Technologies de l' Internet

Claude Guéganno

15 novembre 2004

Table des matières

Première partie

HTML

Chapitre 1

Vue d'ensemble de HTML

1.1 Généralités

1.1.1 Historique

La connexion de plusieurs réseaux d'ordinateurs apparaît en 1961. En 1969, le ministère de la défence des U.S.A. crée le réseau ARPANET. Un peu plus tard, les universités sont elles aussi connectées à ARPANET, mais les différences notables entre les différent réseaux locaux posent problèmes.

En 1982, création de TCP/IP (Transmission Control Protocol, Internet Protocol) : IP et TCP proviennent respectivement de la couche réseau et de la couche transport de ARPANET. On commence à parler d'internet, c'est le début du World Wide Web.

À partir de 1989, le langage HTML[1] associé au protocole http[2] normalise les échanches sur internet. Mais il faut attendre 1993 pour voir le premier navigateur graphique (Mosaic qui deviendra plus tard Netscape). l' Internet Explorer de microsoft est présenté en 1995.

La dernière norme XHTML 1.0 [2000] rend le langage HTML conforme à XML[3].

Lien hypertexte : une page web peut contenir des textes, des tableaux, des images, des vidéos, des sons, des formulaires. Chacun de ces éléments peut être lié à un autre document pouvant appartenir à un site différent.

1.1.2 Sites internet

Pour créer un site, il suffit d'un ordinateur, même non connecté à internet, et disposant de quelques logiciels de mise au point.

[1] *HyperText Markup Language*

[2] *HyperText Transfer Protocol*

[3] *eXtensible Markup Language* : langage extensible de balisage, *XML* est un langage pour définir d'autres langages

Exemple de configuration

- Ordinateur sous `Linux` ou `Windows XP`
- Serveur `http Apache` avec module `PHP`
- Éditeur de texte (`emacs` avec modules `HTML` et `PHP`)
- Quelques navigateurs (`netscape`, `mozilla`, ...)
- Serveur de base de données `mysql`
- Gestionnaire de base de données `phpMyAdmin`
- Logiciel de dessin (`gimp`, ...)

Cette configuration permet de réaliser des sites complexes mettant en œuvre des scripts côté serveur et de créer des pages dynamiquement à partir des données d'une base. Notez que la configuration sous `Linux` donne en final une configuration entièrement gratuite, et que les logiciels cités existent et fonctionnent correctement sous `Windows XP`.

Outils intégrés : `Frontage`, `Golive`, `Websphere`, `DreamWeaver` ...

Pour mettre un site en ligne, il faut disposer
- d'un accès à `internet`,
- d'un hébergeur pour le site. Souvent, le fournisseur d'accès (F.A.I.) propose d'héberger les pages de ses abonnés. Il existe également des hébergeurs gratuits (`freesurf`, `free`, ...)

Mettre le site en ligne revient simplement à transférer chez l'hébergeur le système de fichiers mis au point sur l'ordinateur de développement. Ce transfert se fait avec `ftp`[4].

Attention de ne pas mettre en ligne des informations ou des documents portant atteint aux droits d'auteurs, droits d'éditeurs ou décision de justice : *scans* de livres, fichiers `MP3`, documents interdits ...

Attention également aux photos de personnes : *«toute personne a sur son image et sur l'utilisation qui en est faite un droit absolu qui lui permet de s'opposer à sa reproduction et à sa diffusion sans son autorisation expresse et par écrit quel que soit le support utilisé»*.

`HTML` est l'acronyme de *Hypertext Markup Language*[5]. Il s'agit d'un ensemble de balises qui définissent la façon dont le navigateur affiche les pages.

Il existe de nombreux logiciels qui permettent de générer du code `HTML`. La plupart des traitements de textes le font. Cependant, une programmation efficace en `PHP` ou en `JavaScript` passe par une bonne connaissance des bases de `HTML`.

Dans ce chapitre nous verrons le `HTML` «de base» c'est à dire les balises qui influent sur la structure et l'aspect des pages.

On appelle *balise* un mot clé du langage enfermé entre les signes `<` et `>`. Ainsi, `<B>` est une balise qui demande au navigateur d'afficher le texte qui

[4] *File Transfer Protocol*

[5] Langage de marquage (ou balisage) hypertexte

va suivre en caractères gras (*bold*). La balise `</B>` met fin à cette consigne. Ainsi, pour afficher le message **Hello World !** en caractères gras, on écrira en `HTML` :

```
<B>Hello World !<B>
```

Remarque : l'utilisation de majuscules ou de minuscules est indifférent : `<B>` ⇔ `<b>`.

1.2 Notions de base du codage `HTML`

`HTML` est interprété par un navigateur ou *browser* (côté client). Le rendu peut varier d'un navigateur à l'autre. Il est souhaitable de tester les pages sur les navigateurs les plus répandus.

1.2.1 Structure d'une page

Une page `HTML` est composée d'un en–tête et d'un corps (figure 1.1).

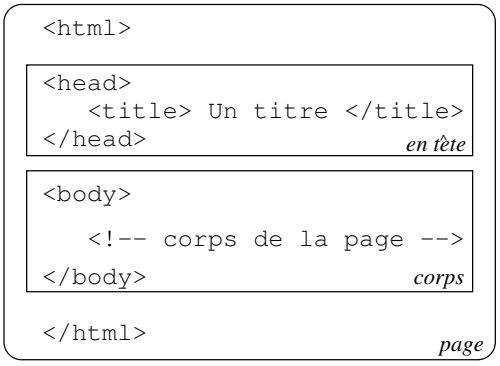


FIG. 1.1 – *Vue macroscopique d'une page* ***HTML***

On retiendra déjà les balises suivantes :

`<html>` ...`</html>`	Délimitent la page `HTML`
`<head>` ...`</head>`	Délimitent l'en–tête. Un en–tête est unique dans le document. Il contient des informations utiles pour le navigateur
`<!--` ... `-->`	C'est un commentaire : il sera ignoré par le navigateur.
`<title>` ...`</title>`	Donne le titre de la page. Il sera affiché dans le bandeau du navigateur (et non pas dans la page elle même).
`<body>` ...`</body>`	Délimitent le véritable contenu de la page.

Exemple :

```
<html>
<head>
<title>Le titre</title>
</head>
<body>
<h1>Structure d'une page</h1>
<h2> Deuxième titre </h2>
<h3> Troisième titre </h3>
<h4> Quatrième titre </h4>
<h5> Cinquième titre </h5>
<h6> Sixième ...</h6>
</body>
</html>
```

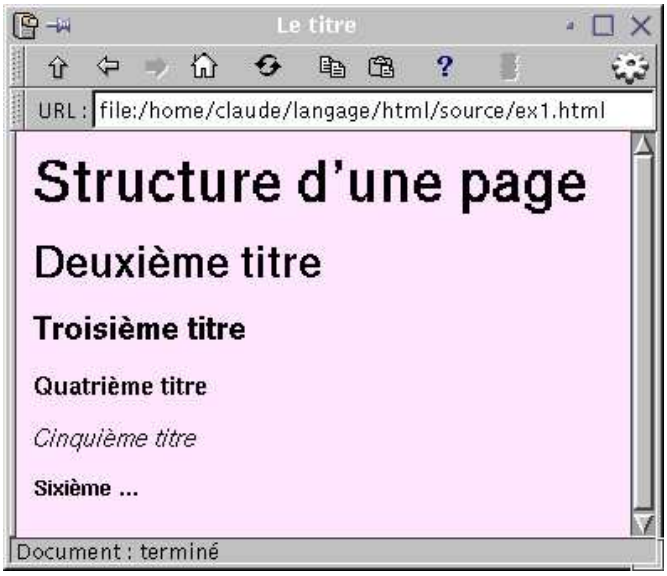


FIG. 1.2 – *Rendu du premier exemple avec* `kfm`

1.3 Caractères

Les effets de bases sont donnés par les balises de la table 1.1.

Les caractères spéciaux utilisés par le langage `HTML` sont affichables dans une page `HTML` grâce aux séquences de caractères de la table 1.2.

Pour garantir leur portabilité, les caractères accentués peuvent être représentés par des séquences de caractères du code `ASCII` de 7 *bits* (table 1.3).

`<b>...</b>`	Affichage de caractères «gras»
`<big>...</big>`	Affichage de gros caractères (le comportement peut beaucoup varier d'un navigateur à l'autre.)
`<blockquote>...</blockquote>`	Affichage d'une très longue citation
`<code>...</code>`	Les caractères sont affichés dans une police `monospace`, c'est la norme pour afficher du code.
`<i>...</i>`	Affichage de caractères en italique
`<s>...</s>`	Affichage de caractères barrés
`<strike>...</strike>`	Affichage de caractères barrés
`<strong>...</strong>`	Le texte est mis en valeur ($\Leftrightarrow$ `<b>...</b>`)
`<sup>...</sup>`	Texte mis en exposant
`<sub>...</sub>`	Texte mis en indice
`<small>...</small>`	Petite police.
`<tt>...</tt>`	texte télétype (même largeur pour tous les caractères) $\Leftrightarrow$ `<code>...</code>`
`<u>...</u>`	Texte souligné
`<var>...</var>`	Police adaptée pour l'affichage de variables. Les formules peuvent être affichées avec cet environnement

TAB. 1.1 – Formats de caractères

`&gt ;`	caractère >
`&lt ;`	caractère <
`&amp ;`	caractère &
`&quot ;`	caractère "
`  ;`	espace

TAB. 1.2 – Caractères spéciaux

signe	**mot-clé**	**caractères**	**exemple**	**HTML**
’	`acute`	A, E, I, O, U, Y, a, e, i, o, u, y	é *et* É	`&eacute ;` *et* `&Eacute ;`
‘	`grave`	A, E, I, O, U, a, e, i, o, u	è *et* È	`&egrave ;` *et* `&Egrave ;`
^	`circ`	A, E, I, O, U, a, e, i, o, u	ê *et* Ê	`&ecirc ;` *et* `&Ecirc ;`
~	`tilde`	A, N, O, a, n, o	ñ *et* Ñ	`&ntilde ;` *et* `&Ntilde ;`
"	`uml`	A, E, I, O, U, a, e, i, o, u, y	ë *et* Ë	`&euml ;` *et* `&Euml ;`
æ	`lig`	AE, ae	æ *et* Æ	`&aelig ;` *et* `&AElig ;`
ç	`cedil`	C, c	ç et Ç	`&ccedil ;` *et* `&Ccedil ;`
/	`slash`	O, o	Ø	`&Oslash ;`

TAB. 1.3 – Caractères accentués

1.4 Paragraphes

L'organisation d'une page en paragraphes et la gestion des sauts de lignes utilisent les balises de la table 1.4

` `	Provoque un saut de ligne.
`<p>...</p>`	Délimite un paragraphe
`<hr>`	Trace une ligne horizontale

TAB. 1.4 – Paragraphes et mise en page

1.5 Les tableaux

Un tableau `HTML` est défini par une liste de lignes contenant chacune des cellules.

Trois balises élémentaires suffisent pour définir un tableau :

1. `<table>` ... `</table>` : donnent respectivement le début et la fin de la définition d'un tableau.
2. `<tr>` ... `</tr>` : définissent le début et la fin d'une ligne
3. `<td>` ... `</td>` : idem pour une cellule.

Remarque : les balises `</tr>` et `</td>` sont facultatives.

Exemple :

1	2	3
4	5	6
7	8	9

```
<table border="1">
 <tr> <td>1</td> <td>2</td> <td>3</td> </tr>
 <tr> <td>4</td> <td>5</td> <td>6</td> </tr>
 <tr> <td>7</td> <td>8</td> <td>9</td> </tr>
</table>
```

1.6 Les listes (ordonnées ou non)

Liste ordonnée :

1. ceci
2. est
3. une liste
4. ordonnée

Traduction HTML

```
<ol>
  <li>ceci
  <li>est
  <li>une liste
  <li>ordonnée
</ol>
```

Liste non ordonnée :

- celle
- ci
- ne l'est
- pas

Traduction HTML

```
<ul>
  <li>celle
  <li>ci
  <li>ne l'est
  <li>pas
</ul>
```

1.7 Les liens

1.7.1 Notion d'URL

URL est l'acronyme pour Uniform Resource Location. C'est l'adresse complète d'une ressource accessible sur internet. On peut y trouver :

- le protocole à utiliser,
- la machine ou se trouve la ressource,
- le port de communication,
- le nom de la ressource,

Exemple d'URL :

```
http://claude.gueganno.free.fr:80/iris/stage.html
```

1. http est le protocole à utiliser,
2. 80 est le numéro du port ou il faut établir la connexion, si on ne met rien, c'est le port 80 qui est pris pour HTTP.
3. claude.gueganno.free.fr est la machine à contacter,
4. /iris/stage.html est la ressource proprement dite.

Il s'agit exclusivement des balises <a> et (*anchor*). Elles permettent, soit de gérer les liens à l'intérieur d'une ressource, ou bien entre des ressources différentes.

Liens à l'intérieur d'une page : à chaque fois que l'on trouve en bas d'une page, quelque chose comme :

retour en haut de la page

le code HTML est :

```
<a href="#debut">retour en haut de la page</a>
```

Et en haut de la page, une étiquette a été positionnée avec la même balise, elle permet de définir une ancre.

```
<html>
<body>
<a name="debut"></a>
 ...
 ...
</body>
</html>
```

Lien vers un autre site : par exemple, si sur un site donné on souhaite faire un lien vers www.altavista.com, il suffit d'écrire dans la page l'instruction :

```
<a href="http://www.altavista.com"> Moteur de recherche </a>
```

Ce qui aura pour effet d'afficher

Moteur de recherche

dans la page. En cliquant sur ce lien, le navigateur essaiera d'atteindre l'adresse précisée dans le champ `href`.

Lien vers la messagerie :

```
<a href="mailto:claude.gueganno@free.fr">Ecrivez moi!!</a>
```

En cliquant sur le lien Ecrivez moi ! !, le navigateur lance l'éditeur de courrier dont le champ «destinataire» est rempli avec l'adresse précisée.

1.8 Les cadres d'affichage

La fenêtre du navigateur peut être divisée en cadres (les $frames$), ayant chacun leur contenu HTML. Ceci est intéressant pour présenter efficacement un site qui contient, par exemple, un bandeau de titre, une colonne de menu, une fenêtre principale d'affichage des pages et un pied de page.

HTML propose deux concepts pour la présentation par cadres :

1. les $frames$, qui sont les composants de base,
2. les $framesets$, littéralemnt «ensembles de fenêtres» destinés à contenir soit des cadres, soit d'autres ensembles de fenêtres.

Une fenêtre composée de plusieurs cadres est nécessairement un $frameset$.

1.8.1 Mise en œuvre par un exemple

On désire réaliser un site dont l'aspect est illustré par la figure 1.3. À gauche apparaît un menu composé de liens dont la sélection agit sur la fenêtre de droite.

La réalisation d'une telle page conduit à concevoir au moins 5 fichiers HTML (figure 1.4) :

FIG. 1.3 – *Cadres.*

– un fichier principal qui lie tous les autres (*index.html*)
– un fichier pour le bandeau (*bandeau.html*)
– un fichier pour le menu (*menu.html*)
– un fichier pour le pied de la page (*pied.html*)
– un fichier par page d'informations (*page*1.*html*, *page*2.*html*).

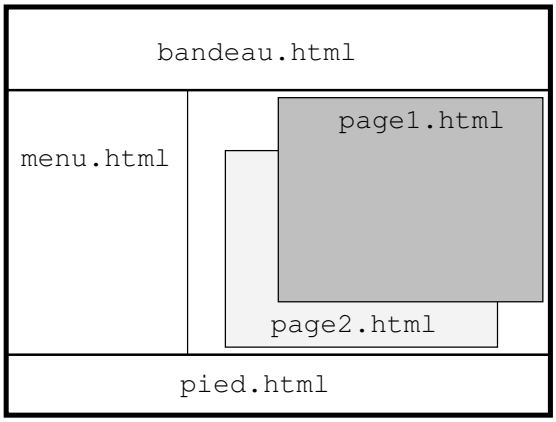


FIG. 1.4 – *Partage en fichiers*

Le fichier *index.html* qui relie l'ensemble des cadres est

```
<html>
<head>
  <title>Les cadres</title>
</head>
<frameset frameborder="no" frameborder="0" border="0" rows="50,*,40">
  <frame src="bandeau.html" scrolling="no" noresize>
  <frameset frameborder="no" frameborder="0" border="0" cols="200,*">
     <frame src="menu.html" scrolling="auto" noresize>
     <frame src="page1.html" scrolling="auto" name="page">
  </frameset>
  <frame src="pied.html" scrolling="no" noresize>
</frameset>
</html>
```

Les fichiers `HTML` définissant le contenus des cadres sont les suivants :

bandeau.html

```
<html>
<head> <title>Bandeau</title> </head>
<body bgcolor="#A0A0A0">
  <center><h6>TITRE et PUBLICIT&Eacute; </h6></center>
</body>
</html>
```

---

menu.html

```
<html>
<head>
 <title>Menu</title>
</head>
<body bgcolor="blue">
<h3>Menu</h3>
<ul>
 <li><a href="page1.html"
        target="page">page 1</a></li>
 <li><a href="page2.html"
        target="page">page 2</a></li>
</ul>
</body>
</html>
```

page1.html

```
<html>
<head>
  <title>page 1</title>
</head>
<body>
  <h1>page 1</h1>
  Contenu de la page 1
</body>
</html>
```

---

pied.html

```
<html>
<head> <title>Pied</title> </head>
<body bgcolor="black">
  <center><font color="white">...liens... </font></center>
</body>
</html>
```

1.8.2 Élément `frameset`

La balise `<frameset>` remplace la balise `<body>`. Elle doit contenir au moins un élément `frame`. Ses attributs sont :

- `frameborder` : bordures visibles ou invisibles `"1"` ou `"0"` pour *ie* et `"yes"` ou `"no"` pour *netscape*.
- `border` : précise l'épaisseur des bordures
- `bordercolor` : couleur de la bordure
- `cols` : description des colonnes. L'unité par défaut est le *pixel*. On peut les donner en `%`. Le caractère `'*'` permet d'affecter à l'une des largeurs tout ce qui reste.
  `cols="largeur_1,largeur_2,...largeur_n`
- `rows` : description des lignes. Même chose que pour les colonnes.

1.8.3 Élément `frame`

Ils doivent être contenus dans un `frameset`. Leurs attributs possibles sont :

`src="nom_fich.html"`	`URL` qui doit être affichée dans le cadre
`name="label"`	donne un nom au cadre. Ce nom est utilisé pour utiliser ce cadre comme cible en cas de lien.
`scrolling="no | yes | auto"`	interdit ou autorise les ascenseurs
`noresize`	interdit au navigateur de permettre le redimensionnement du cadre
`marginwidth="x"`	marge à gauche et à droite du document
`marginheight="y"`	marge en haut et en bas du document.

1.8.4 Élément `noframes`

Il est destiné aux navigateurs qui ne reconnaissent pas les $frames$. Placé à l'intérieur des balises `<frameset>` et `</frameset>`, sont contenu est ignoré si le navigateur reconnaît les $frames$.

```
<frameset ...>
  <!-- ici, les frames -->

  <noframes>
    <body>
       <!-- ici, une page sans frame -->
    </body>
  </noframes>
</frameset>
```

1.8.5 Élément `iframe`

`longdesc="url"`	lien vers une longue description
`name="nom"`	nom pour en faire une cible
`src="url"`	document à charger dans le cadre flottant
`frameborder="1 | 0"`	présence ou non de cadre
`marginwidth="x% | x"`	largeur de la marge dans le cadre en pourcentage ou pixels
`marginheight="y% | y"`	hauteur de la marge dans le cadre en pourcentage ou pixels
`title`	un titre pour informations sur le contenu
`scrolling="no | yes | auto"`	présence de la barre de défilement
`align="top | middle | bottom | left | right"`	alignement horizontal ou vertical
`height="y"`	hauteur du cadre dans la page *web*
`width="x"`	largeur du cadre dans la page *web*

Exemple. On intègre dans une page ordinaire un cadre intégré qui contient l'exemple précédent sur les *framesets* :

```
<html>
<head>
  <title>iframe</title>
</head>
<body>
  <center>Les frames :<br>
  <iframe src="http://192.168.1.60/html/frame/" height="280" width="410">
    Texte pour les navigateurs qui ne connaissent pas iframe
  </iframe>
  </center>
</body>
</html>
```

Le résultat avec un navigateur compatible est donné par la figure 1.5, et avec un navigateur incompatible par la figure 1.6.

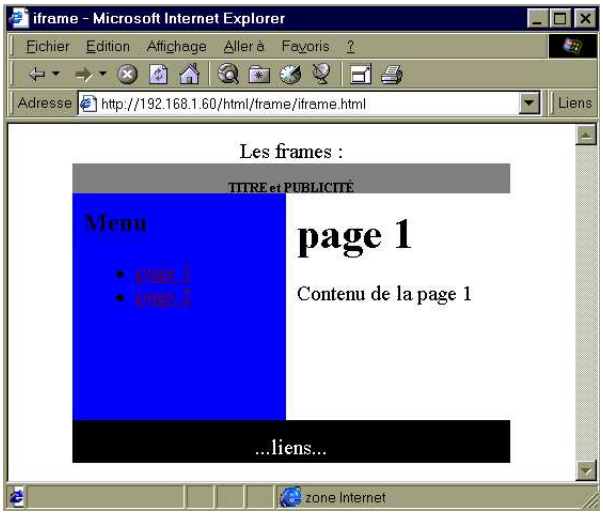


FIG. 1.5 – *Une fenêtre intérieure est intégrée à la page.*

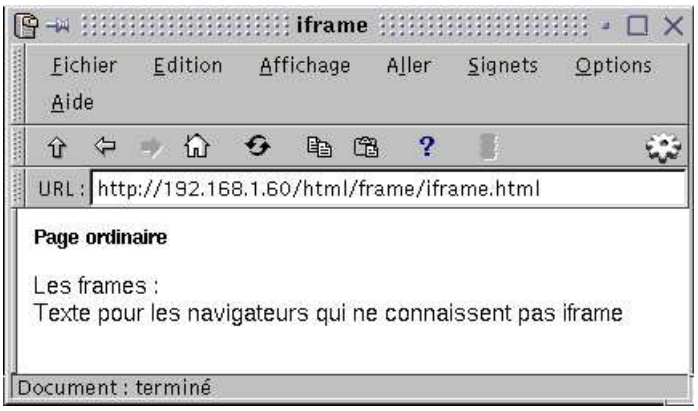


FIG. 1.6 – *Le navigateur ne reconnaît pas* `<iframe>`.

Chapitre 2

Les feuilles de styles

2.1 Généralités

Le but des feuilles de style est de contrôler la présentation du document (marges, polices, couleur). On parle de `CSS` : *Cascading Style Sheet*.

Les `CSS` sont utilisées pour séparer l'affichage de la structure du document, tous les attributs déclassés de `HTML 4.0` peuvent être définis en utilisant les propriétés de style.

Il existe trois types de méthodes :

1. **locale** (ou interne) qui affecte un style à un élément `HTML` d'une page ;
2. **incorporée** (ou intégrée) qui affecte des règles de style à toute la page ;
3. **externe** (ou attachée) qui définit des règles de style dans un fichier et peut être appelé dans toute page `HTML`.

Principes

- On peut diviser les balises en classes.
- À une balise (ou une classe de balise), on associe des paramètres d'affichage (taille, couleur, ...), ou attributs de styles.

Les attributs de styles les plus courants sont présentés dans la section 2.5.

2.2 Les feuilles de style internes

Elles sont intéressantes pour caractériser un style particulier qui apparaît une seule fois dans le document.

Exemple :

```
<p style="font-size:12pt; color:red; font-style:arial">
  le paragraphe ...
</p>
```

La feuille de style interne correspond à la valeur du paramètre `style`. D'une manière générale, un style est défini par une suite :

```
propriété1:valeur1; propriété2:valeur2 ...
```

Les propriétés sont séparées par des `;`. Le dernier `;` n'est pas obligatoire.

Dans l'exemple précédent, on choisit une police, une taille de caractères et une couleur pour tout un paragraphe. Les paragraphes suivants ne seront pas affectés par ces valeurs.

La taille des caractères peut être précisée dans l'une des unités de la table 2.1.

suffixe	**unité**
`px`	pixels
`pt`	points
`cm`	centimètres
`mm`	millimètres
`%`	pourcentage de largeur d'écran

TAB. 2.1 – *Unités pour la taille des caractères*

2.3 Les feuilles de styles incorporées

2.3.1 Définition

Les styles sont définis à l'intérieur d'une balise `<style>`, placée entre l'en-tête et le corps du document. Une feuille de style incorporée permet de fixer un style pour toute une page.

La feuille de style incorporée se place entre les balises `</head>` et `<body>`.

Exemple

```
<html>
<head> <title> Feuilles de styles</title></head>

<style>
   body  {background-color:blue}
   h1    {color:white; font-size:22pt; text-align:center; bgcolor:black}
   p     {font-style:arial; color:yellow}
</style>

<body>
 <h1>Page avec un style incorporé</h1>
 <p> Le style est valide pour cette page seulement</p>
</body>
</html>
```

2.3.2 Organisation d'une feuille de style en groupes

Le traitement par groupes permet de factoriser un certain nombre d'attributs communs à plusieurs balises. Ainsi, la feuille suivante

```
h1 {font-family:arial; background-color:#4060F0; color:blue; font-size:14pt}
h2 {font-family:arial; background-color:#4060F0; color:blue; font-size:12pt}
h3 {font-family:arial; background-color:#4060F0; color:blue; font-size:10pt}
```

peut être remplacée par :

```
h1 h2 h3 {font-family:arial; background-color:#4060F0; color:blue}
h1 {font-size:14pt}
h2 {font-size:12pt}
h3 {font-size:10pt}
```

2.3.3 Organisation d'une feuille de style en classes

Classes générales : on peut définir une classe valide pour toutes les balises pour lesquelles les attributs définis ont un sens :

```
<style>
  .centre {text-align:center}
</style>
<body>
  <h1 class="centre"> Titre centré </h1>

  <p class="centre">
   Paragraphe centré ...
  </p>
</body>
```

L'appartenance d'une balise à la classe définie se fait avec le paramètre `class="NomDeClasse"`.

Classes spécialisées : pour spécialiser une balise.

Exemple

```
<style>
 h1 {font-size:14pt}
 h1.centre {text-align:center}
 h1.droite {text-align:right}
 h1.gauche {text-align:gauche; font-size:12pt}
</style>
```

Dans cet exemple, l'attribut `font-size` est défini pour toutes les classes, sauf la classe `gauche`.

2.4 Les feuilles de styles externes

L'idée est de séparer complètement le contenu de la présentation. La présentation est décrite dans un fichier (ou feuille de style attachée) qui est incluse dans la page `html` au moment de son affichage. Ainsi, une feuille de

style peut être partagée entre plusieurs documents. Un autre avantage, est d'arriver à un code `html` de la page très simplifié.

Le fichier contenant la feuille de style externe porte l'extension *.css*.

Exemple

Dans un fichier *presentation.css*, on écrit le code suivant :

```
body  { background-color:blue}
h1 {color:white; font-size:22pt; text-align:center; bgcolor:black;}
p {font-style:arial; color:yellow}
p.rouge {color:red; text-align:right}
```

Le fichier `html` suivant utilise cette feuille de style :

```
<html>
<head>
  <title>Feuilles de style attachée</title>
  <link rel=stylesheet href="presentation.css" type="text/css">
</head>

<body>
  <h1>Feuilles de styles attachées</h1>

<p> On sépare le contenu de la présentation ... </p>
<p class="rouge">
  Il ne reste qu'à préciser  (éventuellement) la classe d'appartenance
  d'une balise !! </p>
</p>
</body>
</html>
```

2.5 Les attributs de style

2.5.1 Les attributs liés aux polices

`font-family` : suivi d'une liste de polices de caractères, dans l'ordre de préférence. Le navigateur utilise la première police présente sur l'ordinateur. Il est préférable de cloturer la liste par une famille générique :

- `serif` : polices avec empattements (`Times`...)
- `sans-serif` : police sans empattement (`Helvetica`...)
- `cursive` : polices proches de l'écriture manuscrite (`Zapf Chancery` ...)
- `fancy` : polices décoratives à usage spécial (`Comic book sans` ...
- `monospace` : polices ou chaque caractère occupe le même espace (`Courier` ...)

Exemple `p {font-family :"Zapf chancery",cursive}`

`font-style` : `normal`, `oblique` ou `italic`.

`font-variant` : `normal` ou `small-caps`. La valeur `small-caps` demande l'utilisation de petites majuscules.

`font-weight` : `normal` ou `bold` (caractères gras)

`font-size` :

- **taille absolue :** `xx-small`, `x-small`, `medium`, `large`, `x-large`, `xx-large`, ou bien directement une valeur numérique, par exemple, `11pt` (11 points).
- **Taille relative** : `smaller` ou `larger`.
- **Poucentage :** on donne la taille de la police en fonction de la police parent par un pourcentage
  Exemple : `h2 {font-size :150%}`

2.5.2 Les attributs liés à la couleur et au fond

`color` : spécifie la couleur de l'élément de texte. Quelques couleurs sont prédéfinies : `black`, `silver`, `gray`, `white`, `maroon`, `red`, `purple`, `fuchsia`, `green`, `lime`, `olive`, `yellow`, `navy`, `blue`, `teal` et `aqua`. Les autres couleurs doivent être codées avec le triplet (rouge, vert, bleu) selon la syntaxe `#RRVVBB` (`RR` est le codage en hexadécimal du niveau de rouge ...) ou bien `rgb(r, v, b)` ou les codes sont donnés en valeur décimale (entre 0 et 255). **Ex.** `p {color :#2365B8}`

`#FF0000` $\Leftrightarrow$ `red` $\Leftrightarrow$ `rgb(255,0,0)`

`background-color` : permet pour chaque style de préciser une couleur de fond.

```
h1 {background-color:black; color:white}
```

`background-image` : pour doter un style d'une image de fond. L'image est recopiée et/ou découpée pour que seul le style ait l'image pour fond.

`background-repeat` : les valeurs possibles sont `repeat`, `repeat-x` ou `repeat-y`. La valeur par défaut est `repeat`, qui indique que l'image est placée en mosaïque. `repeat-x` limite à une recopie selon l'axe horizontal et `repeat-y` à une recopie selon l'axe vertical.

`background-attachment` : défini si l'image est attachée au texte (`scroll`) ou au fond (`fixed`). La différence n'apparaît que lorsque l'utilisateur fait défiler une sélection de texte.

2.6 Les attributs liés à la présentation du texte

`word-spacing` : indique la distance supplémentaire placée entre deux mots. Elle s'ajoute à la distance prévue par le navigateur. On l'exprime avec l'unité `em` qui représente la largeur de la lettre `m` dans la police considérée.

```
body {word-spacing:1.1em}
```

`letter-spacing` : même chose pour les lettres d'un mot.

`text-decoration` : donne les enrichissements pour le texte (soulignement, barré, clignotement). Les valeurs possibles sont
- `none` : aucun
- `overline` : surligné
- `underline` : souligné
- `line-through` : barré
- `blink` : clignotant

`vertical-align` : définit la position verticale de l'élément par rapport à l'élément parent. Utile, par exemple pour positionner des petites images à l'intérieur d'un texte.
- `baseline` : la base de style est alignée sur celle de l'élément parent ;
- `sub` : en indice ;
- `super` : en exposant ;
- `text-top` : aligne le haut de l'élément avec le haut du texte parent :
- `text-bottom` : aligne le bas de l'élément avec le bas du texte parent :
- `middle` : aligne le point médian vertical du texte sur la ligne de base du texte parent augmentée de la hauteur de la lettre `x` du texte parent ;
- `top` : aligne le sommet de l'élément avec le plus grand élément de la ligne courante ;
- `bottom` : aligne le bas de l'élément avec le plus petit élément de la ligne courante ;
- `%` : valeur de pourcentage positive ou négative : hausse ou rabaisse l'élément par rapport à la ligne de base du parent.

Exemple :

```
<style>
  img {vertical-align:text-bottom}
</style>
...
<p> Texte de référence <img src="photo.gif"> </p>
```

`text-transform` : Contribue à rendre une présentation homogène en agissant sur la casse des caractères.
- `capitalize` : la première lettre en majuscule ;
- `uppercase` : toutes les lettres en majuscules ;
- `lowercase` : toutes les lettres en minuscules ;
- `none` : pas de transformation.

Exemple :

```
<style>
 h1 {text-transform:uppercase}
</style>
...
<h1>chapitre 1</h1>
```

`text-align` : justification du texte (`left`, `right`, `center` ou `justify`).

`text-indent` : permet de fixer l'indentation de la première ligne d'un paragraphe.

Exemple :

```
p {text-indent:5em}
```

`line-height` : définit la distance entre les bases de deux lignes consécutives.

Exemple :

```
p {line-height:1.5; font-size:12pt}
```

Ici, la police est en 12 points et la distance entre deux lignes est de 18 points.

2.7 Marges, remplissages et bordures

`margin` : spécifie une ou quatre valeurs de marges. Si une seule valeur est précisée, elle est appliquée au quatre marges, dans l'ordre haut,droit,bas et gauche.

`margin-top margin-bottom margin-right margin-left` : pour spécifier séparément les quatres marges.

`padding` : spécifie une ou quatre valeurs de distance entre le contenu d'un élément et son encadrement.

`padding-top padding-bottom padding-right padding-left` : pour spécifier séparément les quatres valeurs de `padding`.

`border-top border-bottom border-right border-left` : permet de matérialiser la bordure de l'encadrement (séparément pour les 4 côtés). Les valeurs possibles sont : `none`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset` et `outset`.

`border-style` : même chose, mais pour les 4 côtés.

`border-top-width border-bottom-width border-right-width border-left-width` : épaisseur de la bordure.

`border-width` : épaisseur de la bordure, commune aux 4 côtés.

`border-color` : couleur.

`border` : permet de spécifier globalement tous les paramètres de la bordure.

`height` **et** `width` : hauteur et largeur de la boîte d'encombrement (surtout utilisé pour les images).

Exemple. On définit une classe de paragraphe encadré. Ici, tous les paramètres de marges d'une part et de `padding`, d'autre part, sont mis à la même valeur. La figure 2.1 donne le rendu de cette page.

```
<html>
<head>
        <title>Feulles de styles</title>
</head>

<style type="text/css">
 P.cadre {
         color:black; font-size:10pt; text-align:justify;
         background-color:#A0A0A0; line-height:1.5;
         padding:25px;
         border:double 1.2pt;
         border-color:red;
         margin:50px;
 }
</style>

<body bgcolor="#C0C0C0">
 <p class="cadre">
  Lorsque la carte est activée par la détection du champ
  magnétique, elle s'alimente puis  reçoit les trames émises
  par le lecteur. Elle répond au lecteur en modulant le
  coefficient de qualité de son antenne suivant un taux
  qui dépend des constructeurs.
 </p>
</body>
</html>
```

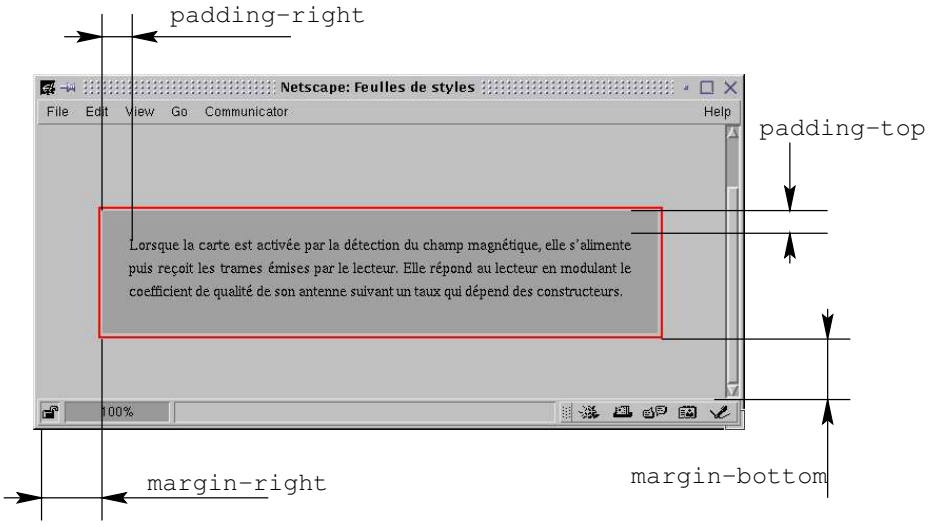


FIG. 2.1 – *margin et padding*

2.8 Les attributs liés aux listes

`list-style`	Définit le type de marqueurs pour une liste : `disc`, `square`, `decimal`, `lower-roman`, `upper-roman`, `lower-alpha`, `upper-alpha`
`list-style-image`	Au lieu d'un marqueur, on peut préciser une image `li.image {list-style-image:url(/images/im1.jpg)}`
`list-style-position`	Détermine l'alignement de la liste par rapport au titre : `inside` ou `outside`
`list-style`	Permet de spécifier globlalement tous les paramètres précédents `ul {list-style: url(/images/im1.jpg) outside}`

Exemple :

```
<html>
<head>
        <title>Attributs liés aux listes</title>
</head>
<style type="text/css">
  ol {list-style:lower-roman inside}
  ul {list-style:square outside}
</style>

<body>
 <ol>
    <lh>Liste ordonnée: </lh>
   <li> premier élément  </li>
   <li> deuxième élément </li>
   <li> troisième élément </li>
 </ol>
 <ul>
   <lh> Liste non ordonnée </lh>
   <li> premier élément  </li>
   <li> deuxième élément </li>
   <li> troisième élément </li>
 </ul>

</body>
</html>
```

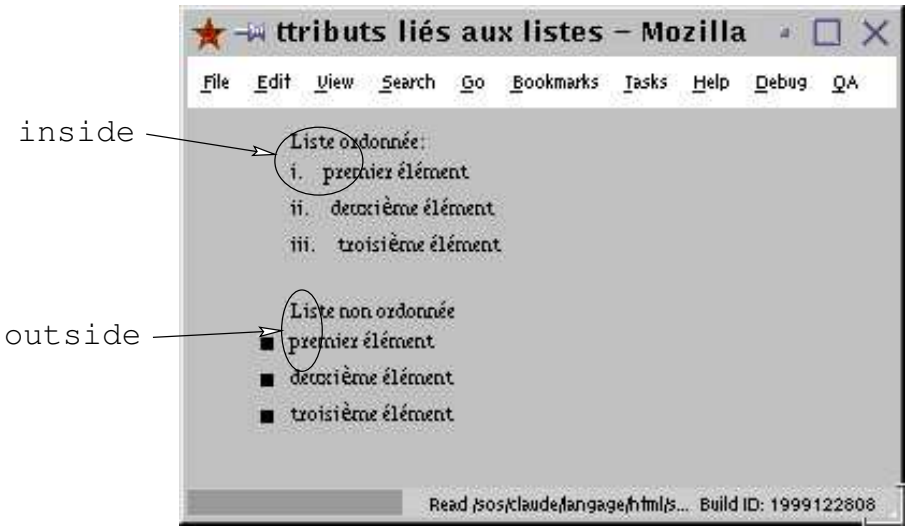


FIG. 2.2 – *Attributs liés aux listes.*

Chapitre 3

Les formulaires

3.1 Objectifs

Un formulaire permet de saisir des données sur un navigateur, et de les exporter vers le serveur. Le traitement d'un formulaire met en œuvre des éléments de la technologie `internet` à différents niveaux :

1. `HTML` pour l'affichage du formulaire,
2. `javascript` pour un premier contrôle de la saisie au niveau du navigateur
3. `PHP`, `ASP` ou un script `CGI` au niveau du serveur pour le traitement de l'information.

Un formulaire est caractérisé par un ensemble d'informations saisies par un utilisateur et un programme cible chargé d'exploiter ces informations.

3.1.1 Éléments graphiques possibles

Afin de faciliter et de standardiser la saisie, il est possible d'utiliser :
- des boîtes de saisie de texte ;
- des boîtes de saisie masquant le texte (password) ;
- des cases à cocher ;
- des boutons radio ;
- un sélecteur de fichier ;
- des listes de choix ;
- des listes menu ;
- des champs de saisie.

Exemple : la figure 3.1 est un formulaire qui met en œuvre les différents éléments graphiques possibles.

3.2 Élément `<form> </form>`

Signale le début et la fin d'un formulaire.
Les attributs autorisés sont :

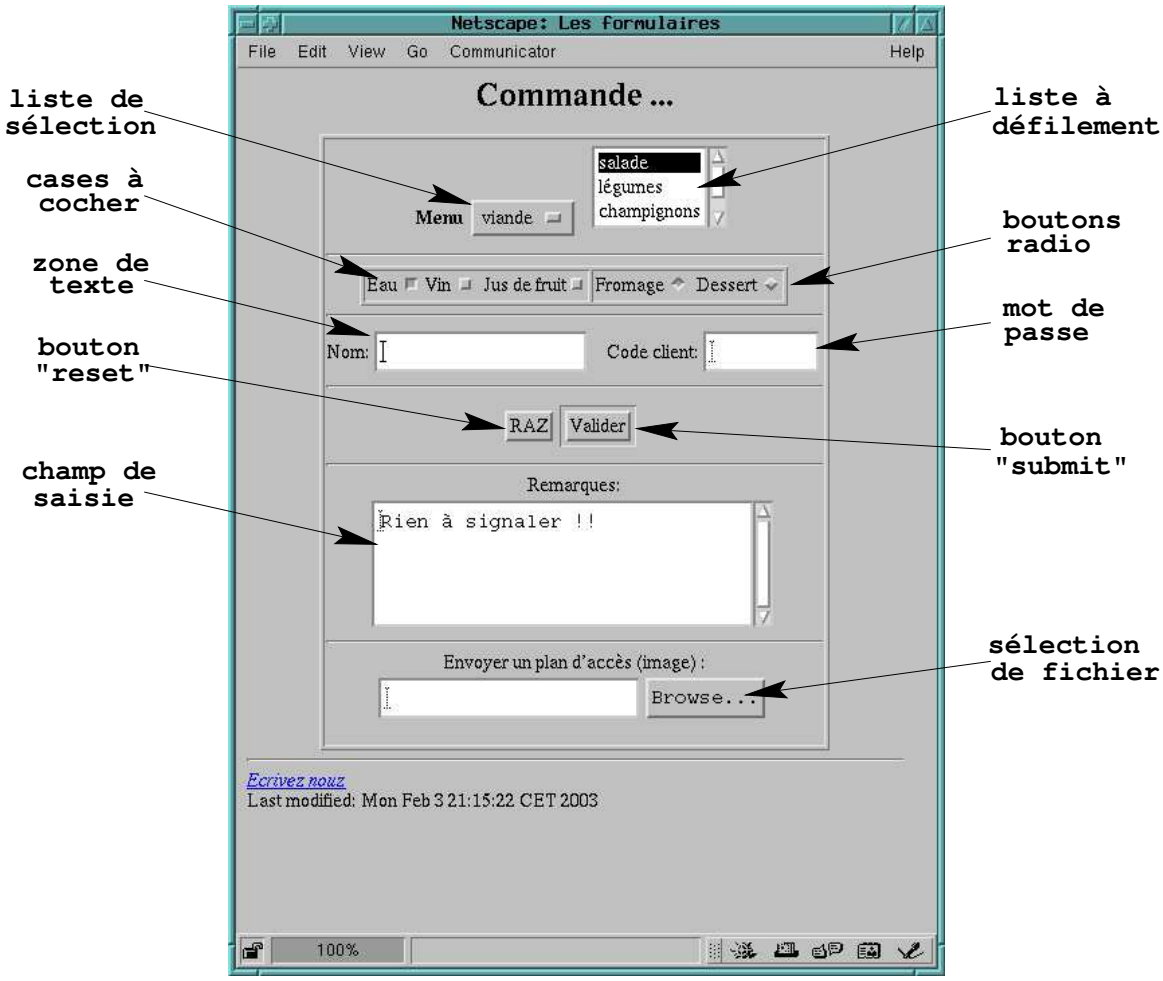


FIG. 3.1 – *Formulaire*

- `method="post | get"` méthode de requête HTTP pour soumettre les données du formulaire ;
- `action="url"` pour indiquer le script à exécuter quand on clique sur le bouton de soumission ;
- `name` nom du formulaire qui expédie les données vers le script (CGI, PHP), utilisé par le serveur et par JavaScript ;
- `target="_blank | _top | _parent | _self"` fenêtre où sera affiché le résultat du script appelé ; peut être un nom défini par l'utilisateur.
- `enctype="encodage"` encodage des valeurs envoyées au serveur, par défaut : `application/x-www-form-urlencoded`, si on souhaite envoyer un fichier (c.f. `INPUT type="file"`) il faut donner comme encodage `multipart/form-data`

Exemple

```
<form enctype="multipart/form-data" action="menu.php" method="post">
  ...
</form>
```

3.3 Les éléments `<input>`

3.3.1 Boîtes de saisie de texte : `type="text"`

Syntaxe

```
<INPUT type="text" size="x" maxlength="m" name="label"
       value="texte">
```

Attributs :

- `size` longueur de la zone du texte ;
- `maxlength` nombre de caractères maximal autorisé ;
- `name` identification ;
- `value` pour inscrire un texte dans le champ.

Exemple


```
Nom:
<input type="text" size="16" maxlength="32"
       name="nom" value="">
```

3.3.2 Boîtes de saisie masquées : `type="password"`

Syntaxe

```
<INPUT type="password" size="x" maxlength="m" name="label"
       value="texte">
```

Attributs identiques à ceux de `type="text"`

Exemple


```
Code client:
<input type="password" size="8"
       maxlength="8" name="mdp"
       value="">
```

3.3.3 Cases à cocher : `type="checkbox"`

Syntaxe

```
<INPUT type="checkbox" name="label" value="texte">
```

Attributs :

- `name` pour nommer les éléments de la case à cocher (quand le formulaire est soumis seules les cases cochées sont envoyées au serveur) ;
- `value` pour spécifier la valeur à envoyer, par défaut la valeur on est envoyée ;
- `checked` positionne par défaut le bouton en mode validé.

Exemple


```
Eau
<input type="checkbox"
       checked name="eau">
  Vin
<input type="checkbox"
       namee="vin">
  Jus de fruit
<input type="checkbox"
       name="jdf">
```

3.3.4 Boutons radio : `type="radio"`

Syntaxe

```
<INPUT type="radio" name="label_groupe" checked value="label">
```

Attributs :

- `name` pour nommer un groupe de boutons (il faut donner le même nom à tous les boutons radio du même groupe pour qu'ils soient exclusifs) ;
- `value` pour spécifier la valeur à renvoyer ;
- `checked` pour sélectionner par défaut.

Exemple


```
Fromage
<input type="radio" name="fin"
       checked value="fromage">
  Dessert
<input type="radio" name="fin"
       value="dessert">
```

3.3.5 Données cachées : `type="hidden"`

Syntaxe

```
<INPUT type="hidden" name="label" value="texte">
```

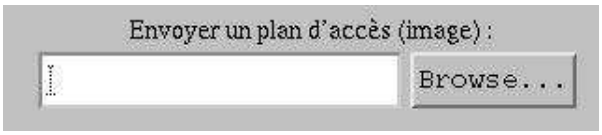
Ajouté au formulaire avant l'envoi des données.

3.3.6 Sélecteur de fichier : `type="file"`

Envoi d'un fichier du client vers le serveur.

```
<INPUT type="file" name="label" value="texte">
```

Exemple




```
Envoyer un plan d'accès (image) :<br>
<input type="file" name="plan"
       value="fichier">
```

3.3.7 Soumission du formulaire : `type="submit"`

Chaque formulaire doit contenir ce bouton qui déclenche l'envoi du formulaire.

Syntaxe

```
<INPUT type="submit" value="texte_du_bouton">
```

Exemple


```
<input type="submit" value="Valider">
```

3.3.8 Soumission du formulaire : `type="image"`

Syntaxe

```
<INPUT type="image" src="url image" alt="texte">
```

3.3.9 Effacer le formulaire : `type="reset"`

Syntaxe

```
<INPUT type="reset" value="texte_du_bouton">
```

Exemple


```
<input type="reset" value="RAZ">
```

3.3.10 Boutons : `type="button"`

Syntaxe

```
<INPUT type="button" name=label value="texte_du_bouton">
```

Pour `JavaScript` uniquement.

Attributs :

- `name` pour nommer le bouton à l'intérieur du formulaire ;
- `value` texte inscrit sur le bouton.

3.4 Élément `<button> </button>`

Syntaxe

```
<BUTTON type="submit | reset | button" name="label" value="texte">
    texte, code HTML
</BUTTON>
```

Exemple

```
Nom <INPUT type="text"><BR>
Prénom <INPUT type="text"><BR>
Mot de passe <INPUT type="password"><BR>
     <BUTTON type="submit">
        <IMG src="dessin.jpg" align="center"><br>
        Et hop !!!
     </BUTTON>
```

3.5 Élément `<select> </select>`

Listes de choix et menu déroulant :

Syntaxe

```
<SELECT name="label[]" size="n" multiple>

    <OPTION selected>choix1</OPTION>
    <OPTION>choix2</OPTION>
    ...

</SELECT>
```

Attributs de l'élément `<select>` :
- `name` nom pour la donnée, ce nom doit être suivi de `[]` lorsque le choix est multiple ;
- `size` nombre de lignes affichées pour une liste multiple ;
- `multiple` autorise la sélection simultanée de plusieurs items de la liste ;

Attributs de l'élément `<option>` :
- `value` définit la valeur passée au script ;
- `selected` pré-sélection d'un item de la liste ou valeur par défaut du menu.
- `width` largeur du menu en pixels ;

Exemple


```
<select name="plat" >
  <option selected>viande</option>
  <option>poisson</option>
</select>
```

---

la norme HTML 4.0 introduit l'élément `optgroup` qui permet de regrouper des choix logiquement ce qui peut être utile lorsque le client doit choisir dans une liste de nombreuses options (menus hiérarchiques). Les éléments `optgroup` sont compris dans l'élément `select` et entourent des éléments `option`.

3.6 Élément `<textarea> </textarea>`

Champs de saisie :

Syntaxe

```
<TEXTAREA name="label" rows="n" cols="m" wrap="virtual | off | physical">
 texte dans le champ de saisie
</TEXTAREA>
```

Attributs :
- `name` nom de la donnée ;

– `rows` nombre de lignes ;
– `cols` nombre de caractères pour une ligne ;
– `wrap` passage à la ligne automatique dans le champ de saisie :

`off` :	pas activé
`virtual` :	affiche le passage à la ligne mais n'envoie pas les retours à la ligne au serveur
`physical` :	affiche et transmet les retours à la ligne

Exemple


```
Remarques:<br>
<textarea name="rem" rows="5"
          cols="30" wrap="virtual">
 Rien à signaler !!
</textarea>
```

3.7 Attacher une information à un contrôle

Syntaxe

```
<LABEL for="id"> ... </LABEL>
```

L'attribut `for` associe un label a un contrôle du formulaire (l'identificateur doit donc correspondre à un label d'un contrôle).

Exemple :

```
<LABEL for="idnom">Nom</LABEL>
    <INPUT type="text" name="nom" id="idnom">
<LABEL for="idprenom">Prenom</LABEL>
    <INPUT type="text" name="prenom" id="idprenom">
```

3.8 Traitement d'un formulaire

avec `JavaScript` pour :

1. la vérification des données contenues dans le formulaire avant l'envoi ;
2. l'envoi de ce formulaire vers une adresse `mail` plutôt qu'au serveur.

avec `PHP` pour :

1. la récupération des données envoyées ;
2. le traitement du formulaire et la réponse au client.

Deuxième partie

JavaScript

Chapitre 4

JavaScript dans HTML

4.1 Généralités

- JavaScript a été créé par Netscape (1$^{\text{er}}$nom : *LiveScript 1.0*)
- Script = ensemble d'instructions permettant de réaliser une action.
- Un script JavaScript est embarqué dans la page HTML ⇒ lisible pour le client.
- C'est un langage de programmation dont la syntaxe est proche du C++ .
- Il permet de gérer des événements provoqués par le client (souris, modification de champ de texte ...)
- Il est exécuté par le navigateur sans recours au serveur.
- Il n'y a pas d'analyse du code ⇒ les erreurs de syntaxe ne sont pas signalées.
- L'aspect objet est succint ; il n'y a pas d'héritage entre les classes.
- Netscape a rendu public JavaScript en 1997 (ECMAScript – ECMA-262) standard de l'*European Computer Manufacturers Association*.

Nous utiliserons en général JavaScript pour rendre des pages dynamiques et vérifier en «temps-réel» les saisies dans un formulaire.

4.2 Élément script

1. Pour intégrer du JavaScript dans du code HTML il faut utiliser l'élément script :

```
<script language="nom" type="type-mime">
code JavaScript...
</script>
```

   Le code compris entre <script> et </script> n'est pas affiché, il est interprété par le navigateur.
   - attribut language définit la version du langage de script utilisé (VBScript, JavaScript)

   Exemple :

```
language="JavaScript"
language="JavaScript1.1"
language="JavaScript1.2"
```

Cet attribut est déclassé en *HTML 4*. Il est remplacé par l'attribut `type` (attention, les navigateurs antérieurs aux versions 4.0 ne reconnaissent pas l'attribut `type`)

```
type="text/javascript"
```

– attribut spécifique : `src`, pour charger du `JavaScript` dans un fichier séparé

```
src="nomfichierscript.js"
```

2. Position du code dans un fichier `HTML` :
   – dans l'en-tête : pour définir des fonctions dès le chargement ;
   – entre l'en-tête et le corps ;
   – dans le corps : pour gérer les événements.

4.3 Élément `noscript`

Il sert à masquer les scripts pour les navigateurs non compatibles : si un navigateur n'est pas capable d'exécuter les scripts, alors l'élément `script` n'aura pas de sens pour lui et il fera ce que fait tout navigateur qui ne connaît pas un élément : il ignorera l'élément. Les éléments `script` et `/script` seront donc ignorées, par contre le texte contenu entre les deux (qui est du code `JavaScript`) sera lu comme du texte `HTML` et sera affiché dans la page. Pour éviter cela, il faut placer des commentaires `HTML`

```
<!--   texte en commentaire -->
```

qui auront pour effet de masquer le code `JavaScript` aux navigateurs qui ne comprennent pas les scripts.

JavaScript connaît le `<!--` qui suit immédiatement l'élément `script`, et il ignore le texte jusqu'à la fin de la ligne (ne pas mettre de code sur cette ligne), par contre il faut lui cacher le `-->` en ajoutant un commentaire `//` `JavaScript` devant.

```
<script>
  <!--
    Code JavaScript
  //-->
</script>
```

Une fois les commentaires `HTML` placés dans l'élément `script`, on peut utiliser l'élément `noscript` pour prévenir les clients qu'il y a du `JavaScript`

```
<script>
 <!--
    Code JavaScript
 //-->
</script>
<noscript>
   texte affiché uniquement si les scripts ne sont pas supportés
</noscript>
```

4.4 Programmation événementielle

On peut insérer du code dans une `URL` d'un élément `HTML` :

```
<a href="JavaScript :code">texte</a>
```

Quand on clique sur texte le code s'exécute (le plus souvent c'est un appel de fonction).

On peut également insérer du code `JavaScript` pour indiquer les actions à effectuer en réaction à un événement provoqué par l'utilisateur. Souvent ce code inséré se contente d'appeler une fonction déclarée plus haut dans le document au sein d'un élément `script`.

Exemple : le fichier suivant contient un script qui exploite l'événement associé aux mouvements de la souris dans le document (voir aussi la figure 4.1).

```
<html>
  <head> <title>&Eacute;vénements</title> </head>

    <script language='javascript'>
    <!--
      document.captureEvents(Event.MOUSEMOVE);

      function mouvementSouris(e) {
         message = "événement: " + e.type;
         message += " [" + e.layerX + "," + e.layerY + "]";
         document.formulaire.coord.value = message;
         return true;
      }

      document.onmousemove=mouvementSouris;
    //-->
    </script>

  <body><center>
    <h3>&Eacute;vénements</h3>
    <h2> Bougez la souris !</h2>
    <form name="formulaire">
      <input name="coord" type="text" size="30">
    </form>
  </center></body>
</html>
```

Les événements pouvant être traités sont résumés dans le tableau 4.1.
La syntaxe générale pour utiliser ces événements est :

```
<ELEMENT onEvenement="code">
```

nom	événement	s'applique à
`onAbort`	interruption de chargement	`image`
`onBlur`	perte du focus	champs de saisie, `window`
`onFocus`	attribution du focus	*idem*
`onClick`	clic sur un objet	`button`, `checkbox`, `radio`, `reset`, `submit`, lien retourne `false` pour annuler l'action
`onChange`	changement de champ de saisie	`file`, `password`, `select`, `text`, `textarea`
`onDblClick`	double clic	lien, `image`, bouton
`onError`	erreur `JavaScript` ou de chargement d'une image	`image`, `window`
`onKeyDown`	touche enfoncée	`document`, `image`, lien, `text` retourne `false` pour annuler
`onKeyPress`	l'utilisateur a appuyé sur une touche	
`onKeyUp`	touche relâchée	
`onLoad`	chargement	`image`, `window`
`onMouseDown`	un bouton de la souris est enfoncé	`document`, lien, `image`, `button` retourne `false` pour annuler
`onMouseUp`	le bouton de la souris est relâché	*idem*
`onMouseMove`	déplacement de la souris	*idem*
`onMouseOut`	la souris vient de quitter une zone	lien, `image`, `layer` retourne `true` pour empêcher l'affichage de l'URL
`onMouseOver`	la souris entre sur une zone	*idem*

`onReset`	annulation des données saisies	formulaire ; retourne `false` pour annuler, exemple :
		`<form action="traiteform.php3"`
		`onSubmit="return Verif()">`

`onSubmit`	soumission du formulaire	*idem*
`onSelect`	sélection d'un champ de texte	champs de saisie
`onUnload`	déchargement d'un document	`image`, `window`

TAB. 4.1 – Événements du navigateur

FIG. 4.1 – *Capture d'événement avec* `JavaScript`

Chapitre 5

Éléments de base du langage

5.1 Types

5.1.1 Chaînes de caractères : `string`

Syntaxe :

```
' voici un texte ' ou " voici un texte "
```

Caractères spéciaux

\b	retour en arrière	backspace
\f	saut de page	form feed
\n	saut de ligne	new line
\r	retour chariot	carriage return
\t	tabulation	tab

Caractère d'échappement

caractère	codage	exemple	JavaScript
'	\'	l'histoire	ch = 'l\'histoire';
"	\"	color="red"	ch = "color=\"red\"";
\	\\		

5.1.2 Les nombres : `number`

Entiers

préfixe	base	exemple
0	octale	0377
0x	hexadécimale	0xFF
	décimale	255

Réels

Syntaxe : `partie entière . partie décimale e exposant_entier`

Exemples `3.4e12 3.14 .4`

5.1.3 Booléens : `boolean`

Valeur : `true` ou `false`

5.2 Fonctions : `function`

Pas de type pour les arguments ou pour la valeur retournée. Une fonction peut être appelée avec un nombre d'arguments différents.

Syntaxe d'une procédure :

```
function nom_procedure(arg1,...,argN){
     instruction1;
     ...
     instructionM;
}
```

Syntaxe d'une fonction :

```
function nom_fonction(arg1,...,argN){
     instruction1;
     ...
     instructionM;
     return val;
}
```

Rappel : une procédure ne retourne aucune valeur, une fonction retourne une valeur.

Appel :

```
nom_procedure(arg1, ..., argN);  /* procédure */
var = nom_fonction(arg1, ..., argN);  /* fonction */
```

5.3 Objets : `object`

5.3.1 Définitions

1. **Classe**
   famille d'objets
   Ex. :`personne`
2. **Propriété** caractéristique d'un objet (elle a un nom et une valeur)
   Ex. : `nom, âge`
3. **Constructeur**
   outil qui permet de créer un élément (instance) d'une classe. Ex.

```
function personne(nom, age){
     this.nom = nom;
     this.age = age;
}
```

`this` permet de faire référence aux instances.

4. **Méthode**
   outil s'appliquant à un objet.

5.3.2 Création d'une instance d'une classe

Syntaxe :

```
instance = new constructeur();
instance = new constructeur(arg1, ..., argN);
```

Exemple :

```
p1 = new personne('Zaza',99)
```

5.3.3 Accès aux propriétés des instances

Syntaxe :

```
nom_instance.propriete
```

Pour modifier l'âge de la variable `p1` :

```
p1.age = 100
```

5.3.4 Méthodes

Pour ajouter une méthode à la classe, il faut écrire une fonction, par exemple :

```
function affiche(){
  document.write("Nom :", this.nom, "<br>");
  document.write("Age :", this.age, "<br>");
}
```

Puis, il faut faire en sorte que cette fonction devienne une méthode de la classe en écrivant, dans le constructeur de cette classe :

```
this.affiche = affiche;
```

D'une manière générale, l'appel d'une méthode se fait selon :

```
nom_instance.methode(arg1, ..., argN)
```

Exemple :

```
p1.affiche()
```

Code complet de l'exemple : Le résultat est donné par la figure 5.1.

```
<html>
<head>
<title> Classes javascript </title>
</head>
<script language="javascript">
<!--
// ---- classe 'personne' -----------------------
function affiche(){
  document.write("Nom :", this.nom, "<br>");
  document.write("Age :", this.age, "<br>");
}

// constructeur
function personne(nom, age){
     this.nom = nom;
     this.age = age;
     this.affiche = affiche; // 'affiche' devient une méthode
}
//-->
</script>

<body>
<h2>Classes <tt>javascript</tt></h2>
<script language="javascript">
<!--
 p1 = new personne("zaza", 99)
 p1.affiche();
//-->
</script>
</body>
</html>
```

5.4 Objets prédéfinis : images, champs de formulaires ...

Classes prédéfinies : `Date()`, `Window()` ...
Pour ajouter des propriétés :

```
nom_classe.prototype.nouvelle_propriete  = valeur
```

Fonctions de conversions :

- `eval(chaine)` évalue la chaîne et retourne sa valeur.
- `parseInt(chaine, base)` traduit en entier si possible (dans la base en option), sinon retourne `NaN`.

FIG. 5.1 – *Classe avec* `JavaScript`

– `parseFloat(chaine)` traduit en réel la chaîne si possible, sinon retourne `NaN`[1].

Pour connaître le type utiliser `typeof(entite)` qui retournera le type de `entite`.

Pour savoir si la valeur est numérique utiliser `isNaN(valeur)` qui retourne `true` ou `false`.

5.5 Opérateurs

Les opérateurs peuvent être :
– **unaires** : ils n'agissent que sur 1 donnée ;
– **binaires** : ils agissent sur 2 données ;

1. **Concaténation (chaînes)**
   Syntaxe : `+`
2. **Arithmétiques**

`+`	addition	binaires
`-`	soustraction	|
`*`	multiplication	|
`/`	division	|
`%`	modulo	|
`++`	incrémentation	unaires
`--`	décrémentation	|

3. **Logiques**

[1] *Not a Number*

&&	et	binaires
||	ou	|
!	négation	unaire

a	b	a et b	a ou b	non a
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

avec 1 = vrai, 0 = false

4. **Binaires (*bit* à *bit*)**

   Entiers sur 32 bits.

&	et (1 & 1) → 1
|	ou (1 | 0) → 1
^	ou exclusif (1 ^ 1) → 0
<<	décalage à gauche
>>	décalage à droite
>>>	idem sans conservation du signe

5. **Affectation**

signe	utilisation	équivalent
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x \^= y	x = x \^ y
|=	x |= y	x = x | y
++	x++	x = x +1
--	x--	x = x - 1

6. **Relationnels** (nombres et chaînes)

   ==, !=, <=, >=, >, <

Priorité des opérateurs

`[ ] ( ) . x++ x--`
`++x --x +x -x ~ !`
`* / %`
`+ -`
`<< >> >>>`
`< > <= >=`
`== !=`
`&`
`^`
`|`
`&&`
`||`
`? :`
`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

5.6 Instructions

Toutes les instructions se terminent par `;` Cependant `;` peut être omis s'il y a un passage à la ligne

```
ch = "ceci est une chaine";
ch = "ceci est une chaine"
```

5.6.1 Commentaires

`//` ou `/* ··· */`

5.6.2 Variables

```
var nom_variable ;
var nom_variable = valeur ;
```

avec, `nom_variable` :
- caractères autorisés : lettres, chiffres (sauf le premier caractère), et souligné _
- attention à la casse : `nom` différent de `noM`

la déclaration des variables est optionnelle mais fortement recommandée !
Classes d'allocation :
- Variables locales : déclarées dans une fonction
- Variables globales : déclarées en dehors du corps des fonctions.

5.6.3 Boucles

Faire tant que condition vraie

do{
instruction1;
...
instructionN;
}
while (condition)
	var i = 1;
	var x = 2;
	do{
	x *= x;
	i++;
	} while (i<3);

Tant que condition vraie faire

while(condition){
instruction1;
...
instructionN;
}
	var i = 1;
	var x = 2;
	
	while(i<3) {
	x *= x;
	i++;
	}

Pour expression initiale jusqu'à condition finale faire

for(expression; condition; incrément){
instruction1;
...
instructionN;
}
	var i;
	var x = 2;
	for(i=0; i<3; i++)
	x *= 2;
	--- 	
	var i;
	var x = 0;
	for(i=50; i<90; i+=10)
	x += i;

Pour variable dans un ensemble faire

ces instructions n'existe pas en C++ .

`for(variable in nom_tableau){`
`  instruction1;`
`  ...`
`  instructionN;`
`}`
	`for(variable in nom_objet){`
	`  instruction1;`
	`  ...`
	`  instructionN;`
	`}`

Exemple : nous allons vérifier si nous avons le fruit demandé par un client (nommé `fruit_cherche`) dans notre stock (dans le tableau `fruits`) et dans le cas ou nous trouvons ce fruit nous allons afficher un message dans une fenêtre de dialogue :

1. Solution avec `for`

```
var fruits = new Array('pomme', 'poire', 'fraise', 'abricot');

for(i = 0 ; i < 4 ; i++)
      if(fruits[i] == fruit_cherche)
            window.alert("nous avons ce fruit en stock");
```

2. Solution avec `for .. in`

```
var fruits = new Array('pomme', 'poire', 'fraise', 'abricot');

for(i in fruits)
      if(fruits[i] == fruit_cherche)
             window.alert("nous avons ce fruit en stock");
```

   Avantage : aucun risque de se tromper sur la taille du tableau.

Interrompre la boucle `break`

Dans l'exemple précédent, on parcourt toutes les cases du tableau fruits pour voir si `fruit_cherche` est dedans. Pour gagner du temps on peut interrompre la boucle dès qu'on a trouvé le fruit avec `break` (NB, `break` s'applique à toutes les boucles : `while`, `for` ...).

Dans les solution apportées précédemment à l'exemple, la recherche s'effectue ainsi :

```
pomme == fruit_cherche ?
poire == fruit_cherche ?
fraise == fruit_cherche ?
abricot == fruit_cherche ?
```

Ajoutons à présent un `break` :

```
var fruits = new Array('pomme', 'poire', 'fraise', 'abricot');

for(i in fruits)
    if(fruits[i] == fruit_cherche) {
        window.alert("nous avons ce fruit en stock");
        break;
    }
```

si le fruit cherché est une poire, la recherche s'arrête dès que le fruit poire est rencontré :

```
pomme = fruit_cherche ?
poire = fruit_cherche ?
break
```

Itération suivante dans la boucle : `continue`

Permet de passer directement à la boucle suivante sans terminer la boucle courante.

```
for(i=0 ; i<5; i++) {
    if(i == 3)
        continue;
    document.writeln(i);
}
```

affichera `0 1 2 4`

5.6.4 Instructions conditionnelles

1. Si ... sinon

`if(condition){`
`  instruction1;`
`  ...`
`  instructionN;`
`} else{`
`  instruction1;`
`  ...`
`  instructionN;`
`}`
	`if(a < b)`
	`  window.alert(a+" inferieur a "+b);`
	`else`
	`  window.alert(a+" superieur a "+b);`

2. Selon valeur expression faire

`switch (expression){`
`  case val1 : ... ; break;`
`  case val2 : ... ; break;`
`  ...`
`  case valN : ... ; break;`
`  default : ...`
`}`
	`today = new Date();`
	`switch (today.getDay()) {`
	`  case 0 : jour = "dimanche"; break;`
	`  case 1 : jour = "lundi"; break;`
	`  case 2 : jour = "mardi"; break;`
	`  case 3 : jour = "mercredi"; break;`
	`  case 4 : jour = "jeudi"; break;`
	`  case 5 : jour = "vendredi"; break;`
	`  case 6 : jour = "samedi"; break;`
	`}`
	`window.alert(jour);`

Si le `break` n'est pas précisé :

```
switch (lettre) {
    case 'o' :
    case 'O' : Ouvrir(); break;
    case 'q' :
    case 'Q' : Quitter(); break;
    case 's' :
    case 'S' : Sauver(); break;
}
```

quand la lettre est un petit `o` alors on exécute tout ce qu'on trouve jusqu'au `break` (appel de la fonction `Ouvrir()`).

5.6.5 `with`

Permet d'éviter de répéter un nom d'objet tout le long du bloc (le code devient plus lisible).

```
with(document) {
   write("<h1>Titre</h1>");
   write("<h2>Sous-titre</h2>");
}
```

Chapitre 6

Classes prédéfinies

6.1 Tableaux (classe `Array`)

6.1.1 Déclaration d'un tableau

```
var mon_tableau = new Array();
var mon_tableau = new Array(taille);
var mon_tableau = new Array(el1, el2, ..., elN);
var mon_tableau = [el1, el2, ..., elN] /* depuis JavaScript 1.2) */
var tab = new Array(3, 'pomme', 4.5);
var tab = [3, 'pomme', 4.5];
```

6.1.2 Propriétés

`mon_tableau.length` : taille du tableau

Exemple :

```
i = tab.length;
```

6.1.3 Accès aux cases

```
mon_tableau[0]
mon_tableau[1]
...
mon_tableau[N-1]
```

NB : pour le parcours du tableau pensez à la boucle

`for (i in mon_tableau)`.

6.1.4 Méthodes

`join(separateur)`	concatène les cases du tableau avec le separateur (JavaScript 1.1, ECMA)
`pop()`	dépile et retourne le dernier élément du tableau (Navigator 4)
`push(val,...)`	ajoute des éléments au tableau (Navigator 4)
`reverse()`	renverse le tableau (JavaScript 1.1, ECMA)
`shift()`	supprime et retourne le 1er élément (Navigator 4)
`slice(deb, fin)`	retourne une portion du tableau (JavaScript 1.2)
`sort()`	tri des éléments (JavaScript 1.1, ECMA)
`splice(deb, nbel, val, ..)`	retire des éléments du tableau (Navigator 4)
`toString()`	conversion du tableau en chaîne (JavaScript 1.1, ECMA)
`unshift(val,...)`	insère des éléments (Navigator 4)

Exemples

`var tab = new Array();`	«vide»
`tab.push('pomme', 'ananas', 'poire', 'cerise')`	`pomme ananas poire cerise`
`tab.push('abricot', 'raisin')`	`pomme ananas poire cerise abricot raisin`
`tab.reverse()`	`raisin abricot cerise poire ananas pomme`
`fruit = tab.pop()`	`raisin abricot cerise poire ananas` `fruit = pomme`
`fruit = tab.shift()`	`abricot cerise poire ananas` `fruit = raisin`
`tab.unshift('kiwi')`	`kiwi abricot cerise poire ananas`
`liste_fruits = tab.splice(2,4)`	`kiwi abricot` `liste_fruits = ["cerise", "poire", "ananas"]`
`tab.push('pomme', 'ananas', 'poire', 'cerise')`	`kiwi abricot pomme ananas poire cerise`
`tab.sort()`	`abricot ananas cerise kiwi poire pomme`
`liste_fruits = tab.join(' - ')`	`liste_fruits = abricot - ananas - cerise - kiwi - poire - pomme`
`liste_fruits = tab.slice(1,4)`	`abricot ananas cerise kiwi poire pomme` `liste_fruits = ["ananas", "cerise", "kiwi"]`

6.2 Chaînes (classe `String`)

6.2.1 Constructeur

Deux façons pour obtenir un objet `String` :

```
var chaine1 = new String('ma chaine');
var chaine2 = 'mon autre chaine!'
```

6.2.2 Propriété : `length`

`ch.length` est le nombre de caractères contenus dans la chaîne `ch`.

6.2.3 Méthodes

`charAt(i)`	caractère qui correspond à l'index `i`
`charCodeAt(i)`	entier correspondant au code *ascii* du caractère en `i`
`+`	`chaine = chaine1 + chaine2`
`indexOf(ch,i)`	index de la première occurence de `ch` à partir de `i` (optionnel)
`lastIndexOf(ch,i)`	index de la dernière occurence de `ch` à partir de `i` (recherche arrière)
`match(exp)`	applique l'expression rationnelle `exp` et renvoie les correspondances
`replace(exp,ch)`	remplace les correspondances de `exp` et de la chaine par `ch`
`search(exp)`	index de la première correspondance entre `exp` et `chaine`
`slice(i,j)`	donne la sous-chaîne qui commence en `i` et finit en `j`
`split(sep)`	donne les sous-chaînes quand la chaîne est découpée avec `sep`
`substr(d,i)`	sous-chaîne de longueur `l` commençant en `i`
`substring(i,j)`	sous-chaîne entre `i` et `j`
`toLowerCase()`	donne la chaîne en minuscules
`toUpperCase()`	donne la chaîne en majuscules
`anchor(ancre)`	insère la chaîne dans les éléments `<A NAME="ancre">...</A>`
`big()`	insère la chaîne dans les éléments `<BIG>...</BIG>`
`blink()`	insère la chaîne dans les éléments `<BLINK>...</BLINK>`
`bold()`	insère la chaîne dans les éléments `<B>...</B>`
`fixed()`	insère la chaîne dans les éléments `<TT>...</TT>`
`fontcolor(couleur)`	insère la chaîne dans les éléments `<FONT color="couleur">...</FONT>`
`fontsize(taille)`	insère la chaîne dans les éléments `<FONT size="taille">...</FONT>`
`italics()`	insère la chaîne dans les éléments `<I>...</I>`
`link(URL)`	insère la chaîne dans les éléments `<A HREF="URL">...</A>`
`small()`	insère la chaîne dans les éléments `<SMALL>...</SMALL>`
`strike()`	insère la chaîne dans les éléments `<STRIKE>...</STRIKE>`
`sub()`	insère la chaîne dans les éléments `<SUB>...</SUB>`
`sup()`	insère la chaîne dans les éléments `<SUP>...</SUP>`

Exemple :

instruction	résultat
`var ch = "bonjour"`	bonjour
`ch.charAt(3)`	j
`ch.charCodeAt(3)`	106
`ch += " tout le monde"`	bonjour tout le monde
`ch.indexOf("ou")`	4
`ch.indexOf("ou",5)`	9
`ch.lastIndexOf("ou")`	9
`ch.lastIndexOf("ou",8)`	4
`ch.slice(3, 7)`	jour
`tab = ch.split(' ')`	tab[0]=bonjour tab[1]=tout tab[2]= le tab[3]=monde
`ch.substr(3, 7)`	jour to
`ch.toUpperCase()`	BONJOUR TOUT LE MONDE
`ch.toLowerCase()`	bonjour tout le monde
`ch.fontcolor("red")`	<FONT color="red">bonjour tout le monde</FONT>

6.3 Expressions rationnelles `RegExp`

But : traitement des chaînes de caractères (substitution, recherche, découpage...)

6.3.1 Définition d'un modèle

1. **Constructeur**

```
reg = new RegExp('modele','option1, option2...');
reg = /modele/options
```

Options :
- `g` : rechercher toutes les correspondances
- `i` : pas de distinction entre les majuscules et les minuscules

2. **Caractères spéciaux**

`.`	tout caractère
`\`	le caractère suivant n'est pas spécial
`\f`	saut de page
`\n`	saut de ligne
`\r`	retour chariot
`\t`	tabulation

Pour mettre les caractères spéciaux `* +? . / \` dans une expression rationnelle il faut les faire précéder d'un \.

3. **Classes de caractères**

`[a-zA-Z]`	caractères présents dans les crochets
`[^cp]`	interdit les caractères c et p
`\d`	chiffre (`\d{2}` = nombre à deux chiffres)
`\D`	équivalent à `[^0-9]`
`\w`	lettres, chiffres, souligné
`\W`	tout ce qui n'est pas lettre, chiffre, souligné
`\s`	espace, retour à la ligne, retour chariot, tabulation
`\S`	tout ce qui n'est pas espace
`*`	le caractère peut apparaître 0 ou n fois
`+`	le caractère doit apparaître au moins une fois
`?`	le caractère doit apparaître 0 ou 1 fois
`{n}`	le caractère doit apparaître n fois
`{n,}`	il doit apparaître au moins n fois
`{n,m}`	au moins n fois et au plus m fois.
`a | b`	a ou b

4. **Position**

`^`	correspondance en début de chaîne
`$`	correspondance en fin de chaîne
`\b`	correspondance en début de mot
`\B`	correspondance en fin de mot

5. **Mémorisation des correspondances trouvées**

   ```
   reg = /\w+\s(\w+)\s(\d+)/
   ```

   accès par `$1`, `$2` aux données entre parenthèses. Ces attributs sont utilisés en particulier pour les fonctions sur les chaînes de caractères, lorsqu'il s'agit de les modifier (inversion ou suppression de mots ...).

   ```
   <script type="text/javascript">
   <!--
   reg = /(\w+)\s*(\w+)/
   nom = "Guéganno claude";
   document.write("Nom: ", nom, "<br>");
   nom = nom.replace(reg,"$2 $1");
   document.write("Nom: ", nom, "<br>");
   //-->
   </script>
   ```

   Ce script affiche

   ```
   Nom: Guéganno claude
   Nom: claude Guéganno
   ```

6. **Méthodes**
   - `exec(chaine)` recherche le modèle dans la chaîne
   - `test(chaine)` renvoie `true` si une correspondance modèle-chaîne a été trouvée

   Exemple :

   ```
   chaine = "toto titi tata";
   reg = /t.t./;
   alert(reg.exec(chaine));
   ```

NB : les expressions rationnelles sont utilisées dans les méthodes de la classe `String` (`match`, `replace`, `search`).

Exemples

1. **Vérification d'une adresse** en appliquant les règles suivantes :
   - il faut au moins 1 caractère avant le `@` ;
   - il peut y avoir un `.` ou un `-` avant le `@`, mais pas les deux ;
   - Il doit y avoir un `@` ;
   - il peut y avoir un `.` ou un `-` après le `@`, mais pas les deux ensemble ;
   - L'adresse se termine par un `.` suivi d'au moins deux caractères.

   Le code suivant donne l'expression régulière et son utilisation.

```
<script language="javascript">
<!--
reg = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,})+$/;
ch1 = "bob-mary@chez";
ch2 = "bob@chez.com";
if (reg.test(ch1) == true)
   document.write(ch1, " est une adresse valide")
else document.write(ch1, " n'est pas une adresse valide")
document.write("<br>");
if (reg.test(ch2) == true)
   document.write(ch2, " est une adresse valide")
else document.write(ch2, " n'est pas une adresse valide")
document.write("<br>");
//-->
</script>
```

2. **Vérification d'une date** avec les règles suivantes :
   - le format est `dd/mm/aaaa` ;
   - les jours sont compris entre 01 et 31 et affichés avec 2 digits ;
   - les mois sont compris entre 01 et 12 et affichés avec 2 digits ;
   - l'année est affichée sur 4 digits.

   On en déduit l'expression régulière :

```
<script language="javascript">
<!--
reg = /^(3[01]|0[1-9]|[12]\d)\/(0[1-9]|1[012])\/\d{4}$/;
date1 = "3/08/1999"
date2 = "23/09/2003"
if (reg.test(date1) == true)
   document.write(date1, " est une date valide")
else document.write(date1, " n'est pas une date valide")
document.write("<br>");
if (reg.test(date2) == true)
   document.write(date2, " est une date valide")
else document.write(date2, " n'est pas une date valide")
//-->
</script>
```

6.4 Mathématiques : classe `Math`

Cette classe ne contient que des attibuts et métodes statiques : il n'y a pas de constructeur. Les constantes sont accessibles avec la syntaxe :

```
Math.constante
```

et les méthodes :

```
Math.fonction(paramètres)
```

6.4.1 Constantes mathématiques

`E`	$\exp 1$
`LN10`	$ln10$
`LN2`	$ln2$
`PI`	π
`SQRT1_2`	$\sqrt{\frac{1}{2}}$
`SQRT2`	$\sqrt{2}$

6.4.2 Méthodes mathématiques

`abs(x)`	$\|x\|$
`acos(x)`	$\arccos x \quad x \in [-1, 1]$
`asin(x)`	$\arcsin x \quad x \in [-1, 1]$
`atan(x)`	$\arctan x$
`atan2(rho,phi)`	angle associé aux coordonnées polaires de (ρ, ϕ)
`ceil(x)`	plus petit entier supérieur ou égal à n
`cos(x)`	$\cos x$
`exp(x)`	$\exp x$
`floor(x)`	plus grand entier $\leq x$
`log(x)`	$\log x$
`max(a, b)`	retourne le paramètre le plus grand
`min(a, b)`	retourne le paramètre le plus petit
`pow(x, y)`	x^y
`random()`	nombre pseudo-aléatoire $\in [0, 1]$
`round(x)`	entier le plus proche de x
`sin(x)`	$\sin x$
`sqrt(x)`	$\sqrt{x}$
`tan(x)`	$\tan x$

6.5 Dates : classe `Date`

Constructeur

```
var date_courante = new Date();
ma_date = new Date(annee, mois, jour);
ma_date = new Date(annee, mois, jour, heure, minute, seconde, milliseconde);
```

Méthodes pour les heures, minutes, secondes, millisecondes

temps	lire	affecter
heure (0..23)	`getHours()`	`setHours(h)`
minutes (0..59)	`getMinutes()`	`setMinutes(mn)`
secondes (0..59)	`getSeconds()`	`setSeconds(s)`
millisecondes (0..999)	`getMilliseconds()`	`setMilliseconds(m)`

temps	lire	affecter
heure (0..23)	`getUTCHours()`	`setUTCHours(h)`
minutes (0..59)	`getUTCMinutes()`	`setUTCMinutes(mn)`
secondes (0..59)	`getUTCSeconds()`	`setUTCSeconds(s)`
millisecondes (0..999)	`getUTCMilliseconds()`	`setUTCMilliseconds(m)`

Méthodes pour les jours, mois, années

temps	lire	affecter
jour de la semaine anglaise (0..6) qui se traduit par dimanche..samedi	`getDay()`	
jour du mois (1..31)	`getDate()`	`setDate(jour)`
mois (0..11)	`getMonth()`	`setMonth(mois)`
`Netscape` : nombre d'années depuis 1900 ; `IE` : année sur 4 chiffres ($\Rightarrow$ à éviter)	`getYear()`	`setYear(annee)`
année sur 4 chiffres	`getFullYear()`	`setFullYear(annee)`

Fonctions `UTC` `getUTCDay()`, `getUTCDate()`, `setUTCDate(jour)`, `getUTCMonth()`, `setUTCMonth(mois)`, `getUTCFullYear()`, `setUTCFullYear(annee)`.

```
<html><!-- date.html -->
<head>
   <title>Detecte heure</title>
</head>

<body>
<h1>Heure</h1>
<script language="JavaScript">
// <!--
   today = new Date();

   document.write(today.getHours(),' h ');
   document.write(today.getMinutes(),' mn ');
   document.write(today.getSeconds(),' s ');
   document.write(today.getMilliseconds(),' ms');
// -->
</script>

<h1>Date</h1>
```

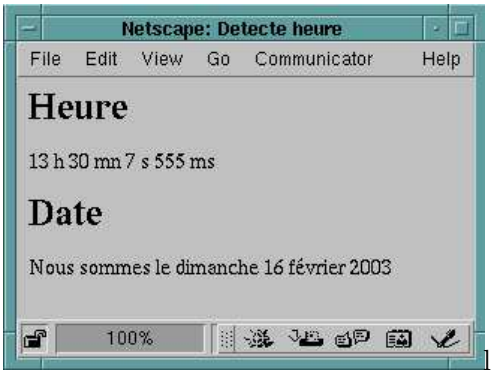
```
<script language="JavaScript">
   // <!--
   var Tjour = new Array("dimanche","lundi","mardi","mercredi",
                         "jeudi","vendredi","samedi");
   var Tmois = new Array("janvier","février","mars","avril","mai",
                         "juin","juillet","août", "septembre",
                         "octobre","novembre","décembre");

   today = new Date();
   var jsem = today.getDay();
   var mois = today.getMonth();

   var ch ="Nous sommes le " + Tjour[jsem] + " " + today.getDate() +
           " " + Tmois[mois] + " " + today.getFullYear();

   document.write(ch);
   // -->
   </script>

</body>
</html>
```



Aperçu de ***date.html***

Autres méthodes

`getTime()`, `setTime(ms)`	nombre de *ms* depuis le 01/01/1970 Ex : `1045400238785`
`getTimeZoneOffset()`	nombre de minutes de décalage heure locale - heure GMT Ex : `-60`
`toUTCString()`	retourne une chaîne Ex :`Sun, 16 Feb 2003 12 :57 :18 GMT`
`toLocaleString()`	retourne une chaîne Ex : `dim 16 fév 2003 13 :57 :18 CET`

6.6 Images : classe `Image`

Constructeur

```
var mon_image = new Image();
```

Propriétés

`border`	largeur du cadre
`height`	hauteur
`hspace`	espace horizontal
`lowsrc`	URL de l'image basse résolution
`name`	nom de l'image
`src`	URL de l'image
`vspace`	espace vertical
`width`	largeur de l'image
`complete`	`false` tant que le chargement n'est pas fini

Remarque : On peut accéder à une image par l'intermédiaire d'un tableau `window.document.images[num]` dont la taille est donnée par

`window.document.images.length`

ou par son nom (celui qu'on lui a donné dans la balise `IMG`) `window.document.nom`.

Méthodes

`onabort`	chargement interrompu par le client
`onerror`	erreur au chargement
`onload`	pendant le chargement

Exemple

Cet exemple crée une page `html` qui contient une image et un bouton. À chaque appui sur le bouton, l'image est changée par une autre. Les changements d'images se font de manière circulaire sur un tableau de 3 images.

```
<html>
  <head>
    <title>images ...</title>
    <script language="javascript">
      var T = new Array("a.jpg", "l.jpg", "n.jpg");
      var imageCourante = 0;
      function changeImage(){
         imageCourante++;
         imageCourante %= 3;
         window.document.images[0].src=T[imageCourante];
      }
   </script>
  </head>
```

```
<body>
   <h1>images ...</h1>
   <center>
     <img src="a.jpg">
     <form>
       <input type="button" name="Change" onclick="changeImage()">
     </form>
   </center>
   <hr>
 </body>
</html>
```

6.7 Classes instanciées par le navigateur

6.7.1 `Navigator`

Un seul objet est instancié : `navigator`

Propriétés de `navigator`

`appCodeName`	nom de code du navigateur Ex : `Mozilla`
`appName`	nom d'application du navigateur Ex : `Netscape`
`appVersion`	plate-forme et version du navigateur Ex : `4.72 [en] (X11 ; I ; Linux 2.2.14 i586)` `parseFloat(navigator.appVersion)` → 4.5
`language`	langage du navigateur Ex : `en`
`mimeTypes`	tableau des types *MIME* reconnus par le navigateur
`platform`	système d'exploitation Ex : `LinuxELF2.2`
`plugins`	tableau des *plugins* installés
`userAgent`	chaîne envoyée lors d'une requête `http` Ex : `Mozilla/4.72 [en] (X11 ; I ; Linux 2.2.14 i586)`

Exemple : pour accéder au nom d'application du navigateur il suffit donc d'écrire : `navigator.appName`.

Tableau `plugins`

– Propriétés des éléments du tableau

$$\begin{cases} \texttt{name} : \text{nom du } \textit{plug-in} \\ \texttt{description} : \text{description du } \textit{plug-in} \end{cases}$$

– Accès aux propriétés par index
`navigator.plugins[i]`
Ex : informations sur le plugin de la case 0 :
  – `navigator.plugins[0].name` : `Netscape Default Plugin`
  – `navigator.plugins[0].description` : `The default plugin handles plugin data for mimetypes and extensions that are not specified and facilitates downloading of new plugins.`
– Accès aux propriétés par le nom du plugin
`navigator.plugins['nomplugin']`

Tableau type *MIME*

– Qu'est-ce que le *Mime* ? *Multipurpose Internet Mail Extension* (Bell Communications 1991). C'est un standard pour étendre les possibilités du courrier électronique, et insérer dans un *e-mail*, du texte, des images ou du son. Avec `http` le serveur donne le type *Mime* avant d'envoyer le document :

```
Content-type:type_mime/sous_type
```

Exemples :

$\begin{cases} \texttt{Content-type :text/html} \rightarrow \text{document } \textit{.html} \\ \texttt{Content-type :image/gif} \rightarrow \text{image au format } \textit{gif} \end{cases}$

– Intérêt :
  – savoir si le navigateur est *Netscape* ou *Internet Explorer*
  – savoir si le client peut visualiser des images `PNG` :

```
if(navigator.mimeTypes['image/png'])
     action;
else
     autre action;
```

– Propriétés des éléments du tableau *MIME*

`type`	nom du type *mime*
`description`	description du type
`enabledPlugin`	référence vers le plugin qui le gère
`suffixes`	listes des suffixes de fichiers gérés par ce type

– Accès aux propriétés par le nom du type

```
navigator.mimeTypes['nomtype']
```

Exemple : informations sur le type mime `image/gif`,
`navigator.mimeTypes['image/gif'].description` donne : `«GIF Image»`
– Accès aux propriétés par index

```
navigator.mimeTypes[i]
```

Exemple : informations sur la case 1 du tableau *mime* :
`navigator.mimeTypes[1].type` → `video/x-sgi-movie`

6.7.2 `Screen`

Un seul objet est instancié : `screen`

Propriétés

`height`	hauteur de l'écran en pixels
`width`	largeur de l'écran en pixels
`availHeight`	hauteur moins les barres de menu ou de tâches
`availWidth`	largeur disponible
`pixelDepth`	nombre de bits pour coder un pixel dans le navigateur

6.7.3 Classe `Window`

Un objet `window` est instancié pour chaque zone du navigateur (fenêtre, *frame*).

Propriétés

– Référence aux objets

```
window.nom_propriete.propriete_specifique
```

`closed`	pour savoir si la fenêtre a été fermée ou non
`defaultStatus`	texte à afficher par défaut dans la barre d'état en bas
`document`	accès à toutes les informations de la zone
`frames[]`	tableau d'objets pour les *frames* de la zone

`window.frames.length`
`window.frames[num]`
`history`
	pour naviguer

`length`	nombre de cadres dans la fenêtre
`location`	informations sur l'URL de la zone
`name`	nom de la zone
`navigator`	réfère à l'objet `navigator`
`offscreenBuffering`	mises à jour dans des tampons
`opener`	fenêtre à partir de laquelle a été créée la zone
`parent`	document contenant `<frame>` associé au *frame*
`screen`	réfère à l'objet `screen`
`self`	la zone elle-même
`status`	texte à afficher dans la barre d'état
`top`	la fenêtre d'un cadre
`window`	la fenêtre elle-même

– Le champ `history`

Propriétés de `window.history`

`current`	URL du document courant
`length`	nombre d'éléments dans l'historique
`next`	URL suivante dans l'historique
`previous`	URL précédente dans l'historique

Méthodes de `window.history`

`back()`	pour simuler le bouton back
`go(+/-n)`	effectuer un retour ou une avancée dans l'historique de +/- n pages
`forward()`	pour simuler le bouton *forward*

Exemple

```
<input type="button" value="back" onClick="window.history.back()">
<input type="button" value="back2" onClick="window.history.go(-2)">
<input type="button" value="forward" onClick="window.history.forward()">
```

– Le champ `location`

Propriétés de `window.location`

`href`	totalité de l'URL
`hash`	partie ancre de l'URL (après `'#'`)
`host`	nom du serveur et port
`hostname`	adresse IP ou nom
`pathname`	chaîne après le nom du script et avant `'?'`
`port`	numéro de port
`protocol`	nom du protocole (`http`)
`search`	partie requête (après `'?'`)

Méthodes de `window.location`

`reload()`	pour recharger le document
`replace(URL)`	pour charger le document situé à l'adresse URL

```
<input type="button" value="recharger" onClick="window.location.reload()">
<input type="button" value="remplacer"
       onClick="window.location.replace('/index2.html')">
```

– Le champ `document`

Propriétés de `window.document`

`bgColor`	couleur de fond
`fgColor`	couleur du texte par défaut
`alinkColor`	couleur des liens activés
`linkColor`	couleur des liens non visités
`vlinkColor`	couleur des liens visités
`anchors[num]`	cible d'un lien hypertexte `window.document.anchors.length` → nombre d'ancres `window.document.anchors[i].name` → nom de l'ancre
`applets[num]`	*applets* d'un document les propriétés sont celles des champs publics de l'*applet* `java`, les méthodes sont les méthodes publiques de l'*applet* `java`
`cookie`	cookies du document
`embeds[num]`	objets incorporés dans le document
`forms[num]`	formulaires du document
`images[num]`	images du document
`links[]`	tableau pour les liens du document au nombre de `window.document.links.length` attributs : `hash`, `host`, `hostname`, `href`, `pathname`, `port`, `protocol`, `search`, `target`
`plugins[num]`	objets incorporés dans le document
`lastModified`	date de modification du document
`referrer`	URL du document provenant du lien
`title`	titre du document
`URL`	URL du document

Méthodes de `window.document` :

`write(param1, ..., paramN)`	insertion dynamique dans le document.
`writeln(param1, ..., paramN)`	idem avec retour à la ligne
`open()`	vide le document et en crée un nouveau
`close()`	affichage de tout ce qui a été écrit avec `write()` ou `writeln()`

Méthodes de `window`

`open(arguments)`	ouvrir une nouvelle fenêtre
`close()`	fermer une fenêtre
`alert(texte)`	ouvre une fenêtre d'alerte contenant le texte
`blur()`	faire perdre le focus
`focus()`	donner le focus
`prompt(txt, val)`	ouvre une fenêtre avec un message *txt* et un champ de texte contenant *val*, retourne le texte
`confirm(texte)`	fenêtre de confirmation qui retourne un booléen
`back()`	bouton back du navigateur
`find()`	recherche dans le document
`forward()`	bouton forward du navigateur
`home()`	bouton accueil du navigateur
`stop()`	bouton stop du navigateur
`setTimeout(code, delai)`	compte à rebours, *delai* en ms
`clearTimeout(id)`	désactiver un compte à rebours avant son terme
`setInterval(code, int)`	déclenche l'exécution périodique d'une fonction ; *int* en ms
`clearInterval(id)`	stoppe de façon périodique l'exécution du code
`moveBy(x,y)`	déplace la fenêtre (zone, couche) de x pixels horizontaux et y verticaux
`moveTo(x,y)`	déplace le coin gauche en (x, y)
`resizeBy(l,h)`	pour retrécir ou agrandir la fenêtre
`resizeTo(l,h)`	largeur et hauteur de la fenêtre à agrandir ou réduire
`scrollBy(x,y)`	déplacement horizontal et vertical du document dans la fenêtre (ascenseurs)
`scrollTo(x,y)`	déplacement du coin x, y du document en haut de la fenêtre

Exemple qui complète celui de la page 62 pour changer l'image automatiquement chaque seconde.

```
<script language="javascript">
   setInterval("changeImage()", 1000);
</script>
```

La méthode `open(URL, nom, options)` prend comme paramètres :
- `URL` : URL du document à afficher dans la fenêtre (optionnel)
- `nom` : nom de la fenêtre (`target` pour les autres documents HTML)

– `option` :

$$\begin{cases} \texttt{height} : \text{hauteur initiale} \\ \texttt{width} : \text{largeur initiale} \\ \texttt{left} : \text{distance bord gauche fenêtre - bord gauche écran} \\ \texttt{location} = yes|no : \text{barre de localisation} \\ \texttt{menubar} = yes|no : \text{barre de menus} \\ \texttt{resizable} = yes|no : \text{autorisation de redimensionnement} \\ \texttt{scrollbars} = yes|no : \text{barres de défilement} \\ \texttt{status} = yes|no : \text{barre de statut} \\ \texttt{toolbars} = yes|no : \text{barre d'outils} \end{cases}$$

Événements

`onBlur`	la fenêtre perd le focus
`onFocus`	la fenêtre prend le focus
`onLoad`	la fenêtre est chargée dans le navigateur
`onUnload`	le navigateur quitte la page
`onError`	erreur de code `JavaScript` ou de chargement d'une image
`onResize`	la fenêtre est redimensionnée

Exemple

```
<body onBlur="fonction1" onFocus="fonction2">

<!-- Pour supprimer l'affichage des erreurs :-->
<script language="javascript">
     window.onError = null
</script>
```

Chapitre 7

Interactivité

7.1 Accès aux données du formulaire

Nous allons prendre l'exemple du formulaire de la figure 3.1 (page 29) afin d'illustrer l'accès aux champs pour lire les données ou pour les modifier.

```
<form name="commande" action=...>
```

7.1.1 Accès aux textes contenus dans les champs

Les éléments de formulaire concernés sont :
- boîtes de saisie de texte (`input type="text"`)
- saisie de textes sur plusieurs lignes (`textarea`)
- boîtes de saisie masquées (`input type="password"`)
- boîtes de soumission de fichier (`input type="file"`)
- données masquées (`input type="hidden"`)

Exemple : on vérifie que le code client est composé de 4 chiffres (exactement). Le code `html` de l'entrée `password` devient :

```
Code client:
 <input type="password" size="8" maxlength="8"
        name="mdp" value=""
        onChange="verifCode(value)">
```

La fonction `verifCode` est appelée dès que l'entrée `password` perd le *focus*. Un codage `JavaScript` possible pour cette fonction :

```
/* Renvoie Vrai si la zone de texte ne contient que des chiffres */
function verifCode(chaine){
    var regnb = /^\d{4}$/; // ^:début de chaîne; $:fin de chaîne
                           // \d{4}: exactement 4 chiffres
    if(!regnb.test(chaine)) {
        window.alert("Le code doit contenir 4 chiffres");
        document.commande.mdp.focus();
        document.commande.mdp.select();
        return false;
    }
    else return true;
```

```
}
```

Si le code est erroné, le *focus* est redonné à l'entrée `password`.

7.1.2 Accès aux choix cochés par le client

Les éléments de formulaire concernés sont :
- cases à cocher (`input type="checkbox"`)
- boutons radio (`input type="radio"`)

Recherche des choix cochés

```
window.document.nom_formulaire.nom_element[num_case].checked
```

retourne une valeur `true/false`.

```
window.document.nom_formulaire.nom_element[num_case].value
```

retourne le nom donné à la boîte.

Exemple pour les boutons radio : à chaque intervention sur les boutons radio, on affiche le choix dans la zone de texte réservée aux remarques. L'événement à capter est `onClick` :

```
Fromage <input type="radio" name="fin" checked
                value="fromage" onClick="afficheFin()">
  Dessert <input type="radio" name="fin" value="dessert"
                      onClick="afficheFin()">
```

La fonction `javascript` est :

```
/* Affiche dans la zone de texte le bouton radio sélectionné: */
function afficheFin() {
  with (window.document.commande){
    rem.value=""; // effacement
    if (fin[0].checked) rem.value="Fromage";
    else rem.value="Dessert";
  }
}
```

7.1.3 Accès aux choix dans les listes de choix

L'élément de formulaire concerné est `select`. Ses propriétés sont :
- `length` : le nombre de choix possibles dans la liste
- `options[]` : tableau dans lequel se trouve chacune des options de la liste de sélection.
- `options[num_case].text` : retourne le texte `HTML` associé à une option.
- `options[num_case].selected` : valeur booléenne indiquant, ou permettant d'indiquer que le choix est sélectionné
- `options[num_case].value` : retourne le nom donné au choix
- `selectedIndex` : dans le cas d'un choix unique, donne directement l'index de l'élément sélectionné.

Exemple pour la liste à choix multiples : on décide de rendre le formulaire plus interactif en désélectionnant systématiquement toutes les entrées dès que l'option «rien» est choisie. Le code HTML est :

```
<select name="acc"  size=3  multiple onChange="traiteChoix()">
   <option selected value="salade">salade</option>
   <option value="legumes">légumes</option>
   <option value="champignons">champignons</option>
   <option value="rien">rien</option>
</select>
```

...et la fonction JavaScript associée :

```
/* Si le choix "rien" est sélectionné, tous les autres disparaissent */
function traiteChoix(){
   with (window.document.commande){
     var i
     if (acc.options[acc.length-1].selected) {
       for (i=0; i<acc.length-1; i++) {
          acc.options[i].selected = false;
       }
     }
  }
}
```

Amélioration : supprimer la sélection «rien» dès qu'un nouveau choix est formulé.

Remarque : dans cet exemple, la liste de sélection est nommée «acc», ce qui permet, avec `acc.options[i]` de lire l'état des choix. L'accès au formulaire sur le serveur en php impose que le nom de la liste soit «acc[]». Mais dans ce cas, ce nom pose un problème en JavaScript ...

7.1.4 Accès aux textes sur les boutons

Les éléments de formulaire concernés sont :
- boutons de soumission (`input type="submit"`)
- boutons de soumission avec image(`input type="image"`)
- boutons d'annulation (`input type="reset"`)
- boutons simples (`input type="button"`)

```
window.document.nom_formulaire.nom_element.value
```

Exemple avec un bouton ordinaire : l'événement capturé est le clic sur le bouton. Le texte du bouton peut être modifié.

```
<html>
  <head>
    <title>mouse</title>
  </head>

    <script language='javascript'>
      function change(){
```

```
        with (window.document.f1.b1) {
          if (value=="on") value="off"; else value="on";
        }
      }
    </script>

  <body><center><h1>Bouton</h1>
    <form name="f1">
      <input type="button" name="b1" value="on" onClick="change()">
    </form>
  </center>
 </body>
</html>
```

FIG. 7.1 – *Exécution de* `bouton.html`

7.2 Accès par tableau

Nous avons pour l'instant fait des accès par nom, il est possible d'accéder aux éléments du formulaire par l'intermédiaire du tableau `forms[num_form]`. Ce tableau est un attribut de `window.document` :

`window.document.forms[0].elements.length :` nombre d'éléments dans le formulaire

`window.document.forms[0].elements[i].type :` type de l'élément i

Pour les types `text`, `password`, `hidden`, `file`, `textarea` :

`window.document.forms[0].elements[i].name :` nom donné à l'élément

`window.document.forms[0].elements[i].value :` texte dans l'élément

Pour les types `radio`, `checkbox` :

`window.document.forms[0].elements[i].ckecked :` pour savoir si l'élément est coché

`window.document.forms[0].elements[i].value :` valeur de l'élément i

Pour les types `select` unique et multiple :

`window.document.forms[0].elements[i].options[j].selected :` pour savoir si une option est sélectionnée

`window.document.forms[0].elements[i].options[j].value :` valeur de l'option

Pour les types `button`, `submit`, `reset` :

`window.document.forms[0].elements[i].value :` texte sur l'élément bouton

Autres informations disponibles sur le formulaire :

`window.document.forms[i].action :` valeur de l'attribut `action` du formulaire numéro i

`window.document.forms[0].method :` valeur de l'attribut `method` du formulaire numéro i

`window.document.forms[0].name :` valeur de l'attribut `name` du formulaire numéro i

`window.document.forms[0].target :` valeur de l'attribut `target` du formulaire numéro i

7.2.1 Événements

Lorsque le client clique dans le formulaire, modifie des champs, coche des boutons radio..., il est possible d'exécuter du code `JavaScript` . Ce code sert généralement à :

1. vérifier la validité des données entrées (pas de lettres dans un numéro de fax)
2. donner des informations (affichage d'un message dans la barre d'état lorsqu'on clique dans un champ de formulaire)

Les événements se placent dans l'élément de formulaire comme des attributs, ils sont de la forme `nom_evenement = "code JavaScript"` le code `JavaScript` étant la plupart du temps un appel à une fonction `JavaScript` définie plus haut dans la page, ou dans un fichier externe.

Exemple

```
<input type="button" value="on" name="b1" onClick="change()">
```

Nous prendrons pour illustrer les événements le formulaire déjà utilisé pour montrer l'accès aux éléments :

`onClick` **(clic sur un élément)** Éléments concernés :
- cases à cocher (`input type="checkbox"`)
- boutons radio (`input type="radio"`)
- boutons de soumission (`input type="submit"`)
- boutons de soumission avec image (`input type="image"`)
- boutons d'annulation (`input type="reset"`)
- boutons simples (`input type="button"`)

Exemple 1 – Événement `onClick` pour un bouton *A propos...*

```
<HTML>
<HEAD>...</HEAD>
 <SCRIPT language="JavaScript">
  <!--
    function Info() {
       window.alert("Version 1.0 du 1er mars 2003");
    }
  // -->
 </SCRIPT>
<BODY>
  <FORM ...>
  ...
  <INPUT type="button" value="A propos.." name="binfo" onClick="Info()">
  ...
  </FORM>
</BODY>
</HTML>
```

Lorsque l'utilisateur clique sur le bouton Informations, une fenêtre d'alerte apparaît avec le texte : *«Version 1.0 du 1er mars 2003»*

Exemple 2 – Événement `onClick` pour un bouton `submit`
Fonction JavaScript déclarée dans `<SCRIPT...>...</SCRIPT>` :

```
function Confirmer() {
    return window.confirm("Voulez-vous vraiment soumettre ?");
}
```

Élément de formulaire dans l'élément `FORM`

```
<INPUT type="submit" value="OK" onClick="return Confirmer()">
```

lorsque l'utilisateur clique sur le bouton `OK`, une fenêtre de confirmation apparaît avec le texte, s'il choisit d'annuler, alors la soumission est annulée grâce aux deux `return`

Exemple 3 – Événement `onClick` pour les boutons `radio`
Fonction `JavaScript` déclarée dans `<SCRIPT...>...</SCRIPT>`

```
function afficheStatus(choix){
  switch(choix){
   case 1: window.status="Notre fromage: roquefort"; break;
   case 2: window.status="Notre dessert: une mousse au chocolat"; break;
   default: break;
  }
}
```

Éléments de formulaire dans l'élément `FORM` :

```
Fromage<input type="radio" name="fin" checked
               value="fromage" onClick="afficheStatus(1)">
Dessert<input type="radio" name="fin" value="dessert"
              onClick="afficheStatus(2)">
```

`onChange` **(modification dans l'élément)** Éléments concernés :
- boîtes de saisie de texte (`INPUT type="text"`)
- saisie de textes sur plusieurs lignes (`TEXTAREA`)
- boîtes de saisie masquées (`INPUT type="password"`)
- boîtes de soumission de fichier (`INPUT type="file"`)
- listes (`SELECT`)

`onFocus, onBlur` **(attribution, perte du focus)** Éléments concernés :
- boîtes de saisie de texte (`INPUT type="text"`)
- saisie de textes sur plusieurs lignes (`TEXTAREA`)
- boîtes de saisie masquées (`INPUT type="password"`)
- boîtes de soumission de fichier (`INPUT type="file"`)
- listes (`SELECT`)

Exemple : événement `onFocus` pour un champ masqué élément de formulaire dans l'élément `FORM` :

```
Code client: <input type="password" size="8" maxlength="8"
                    name="mdp" value=""
                    onFocus="window.status='Entrez votre code'">
```

Lorsque le client clique dans le champ, le message *«Entrez votre code»* apparaît dans la barre d'état.

`onSelect` **(sélection dans un champ)** Éléments concernés :
- boîtes de saisie de texte (`INPUT type="text"`)
- saisie de textes sur plusieurs lignes (`TEXTAREA`)
- boîtes de saisie masquées (`INPUT type="password"`)
- boîtes de soumission de fichier (`INPUT type="file"`)

`onReset, onSubmit` Lorsque le formulaire est soumis en appuyant sur le bouton de soumission il est possible d'appeler un événement `onSubmit` (pour vérifier par exemple la validité des données avant d'envoyer le formulaire, pour demander la confirmation de la soumission).

L'événement `reset` est appelé lorsqu'on a cliqué sur le bouton `annuler`.

Exemple :

```
<FORM name="f1" action="traiteform.php3" onSubmit="return verif()">
```

Lorsque l'utilisateur clique sur le bouton `Valider`, une fonction de vérification `verif()` est appelée, si elle retourne la valeur `true`, le formulaire est envoyé au serveur, sinon la soumission est annulée.

7.2.2 Poster un formulaire

Envoyer le contenu d'un formulaire à une adresse *mail*

Le formulaire est habituellement envoyé à un script qui traite les données par l'intermédiaire de l'attribut action de l'élément `FORM`. Il est possible

d'utiliser l'URL donné dans action pour envoyer le contenu du formulaire à une adresse *mail* (il suffit d'utiliser `mailto`).

Exemple :

```
<form action="mailto:claude.gueganno@free.fr" enctype="text/plain">
```

7.2.3 Utilisation des classes pour rendre une page interactive

La notion de classe a été introduite au chapitre 2. Par exemple, la feuille de style suivante défini deux classes :

```
.c1 {color:red; font-weight:bold; }
.c2 {color:#0e682f; font-weight:normal; }
```

Un paragraphe compris entre les balises `<p class="c1">` et `</p>` apparaîtra donc en caractères rouges et gras.

Un nom de classe peut être changé par une fonction `javascript`. Si on utilise les événements liés à la souris pour déclencher une telle fonction, on peut augmenter de manière significative l'interactivité de la page.

Les événements généralement utilisés sont :

- `onMouseover` : quand la souris arrive sur un élément,
- `onMouseout` : quand la souris sort d'un élément.

Exemple

```
<html>

<head>
 <style>
 .c1 {color:red; font-weight:bold; }
 .c2 {color:#0e682f; font-weight:normal; }
 </style>
</head>

<script language="javascript">
   function sOver(src){ src.className="c1";}
   function sOut(src){ src.className="c2";}
</script>


<body>
  <ul>
    <li><a class="c2"
           onMouseover="sOver(this);" onMouseout="sOut(this);"
           href="p1.html">page 1</a></li>
    <li><a class="c2"
           onMouseover="sOver(this);" onMouseout="sOut(this);"
           href="p2.html">page 2</a></li>
  </ul>
</body>
</html>
```

Troisième partie

PHP

Chapitre 8

Bases du langage

8.1 Introduction

8.1.1 Historique

1994 : Rasmus Lerdof crée un ensemble de *scripts Perl* (`Personal Home Pages`) pour son utilisation personnelle.

1995 : mise à la disposition du public de `PHP`. `PHP3` exécute le code en le lisant.

2000 : `PHP4` compile le code et l'exécute.

8.1.2 Caractéristiques

En bref :
- c'est un langage de `script` ;
- module intégré dans `Apache` ;
- écrit dans la page `html` : les codes `HTML`, `JavaScript` et `php` sont mélangés dans la page ;
- traité par le serveur (la page reçue ne contient plus de code `php`) ;
- résultat de requêtes à des `SGBD`, de calculs -> `HTML` dynamique : la page est fabriquée en fonction des résultats de calculs ou de requètes :
- gratuit ;
- syntaxe proche du `C` ;
- adresse officielle (beaucoup de documentations) `http ://www.php.net` ;
- les fichiers contenant du code `php` finissent par *.php3* ou *.php* ;
- utilisation sous `Linux` ou `Windows` avec le serveur `Apache`.

8.1.3 Différences entre `PHP` 3 et 4

`PHP4` compile le code et l'exécute, alors que `PHP3` exécutait au fur et à mesure qu'il lisait le code. Pour des scripts complexes `PHP4` peut être de 5 à 200 fois plus rapide que `PHP3`. En `PHP3`, le *script* étant exécuté il pouvait s'interrompre après avoir commencé le traitement à cause d'une erreur à la ligne 200 par exemple. Avec `PHP4`, un code erroné ne s'exécute pas.

Le succès de PHP3, a fait que le développement de PHP4 a été réalisé de manière à garder la compatibilité. Quelques incompatibilités connues :

- en PHP3 les *cookies* sont envoyés dans l'ordre inverse par rapport aux appels multiples à la fonction `setcookie()`. Dans PHP4 ils sont envoyés dans le même ordre que les appels à cette fonction.
- Les variables statiques en PHP4 acceptent uniquement des scalaires alors qu'en PHP3 toute expression valide est acceptée.
- un `return` dans un fichier inclus par `require(file)` ne fonctionne pas (il suffit d'inclure le fichier avec `include(file)`).
- dans une chaîne, la combinaison suivante `"{$"` produit une erreur dans PHP4, il faut utiliser `"\{$"`
- la chaîne 0 est considérée comme vide en PHP4.

8.2 PHP dans HTML

Le code PHP apparaît dans la page entre des balises :

Le '<' et le '>' sont des indicateurs de début et de fin de balise comme en html, pour les distinguer des balises html, trois approches sont autorisées :

1. `<? code php ?>`
2. `<?php code php ?>`
3. `<SCRIPT language="php"> code php </SCRIPT>`

Les instructions dans les balises sont séparées par des `;`

Tout ce qui est entre `/*` et `*/` est un commentaire.

Exemple de fichier php, le fichier suivant est sur le serveur :

```
<HTML>
  <HEAD>
    <title>premiers pas en PHP</title>
  </HEAD>

  <BODY>
    <H1>Essai de script PHP</H1>

    <?php
       echo "Bonjour, il est ";
       echo date ("H:i:s");
    ?>
</BODY>
</HTML>
```

La page envoyée au navigateur est :

```
<HTML>
  <HEAD>
    <title>premiers pas en PHP</title>
  </HEAD>

  <BODY>
```

```
    <H1>Essai de script PHP</H1>
Bonjour, il est 17:05:38</BODY>
</HTML>
```

Remarque : on note que tout ce qui était entre les balises `<?` et `?>` a été interprété et transformé en code HTML.

8.3 Les données

8.3.1 Les variables

- La syntaxe pour déclarer une variable : `$ma_variable = expression ;`
- attention à la casse (`$i` différent de`$I`) ;
- le nom de la variable commence avec une lettre ou un souligné et est constitué de lettres, chiffres, souligné ;
- pas de déclaration préalable ;
- en PHP3 les variables sont assignées uniquement par valeurs.

Exemple

```
<?php
  $reel = 0.3;
  $entier = 22;
  $chaine = "World !";
  $phrase1 = "Hello $chaine!";
  $phrase2 = 'Hello $chaine$';

  print("un réel : $reel<BR>");
  print("un entier : $entier<BR>");
  print("une chaîne : $chaine<BR>");
  print("une phrase : $phrase1<BR>");
  print("du bon usage des guillemets : $phrase2");
?>
```

Données produites :

```
un réel : 0.3
un entier : 22
une chaîne : World !
une phrase : Hello World !!
du bon usage des guillemets : Hello $chaine$
```

Affichage des variables. Les lignes suivantes sont équivalentes :

```
echo "N = ".$N."<br>";
echo "N = $N<br>";
print("N =$N<br>");
```

Le caractère `$` est détecté par PHP et `$N` est remplacé par sa valeur. Si $N = 1234$, l'affichage produit par ces lignes est donc :

```
N = 1234
N = 1234
N = 1234
```

La substitution ne se fait pas si on utilise le caractère `'` pour délimiter la chaîne de caractères. Ainsi,

```
echo 'N = $N';
```

affiche

```
N = $N
```

– En PHP4 les variables peuvent être assignées par références (elles deviennent un alias sur la variable qu'elles référencent) :

```
<?php
$chaine1 = "Bonjour";
$chaine2 = &$chaine1;   // Reference $chaine1 par $chaine2.
$chaine2 = "$chaine2 tout le monde";
print("$chaine1\n$chaine2");
?>
```

donne :

```
Bonjour tout le monde
Bonjour tout le monde
```

– type donné automatiquement (`integer`, `double`, `string`, `array`, `object`).
– pour écrire un guillemet dans une chaine, utiliser `\"`.

```
<?php
  $chaine = "<a href=\"index.html\">accueil</a>";
  echo $chaine;
?>
```

donne :

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

Fonctions de conversion

1. `intval`, `strval`, `doubleval`
   Exemple : `$entier = intval($reel) ;`
2. `(integer)`, `(string)`, `(double)`
   Exemple : `$entier = (integer)$reel ;`
3. `int settype(var, type)` : force la variable *var* à prendre le type *type* et renvoie `0` en cas d'échec.
   Exemple : `settype($i,"integer") ;`
   Les type possibles sont `"integer"`, `"double"`, `"string"`, `"array"` et `"object"`.
4. `strval(argument)` retourne l'argument sous forme de chaîne.

Fonctions pour connaître le type

– `gettype($v)` : renvoie une chaîne de caractère donnant le type de `$v`.
– `is_array($v)`, `is_double($v)`, `is_integer($v)`, `is_object($v)`, `is_string($v)` donnent 1 si `$v` est du type précisé et 0 sinon.

Variables d'environnement

Il existe des variables prédéfinies, la plupart dépendent du serveur qui appelle le script `php`.

Pour obtenir la liste complète il suffit d'exécuter le *script* ci-dessous, une page sera affichée contenant des tableaux qui décrivent la configuration, les variables d'environnement `Apache` sont dans le tableau «`Apache Environment`».

```
<?php
   phpinfo();
?>
```

Quelques unes des variables d'environnement (`Apache`)

`HTTP_USER_AGENT`	navigateur du client
`REQUEST_METHOD`	méthode utilisée pour appeler la page (`GET`, `HEAD`, `POST`, `PUT`) ;

Variables `PHP`

`HTTP_GET_VARS`	tableau associatif pour la methode get (nom, valeur)
`HTTP_POST_VARS`	tableau associatif pour la methode `post` (nom, valeur)
`HTTP_COOKIE_VARS`	tableau associatif pour la methode `cookie` (nom du `cookie`, contenu)
`GLOBALS`	tableau associatif qui contient les variables globales du *script*, la clé est le nom de la variable (sans le dollar) et la valeur est le contenu de la variable.

Variables dynamiques

Il est possible de définir un nom de variable déclaré et utilisé dynamiquement. Une variable dynamique prend comme nom la valeur d'une autre variable.

- Affectation : `$$nom_var = valeur ;`
- Lecture de la valeur : `$$nom_var`

```
<?php
  $ch = "bonjour";
  $$ch = "tout le monde"; // '$bonjour' devient aussi une variable
  echo "$ch<BR>";
  echo "${$ch}<BR>";
  echo "$bonjour<BR>";
?>
```

donne :

```
bonjour
tout le monde
tout le monde
```

8.3.2 Les constantes

Définition d'une constante

Une constante est définie par l'instruction `define` :

```
define ("NOM_CONST", valeur);
```

On y accède directement par son nom (Ex : `NOM_CONST`. On peut vérifier qu'une constante est définie avec le prédicat `defined` :

```
if ( defined(NOM_CONST) )  ...
```

Exemple

```
<?php
   define("SITE", "http://claude.gueganno.free.fr");
   print("à visiter: ".SITE);
?>
```

Constantes prédéfinies

`TRUE`	vrai
`FALSE`	faux
`__FILE__`	nom du fichier du *script* interprété
`__LINE__`	numéro de la ligne du script
`PHP_VERSION`	version de PHP
`PHP_OS`	système sur lequel PHP exécute le script

```
<?php
  echo PHP_VERSION."<BR>";
  echo PHP_OS."<BR>";
  echo __LINE__."<BR>";
  echo __FILE__."<BR>";
?>
```

8.3.3 Les tableaux

Tableaux à une dimension

Il y a trois possibilités équivalentes pour initialiser un tableau :

```
$tab = array("b","o","n");
$tab[0] = "b"; $tab[1] = "o"; $tab[2] = "n";
$tab[] = "b"; $tab[] = "o"; $tab[] = "n";
```

(pas de déclaration préalable).

L'accès à un élément dans tableau de n cases se fait :

```
$tab[$indice]
```

avec `$indice` $\in [0, n-1]$.

Nombre d'éléments dans un tableau : avec la fonction `count`

```
$nb = count($tab);
```

Exemple :

```
<?php
  echo "essai avec array : ";
  $tab = array("b", "o", "n");
  $taille = count($tab);
  for($i=0; $i<$taille; $i++)
      echo $tab[$i];

  echo "<BR>essai avec [0], [1], ... : ";
  $tab2[0] = "h";  $tab2[1] = "e";  $tab2[2] = "l";
  $tab2[3] = "l";  $tab2[4] = "o";
  $taille = count($tab2);
  for($i=0; $i<$taille; $i++)
      echo $tab2[$i];

  echo "<BR>essai de [] : ";
  $tab[] = "j";  $tab[] = "o";  $tab[] = "u";  $tab[] = "r";
  $taille = count($tab);
  for($i=0; $i<$taille; $i++)
     echo $tab[$i];
?>
```

Sortie du programme :

```
essai avec array : bon
essai avec [0], [1], ... : hello
essai de [] : bonjour
```

Tableaux associatifs

L'accès aux données se fait par un nom (une clé) et non plus par un index.

Initialisations :

1. par éléments :

   ```
   $nomtab["cle"] = valeur;
   ```

2. tout (ou partie) du tableau :

   ```
   $nomtab = array("cle1"=>valeur1,"cle2"=>valeur2,...);
   ```

L'accès à un élément se fait par la clé qui lui est associé :

```
$nomtab["cle"];
```

Exemples de parcours

```
/* avec une boucle */
reset($nomtab);
while ($cle = key($nomtab)){
   $val = pos($nomtab);
   print("$cle = $val<BR>\n");
   next($nomtab);
}
```

```
reset($nomtab);
while (list($cle, $valeur) = each($nomtab)){
   echo "<BR>$cle = $valeur";
}
```

Tableaux de dimension ≥ 2

```
$saison = array(
  "ete"=>array("juin","juillet","aout","septembre"),
  "automne"=>array("septembre","octobre","novembre","decembre"),
  "hiver"=>array("decembre","janvier","fevrier","mars"),
  "printemps"=>array("mars","avril","mai","juin"));

print($saison["printemps"][2]);
```

affiche `mai`.

Fonctions de manipulation des tableaux

`int count(tab)`	nombre d'éléments
`int sizeof(tab)`	nombre d'éléments
`array explode(sep, chaine)`	transforme une chaine en tableau en fonction du séparateur.
`string implode(tab, sep)`	opération inverse
`val max(tab)`	retourne la plus grande valeur d'un tableau
`val min(tab)`	retourne la plus petite valeur du tableau
`shuffle(tab)`	change aléatoirement l'ordre des éléments
`array_walk(tab,'fonc')`	applique la fonction `fonc` au tableau `tab`

Itérateurs

PHP propose quelques fonctions de parcours d'un tableau. Chaque élément du tableau est en fait une paire $\{cl, valeur\}$. Les opérations portent soit sur la valeur, soit sur la clé.

Exemple

```
$T = array( "A", "B", "C", "D", "E");
```

Les clés sont attribuées automatiquement (entiers de 0 à 4).

`current(tab)`	valeur de l'élément courant ; `pos(tab)` fait la même chose.
`tableau each(tab)`	clé et valeur pour l'endroit où est l'itérateur (fait avancer de 1 l'itérateur)
`valeur end(tab)`	place l'itérateur sur le dernier élément du tableau
`clé key(tab)`	index de l'élément courant
`valeur next(tab)`	avance l'itérateur d'une case, retourne sa valeur
`valeur prev(tab)`	recule d'une case et retourne sa valeur
`valeur reset(tab)`	itérateur sur le premier élément du tableau

Exemple

```
<html>
<head>
<title> Tableaux en PHP </title>
</head>

<body>
 <h1>Tableaux</h1>
<?php
$T = array( "A", "B", "C", "D", "E");
echo implode($T,"; ");
echo "<br>";
echo "current(\$T) --> ".current($T)."<br>";
$T2 = each($T);
echo "each(\$T) --> [".$T2[0]." , ".$T2[1]."]<br>";
echo "current(\$T) --> ".current($T)."<br>";
echo "key(\$T) --> ".key($T)."<br>";
echo "next(\$T) --> ".next($T)."<br>";
echo "end(\$T) --> ".end($T)."<br>";
echo "prev(\$T) --> ".prev($T)."<br>";
echo "current(\$T) --> ".current($T)."<br><br>";
?>
</body>
</html>
```

Les tris

On peut les séparer en deux catégories :

1. ceux qui se font sans modification des clés (`arsort`, `ksort`, `asort`, `uasort`)
2. ceux qui effacent les clés.

FIG. 8.1 – *Fonctions de parcours de tableau.*

`arsort(tab)`	trie par ordre décroissant, l'index associé se déplace avec les valeurs
`asort(tab)`	trie par ordre croissant, l'index associé se déplace avec les valeurs
`ksort(tab)`	trie le tableau par ses clés
`ksort(tab)`	trie le tableau en ordre inverse par ses clés
`rsort(tab)`	trie par ordre décroissant, efface les valeurs des clés
`sort(tab)`	trie par ordre croissant, efface la valeur des clés
`uasort(tab, comp)`	trie selon la fonction `comp` (définie par l'utilisateur), conserve la valeur des clés
`usort(tab, comp)`	trie selon la fonction, efface la valeur des clés
`uksort(tab, comp)`	trie les clés selon la fonction `comp` définie par l'utilisateur

```
<html>
<head>
<title> Tableaux en PHP </title>
</head>

<body>
 <h1>Tableaux</h1>

<?php

/* Affiche les couples [clé, valeur] tu tableau $tab */
function affiche($tab) {
    $n = sizeof($tab);
    reset($tab);
```

```
    for ($i=0; $i<$n; $i++) {
        $t = each($tab);
        echo "[".$t[0].",".$t[1]."]; ";
    }
}


$T1 = array( "D", "C", "E", "B", "A");
$T2=$T1;
echo "<b>Valeur initiale du tableau avant chaque appel:</b><br>\n";
echo "\$T2 = ".implode($T2,"; ")."<br>\n";
affiche($T2);
echo "<hr>";

arsort($T2);
echo "arsort(\$T2) --> "; affiche($T2); echo "<br>\n";

$T2 = $T1; asort($T2);
echo "asort(\$T2) --> "; affiche($T2); echo "<br>\n";

$T2 = $T1;   ksort($T2);
echo "ksort(\$T2) --> "; affiche($T2); echo "<br>\n";

$T2 = $T1; sort($T2);
echo "sort(\$T2) --> "; affiche($T2); echo "<br>\n";

$T2 = $T1; rsort($T2);
echo "rsort(\$T2) --> "; affiche($T2); echo "<br>\n";

echo "<hr>\n";
?>
</body>
</html>
```

Fonctions liées à la notion d'ensemble

`tableau array_count_values (tab)`	retourne un tableau associatif qui donne en clé les valeurs du tableau et en valeur le nombre d'apparitions
`tableau array_diff (tab1, tab2, ...)`	retourne un tableau contenant toutes les valeurs de `tab1` qui ne sont pas dans `tab2, ...`
`tableau array_keys (tab)`	retourne un tableau contenant toutes les clés de `tab`.
`tableau array_values (tab)`	retourne un tableau contenant toutes les valeurs de `tab`.
`bool in_array (val, tab)`	retourne *vrai* si la valeur `val` est dans le tableau `tab`

FIG. 8.2 – *Fonctions de tri de tableau.*

Remarque : `tableau array_count_values()` et `tableau array_diff()` à partir de PHP4.

Gestion d'un tableau en pile ou en file

`valeur array_pop (tab)`	retire et retourne le dernier élément du tableau
`array_push (tab, e1, e2, ...)`	empile les éléments `e1`, `e2`, ... à la fin du tableau `tab`
`array_unshift(tab, e1, e2, ...)`	insère les éléments `e1`, `e2`, ... au début du tableau `tab`
`array_shift(tab)`	retire et retourne le premier élément du tableau `tab`

8.4 Les classes

Une classe est une collection de variables (ou attributs) et de fonctions (ou propriétés).

Définition d'une classe. Cela revient à définir un nouveau type. Syntaxe générale :

```
class nomclasse{
   /* attributs */
   var $nomvar_1;
   ...
   var $nomvar_n;

   /* constructeur */
   function nomclasse($arg1=defaut1, $arg2=defaut2, ...){
      instructions;
   }

   /* propriétés */
   function nomfonc_1(){
     instructions;
     return x;
   }
   ...
   function nomfonc_n($arg1=defaut11, $arg2, ...){
     instructions;
   }
}
```

Le constructeur est facultatif. Il permet d'initialiser les attributs de la classe. Il ne peut pas être surchargé, mais on peut préciser des arguments «par défaut», ce qui revient finalement au même.

Création d'une instance de la classe : elle se fait avec l'opérateur `new`.

```
objet = new nomclasse(...);
```

Si le constructeur reçoit des arguments, ils doivent être obligatoirement passés. S'il n'y a pas de constructeur, ou si celui–ci ne reçoit pas d'argument :

```
objet = new nomclasse;
```

Héritage : il se fait avec le mot-clé `extends`. Le constructeur de la classe parent n'est pas appelé systématiquement. Cependant, si la classe dérivée n'a pas de constructeur, c'est celui de la classe parent qui est appelée.

Exemple

```
<html>
<head>
<title> Classes </title>
</head>
```

```
<body>
 <h1>Classes</h1>

<?php
/* --------- classe pile ----------------- */
class pile {
    /* attributs */
    var $N;
    var $tab;
    var $nom;

    /* constructeur */
    function pile($nom="P"){
        $this->N=0;
        $this->nom = $nom;
        echo "Création d'une pile<br>";
    }

    /* propriétés */
    function ajoute($x){
        $this->tab[$this->N] = $x;
        $this->N++;
    }

    function retire() {
        $this->N--;
        return $this->tab[$this->N];
    }
}

/* --------- classe pile2 ---------------- */
class pile2 extends pile {

    function affiche() {
        echo $this->nom." --> ";
        for ($i=0; $i<$this->N; $i++)
            echo $this->tab[$i]."; ";
    }
}

/* -------- programme PHP ---------------- */
$P = new pile2("cours PHP");
$P = new pile2;
$P->ajoute("php");
$P->ajoute("java");
$P->affiche(); echo "<br>";
echo $P->retire()."<br>";
$P->affiche(); echo "<br>";
?>

</body>
```

```
</html>
```

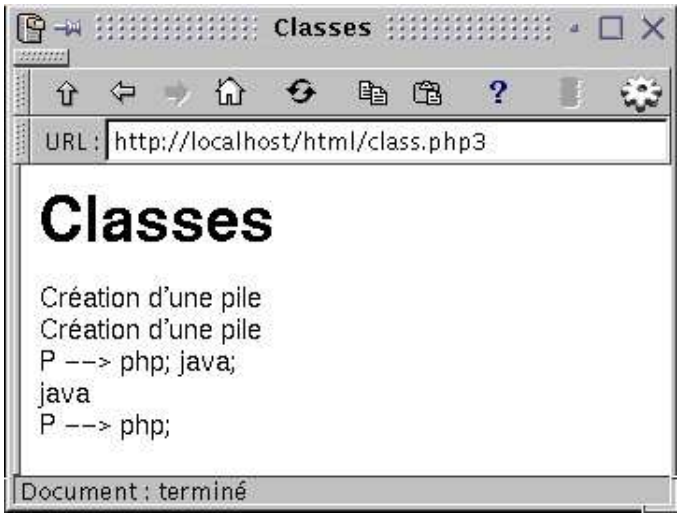


FIG. 8.3 – *Exemple sur les classes.*

8.5 Expressions et instructions

8.5.1 Les opérateurs

Opérateurs arithmétiques

`+`	addition	binaires
`-`	soustraction	|
`*`	multiplication	|
`/`	division	|
`%`	modulo	|
`++`	incrémentation	unaires
`--`	décrémentation	|

Opérateurs logiques

`&& and`	et	binaires
`|| or`	ou	|
`! not`	négation	unaire

Opérateurs sur les *bits*

`&`	et `(1 & 1)` $\rightarrow 1$
`|`	ou `(1 | 0)` $\rightarrow 1$
`^`	ou exclusif `(1 ^ 1)` $\rightarrow 0$
`~`	non

Opérateurs d'affectation

signe	utilisation	équivalent
`=`	`$x = $y`	`$x = $y`
`+=`	`$x += $y`	`$x = $x + $y`
`-=`	`$x -= $y`	`$x = $x - $y`
`*=`	`$x *= $y`	`$x = $x * $y`
`/=`	`$x /= $y`	`$x = $x / $y`
`%=`	`$x %= $y`	`$x = $x % $y`
`++`	`$x++`	`$x = $x + 1`
`--`	`$x--`	`$x = $x - 1`

Opérateurs relationnels

`$x == $y`	vrai si `$x = $y`
`$x != $y`	vrai si `$x` différent de `$y`
`$x <= $y`	vrai si `$x` inférieur ou égal à `$y`
`$x >= $y`	vrai si `$x` supérieur ou égal à `$y`
`$x > $y`	vrai si `$x` supérieur à `$y`
`$x < $y`	vrai si `$x` inférieur à `$y`
`$x === $y`	PHP4 : vrai si `$x` = `$y` et qu'ils sont du même type
`$x !== $y`	PHP4 : vrai si `$x` différent de `$y` ou qu'ils n'ont pas le même type

8.5.2 Les structures de commande

Convention pour les valeurs booléennes

- `false` → 0
- `true` → tout le reste.

Structure conditionnelle `if`

Si l'expression est vraie exécuter l'instruction ou les instructions dans le bloc.

```
if (expression){
  instruction1;
  ...
  instructionN;
}
```

Les accolades sont facultatives lorsque le bloc ne contient qu'une seule instruction :

```
if (expression)  instruction1;
```

Cette remarque reste vraie pour toutes les structures qui suivent.

Structure conditionnelle `if ...else`

Si l'expression est vraie exécuter les instructions du bloc 1 sinon exécuter celles du bloc 2.

```
if (expression){
  instruction1;
  ...
}
else {
  instruction1;
  ...
}
```

Cette structure peut être remplacée par l'opérateur ternaire `? :`

```
condition ? instruction si true : instruction si false;
```

Structure conditionnelle `if ...elseif ...else`

Si l'expression est vraie exécuter le bloc 1, sinon, si l'expression 2 est vraie exécuter le bloc 2, sinon si ..., sinon exécuter le dernier bloc.

```
if (expression){
  instruction1;
  ...
}
elseif (expression2){
  instruction2;
  ...
}
elseif (expression3){
```

```
  instruction3;
  ...
}
else {
  instructionN
  ...
}
```

Structure conditionnelle `switch`

Selon l'expression exécuter des instructions.

```
switch (expression){
   case val1 : instructions_1; break;
   case val2 : instructions_2; break;
   ...
   default : instructions_N;
}
```

Les valeurs ne peuvent pas être des tableaux ou des objets.

Structure répétitive `while`

Tant que l'expression est vraie faire les instructions

```
while (expression) {
   instructions;
}
```

Structure répétitive `do ... while`

Faire les instructions tant que l'expression est vraie

```
do{
  instructions;
}
while (expression);
```

Structure répétitive `for`

C'est une variante du `while`

```
for (instruction1; expression; instruction2){
   instructions;
}
```

Cette écriture équivaut à :

```
instruction1;
while (expression) {
   instructions;
   instruction2;
}
```

Structure répétitive `foreach`

Permet d'appliquer un bloc d'instructions *pour chaque* élément d'un tableau (PHP4 uniquement).

```
foreach($tab as $value){
   instructions;
}
```

Pour tout le tableau `$tab`, à chaque itération la valeur de l'élément courant est assignée à la variable `$value` et le pointeur sur le tableau est avancé d'une case. Au départ le pointeur sur le tableau est automatiquement mis sur la première case par le `foreach`.

Exemple :

```
$tab = array(1,2,3,4);
foreach($tab as $v){
   $v++;
}
```

Interruption de boucles

- `break` interrompt les boucles `for`, `while`.
- `continue` interrompt l'exécution d'une itération et reprend à l'itération suivante.
- `exit` interrompt le script

8.5.3 Les fonctions

8.5.4 Définition d'une fonction

Il faut utiliser le mot–clé `function` :

```
function foo ($arg_1, $arg_2, ..., $arg_n) {
   instructions;
   return $retour;
}
```

Une fonction doit être définie avant d'être utilisée.

Passage des arguments

```
function foo ($arg_1, &$arg_2, $arg3="defaut", ... ) {
   ...
}
```

dans cet exemple,

- `$arg1` est passé par valeur ;
- `$arg2` est passé par référence ; si sa valeur est modifiée dans la fonction, elle est modifiée pour tout le programme ;
- `$arg2` est optionnel, il peut ne pas être passé lors de l'appel de la fonction puisqu'il a une valeur par défaut.

Exemple Le fichier suivant contient un fonction chargée de dessiner un échiquier carré dont le nombre de lignes est passé en argument. La valeur par défaut de cet argument est 4. Le rendu de la page est donné par la figure 8.4.

```
<html>
<head>
<title> Fonctions PHP </title>
</head>

<body>
 <h1>Fonctions</h1>

<?php
function damier($n=4){
    echo "<table border=\"1\">\n";
    $couleur1 = 0;
    $i = 0;
    while ($i<$n) {
        $couleur1 = 1 - $couleur1;
        $couleur2 = $couleur1;
        echo "<tr>";
        for ($j=0; $j<$n; $j++) {
            echo "<td bgcolor=\"".($couleur2?"white":"black")."\">";
            echo "   </td>";
            $couleur2 = 1 - $couleur2;
        }
        echo "</tr>\n";
        $i++;
    }
    echo "</table>";
}

echo "<center>\n";
damier();
echo "<br>";
damier(7);
echo "</center>\n";
?>

</body>
</html>
```

8.5.5 Inclusion de fichiers

Inclusion avec `include`

L'instruction `include` réalise l'inclusion et l'évaluation d'un fichier. Le fichier est réévalué à chaque nouvelle inclusion.

Exemple :

```
$files = array ('first.inc', 'second.inc', 'third.inc');
```

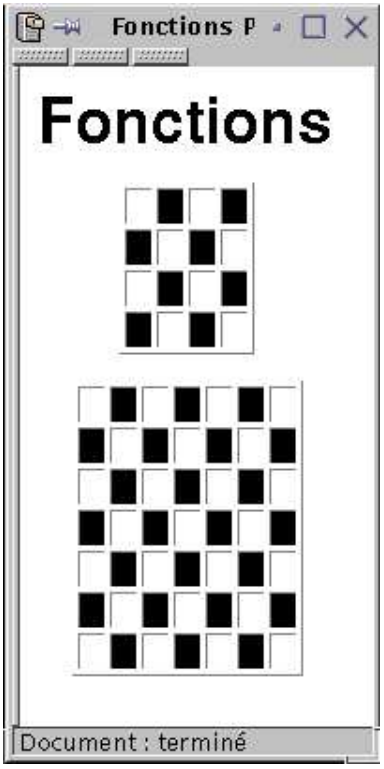


FIG. 8.4 – *Exemple de fonction.*

```
for ($i = 0; $i < count($files); $i++) {
   include $files[$i];
}
```

Attention à l'utilisation conditionnelle de cette instruction : puisqu'elle est remplacée par les instructions du fichier, il faut qu'elle figure dans un bloc d'instructions (marqué par des accolades).

```
/* Mauvaise écriture */
if ($condition) include($fichier);
else include($autreFichier);
```

```
/* Bonne écriture */
if ($condition) { include($fichier); }
else { include($autreFichier);}
```

Inclusion avec `require`

Cette instruction est semblable à `include`, à ceci près que le code est inséré même s'il n'est pas exécuté. Il faut donc éviter d'utiliser le `require` dans une structure conditionnelle.

8.6 Envoi de données vers un script

8.6.1 Utilisation de la fenêtre «URL» du navigateur

Il suffit de compléter l'URL avec la liste des paramètres séparés par le caractère `&`. Un `?` sépare le nom du fichier de la liste de paramètres.

Exemple :

```
http://localhost/html/recdata.php3?N=6&nom=Zaza&X=3.14
```

8.6.2 Appel à partir d'un lien HTML

La même méthode fonctionne en général. Il est cependant conseillé de remplacer le `&` par son expression HTML `&`.

Exemple

```
<html>
<head><title>Envoi de données</title></head>
<body>
 <h3>Envoi de données</h3>
 <a href="recdata.php3?N=99&amp;nom=Zaza&amp;X=3.1415">
    Cliquer ici!
 </a>
</body>
</html>
```

8.6.3 Récupération des données

On fait précéder, dans le script, le nom des paramètres par un `$` pour en faire des variables PHP.

Exemple

```
<html>
<head>
  <title>Réception de données</title>
</head>
<body>
 <h3>Réception de données</h3>
<?php
echo "N = ".$N."<br>";
echo "Nom = ".$nom."<br>";
echo "X = ".$X."<br>";
?>
</body>
</html>
```

FIG. 8.5 – *Envoi et réception de données*

8.7 Données d'un formulaire

Les données d'un formulaire sont directement accessibles par le nom affecté à chaque variable dans le fichier HTML. Il suffit, dans le script d'ajouter le caractère `'$'` devant ce nom.

Exemple : le formulaire suivant met en œuvre les principaux éléments d'un formulaire.

```
<html>
  <head>
    <title>Formulaires / PHP</title>
  </head>

<body>
<form ENCTYPE="multipart/form-data"  action="traiteform.php3" method="post">
  <select name="civilite" >
    <option selected>Monsieur</option>
    <option>Madame</option>
    <option>Mademoiselle</option>
  </select>
   Nom : <input type="text" size="16" maxlength="32" name="nom" value="">
  <hr>
  Langue :
  <select name="langue[]"  size=3 multiple>
        <option selected value="fr">Français</option>
        <option value="en">Anglais</option>
        <option value="de">Allemand</option>
  </select> <br>
  <input type="hidden" name="N" value="5">
  <br>
  Langage :
```

```
 <input type="checkbox" checked name="js"> javascript   
 <input type="checkbox" name="php"> php   
 <input type="checkbox" name="java"> java
<br>
Système :
 <input type="radio" name="syst" checked value="unix"> UNIX
                 ou
 <input type="radio" name="syst" value="windows"> Windows
<hr>
Mot de passe: <input type="password" size="8" maxlength="8"
                     name="mdp" value="">
<br>
Message : <textarea name="remarque" rows="3" cols="30"
                    wrap="virtual">&Eacute;crivez ici !</textarea>
          <hr>
<input type="submit" value="Valider">
</form>
</body>
</html>
```

Le rendu de la page est donné par la figure 8.6.

Les données du formulaire sont postées vers le fichier de traitement *traiteform.php*3. Le traitement consiste simplement ici à afficher les variables reçues sur le serveur.

```
<html>
<head>
<title>Formulaire / PHP</title>
<style>
td {text-align:center}
h1 {font-size:200%}
</style>
</head>

<body bgcolor="white">
<ol>
<?php
echo "<li>Civilité: ".$civilite;
echo "<li>Nom: ".$nom;
echo "<li>Langue : ".$langue[0]."; ".$langue[1]."; ".$langue[2];
echo "<li>N = ".$N;
echo "<li>javascript : ".$js;
echo "<li>PHP : ".$php;
echo "<li>Java : ".$java;
echo "<li>Système : ".$syst;
echo "<li>Mot de passe : ".$mdp;
echo "<li>Remarques : ".$remarque;
?>
<hr>
</body>
</html>
```

L'exécution du script produit le résultat de la figure 8.7.

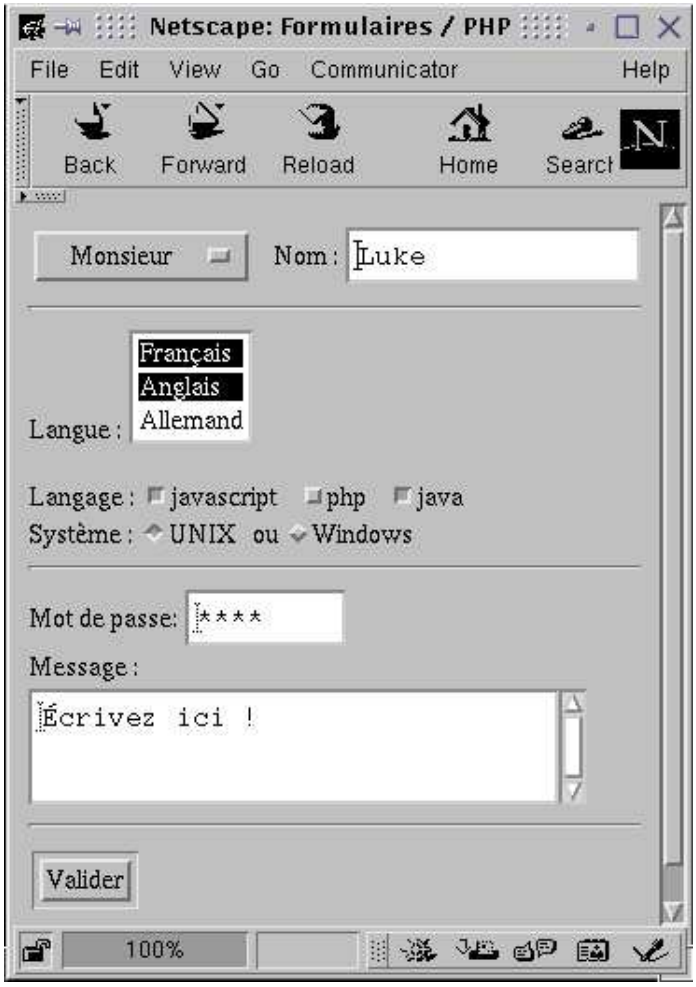


FIG. 8.6 – *Formulaire*

8.8 Les `cookies`

Les *cookies* sont de petites informations à durée de vie limitée qu'un serveur peut écrire sur le disque d'un client. Avec `netscape`, les *cookies* sont dans *~/.netscape/cookies*, avec `windows`, ils figurent dans
`c:\windows\cookies\'nom utilisateur'.txt`

8.8.1 Envoyer un *cookie*

Avec la fonction `setcookie` :

```
int setcookie(string nom, string valeur, int date_expiration,
              string chemin, string domaine, int securite);
```

Les *cookies* doivent être envoyés avant l'en-tête, donc, la fonction `setcookie()` doit être appelée en début de fichier, avant la balise `<html>`.

FIG. 8.7 – *Traitement du formulaire*

Tous les arguments sont optionnels, sauf le premier. Le domaine permet de limiter la visibilité à un site, le chemin d'accès restreint la visibilité à une partie de l'arborescence . Si la date d'expiration n'est pas précisée, le *cookie* disparaît à la fin de la session.

Lors d'une connexion à un site, le navigateur envoie automatiquement les cookies qui concernent le site (en fonction du domaine et du chemin d'accès). En `PHP` il est possible de connaître la liste des cookies et de leurs valeurs avec le tableau associatif `$HTTP_COOKIE_VARS`. Chaque cookie est accessible avec son nom comme nom de variable.

Exemple

```
<?php
setcookie("TestCookie","Valeur de test",time()+36000);
?>
<html>
<head>
<title> Cookies en PHP </title>
</head>
<body>
 <h1>Cookies</h1>

<?php
 echo "Valeur du cookie =".$HTTP_COOKIE_VARS["TestCookie"];
?>
</body>
</html>
```

Dans cet exemple, la valeur du *cookie* ne s'affiche que si le *cookie* a déjà été déposé lors d'une session antérieure.

La valeur ne doit pas occuper plue de 4096 caractères.

8.9 Envoi d'un *mail*

C'est possible en PHP avec la fonction `mail`. Cette fonction est parfois désactivée chez certains hébergeurs.

```
<html>
<head>
<title>essai mail</title>
</head>
<body>
 <h4>Envoi de mail avec PHP</h4>
<?php
$dest = "claude.gueganno@free.fr"; /* destinataire */
$sujet = "essai de mail"; /* sujet */
$contenu = "Ce message a été envoyé automatiquement...\n";
$enTete = "From: claude.gueganno@univ-ubs.fr\n";
mail($dest, $mail, $sujet, $contenu, $enTete);
?>
</body>
</html>
```

8.10 Envoi de fichier au serveur

```
<!-- fichier envF.php3 -->
<html>
<head>
<title>envoi de fichier au serveur</title>
</head>
<body>

<?php
if(isset($fp)){ // faux au premier appel
  unlink($fp);  // efface l'ancien fichier
  echo "pointeur  : ".$fp." <br>";
  echo "nom : ".$fp_name." <br>";
  echo "taille : ".$fp_size." <br>";
  echo "type : .".$fp_type." <br>\n";
  echo "<hr>\n";
}
?>
<form enctype="multipart/form-data"
   action = "envF.php3"
   method = "POST">
   <input type="hidden" name="MAX_FILE_SIZE" value="1000">
   <input name="fp" type="file"><br>
   <input type="submit" value="<<Envoyer>>">
</form>
```

```
</body>
</html>
```

Cet exemple intègre en un seul fichierle formulaire d'envoi et la réception des paramètres du fichier. Une trace d'exécution est donnée par la figure 8.8.

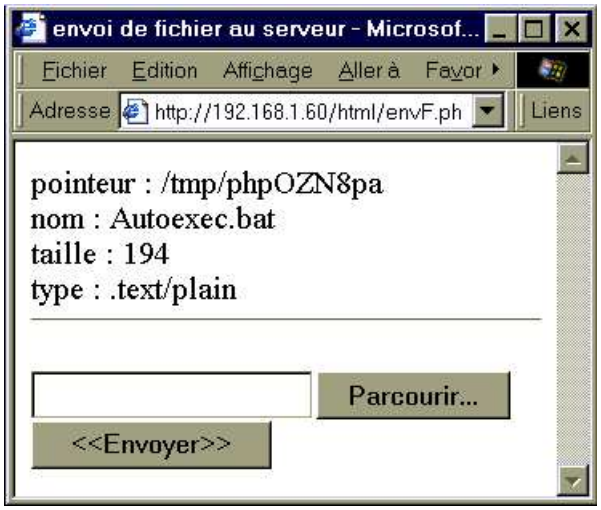


FIG. 8.8 – *Envoi de fichier vers le serveur*

Pour récupérer l'information et la sauver sur le disque du serveur, il suffit d'utiliser les fonctions normales pour les fichiers, en commençant par :

```
// ouverture du fichier source (client) en lecture
 $fi = fopen($fp, "r");
// ouverture d'un fichier sur le serveur
// ici, avec le même nom que le fichier du client
 $fo = fopen($fp_name, "w");
// etc.
```

Chapitre 9

Bibliothèques de fonctions PHP

Ce chapitre ne développe que les fonctions les plus communément utilisées.

9.1 Communication avec le client

9.1.1 Envoi de données vers le navigateur

Ces fonctions qui rapellent les fonctions classiques d'entrées/sorties standards on pour but d'envoyer le code HTML vers le navigateur.

`echo` : envoi de chaînes et de paramètres :

Exemple : `echo "<b>".$i."</b>";`
Le `.` permet de concaténer les différents éléments (constantes ou variables). Il peut être évité ; ainsi, le code

```
<?php
$nom = "Zaza";
$age = 99;
echo "nom : $nom ; age : $age";
?>
```

affiche `nom : Zaza ; age : 99`

`print` : envoi de chaînes :

Exemple : `echo "<h1>PHP</h1>";`

`printf` : envoi de chaînes formatées :

Exemple :

```
printf("Nom : %20s -- Age : %d<br>\n", $nom, $age);
```

Dans cet exemple , `%20s` et `%d` sont des formats associés aux variables respectives `$nom` (chaîne de caractères) et `$age` (entier).

Formats :

`d`	entier décimal
`c`	caractère ASCII
`o`	entier octal
`s`	chaîne
`x`	entier hexadécimal (lettres en minuscules)
`X`	entier hexadécimal (lettres en majuscules)
`f`	double
`b`	entier binaire
`e`	double en notation scientifique

Un format peut contenir des informations optionnelles :
- `-` pour justifier à gauche
- un nombre pour fixer le nombre de caractères à afficher
  Exemples : `%-5d` affiche un entier sur 5 caractères justifiés à gauche. `2.3f` pour un réel fixe à 2 le nombre de chiffres affichés avant la virgule et à 3 ceux qui sont après.

9.1.2 Envoi de données contenues dans un fichier

Voir le paragraphe 9.4.4 page 118.

9.2 Les chaînes de caractères

9.2.1 Séquences d'échappement

Elles permettent d'afficher les caractères spéciaux ou réservés :

`\n`	retour à la ligne
`\t`	tabulation
`\\`	caractère \
`\$`	caractère $
`\"`	caractère "

Exemples :
- `echo "<body bgcolor=\"red\">"` → `<body bgcolor="red">`
- `echo "prix : 4\$"` → `prix : 4 $`

`eval(chaîne)`	évalue la chaîne comme si c'etait du code PHP
`entier strlen(ch)`	retourne la longueur de *ch*
`tab count_chars(ch)`	retourne le nombre d'occurrences de chaque caractère de la chaîne (PHP4)
`chaîne strtok(ch, sep)`	separe *ch* en fonction de *sep*
`chaîne quotemeta(ch)`	retourne la chaîne avec un \ devant les caractères `. \ * ? [ ^ ] ( $ )`
`chaîne str_repeat (ch, n)`	répète *n* fois la chaîne (PHP4)

Exemple avec `strtok` , cette fonction permet de parcourir une chaîne en la découpant en sous-chaînes selon les occurences d'un séparateur. Dans l'exemple suivant, on choisit un découpage en mots. le séparateur est donc l'espace.

```
<html>
<head>
<title> Chaînes en PHP </title>
</head>

<body>
 <h1>Chaînes</h1>
<?php
$ch = "Ceci est une chaîne exemple";
echo "Chaîne : ".$ch."<br>";
echo "<b>strok</b><br>";
$sep = " ";
$mot = strtok($ch,$sep);
while($mot) { /* tant qu'il y a qq chose ... */
    echo "Mot = ".$mot."<br>";
    $mot = strtok($sep);
}
?>
</body>
</html>
```

Au premier appel, la fonction `strtok` est appelée avec 2 arguments. Ensuite, on utilise seulement le séparateur pour parcourir la chaîne. Le résultat du script est donné par la figure 9.1.

FIG. 9.1 – *Fonction* `strtok`.

9.2.2 Fonctions de conversion

`chaîne sprintf(format, arg)`	retourne une chaîne formatée (voir `printf`)
`chaîne chr(entier)`	donne le caractère qui correspond au code ASCII
`int ord(c)`	donne le code ASCII qui correspond au caractère c

9.2.3 Recherche de caractères et sous-chaînes

`entier strpos(ch1, ch2)`	retourne la position de la première occurrence de $ch2$ dans $ch1$
`entier strrpos(ch, car)`	retourne la position de la dernière occurrence du caractère
`entier strspn(ch1, ch2)`	longueur de la sous-chaîne de $ch1$ dont les caractères sont entièrement dans $ch2$
`chaîne strstr(ch1, ch2)`	retourne la portion de $ch1$ à partir de la première occurrence de $ch2$ et jusqu'à la fin
`chaîne stristr(ch1, ch2)`	`strstr` non sensible à la casse
`chaîne substr(ch, i, n)`	renvoie la sous-chaîne de taille n qui commence à l'indice i

9.2.4 Comparaison alphabétique

`entier strcmp(ch1, ch2)`	retourne 0 si $ch1 = ch2$, un nombre négatif si $ch1 < ch2$, un nombre positif si $ch1 > ch2$
`entier strcasecmp(ch1, ch2)`	comme `strcmp` sans tenir compte de la casse

9.2.5 Majuscules/minuscules

`chaîne strtolower(ch)`	convertit en minuscules
`chaîne strtoupper(ch)`	convertit en majuscules
`chaîne ucfirst(ch)`	met la première lettre en majuscule
`chaîne ucwords(ch)`	met tous les mots de ch en majuscules

9.2.6 Espaces

`chaîne chop(ch)`	retourne la chaîne sans les espaces
`chaîne trim(ch)`	supprime les espaces de début et fin de chaîne

9.2.7 Fonctions spéciales HTML

`chaîne htmlspecialchars (ch)`	convertit les caractères spéciaux en entités html
	`<` → `<`
	`>` → `>`
	`&` → `&`
	`"` → `&quot;`

`chaîne htmlentities(ch)`	comme la fonction précédente, sauf que tous les caractères ayant une traduction `HTML` sont convertis
`chaîne nl2br(ch)`	ajoute ` ` devant chaque nouvelle ligne du texte
`parse_str(requete)`	analyse la requête comme si elle venait d'un formulaire posté par `get` (crée les variables et leur valeur)

9.2.8 Expressions rationnelles

Caractères spéciaux à utiliser

`|`	ou
`*`	0 ou plus
`+`	au moins une fois
`?`	0 ou une fois
`{`n`}`	n fois exactement
`{`n`,}`	n fois ou plus
`{`n`,`m`}`	au moins n et au plus m
`.`	un caractère quelconque
`^`	correspondance au début
`$`	correspondance en fin
`[a-z]`	tout caractère minuscule
`[ab]`	a ou b
`[^ab]`	tout sauf a et b
`[:alpha:]`	type de caractère (`alnum`, `blank`, `digit`, `punct`)
`[:<:]`c	mot commence par c
`[:>:]`c	mot finit par c

Les caractères spéciaux (`^ .[]()|?{}\`) utilisés comme données s'obtiennent en ajoutant un \ devant.

Fonctions

`booléen ereg(exp,ch,tabocc)`	évalue l'expression et met les occurrences rencontrées dans *ch* dans le tableau *tabocc* la case 0 contient l'occurrence de l'expression, la case 1 la sous-chaîne. Retourne 0 s'il n'y a pas de correspondance
`chaîne ereg_replace(exp,rep,ch)`	remplace les sous-chaînes de *ch* qui correspondent à l'expression *exp* par la chaîne *rep*. *exp* peut être un code *ascii* **Exemple :** `ereg_replace(10, " ", ch)` remplace les `\n` par des ` `
`booléen eregi(exp,ch,tab)`	comme `ereg` sans tenir compte de la casse
`booléen eregi_replace(exp,rep,ch)`	comme `ereg_replace` sans tenir compte de la casse
`tableau split(exp,ch,limite)`	retourne un tableau des sous-chaînes

```
<html>
<head>
<title> ereg </title>
</head>

<body>
 <h3>Expressions rationnelles</h3>

<?php
 echo "Chaîne originale : ".$HTTP_USER_AGENT."<br>\n";
 ereg("^([A-Za-z]+)/.*$", $HTTP_USER_AGENT, $T);
 echo "navigateur = ".$T[1]."<br>";
?>
</body>
</html>
```

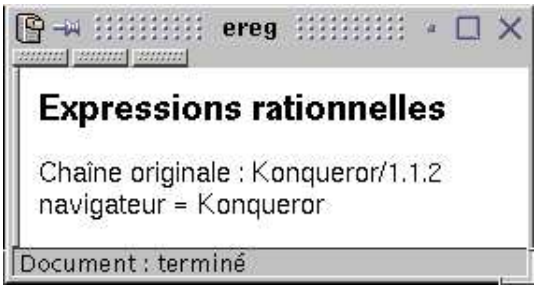


FIG. 9.2 – *Fonction* `ereg` *exécution avec Konqueror.*

FIG. 9.3 – *Fonction* `ereg` *exécution avec Netscape.*

9.3 Fonctions mathématiques

Trigonométrie

`décimal acos(x)`	arc cosinus
`décimal asin(x)`	arc sinus
`décimal atan(x)`	arc tangente
`décimal cos(x)`	cosinus
`décimal sin(x)`	sinus
`décimal tan(x)`	tangente de l'angle x

Autres fonctions

`nombre abs(x)`	valeur absolue ($\|x\|$)
`décimal exp(x)`	exponentielle (e^x)
`décimal log(x)`	logarithme
`décimal pi()`	retourne la valeur de π
`décimal pow(x, y)`	élève x à la puissance y (x^y)
`décimal sqrt(x)`	racine carrée de x ($\sqrt{x}$)

Conversions

`chaîne decbin(entier)`	retourne la représentation binaire de `entier`
`entier bindec(chaîne)`	retourne la représentation décimale de la chaîne binaire
`chaîne dechex(entier)`	retourne la représentation hexadécimale de `entier`
`chaîne decoct(entier)`	retourne la représentation octale de *entier*
`décimal deg2rad(angle)`	conversion degré → radian
`entier hexdec(chaînehex)`	conversion chaîne hexadécimale en un entier
`décimal octdec(chaîneoct)`	conversion chaîne octale en un entier
`décimal rad2deg(angle)`	conversion radian → degré

Arrondis

`nombre abs(x)`	valeur absolue
`entier ceil(x)`	retourne le plus petit entier $> x$
`entier floor(x)`	partie entière
`entier round(val)`	arrondi à l'entier le plus proche

nombres aléatoires

`entier getrandmax()`	renvoie le nombre aléatoire maximum pour `rand`
`entier rand(inf, sup)`	renvoie un nombre aléatoire compris entre les entiers inf et sup
`entier srand(entier)`	pour initialiser le générateur de nombres aléatoires

9.4 Les fichiers

9.4.1 Ouverture, fermeture

Ouverture

```
$f = fopen("nom","mode");
```

La fonction renvoie un pointeur sur le fichier. En cas d'échec, la valeur 0 est retournée. Cette fonction permet aussi d'établir une connexion `HTTP` ou `FTP`.

Exemples :

```
$fp = fopen("/home/rasmus/file.txt", "r");
$fp = fopen("http://www.php.net/", "r");
$fp = fopen("ftp://user:password@example.com/", "w");
```

Sous `Windows`, le séparateur \ doit être doublé (caractère spécial).

```
$fp = fopen("c:\\data\\info.txt", "r");
```

Modes d'ouvertures :

`"r"`	lecture
`"w"`	écriture
`"a"`	ajout
`"r+"`	lecture et écriture
`"w+"`	lecture et écriture, supprime le contenu précédent
`"a+"`	lecture et écriture en fin de document

Remarque : pour un fichier binaire il faut ajouter après le mode un `b`, par exemple, `"rb"` ...

Traitement des ereurs

```
if($fp == 0){
  echo "erreur d'ouverture du fichier";
  exit; /* fin du script */
}
```

Ouverture et stockage dans un tableau : ces 2 opérations sont enchainées par la fonction `file`. Chaque élément du tableau est une ligne du fichier terminée par `'\n'`.

```
$ft = file("nom.txt");
for($i = 0; $i < count($ft); $i++) {
   print($ft[$i]);
}
```

Fermeture

```
fclose($fp);
```

9.4.2 Lecture et écriture

`booléen feof(fp)`	Renvoie 0 lorsque la fin de fichier est atteinte
`chaîne fgets(fp, n)`	Lecture de $n-1$ caractères dans le fichier fp. La lecture s'arrête à la fin de fichier ou à la fin de ligne ou à $n-1$ caractères.
`car fgetc(f)`	Lecture d'un caractère
`fputs(fp,str)`	Écriture dans le fichier fp de la chaîne str.
`fputc(fp,c)`	Écriture du caractère c

9.4.3 Manipulation de fichiers et répertoires

Modifications (nom et droits d'accès)

`booléen chgrp("nomfich","nomgroupe")`	change le groupe d'appartenance
`booléen chmod("nomfich",mode)`	change les droits d'accès
`booléen chown("nomfich","propriétaire")`	change le propriétaire.
`int rename("nomfich1", "nomfich2")`	change le nom du fichier.

Informations sur les attributs des fichiers

`chaîne basename(chemin)`	retourne le nom de fichier d'un chemin
`entier fileatime("nomfich")`	temps écoulé depuis le dernier accès au fichier.
`entier filesize("nomfich")`	taille en octets du fichier.
`booléen file_exists("nomfich")`	1 si le fichier existe.
`booléen is_file("nomfich")`	1 si nomfich est un fichier.
`booléen is_executable("nomfich")`	1 si fichier existe et est en mode exécutable.
`booléen is_readable("nomfich")`	1 si fichier existe et est en mode lecture.
`booléen is_writeable("nomfich")`	1 si fichier existe et est en mode écriture.
`chaîne tempnam(rep, ch)`	création d'un fichier temporaire unique dans le répertoire rep. Le préfixe est optionnel, retourne le nom du fichier temporaire.

Gestion des répertoires

`chaîne dirname("chemin")`	retourne le répertoire d'un chemin
`booléen is_dir("nom")`	1 si nom est un répertoire.
`booléen chdir("repertoire")`	change de répertoire, renvoie 1 si ok.
`closedir(pointeur)`	ferme un répertoire ouvert par `opendir`
`booléen copy("source","destination")`	copie d'un fichier vers un répertoire.
`booléen mkdir("nomrep")`	crée un nouveau répertoire
`entier opendir("nomrep")`	ouvre un répertoire.
`booléen rmdir("nomrep")`	supprime le répertoire.
`chaîne readdir(pointeur)`	pour lire un élément du répertoire (retourne un nom de fichier ou de répertoire). Le pointeur est retourné par `opendir`.
`rewinddir(pointeur)`	revenir au début du répertoire.

Exemple d'affichage de répertoires .

```
<html>
<head>
<title> readdir </title>
</head>

<body>
<h3>Répertoire /</h3>
<?php
$rp = opendir("/");
while($x = readdir($rp)) echo $x." ";
closedir($rp);
?>
</body>
</html>
```

L'exécution du script est donné par la figure 9.4. C'est bien la racine du système qui est prise en compte et non la racine du site *web*. Dans le cas d'un service *internet*, il est donc important de soigner la sécurité des fichiers et des répertoires.

FIG. 9.4 – *Fonction* `readdir`.

9.4.4 Envoi de fichiers vers le navigateur

`fpassthru` : le contenu du fichier est envoyé vers le navigateur en une seule instruction.

```
$fp = fopen("fich.html", "r");
if ($fp) {
  fpassthru($fp);
  fclose($fp);
}
```

`readfile` : Lit le fichier et l'envoie au navigateur, retourne le nombre d'octets lus.

```
readfile("fich.html");
```

L'ouverture du fichier, sa lecture et l'envoi vers la sortie standard (le navigateur) son enchaînés. Comme pour `fopen`, le nom de fichier peut commencer par `http://` ou `ftp://`.

9.5 Réseau

9.5.1 Fonction `fsockopen`

Cette fonction permet d'établir une connexion avec un serveur sur un port donné.

```
$fp  = fsockopen($serveur, $port ,[&$errno, &$errstr, $timeout]);
```

Les trois derniers paramètres sont optionnels.

Exemple : connexion sur un serveur `POP`. Le numéro de port attribué au service `POP` est le 110. Le protocole est relativement simple (identification avec les requètes `USER` et `PASS`, puis interrogation du serveur).

Dans l'exemple, on réalise une connexion avec le serveur, puis on lui demande le nombre de messages en attente (requète `STAT`).

Les paramètres et les fonctions utiles ont été encapsulées dans une classe `message` :

```
/* Fichier 'message.php' */
<?php
class message {
    var $server; // nom du serveur
    var $user;   // utilisateur
    var $pass;   // mot de passe
    var $stream; // pointeur pour la communication

    // Constructeur
    function message($s,$u,$p){
        $this->server=$s;
        $this->user=$u;
        $this->pass=$p;
    }

    /* ouverture du port */
    function open(){
        $this->stream  = fsockopen($this->server, 110,&$errno, &$errstr, 30);
        if(!$this->stream) return 0; else return 1;
    }

    /* Connexion => lecture de l'en-tête + user + password */
    function connect(){
        /* Lecture en-tête */
        $c = fgetc($this->stream);
        $b = $c.fgets($this->stream, 128);
        echo "<br> b = ".$b."<br>";
        if (!strpos($b, "OK")) return false;
        /* Nom d'utilisateur */
        fputs($this->stream,"USER ".$this->user."\n");
        $b = fgets($this->stream, 128);
        echo $b."<br>";
        if (!strpos($b, "OK")) return false;
        /* mot de passe */
```

```
        fputs($this->stream,"PASS ".$this->pass."\n");
        $b = fgets($this->stream, 128);
        if (strpos($b, "OK") != 1) return false;
        echo $b."<br>";
        return true;
    }

    /* Nombre de messages sur le serveur */
    function stat() {
        fputs($this->stream,"STAT\n");
        $b = fgets($this->stream, 128);
        if (strpos($b, "OK") != 1) return false;
        return $b;
    }

    function close(){
        fputs($this->stream,"QUIT");
        fclose($this->stream);
    }
}
?>
```

L'utilisation de la classe est donnée dans le script principal :

```
<html>
<body>
<h3>fsockopen</h3>
<?php
require('message.php'); // classe pour 'pop'
echo "<h2>Local</h2>";
$user = "   ";   // à remplir
$motdepasse = "    "; // à remplir
$M = new message("pop.free.fr",$user,$motdepasse);
echo "open:".$M->open()."<br>";
echo "connect:".$M->connect()."<br>";
echo "nombre de message(s) = ".$M->stat();
$M->close();
?>
</body>
</html>
```

9.5.2 Fonction `header`

Cette fonction sert à modifier l'en-tête `HTTP` du fichier `HTML` envoyé vers le navigateur. Les appels doivent être faits avant les sorties `html`.

Redirection d'adresse. La modification du champ `Location` de l'en-tête permet de rediriger le navigateur vers un autre site.

```
header("Location: http://www.autresite.fr");  /* Redirection du navigateur */
exit;  /* Rien ici après la redirection */
```

Désactivation des caches. Lorsqu'un script PHP génère du code html qui ne doit pas être pris en compte par le cache d'un navigateur ou d'un *proxy*, il faut également intervenir sur l'en-tête HTTP du fichier :

```
/* Expires -> on donne une date du passé ... */
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
/* Last-Modified -> modifié à l'instant
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
header("Cache-Control: no-cache, must-revalidate");  // pour HTTP/1.1
header("Pragma: no-cache");                          // pour HTTP/1.0
```

Avec un tel en-tête, le fichier sera automatiquement rechargé par le navigateur et/ou les serveurs *proxies* intermédiaires entre le serveur et le navigateur.

Utilisation pour le WAP. L'utilisation des scripts PHP pose un problème de nom de fichier dans le cas du WAP : les navigateurs WAP ne veulent pas des fichiers dont l'extension est autre que *.wml*. La solution consiste alors à modifier l'en-tête du fichier avant l'envoi vers le navigateur, comme dans l'extrait suivant :

```
/* fichier 'test.php' */
<?php
   $f = fopen("foo.wml","w");
   fputs($f,"<?xml version=\"1.0\"?>\n");
   fputs($f,"<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"".
         "\"http://www.wapforum.org/DTD/wml_1.1.xml\">\n");
   fputs($f, "<wml>\n<card id=\"rep\">\n");
     ...
     ...
   fputs("</card>\n</wml>\n");
   fclose($f);
   header("Location: http://claude.gueganno.free.fr/wap/foo.wml")
   exit;
?>
```

Ceci permet d'utiliser la technologie PHP vers un terminal mobile.

9.6 Date et gestion du temps

`chaine date(format)` : retourne la date

a	am ou pm	d	jour du mois sans les zéros
D	jour de la semaine en 3 lettres	F	nom du mois
h	heure de 1 à 12	H	heure de 0 à 23
m	minutes	j	jour du mois avec les zéros
l	jour de la semaine	m	chiffre du mois
M	nom du mois en abrégé	y	année sur deux chiffres
Y	année sur 4 chiffres	z	jour de l'année
t	nombre de jours du mois	z	jour de l'année

Exemple : `echo date("a D h m l M Y d F H j m y z")` affiche :

```
pm Tue 03 08 Tuesday Aug 2003 19 August 15 19 08 03 230
```

`tableau gettimeofday() ;` : renvoie la date courante sous la forme d'un tableau associatif. Les clés principales sont :

- `"sec"` : donne le nombre de secondes écoulées depuis le 1er janvier 1970. Cet entier long est une représentation de la date appelée «*timestamp*».
- `"usec"` : donne le nombre de microsecondes écoulées dans la seconde courante.

`tableau getdate(timestamp) ;` : renvoie un tableau associatif correspondant à la date donnée en argument. Les clés sont données dans le tableau suivant :

`"seconds"`	secondes	`"minutes"`	minutes
`"hours"`	heures	`"mday"`	jour du mois
`"wday"`	jour de la semaine (numérique)	`"mon"`	mois (numérique)
`"year"`	année (numérique)	`"yday"`	jour dans l'année
`"weekday"`	jour dans la semaine (textuel)	`"month"`	mois textuel

`entier mktime(h,m,s,mois,j,a)` : renvoie un entier long : le *timestamp*.

Exemple (`date`, `getdate`, `mktime`)

```
<html>
<head>
<title> date </title>
</head>

<body>
<?php
echo "<h4>Construction d'une date formatée (<code>date()</code>)</h4>";
echo "Heure locale :".date("M d Y H:i:s", mktime(1,2,3,4,5,2003));
echo "<br>";
echo "Heure GMT :".gmdate("M d Y H:i:s", mktime(1,2,3,4,5,2003));
echo "<h4>Heure (<code>gettimeofday()</code>)</h4>";
$T = gettimeofday();
echo "Secondes : ".$T["sec"]."  microsecondes : ".$T["usec"];
echo "<h4>Date (<code>getdate()</code>)</h4>";
$T = getdate();
echo "secondes : <b>".$T["seconds"]."</b><br>";
echo "minutes: <b>".$T["minutes"]."</b><br>";
echo "heures : <b>".$T["hours"]."</b><br>";
echo "jour du mois : <b>".$T["mday"]."</b><br>";
echo "jour de la semaine : <b>".$T["wday"]."</b><br>";
echo "mois : <b>".$T["mon"]."</b><br>";
echo "année : <b>".$T["year"]."</b><br>";
echo "jour de l'année : <b>".$T["yday"]."</b><br>";
echo "jour de la semaine : <b>".$T["weekday"]."</b><br>";
echo "mois : <b>".$T["month"]."</b><br>";
?>
</body>
</html>
```

L'affichage produit est donné par la figure 9.5.

FIG. 9.5 – *Date et heure.*

`setlocale("LC_TIME", "fr_CA");` : avec ces paramètres, on obtient la possibilité d'afficher les paramètres textuels en français, avec la fonction `strftime`.

`chaîne strftime(format, [timestamp]);` : renvoie la date sous la forme d'une chaîne formatée. Si le paramètre *timestamp* est ignoré, c'est la date du jour qui est prise en compte. Les formats sont donnés dans le tableau suivant :

%a	jour de la semaine abrégé
%A	jour de la semaine
%b	mois en abrégé
%B	mois
%c	représentation complète date et heure
%d	jour du mois (de 00 à 31)
%H	heure (de 00 à 23)
%I	heure (de 01 à 12)
%j	jour dans l'année (de 001 à 366)
%m	mois (de 1 à 12)
%M	minute
%p	'am' ou 'pm'
%S	second as a decimal number
%U	numéro de semaine (commençant le dimanche)
%W	numéro de semaine (commençant le lundi)
%w	jour de la semaine (dimanche = 0)
%x	date
%X	heure
%y	l'année sans le siècle (de 00 à 99)
%Y	l'année et le siècle (ex. : 2003)
%Z	zone horaire
%%	le caractère '%'

`entier time()` : retourne la valeur du *timestamp* courant.

Exemple (`setlocale`, `strftime`)

```
<html>
<head>
<title> date </title>
</head>

<body>
<?php
setlocale ("LC_TIME", "fr_CA");
print(strftime("%A %d %B %Y", mktime(1,1,1,7,14,2003)));

echo "<br> Nous sommes le ".strftime("%A %d %B %Y")."<br>";
echo "Dans 360 jours nous serons le ";
$td = mktime(0,0,0,date("m"),date("d")+360,date("Y"));
echo strftime("%A %d %B %Y", $td);
echo "<td>";
?>
</body>
</html>
```

L'affichage produit est donné par la figure 9.6. On remarque la robustesse de la fonction `mktime` qui accepte ici un numéro de jour > 7.

FIG. 9.6 – *Date et heure en français.*

9.7 Base de données

PHP propose des fonctions pour accéder à plusieurs bases de données (`mysql`, `mSQL`, `PostgreSQL`, `Informix`, `Oracle`, `access`, ...). Selon la base choisie, l'interface est plus ou moins étoffée, mais on retrouve à chaque fois des fonctions pour :

- se connecter au serveur de base de donnée ;
- se lier à une base du serveur ;
- effectuer une requète ;
- exploiter le résultat de la requète dans un tableau ou séquentiellement ;
- se déconnecter.

Le paragraphe suivant donne un aperçu de l'utilisation d'une base de données avec `mysql`, base largement utilisée sur le *web*.

9.7.1 Connexion au serveur et sélection de la base

entier `mysql_connect(hostname, user, password)` : ouvre une connexion avec le serveur. Un entier > 0 est renvoyé en cas de réussite. En cas d'échec, la fonction retourne 0. Si deux appels sont faits à la fonction, il n'y a pas 2 connexions simultanées : c'est toujours la même qui court. La connexion est annulée à la fin du script, ou après un appel à `mysql_close`.

`entier mysql_select_db(nom_base, [lien]) ;` : sélectionne une base de donnée sur le dernier lien ouvert avec `mysql_connect`. Le paramètre *lien* est donc facultatif.

Renvoie *vrai* en cas de réussite et $faux$ sinon.

Exemple :

```
function connecter(){
     $host="localhost";
     $user="netcr";
     $password="stsii2";
     $database="netcr";

     /*connection a la base de données*/
     $lien = mysql_connect($host, $user, $password)
```

```
                or die("erreur de connexion");
        mysql_select_db($database, $lien)
                or die("erreur de table");
}
```

Dans la plupart des cas, le choix de l'hôte `localhost` est correct, puisque le script `PHP` s'exécute sur le serveur.

`entier mysql_list_tables(nomBase,[lien]) ;` : renvoie un pointeur sur la liste des noms de tables de la base identifiée par la chaîne *nomBase*.

`chaîne mysql_tablename(resultat, i) ;` : donne le nom de la $i^{\text{ème}}$ table de la base. *resultat* est un paramètre obtenu par un appel à `mysql_list_tables()`.

Exemple :

```
<?php
mysql_connect ("localhost:3306");
$resultat = mysql_list_tables ("netcr");
$i = 0;
while ($i < mysql_num_rows($resultat)) {
  $T[$i] = mysql_tablename ($resultat, $i);
  echo $T[$i] . "<BR>";
  $i++;
}
?>
```

9.7.2 Requète

`entier mysql_query(requete, [lien]) ;` : effectue une requète `SQL` sur la base sélectionnée du dernier lien ouvert. Le paramètre *lien* est facultatif. La requète est une chaîne de caractère qui ne doit pas se terminer par le caractère `' ;'`.

Renvoie un numéro identificateur de résultat > 0 en cas de réussite , et $faux$ (0) sinon.

Exemple :

```
connecter();
$req = "select * from maTable where x<5";
$resultat = mysql_query($req)
or die ("Erreur de requète SQL : ".$req."<br>";
```

9.7.3 Exploitation du résultat de la requète

Pour ces fonctions, le paramètre en entrée est celui qui est retourné par `mysql_query`.

`tab mysql_fetch_array(res)`	Retourne un tableau associatif qui représente tous les champs d'une rangée du résultat. Chaque appel produit la rangée suivante jusqu'à la fin.
`tab mysql_fetch_row(res)`	Retourne un tableau qui représente tous les champs d'une rangée du résultat. Chaque appel produit la rangée suivante jusqu'à la fin.
`objet mysql_fetch_object(res)`	Renvoie un objet dont les propriétés sont les noms des champs. Chaque appel produit un objet corrspondant à la rangée suivante jusqu'à la fin.
`entier mysql_num_rows(res)`	Donne le nombre de lignes
`entier mysql_num_fields(res)`	
`entier mysql_affected_rows([lien])`	renvoie le nombre de rangées affectées par la dernière requète `INSERT`, `UPDATE` ou `DELETE`

9.7.4 Déconnexion

Elle est automatique à la fin de l'exécution du script. Elle peut être provoquée par un appel à `mysql_close()`.

9.8 Autres bibliothèques

En dehors des fonctions présentées ici, notons également des bibliothèques de fonctions pour :

- travailler avec les `URL`,
- concevoir une image,
- concevoir un document `PDF`,
- crypter et décrypter un document,
- compresser et décompresser un document,
- accéder à un annuaire `LDAP`,
- envoyer un $e-mail$,
- consulter une boîte `POP` ou `IMAP`,
- ...

Une documentation complète peut être obtenue sur

```
http://www.php.net
http://www.php.net/download-docs.php
```

Index