sur l'entrée standard (mode POST).

   Dans ce cas, le serveur envoie sur l'entrée standard de l'application une chaîne de caractères de la forme :

   ```
   var1=val2&var2=val2..
   ```

   où var_i et val_i représentent respectivement les noms et les valeurs des variables fournies par l'utilisateur, par exemple à travers un formulaire (section 2.9).

La récupération de cette chaîne s'effectue donc avec un scanf (en C), un cin>> (en C++) et un read (en shell). La longueur de la chaîne (nécessaire pour allouer la place en C et C++) est contenue dans la variable d'environnement `CONTENT_LENGTH`.

Il reste ensuite à décomposer la chaîne pour accéder aux différentes variables.

3. Lecture dans la variable d'environnement (mode GET)

   Dans ce cas, la chaîne précédemment décrite est contenue dans la variable d'environnement `QUERY_STRING`. On y accède donc par `$QUERY_STRING` en shell et la fonction `getenv` en C, C++.

Généralement, une application CGI sait par avance si elle sera appelée en mode POST ou GET. Toutefois elle peut lire le mode dans lequel lui seront passées les informations dans la variable `REQUEST_METHOD`.

5.2 Les variables d'environnement

En fait, de nombreuses informations sont positionnées par le serveur dans l'environnement de votre programme. A titre d'exemple, voici le produit de la commande shell `env` dans un script CGI combiné à un serveur apache.

```
UNIQUE_ID=PhIFz38AAAEAADquCWgAAAAH
HTTP_KEEP_ALIVE=300
HTTP_USER_AGENT=Mozilla/5.0 Galeon/1.2.7 (X11; Linux i686; U;) Gecko/20030131
SERVER_PORT=80
HTTP_HOST=lois
DOCUMENT_ROOT=/var/www/html
HTTP_ACCEPT_CHARSET=ISO-8859-1, utf-8;q=0.66, *;q=0.66
SCRIPT_FILENAME=/var/www/cgi-bin/values.cgi
REQUEST_URI=/cgi-bin/values.cgi?toto=titi
SCRIPT_NAME=/cgi-bin/values.cgi
REMOTE_HOST=localhost.localdomain
HTTP_CONNECTION=keep-alive
REMOTE_PORT=38425
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin
PWD=/var/www/cgi-bin
SERVER_ADMIN=root@localhost
HTTP_ACCEPT_LANGUAGE=fr, en;q=0.50
HTTP_ACCEPT=text/xml,application/xml,..
REMOTE_ADDR=127.0.0.1
SHLVL=1
SERVER_NAME=lois
SERVER_SOFTWARE=Apache/2.0.40 (Red Hat Linux)
QUERY_STRING=toto=titi
SERVER_ADDR=127.0.0.1
GATEWAY_INTERFACE=CGI/1.1
```

```
SERVER_PROTOCOL=HTTP/1.1
HTTP_ACCEPT_ENCODING=gzip, deflate, compress;q=0.9
REQUEST_METHOD=GET
_=/bin/env
```

L'url demandé est :`http ://lois/cgi-bin/values.cgi?toto=titi` cette information est répartie sur plusieurs variables :

- `SERVER_NAME` : lois
- `SCRIPT_NAME` : /cgi-bin/values.cgi. Notez que l'emplacement physique du script est stocké dans `SCRIPT_FILENAME`
- `QUERY_STRING` : toto=titi la valeur des paramètres.

La liste des variables prédéfinies est décrite dans la Table 5.1. Bien que ce tableau paraisse important, il ne faut pas perdre de vue que l'on accède généralement qu'à 3 ou 4 variables dans un même script CGI (dont `QUERY_STRING`,`CONTENT_LENGTH` et `REQUEST_METHOD`).

5.2.1 Envoi de données au Serveur

La première chose que doit faire une application CGI lorsqu'elle désire envoyer des données au serveur est de préciser la nature de ces données. La première ligne affichée par un CGI est donc :

```
Content-type: type mime
```

où `type mime` correspond au type mime (Section 1.1.4) des données envoyées. Un CGI peut donc a priori envoyer du son, des images de la vidéo…Toutefois les types mime les plus utilisés sont :

text/html : Page HTML

text/plain : Page texte

Si l'on ne désire pas positionner de Cookies, la ligne qui suit le Content-type doit être laissée vide.

A titre d'exemple, le code shell décrit dans la Figure 5.1 s'attend à recevoir une chaîne de caractères de la forme `nom=le nom&prenom=le prenom` et renvoie un message du type

```
bonjour prenom nom
```

5.3 Utilisation des Cookies

Un Cookie est une variable que l'on va demander au client de :

1. stocker sur son disque dur (ou en mémoire cache du navigateur),
2. nous retransmettre chaque fois qu'il se reconnectera sur notre site.

Les Cookies sous leurs différentes formes sont fondamentaux dans les technologies Web pour préserver des informations entre deux connexions. Ils servent notamment :

AUTH_TYPE	:	méthode d'authentification utilisée par le client pour accéder au programme CGI
CONTENT_LENGTH	:	Longueur du corps de la requête. Il s'agit de la taille des données envoyées au CGI par l'intermédiaire de la méthode POST d'un formulaire
CONTENT_TYPE	:	Type de données contenu présent dans le corps de la requête. Il s'agit du type MIME des données
HTTP_COOKIE	:	Cookie du client si jamais un cookie est effectivement présent sur le disque du client
HTTP_REFERER	:	URL de la page qui a appelé le script CGI
REQUEST_METHOD	:	type de méthode utilisée pour envoyer les données au programme CGI (GET, HEAD,POST, PUT,DELETE).
HTTP_USER_AGENT	:	type de navigateur et système d'exploitation utilisés par le client
PATH	:	chemin d'accès au script CGI
PATH_INFO	:	partie de l'URL située avant le point d'interrogation
PATH_TRANSLATED	:	chemin d'accès absolu au script CGI
QUERY_STRING	:	partie de l'URL située après le point d'interrogation (méthode GET).
REMOTE_ADDR	:	adresse IP du client appelant le script CGI
REMOTE_USER	:	Cette variable n'existe que si le client a été soumis à une authentification, et contient alors le couple nom_d_utilisateur/mot_de_passe
SCRIPT_FILENAME	:	Chemin d'accès complet au script CGI
SCRIPT_NAME	:	Chemin d'accès relatif (par rapport au chemin d'accès à la racine web) au script CGI
SERVER_PORT	:	port TCP auquel les données ont été envoyées (généralement le port 80)
SERVER_PROTOCOL	:	Nom et version du protocole utilisé pour envoyer la requête au script CGI

Tab. 5.1 – Principales variables utilisées par les applications CGI

```
#!/bin/sh

#On envoi le type mime et la ligne vide
echo "Content-type: text/html"
echo ""

#si on est en mode POST on lit l’entree standart dans QUERY_STRING
if [ "$HTTP_REQUEST_METHOD" = "POST" ]
then
        read QUERY_STRING
fi
#quelque soit le mode l’info est dans QUERY_STRING

#premiere affectation puis partie droite  du =
nom=‘echo $QUERY_STRING|cut -d"&" -f1|cut -d"=" -f2‘
#deuxieme affectation puis partie droite  du =
prenom=‘echo $QUERY_STRING|cut -d"&" -f2|cut -d"=" -f2‘

echo "<h1> bonjour $prenom $nom </h1>"
```

Fig. 5.1 – Un premier script CGI

- à identifier un utilisateur en y affectant un numéro unique,
- à conserver des informations lorsqu’un processus requiert plusieurs étapes nécessitant plusieurs connexions.
- à conserver des statistiques sur les utilisateurs (nombre de commandes, nombre de visites...)

Dans sa forme simple le positionnement d’un Cookie dans un script CGI consiste à afficher les ligne suivante :

```
Content-type: text/html
Set-Cookie: var=val;
```

On remarque que la ligne qui suit `Content-type :` n’est plus vide mais contient le `Set-Cookie`. En fait cette ligne vide est décalée jusqu’au dernier positionnement de Cookie et marque le début du contenu du document.

La récupération de la valeur du ou des Cookies se fait grâce à la variable `HTTP_COOKIE` qui contient une liste d’affectation séparées par des « ; » :

```
var1=val1;var2=val2. . .
```

La syntaxe complète du positionnement d’un cookie est la suivante :

```
Set-Cookie: var=val; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure
```

expires : Cette variable correspond à la date d'expiration du Cookie (date à partir de laquelle le Cookie sera effacé du disque de l'utilisateur). Le format de la date est le suivant :

```
Wdy, DD-Mon-YYYY HH:MM:SS GMT
ce qui nous donne par exemple :
Wednesday, 09-Nov-99 23:12:40 GMT
```

domain la liste des noms de domaines de serveurs auquel le Cookie doit être passé (par exemple `ismra.fr`). Notez que spécifier `ismra.fr` permet à n'importe qu'elle machine de nom `toto.ismra.fr` à recevoir le Cookie. Il est donc préférable de mettre le nom complet du serveur.

path correspond au chemin après le nom de machine. Toutes les applications correspondant au bon domaine et étant des sous chemins du chemin path recevront le Cookie.

secure ne transmettre le Cookie que si le canal est sécurisé (donc crypté).

L'exemple suivant illustre l'envoi du Cookie CUSTOMER avec comme valeur WILE_E_COYOTE, comme nom de domaine ismra.fr, un path qui ne restreint pas l'accès et une date d'expiration le 9 novembre 2005 à 23 heures (la ligne a été coupée en deux uniquement pour rentrer dans la page).

```
Set-Cookie: CUSTOMER=WILE_E_COYOTE; path=/; domain=ismra.fr;
expires=Wednesday, 09-Nov-05 23:12:40 GMT
```

Liste des tableaux

Table des figures

Index

Chapitre 6

Annexes

6.1 Liens utiles

- Version en ligne de ce cours :
  `http://www.greyc.ensicaen.fr/~luc/ENSEIGNEMENT/COURS/COURS_HTML/title.html`
- Site du W3C
  `http://www.w3.org`
- Site du WDG
  `http://www.htmlhelp.com/`

6.2 Différents sources de formulaires

6.2.1 Balise input text

```
    <form method="POST" name="inputtext" action="http://lois/cgi-bin/values.cgi">
      <p>
      Nom :{\nobreakspace}{\nobreakspace}{\nobreakspace}{\nobreakspace}{\nobreakspace}{\nobr
Pr{\'e}nom :{\nobreakspace}{\nobreakspace}{\nobreakspace}{\nobreakspace}<input name="prenom'
      <br/> t{\'e}l{\'e}phone :{\nobreakspace}<input  name="tel" size="14" maxlength="14"> <
      </p>
      Envoyer <input type="submit" value="submit">
      <br/>
    </form>
```

Remarquez qu'il n'y a pas de valeur par défaut. On a également précisé le type d'une des balises. Cela n'a aucun effet dans la mesure où text est la valeur par défaut.

6.2.2 Balise input image

```
    <form method="POST" name="inputtext" action="http://lois/cgi-bin/values.cgi">
      <p>
```

```
    <input NAME="img" type="image"  src="../IMAGES/green-bullet-on-white.gif"> <br/>
  </p>
</form>
```

Un clic sur l'image déclenche le formulaire. Les paramètres passés sont les coordonnées (x,y) du clic dans l'image.

6.2.3 Balise input password

```
<form method="POST" name="inputpasswd" action="http://lois/cgi-bin/values.cgi">
  password: <input type="password" name="pass" size="10"><br/>
  Envoyer <input type="submit" value="submit">
</form>
```

6.2.4 Boutons checkbox

```
<form method="POST" name="inputcheck" action="http://lois/cgi-bin/values.cgi">
  S{\'e}lectionnez vos dipl{\^o}mes :
<ul>
  <li> <input TYPE="checkbox" NAME="bepc" VALUE="oui" checked > BEPC</li>
  <li> <input TYPE="checkbox" NAME="bac" VALUE="science">Bac scientifique</li>
  <li> <input TYPE="checkbox" NAME="bac" VALUE="litteraire">Bac litt{\'e}raire</li>

  <li> <input TYPE="checkbox" NAME="ing" VALUE="oui"> Dipl{\^o}me d'ing{\'e}nieur</li>
</ul>
  Envoyer <input type="submit" value="submit">
</form>
```

6.2.5 Les boutons radio

```
<form method="POST" name="inputradio" action="http://lois/cgi-bin/values.cgi">

  <ul id="sexe">
    Indiquez votre sexe:
    <li><input type="radio" name="sexe" value="F" checked> femme</li>
    <li><input type="radio" name="sexe" value="H" > homme</li>
  </ul>

  <ul>
    Votre age:
    <li><input type="radio" name="age" value="0-20" checked> entre 0 et 20</li>
    <li><input type="radio" name="age" value="20-60" > entre 20 et 60</li>
    <li><input type="radio" name="age" value="60-" > au del{\`a} de  60</li>
  </ul>

  Envoyer <input type="submit" value="submit">
</form>
```

Les boutons radio permettent un choix 1 parmis n. Tous les boutons se rapportant à un même choix doivent avoir le même nom.

6.2.6 La balise select

```
    <form method="POST" name="inputselect" action="http://lois/cgi-bin/values.cgi">
      <div style="float:right;">
      Votre ville d'origine :
      <select name="ville">
        <option> Bordeaux</option>
        <option selected>         Caen</option>
        <option> Grenoble</option>
        <option >Paris</option>
        <option> Toulouse</option>
      </select> <br/>
      </div>
      <div>
      Vos sports favoris
      <select name="sport" size="4" multiple>
        <option>Voile</option>
        <option>natation</option>
        <option value="course">Course {\`a} pied</option>
        <option value="velo">V{\'e}lo</option>
        <option>Rame</option>
        <option>Marathon</option>
      </select>
      </div>
<center>    Envoyer <input type="submit" value="submit"></center>
    </form>
```

Cette balise permet de choisir une valeur parmis une liste déroulante de choix. L'attribut selected permet de présélectionner un choix. Notez également l'utilisation de l'attribut value pour la balise option qui permet de changer la valeur effectivement envoyée au programme spécifié dans l'attribut action.

L'attribut size de la balise select (utilisé pour le sport) permet d'afficher plusieurs éléments du menu. De plus, l'attribut multiple permet de choisir plusieurs élements du menu.

6.2.7 Balise textarea

```
    <form method="POST" name="inputradio" action="http://lois/cgi-bin/values.cgi">
      <div style="margin-left: 5mm;">
        Racontez vos vacances :
      <textarea name="vacances" rows="10" cols="40">
        L'hiver :
```

```
      l'{\'e}t{\'e} :
    </textarea></div>
    <center>Envoyer <input type="submit" value="submit"></center>
  </form>
```

Cette balise permet de rentrer du texte. Les attributs rows et cols spécifient le nombre de lignes et de colonnes de la fenêtre (en caractères).

6.3 Un programme C récupérant les valeurs d'un formulaire

6.3.1 Liste des fonctions

Requete_cgi FAIRE_Requete_cgi() :
Création de l'objet permettant de récupérer les valeurs.

void DEFAIRE_Requete_cgi() :
Destruction de l'objet.

int Requete_cgi_existe(Requete_cgi,char*) : Indique si une variable est positionnée dans les arguments. Renvoie 0 si elle n'existe pas est le nombre de fois où elle est présente dans le cas contraire.

int Requete_cgi_nb_variable(Requete_cgi) :
Indique le nombre de variables positionéées.

void Requete_cgi_display(Requete_cgi) :
Affiche sur la sortie standart la liste des couples (variable,valeur)

char* Requete_cgi_get_value_by_name(Requete_cgi,char*,int) :
Permet de récupérer la valeur de la variable dont le nom est passé en second argument. Si la variable est positionnée plusieurs fois l'on peut récupérer la valeur de sa i^{e}occurence en passant *i* en troisième argument (sinon le troisième argument doit être laissé à 1).

char* Requete_cgi_get_value_by_number(Requete_cgi,int) :
Renvoie la valeur de le i^{e}affectation.

char* Requete_cgi_get_name(Requete_cgi,int) :
renvoie le nom de la variable de la i^{e}affectation.

6.3.2 Code des fonctions

Structures de données

Structure affectation

```
struct affectation
{
        char* name;
        char* value;
};
typedef struct
affectation Affectation;
```

Structure globale

```
struct Requete_cgi
{
   int nb_var;
   Affectation* tab;
   type_methode methode;
};
```

Récupération de la méthode

- Par défaut : méthode get
- Si REQUEST_METHOD=GET : Méthode GET, sinon
- Si REQUEST_METHOD=POST : Méthode POST, sinon
- METHODE_INCONNUE

```
static type_methode get_methode_priv()
{
  char *methode = getenv("REQUEST_METHOD");

  if ( ! methode )
    return METHODE_GET;

  if ( ! strcmp( methode ,"GET" ) )
    return METHODE_GET;

  if ( ! strcmp( methode , "POST" ) )
    return METHODE_POST;

  return METHODE_INCONNUE;
}
```

Récupération du contenu de la requête

- Si la requête est en mode POST on copie QUERY_STRING dans une chaîne.
- Si la méthode est en mode GET et CONTENT_LENGTH est positionné on alloue une chaiñe de taille CONTENT_LENGTH et on lie l'entrée standart, sinon :
- Ne connaissant pas la taille de la chaiñe, on la lie caractère par caractère en re-allouant la chaiîne par paquet de 100 octets.

```
static char* get_requete_priv(type_methode meth)
{
  int longueur;
  char* tmp,*requete;
  char c;
  static int size=100;

  if(meth == METHODE_GET)
    {
      tmp = getenv("QUERY_STRING");
      longueur = strlen(tmp)+1;
      requete = malloc(longueur*sizeof(char));
      strcpy(requete,tmp);
```

```
      return requete;
    }

  longueur = atoi(getenv("CONTENT_LENGTH"));
  if ( longueur != 0 )
    {
      requete = malloc(longueur*sizeof(char));
      scanf("%s",requete);

      return requete;
    }

  requete = malloc(size*sizeof(char));
  while( scanf("%c",&c) != EOF)
    {
      if ( c== '\n')
        break;
      requete[longueur] = c;
      longueur++;
      if(longueur%size == 0)
        requete = realloc(requete,longueur+size);
    }

  return requete;
}
```

Intitialisation du tableau d'affectations

On parcours la chaîne d'entré avec comme séparateurs le & et le = de façon à s'arrêter sur chaque membre d'une affectation. Pour chaque affectation on remplie les champs name et value de l'entré courante du tableau d'affectation.

```
static void init_tab_priv(Requete_cgi self,char* requete)
{
  char* name;
  int i;

  Nb_var = nb_occurence_priv(requete,'&')+1;
  Tab = (Affectation *)malloc(Nb_var*sizeof(Affectation));

  name = strtok(requete,"&=");
  i=0;
  do
    {
      init_affectation(Tab+i,name);
```

```
      i++;
    }
  while((name = strtok(NULL,"&=")));
}
static void init_affectation(Affectation* affect,
                             char* name)
{
  char* value;
  affect->name = malloc((strlen(name)+1)*sizeof(char));
  strcpy(affect->name ,name);
  value = strtok(NULL,"&=");
  affect->value =  malloc((strlen(value)+1)*sizeof(char));
  strcpy(affect->value,value);
}
```

Construction/destruction de l'objet Requete_cgi

```
Requete_cgi FAIRE_Requete_cgi()
{
        Requete_cgi self;
        char * requete;
        type_methode methode = get_methode_priv();


         if ( methode==METHODE_INCONNUE)
                return NULL_REQUETE_CGI;


        requete = get_requete_priv(methode);
        self =(Requete_cgi) malloc(sizeof(struct Requete_cgi));
        assert(self);
        Methode = methode;
        init_tab_priv(self,requete);

        free(requete);

        return self;
}
void DEFAIRE_Requete_cgi(Requete_cgi self)
{
  int i;
  Affectation* ptr;

  assert(self);
  ptr=Tab;
  for(i=0;i<Nb_var;i++)
```

```
    {
      free(ptr->name);
      free(ptr->value);
    }
  free(Tab);
  free(self);
}
```

Liste des méthodes

```
int Requete_cgi_existe(Requete_cgi self,char* quoi)
{
  int i;
  int nb_exist=0;

  assert(self);
  for(i=0;i<Nb_var;i++)
    nb_exist += !strcmp(quoi,Tab[i].name);

  return nb_exist;
}

int Requete_cgi_nb_variable(Requete_cgi self)
{
  assert(self);
  return Nb_var;
}
char* Requete_cgi_get_value_by_name(Requete_cgi self,
                                 char* name,
                                 int number)
{
  int i;
  int nb_meet=0;
  int find;

  assert(self);
  assert(name);
  i=0;
  while((find = !strcmp(Tab[i].name,name)) &&
        nb_meet < number &&
        i < Nb_var)
    if ( find )
      nb_meet++;

  if (find)
    return Tab[i].value;
```

```
  return NULL;
}
char* Requete_cgi_get_value_by_number(Requete_cgi self,int number)
{
  assert(self);
  assert(number >= 0 && number < Nb_var);

  return Tab[number].value;
}

char* Requete_cgi_get_name(Requete_cgi self,int number)
{
  assert(self);
  assert(number >= 0 && number < Nb_var);

  return Tab[number].name;
}
```