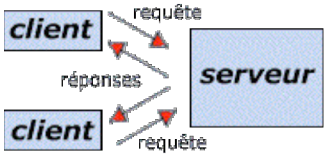
Programmation PHP

Master 1&2

Industries de la Langue

1. Programmation Web

1.1. Rappels de l'architecture client-serveur



- nombreuses applications fonctionnent selon un environnement client/serveur
- **machines clientes** (des machines faisant partie du réseau) contactent un **serveur**, une machine généralement très puissante en terme de capacités d'entrée-sortie, qui leur fournit des **services**.
- services sont des programmes fournissant des données telles que l'heure, des fichiers, une connexion, ...
- services sont exploités par des programmes, appelés **programmes clients**, s'exécutant sur les machines clientes.
- Exemple : client FTP, client de messagerie, ...,
- programme, tournant sur une machine cliente, capable de traiter des informations qu'il récupère auprès du serveur (dans le cas du client FTP il s'agit de fichiers, tandis que pour le client messagerie il s'agit de courrier électronique).

Dans un environnement purement Client/serveur, les ordinateurs du réseau (les clients) ne peuvent voir que le serveur, c'est un des principaux atouts de ce modèle.

1.1.1. Avantages de l'architecture client/serveur

Le modèle client/serveur est particulièrement recommandé pour des réseaux nécessitant un grand niveau de fiabilité, ses principaux atouts sont:

- **des ressources centralisées:** étant donné que le serveur est au centre du réseau, il peut gérer des ressources communes à tous les utilisateurs, comme par exemple une base de données centralisée, afin d'éviter les problèmes de redondance et de contradiction
- **une meilleure sécurité:** car le nombre de points d'entrée permettant l'accès aux données est moins important
- **une administration au niveau serveur:** les clients ayant peu d'importance dans ce modèle, ils ont moins besoin d'être administrés
- **un réseau évolutif:** grâce à cette architecture il est possible de supprimer ou rajouter des clients sans perturber le fonctionnement du réseau et sans modifications majeures

1.1.2. Inconvénients du modèle client/serveur

L'architecture client/serveur a tout de même quelques lacunes parmi lesquelles:

- **un coût élevé** dû à la technicité du serveur
- **un maillon faible**: le serveur est le seul maillon faible du réseau client/serveur, étant donné que tout le réseau est architecturé autour de lui! Heureusement, le serveur a une grande tolérance aux pannes (notamment grâce au système RAID)

1.1.3. Fonctionnement d'un système client/serveur

Un système client/serveur fonctionne selon le schéma suivant:

- Le client émet une requête vers le serveur grâce à son adresse et le port, qui désigne un service particulier du serveur
- Le serveur reçoit la demande et répond à l'aide de l'adresse de la machine client et son port

1.2. Client-serveur Web

- Le Web est un des nombreux services de l'Internet
- Le client Web est aussi connu sous le nom de « Navigateur »
  - Internet Explorer, Netscape Navigator, Safari, Camino…
- Le serveur Web est un logiciel qui renvoie les informations requises par le client selon un protocole et une norme déterminés.
  - Serveurs Web : Apache, Xitami, Fnord, Jigsaw, Sambar, Savant, TomCat…
  - Protocole : HTTP -> Hyper Text Transfer Protocole
  - Norme : HTML -> Hyper Text Mark-up Language
- L'utilisateur par le biais du client requiert un document directement (en le connaissant) ou indirectement (par moteur de recherche ou par lien)
- Le serveur reçoit la demande et renvoie le fichier requis (ou un message d'erreur -> le fameux erreur 404)
- Le client interprète (ou pas) le fichier reçu

1.3. Particularité du Web

- Il n'y a pas qu'un seul serveur -> d'où la nécessité des moteurs de recherche
- Les données sont principalement statiques
- Développement de langages pour dynamiser les pages :
  - Java Script :
    - insère des scripts dans une page qui sont exécutés par le navigateur
    - on transmet le code à exécuter en brut dans le code de la page
    - les scripts font des « petites actions »
  - Java Applets :
    - on envoie un code pré compilé qui est alors exécuté sur le client
    - le code est souvent lourd (en volume) et lent (à s'exécuter)
    - permet des programmes plus évolués

2. Le PHP

2.1. Historique

- 1994 créé par Rasmus Lerdof, ensemble de scripts Perl (Personal Home Pages) utilisation personnelle
- 1995 mise à la disposition du public de PHP
- version 2 PHP/FI
- PHP3 exécute le code en le lisant, efficace sur de petits scripts
- 2000 PHP4 compile le code et l'exécute
- acronyme actuel Hypertext PreProcessor.

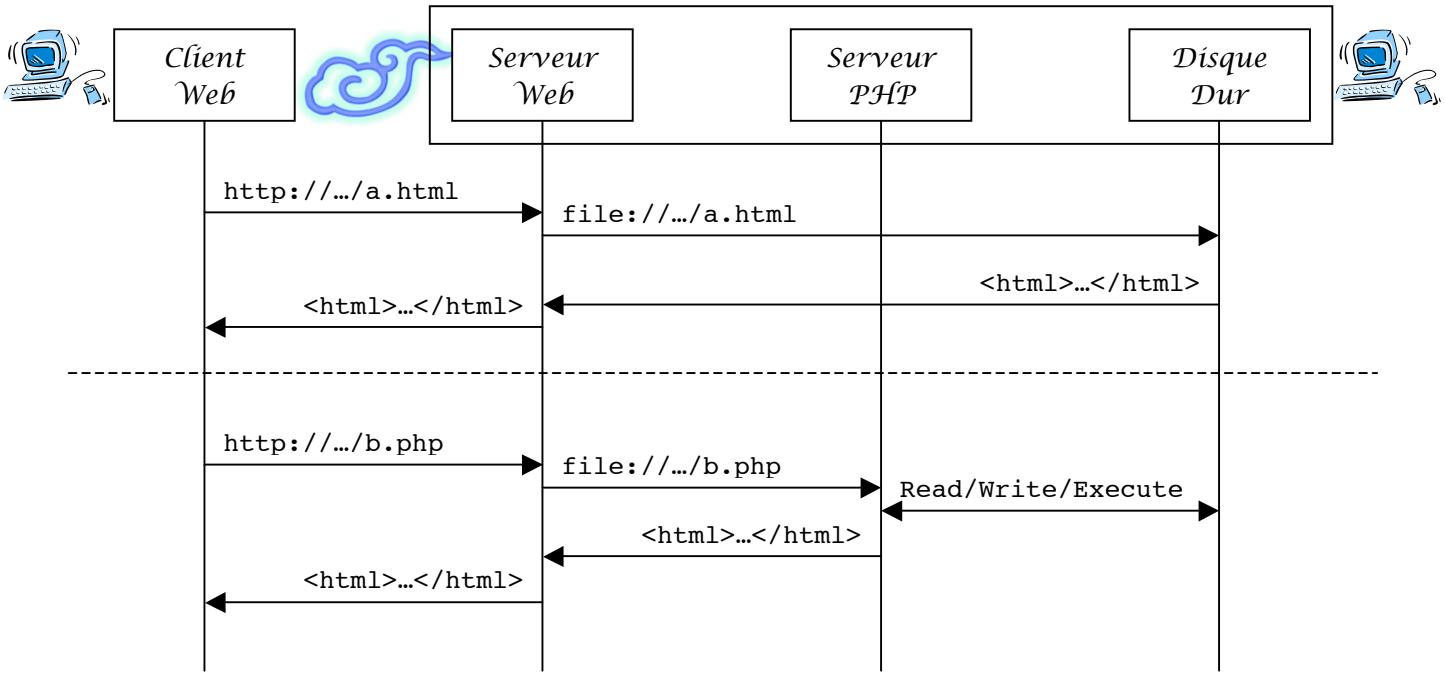
2.2. Caractéristiques

- langage de script ;
- module intégré dans Apache (mais peut tourner en CGI) ;
- 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 ;
- ne supporte pas un grand nombre de requêtes simultanées ;
- gratuit ;
- syntaxe proche du C et du Perl (acronyme non officiel Person Hate Perl) ;
- adresse officielle (beaucoup de documentations) http://www.php.net ;
- les scripts finissent par .php3 ou .php ;
- utilisation sous Linux avec Apache, et sous d'autres systèmes (free permet d'utiliser php3 comme langage de script et l'accès à des bases de données MySql).

2.3. présentation

- PHP = Hypertext PreProcessor
- Programmation serveur : le code s'exécute sur le serveur et renvoie le résultat au client
- Avantage : le programme reste sur le serveur, les données sont sur le serveur et peuvent être utilisées par tous les utilisateurs
- Inconvénient : le programme est exécuté sur le serveur ->charge mémoire et CPU à prendre en compte.
- Un langage à part :
- Un « module » PHP qui s'intègre aux logiciels de service Web
- Un langage qui s'interface avec d'autres ->SQL

- Un langage qui permet d'exécuter des programmes sur le serveur et d'en récupérer les résultats.
- Pas d'exécution directe par le client sur le serveur mais attention toutefois à la sécurité !!!
- Le PHP peut exécuter du PHP au sein de son propre code : eval()



2.4. 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 :

```
<? code php ?>
<?php code php ?>
<SCRIPT language="php"> code php </SCRIPT>
```

- Les instructions dans les balises sont séparées par des ;
- Tout ce qui est entre /* et */ est un commentaire
- page .php3 sur le serveur

```
<HTML>
<HEAD>
    <title>premiers pas en PHP</title>
</HEAD>
<BODY bgcolor="#FFFFFF">
<H1>Essai de script PHP</H1>
<?php
    echo "Bonjour, il est ";
```

```
    echo date ("H:i:s");
?>
</BODY>
</HTML>
```

➢ *page envoyée au navigateur*

```
<HTML>
<HEAD>
    <title>premiers pas en PHP</title>
</HEAD>
<BODY bgcolor="#FFFFFF">
<H1>Essai de script PHP</H1>
Bonjour, il est 17:05:38
</BODY>
</HTML>
```

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

3. Rappel HTML

3.1.Organisation d'un document HTML

- La déclaration de type de document doit précéder tout fichier HTML. Pour HTML 4.0, la déclaration est publique :

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

- La structure générale d'un document HTML est la suivante :

```
<HTML>
<HEAD>
    <!--  l'en-tête regroupe des données qui sont utiles au navigateur ou aux moteurs de recherche, mais qui ne sont pas affichées -->
    <TITLE>     </TITLE>
    <META name="..."  content="...">
    <STYLE type="text/css">
    <!--
          liste des styles appliqués au document de la forme
          selecteur {feuille de style}
    -->
    </STYLE>
    <SCRIPT language="JavaScript">
    <!--
          texte du script
    // -->
    </SCRIPT>
</HEAD>
<BODY>
    <!--  le corps du document est la partie qui est affichée dans la fenêtre du navigateur -->
...
</BODY>
</HTML>
```

3.2. L'en-tête

```
<TITLE>
```

- Obligatoire -- L'élément TITLE indique le nom de la fenêtre. Ce "titre" est aussi utilisé dans les listes de signets.

```
<BASE href="...">
```

- Permet de déréférencer les URL partiels. Si BASE est absent, l'URL du document en cours est utilisé.

```
<META>
```

- attributs de l'élément <META>
- http-equiv : permet de simuler une information du header HTTP
- name... content : une paire d'information nom-contenu

```
<LINK>
```

- utilisé pour les redirections ou pour l'appel des feuilles de styles externes
- attributs de l'élément LINK
- rel indique le type de relation existant entre la page pointée et la page en cours de lecture. L'utilisation principale est pour le lien avec les feuilles de style externes

```
<LINK rel="Stylesheet" href="stylegen.css">
```

- href URL du lien

3.3. Le corps du document

- Le corps du document représente la partie qui est visible à l'intérieur de la fenêtre du navigateur.
- L'élément BODY concerne toute la page encodée. Tous les autres éléments sont imbriqués à l'intérieur de l'élément BODY.

```
<BODY>
```

Attributs de l'élément BODY :

```
bgcolor
```

- codage RGB ("#000000" noir ; "#FFFFFF" blanc)

```
text
```

- couleur du texte

```
link
```

- couleur des liens (bleu : "#0000FF" par défaut)

```
vlink
```

- couleur des liens déjà parcourus

```
alink
```

- couleur flash au moment du clic

```
background
```

- appel d'une image de fond d'écran (image est définie par son URL)

Les attributs de l'élément BODY définissent des valeurs qui s'imposent à tout le document. Ils deviennent "dépréciés", et tendent à être remplacés par des feuilles de style associées à l'élément BODY. Plus généralement, à chaque fois que l'on peut utiliser une feuille de style à la place d'attributs spécifiques des éléments d'un document, cette solution est préférable pour la maintenance et l'évolution des pages concernées.

3.4. Les attributs généraux et les événements intrinsèques

Certains attributs sont applicables à tous les éléments. Il s'agit en particulier de :

`id`

- nom unique d'identifiant pour un élement spécifique. Sert pour les éléments qui ont une position absolue, ou bien qui sont manipulés par des scripts JavaScript. On utilise pour les images l'attribut **name** qui a la même fonction (et devient obsolète... même s'il est aujourd'hui indispensable pour les images sous NetScape).

`class`

- permet de définir des classes d'éléments auxquels on peut appliquer la même feuille de style.

`lang`

- language utilisé et **dir** direction de l'écriture de cette langue de droite à gauche pour les langues européennes, de gauche à droite pour l'arabe et l'hébreu.

`title`

- chaque élément peut avoir un titre spécifique. Celui-ci apparaît en bulle quand le curseur passe dessus l'élément concerné. Utile pour indiquer les destinations des liens ou développer les acronymes

`style`

- pour indiquer un style spécifique directement à l'intérieur du codage HTML. On indique le contenu de la feuille de style entre cotes ("), chaque propriété est séparée par le point-virgule (;).

`bgcolor`

- chaque élément peut recevoir un fond de couleur spécifique. Attention, certains navigateurs ne reconnaissent pas cet attribut pour tous les éléments.

Les éléments HTML peuvent recevoir des événements intrinsèques (intrisic events) qui représentent des actions de l'utilisateur. Pour programmer des pages dynamiques, on capte l'événement dans un handler, qui renvoie à une fonction JavaScript qui traite l'événement et produit les modifications du document.

Les handlers des événements intrinsèques de la souris et du clavier sont :

`onClick`

- l'utilisateur clique sur la boîte correspondant à l'élément concerné

`onDblClick`

- double clic

```
onMouseDown
```

- le bouton de la souris est enfoncé

```
onMouseUp
```

- le bouton de la souris est relaché

```
onMouseOver
```

- le pointeur passe sur la surface de la boîte concernée

```
onMouseOut
```

- le pointeur quitte la boîte

```
onMouseMove
```

- le pointeur se déplace

```
onKeyPress
```

- une touche clavier est pressée

```
onKeyDown
```

- touche clavier maintenue enfoncée

```
onKeyUp
```

- touche clavier relâchée

exemple de script intégré dans un élément :

```
<A href="destination.htm"
   onMouseOver="changecouleur(im1)
   onMouseOut="changecouleur(im1)">
      <IMG src="../image/avionbleu.gif"
            name=im1
            width = 50
            height = 50
            alt = "prends l'avion pour là bas"
            border=0>
</A>
```

Quand le pointeur passe au dessus de l'ancre, la fonction "changecouleur" est appelée avec le nom de l'image. Si bleu figure dans le nom de l'image (avionbleu.gif), on remplace la source de IMG (im1.src) par l'image rouge (avionrouge.gif), ou le contraire. La même fonction est appelée quand la souris quitte la zone définie par la boîte contenant l'ancre, ce qui va faire revenir l'image de départ (avionbleu.gif).

La fonction JavaScript `changecouleur()` *est placée dans l'élément SCRIPT de l'en-tête du document.*

3.5. Les éléments de bloc

```
<DIV>
```

- *distingue des parties d'un document HTML (divisions). Une division munie d'un identifiant (attribut* ***id****="..") est la base de la manipulation par des scripts de blocs de texte ou d'image.*

  attribut :

  ```
  align (left|center|right)
  ```

```
<H1>...<H6>
```

- *titres de section (heading)*

  attribut :

  ```
  align (left|center|right)  (déprécié)
  ```

```
<P>
```

- *Elément paragraphe. On peut omettre la marque de fin </P>, mais nous conseillons de la maintenir. Un paragraphe est suivi d'une ligne blanche.*

```
<UL>
```

- *Liste non-ordonnée.*

  attributs :

  ```
  type (style de la marque : disc, squarre, circle)
  compact (réduit l'espace entre les items - fonctionne mal avec les navigateurs du commerce)
  ```

```
<OL>
```

- *Liste ordonnée.*

  attributs :

  ```
  type :1 (1,2,3) a (a,b,c) A (A,B,C) i (i,ii,iii) I (I,II,III)
  start   valeur de départ
  compact
  ```

```
<DL>
```

- *Liste de définition*

```
<DT>
```

- *Terme à définir*

```
<DD>
```

- *Définition (par défaut en décalage à droite par rapport à l'élément DT)*

```
<PRE>
```

- *texte préformatté (caractères à chasse fixe, par défaut* `courier`*)*

```
<BLOCKQUOTE>
```

- Pour une citation (par défaut apparaît avec une marge gauche. Utilisé fréquemment pour provoquer une marge. Cette utilisation est déconseillée car remplacée par l'utilisation des feuilles de style).

```
<HR>
```

- ligne horizontale

  attributs :

```
align (left|center|right)
noshade
size (en pixels) (hauteur de la barre)
width (en pixels ou en pourcentage) largeur  de la barre
par defaut : 100% (de l'élément englobant)
```

3.6. Les éléments inline

Les éléments de texte (éléments inline) ne provoquent pas de changement de bloc. Ils s'intègrent dans le flot du document.

Les éléments d'enrichissement typographique :

```
<B>
```

- caractères gras

```
<TT>
```

- télétype

```
<I>
```

- Italique (attention : peu lisible à l'écran)

```
<SUB>
```

- en indice

```
<SUP>
```

- en exposant

Les éléments logiques de mise en valeur :

```
<EM>
```

- emphase (par défaut italique)

```
<STRONG>
```

- mise en valeur (par défaut gras)

```
<CITE>
```

- Citations ou références à d'autres sources

```
<ACRONYM>
```

- Pour distinguer les acronymes. En indiquant la signification de l'acronym en titre (attribut title), celle-ci apparaîtra dans une bulle si le pointeur s'attarde sur l'acronyme.

```
<SPAN>
```

- L'élément **<SPAN>** permet de distinguer une partie du flot de texte pour appliquer à cette partie un style spécifique. On utilise l'attribut **id** ou **class** pour indiquer le type de traitement que la feuille de style va appliquer. Très utile pour les premières lettres d'un paragraphe qui sont définies en taille supérieure, voire en couleur différente.

```
<BR>
```

- pour forcer un saut de ligne

3.7. Les éléments fonctionnels

3.7.1. Les liens (ou ancres) <A>

```
<A>
```

- Attributs (bien utiliser les quotes) :

```
name
```

- pour désigner une partie d'un corps de document. Installe un marque page pour pouvoir sauter ensuite directement à ce point du texte. Utilisation déconseillée : éviter les pages HTML trop longues

```
href
```

- lien hypertexte vers un URL (mnémotechnique : Hypertext REFerence)

```
target
```

- définit le cadre (frame) qui va recevoir la page désignée par la référence hypertexte. Nous déconseillons l'utilisation de cadres (FRAMES).

3.7.2. <IMG> les images dans le fil du texte

```
<IMG>
```

- Attributs de l'élément image :

```
Src
```

- URL de l'image ressource

```
alt
```

- équivalent en texte pour les navigateurs sans image

```
align (top | middle | bottom | left | right)
```

- position de l'image par rapport au texte

```
width
```

- largeur de l'image en pixels

```
height
```

- hauteur de l'image en pixels

```
border
```

- en pixels, taille du tour de l'image quand elle est dans une ancre

```
hspace
```

- en pixels, espace à droite et à gauche de l'image

```
vspace
```

- en pixels, espace au dessus et en dessous de l'image

```
usemap
```

- identification de zones sensibles. On indique le nom (name) de l'élément MAP (cet élément doit être présent dans le même document HTML) qui indique les zones sensibles et les liens induits.

3.7.3. <MAP> : images à zone sensible du coté du client

```
<img src="navbar.gif" border=0 usemap="#map1">
<map name="map1">
<area href="guide.html" alt="Access Guide" shape=rect coords="0,0,118,28">
<area href="search.html" alt="Search" shape=rect coords="184,0,276,28">
<area href="shortcut.html" alt="Go" shape=rect coords="118,0,184,28">
<area href="top10.html" alt="Top Ten" shape=rect coords="276,0,373,28">
</map>
```

les formes des zones sont (x et y sont mesurés en pixels à partir du coin haut-gauche de l'image mappée) :

```
    shape=rect coords="x-gauche, y-gauche, x-droit, y-droit"
    shape=circle coords="x-centre, y-centre, rayon"
    shape=poly coords="x1,y1,x2,y2,x3,y3..."
```

3.7.4. <APPLET> ... </APPLET>

(le texte compris entre les deux balises sert aux navigateurs qui n'intègrent pas JAVA
Attributs de l'élément <APPLET>

```
codebase
```

- URL de base pour l'applet (défaut : URL du document en cours)

```
code
```

- URL relatif à codebase pour les appels des classes JAVA

```
alt
```

- texte alternatif

```
name
```

- nom pour l'instance de l'applet, permet la communication entre applets d'une même page

```
width, height
```

- en pixels espace de placement de l'applet

```
align, vspace, hspace
```

- (comme pour l'élément <IMG>)

```
<PARAM>
```

- permet de paser des paramètres à l'applet.

3.8. Les tableaux

Les tableaux sont définis par **<TABLE> ... </TABLE>**. On peut enchasser un tableau dans une cellule d'un autre tableau. Un tableau est composé de lignes (élément <TR>), chaque ligne ayant des cellules juxtaposées (éléments <TH> ou <TD>).

3.8.1. Attributs de l'élément <TABLE>

```
align  (left | center | right)
```

- (alignement du texte à l'intérieur de la table)

```
width
```

- largeur en pixels, ou bien en pourcentage de la taille de l'écran

```
border
```

- en pixel, taille de l'encadrement du tableau (0 -> pas de cadre)

```
cellspacing
```

- en pixels, espace entre les cellules

```
cellpadding
```

- en pixels, espace entre le bord et le contenu d'une cellule

3.8.2. Éléments contenus dans un tableau :

```
<CAPTION>
```

- légende du tableau
  - attribut :

```
align  (top | bottom | left | right)
```

```
<TR>
```

- ligne de tableau (table row) c'est un container pour les cellules. Même si on peut oublier la balise de fin (</TR>), il est préférable de l'utiliser.
  - attributs :

```
align (left | center | right)
valign  (top | middle | bottom)
```

- Les cellules elles-mêmes comportent deux types d'éléments :

```
<TH>
```

- titres de cellules (contiennent du texte mis en gras par défaut)

```
<TD>
```

- cellules de donnés
- (ces distinctions sont utiles pour la typographie, et pour les navigateurs en mode lecture automatique)

3.8.3. Attributs des cellules :

```
nowrap
```

- pas de retour automatique à la ligne

```
rowspan
```

- nombre de lignes occupées par cette cellule

```
colspan
```

- nombre de colonnes occupées par cette cellule

```
align (left | center | right)
```

- (a priorité sur les mentions de <TR>)

```
valign (top | middle | bottom)
```

- (priorité sur les mentions de <TR>)

```
width
```

- (en pixels - attention aux conflits avec les autres colonnes)

```
height
```

- (idem)

```
bgcolor
```

- définit la couleur de fond d'une cellule

3.9. Les formulaires

Le contenu d'un formulaire est compris entre les balises **<FORM>...** **</FORM>**. Le dernier élément d'un formulaire est un bouton d'envoi du formulaire (type = submit ou button).

3.9.1. Les attributs de l'élément <FORM> :

```
id (ou name)
```

- nom du formulaire pour les scripts JavaScript

```
action
```

- donne l'URL du programme auquel est envoyé le contenu du formulaire (en général un programme cgi)

```
method
```

- get | post

```
enctype
```

- par défaut application/x-www-form-urlencoded

3.9.2. Les champs de saisie d'un formulaire : (élément INPUT)

```
<INPUT>
```

- (pas de balise de fermeture, ce n'est pas un container)
- attributs :

```
type text
```

- (défaut) zone de saisie d'une ligne, on précise la taille par size=xx (en nombre de caractères)

```
<input type=text size=40 name="user" value="votre nom">
```

```
password
```

- le contenu est masqué par des points sur l'écran

```
<input type=password size=12 name="pw">
```

```
checkbox
```

- chaque boite à cocher génère une paire nom/valeur

```
<input type=checkbox checked name="etudiant" value="oui">
```

```
radio
```

- permet de choisir une valeur parmi plusieurs

```
<input type=radio name=age value="0-20">
<input type=radio name=age value="20-40" checked>
<input type=radio name=age value="40+">
```

```
submit
```

- envoie à l'URL définit dans l'attribut **action** le contenu du formulaire sous la forme de paires nom/valeur. Le texte du bouton est indiqué par l'attribut value

```
reset
```

- replace les champs du formulaire dans leur état initial

```
file
```

- permet d'attacher un fichier au contenu du formulaire (apparaît comme un champ de texte, avec un bouton de balayage pour désigner le fichier à attacher.

```
Hidden
```

- champ caché, qui sert au suivi de session

```
<input type=hidden name="utilisateur" value="0123-542">
```

3.9.3. Les champs de sélection sur liste (élément SELECT)

```
<SELECT>
```

- menu de sélection au sein d'un formulaire. Il permet de faire apparaître un menu déroulant

```
<SELECT name="couleur">
    <OPTION value="rouge">Rouge
    <OPTION value="vert">Vert
    <OPTION value="bleu">Bleu
</SELECT>
```

Attributs de l'élément de sélection :

```
name
```

```
size
```

- nombre de choix visibles

```
multiple
```

- permet de sélectionner plusieurs valeurs

Attributs de l'élément OPTION

```
value
```

- contient l'information qui est envoyée au programme de traitement du formulaire (la paire nom de la sélection, value de l'option choisie).

```
Selected
```

- option qui apparaît dans le sélecteur avant toute action de l'utilisateur

3.9.4. Les champs de texte (élément TEXTAREA)

```
<TEXTAREA>
```

- champ de texte comportant plusieurs lignes

attributs de l'élément <TEXTAREA> :

```
name
```

```
rows
```

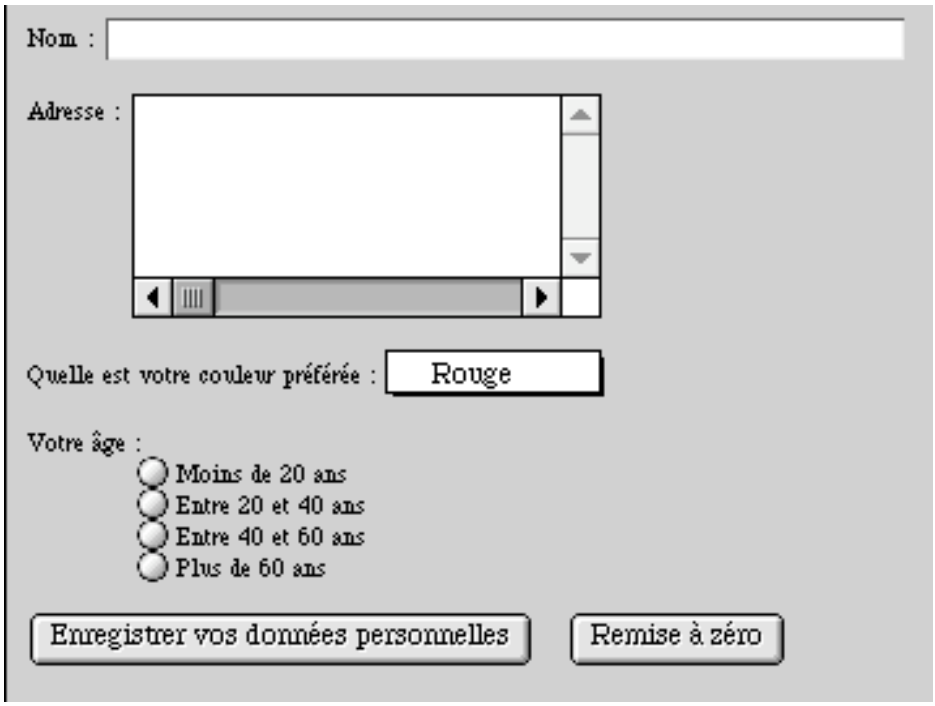
- nb de lignes

```
cols
```

- nb de caractères d'une ligne de texte
- Le texte d'une zone de texte est compris entre la balise <TEXTAREA> et la balise fermante (</TEXTAREA>)

3.9.5. Exemple de formulaire

```
<FORM name = "form1"
     action = "test.cgi"
     method = post>
<P>Nom : <INPUT type=text size="50" name="nom" value=""> </P>
<P>Adresse :
     <TEXTAREA name="adresse" rows=5 cols=25 border=0>  </TEXTAREA>
</P>
<p>Quelle est votre couleur préférée :
     <SELECT name="couleur">
           <OPTION value="rouge" selected>Rouge
           <OPTION value="vert">Vert
           <OPTION value="bleu">Bleu
     </SELECT>
<DL><DT>Votre âge : </DT>
     <DD><INPUT type=radio name="age" value="1">Moins de 20 ans <BR>
         <INPUT type=radio name="age" value="2">Entre 20 et 40 ans <BR>
         <INPUT type=radio name="age" value="3">Entre 40 et 60 ans <BR>
               <INPUT type=radio name="age" value="4">Plus de 60 ans </DD>
</DL>
<P><INPUT type=submit name="go" value="Enregistrer vos données
personnelles">
  <INPUT type=reset name="raz" value="Remise à zéro"></P>
</FORM>
```



4.PHP : le langage de programmation

Définition *Variable*

Case mémoire stockant une donnée pouvant varier au cours de l'exécution d'un programme (par opposition aux constantes).

Définition *Constante*

Un élément de donnée qui ne change jamais.

4.1. Variables

4.1.1. syntaxe

```
$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 :*

```
<?php
$reel = 0.3;
$entier = 22;
$chaine = "bonjour !";
$phrase1 = "bonjour $chaine!";
$phrase2 = 'bonjour $chaine$';
print("un reel : $reel<BR>");
print("un entier : $entier<BR>");
print("une chaine : $chaine<BR>");
print("une phrase : $phrase1<BR>");
print("du bon usage des guillemets : $phrase2");
?>
```

```
un reel : 0.3
un entier : 22
une chaine : bonjour !
une phrase : bonjour bonjour !!
du bon usage des guillemets : bonjour $chaine$
```

- *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");
?>
```

```
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 = "\"Salut tout le monde\"";
echo $chaine;
?>
```

```
"Salut tout le monde"
```

4.1.2. Fonctions de Conversion

- *intval, strval, doubleval*
- *$entier = intval($reel); (integer), (string), (double) : $entier = (integer)$reel;*
- *booleen settype(var, type)*
- *chaine strval(argument) retourne l'argument sous forme de chaine*

```
<?php
$reel = 3.85;
$entier =(integer)$reel;
print("le reel $reel donne l'entier $entier");
?>
```

```
le reel 3.85 donne l'entier 3
```

4.1.3. Fonctions pour connaître le type

```
chaine gettype(variable)
booleen is_array(var)
booleen is_double(var)
booleen is_integer(var)
booleen is_object(var)
booleen is_string(var)
booleen is_type(var, type) renvoie 1 si var est du type : type
```

4.1.4. 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) ;

4.1.5. 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.

4.1.6. 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";
echo "$ch<BR>";
echo "${$ch}<BR>";
echo "$bonjour<BR>";
?
```

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

4.2. Constantes

4.2.1. Syntaxe

- définition de la valeur de la constante :

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

- *utilisation :*

```
NOM_CONST
```

- *vérification d'existence :*

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

```
<?php
define("MAIL_THOMAS", "thomas.lebarbe@u-grenoble3.fr");
print("envoyez un mail &agrave; ".MAIL_THOMAS);
?>
```

```
envoyez un mail à thomas.lebarbe@u-grenoble3.fr
```

4.2.2. Constantes prédéfinies

```
TRUE
```

- *vrai*

```
FALSE
```

- *faux*

```
__FILE__
```

- *nom du fichier du script parsé (attention il y a deux caractères soulignés avant FILE et deux après*

```
__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>";
?>
```

```
3.0.12
Linux
4
/home/thomas/public_html/essaisphp/constantes.php3
```

4.3. Opérateurs

4.3.1. Arithmétiques

```
+
```

- addition

```
-
```

- soustraction

```
*
```

- multiplication

```
/
```

- division

```
%
```

- modulo

```
++
```

- incrémentation

```
--
```

- décrémentation

4.3.2. Chaînes

```
.
```

- concaténation

4.3.3. Booléens

```
Not
!
```

- négation

```
or
||
```

- ou logique

```
and
&&
```

- et logique

4.3.4. Binaires

```
|
```

- ou

```
&
```

- et

```
~
```

- not

```
^
```

- xor

4.3.5. Affectation

```
$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

4.3.6. Comparaison

```
$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 du même type

```
$x !== $y
```

- PHP4 : vrai si $x différent de $y ou qu'ils n'ont pas le même type

4.4. Structures de contrôle

NB :

- false = 0
- true = tout le reste.

4.4.1. if

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

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

4.4.2. if...else

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

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

Un remplaçant du if...else

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

4.4.3. 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){
    instruction1;
    ...
}
elseif (expression3){
    instruction1;
    ...
}
else {
    instruction1
    ...
}
```

4.4.4. switch

Selon l'expression exécuter des instructions.

```
switch (expression){
    case val1 : instructions; break;
    case val2 : instructions; break;
    ...
    default : instructions;
}
```

4.4.5. while

Tant que l'expression est vraie faire les instructions

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

4.4.6. do ... while

Faire les instructions tant que l'expression est vraie

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

4.4.7. for

Pour

```
for(expression; test; expression_inc){
    instructions;
}
```

4.4.8. foreach

Pour chaque (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. NB : au départ le pointeur sur le tableau est automatiquement mis sur la première case par le foreach.

4.4.9. Interruption de boucles

```
break
```

- *interrompt les boucles for, while.*

```
continue
```

- *interrompt l'exécution d'une itération et reprend à l'itération suivante.*

4.4.10. Interruption du script

```
exit
```

4.5. Fonctions

4.5.1. Syntaxe

```
function nomfonction($arg1,&$arg2,$arg3="defaultvalue",...,$argn){
    instructions ;
}
```

NB :

- *arg1 est passé par valeur ;*
- *arg2 est passé par adresse : si sa valeur est modifiée dans la fonction, elle est modifiée pour tout le programme ;*
- *arg3 est optionnel : il peut ne pas être passé lors de l'appel de la fonction puisqu'il a une valeur par défaut.*

Appel :

```
nomfonction(param1,param2,param3..., paramn);
```

Pour retourner une valeur :

```
return
```

Chaque appel de fonction demande du temps CPU, si plusieurs internautes effectuent des requêtes qui appellent plusieurs fois des fonctions, les temps d'attente pourraient devenir très grands.

4.5.2. Portée des variables globales et locales

- *Une variable globale n'est pas visible dans une fonction, pour qu'elle le soit il faut le préciser avec global.*

```
<?php
// fonction Sport1 $sport est une variable locale
```

```
function Sport1(){
  $sport = "volley";
  print ("aimez-vous le $sport ?<BR>");
}
// fonction Sport2 $sport est une variable globale
function Sport2(){
  global $sport;
  $sport = "badmington";
  print ("aimez-vous le $sport ?<BR>");
}
// corps du programme
$sport = "ski";
print ("aimez-vous le $sport ?<BR>");
Sport1();
print ("aimez-vous le $sport ?<BR>");
Sport2();
print ("aimez-vous le $sport ?<BR>");
?>
```

```
aimez-vous le ski ?
aimez-vous le volley ?
aimez-vous le ski ?
aimez-vous le badmington ?
aimez-vous le badmington ?
```

- *Une variable locale peut être mémorisée d'un appel à l'autre de la fonction en lui mettant static.*

```
<?php
// fonction qui alterne les couleurs
function Couleur(){
  static $valcoul;
  if($valcoul == "#FFFFCC")
    $valcoul = "#DDDDDD";
  else
    $valcoul = "#FFFFCC";
  return $valcoul;
}
// corps du programme
print("<TABLE>\n");
for($i = 0; $i < 5; $i++){
  $coul = Couleur();
  print("<TR>");
  print("<TD bgcolor=\"$coul\">");
```

```
  print("ligne $i</TD></TR>\n");
}
print ("</TABLE>");
?>
```

```
<TABLE>
<TR><TD bgcolor="#FFFFCC">ligne 0</TD></TR>
<TR><TD bgcolor="#DDDDDD">ligne 1</TD></TR>
<TR><TD bgcolor="#FFFFCC">ligne 2</TD></TR>
<TR><TD bgcolor="#DDDDDD">ligne 3</TD></TR>
<TR><TD bgcolor="#FFFFCC">ligne 4</TD></TR>
</TABLE>
```

4.5.3. Appel dynamique

Si vous ne savez pas exactement quelle fonction devra être appelée (car ça dépend des données que vous allez recevoir) vous pouvez paramétrer une variable avec un nom de fonction :

```
<?php
function affiche($chaine){
  print($chaine);
}
function afficheln($chaine){
  print("$chaine<BR>");
}
// corps
$afficher = "afficheln";
$afficher("Bonjour, ");
$afficher("...");
$afficher = "affiche";
$afficher("Au revoir ! ");
$afficher("...");
?>
```

```
Bonjour,
...
Au revoir ! ...
```

4.6. inclusion de scripts

Il peut être intéressant de mettre dans un fichier séparé un script qui peut servir à plusieurs autres fichiers de script. Pour appeler le script dans le fichier séparé on peut utiliser include ou require.

4.6.1. include

- *syntaxe :*

```
include("nom_fichier");
```

- PHP analyse le fichier à l'endroit ou include est appelé
- Codé inséré et interprété uniquement si l'instruction est exécutée (sauté si la valeur testée dans un if est false par exemple)

4.6.2. require

- syntaxe :

```
require("nom_fichier");
```

- PHP analyse le fichier à l'endroit ou require est appelé
- Inconvénient : insère toujours le code, même si l'instruction n'est pas exécutée (ne pas mettre dans un if, switch, while, for)
- Avantage : en PHP3 le code est inséré au premier passage de l'analyseur PHP -> rapidité.

4.7. Chaînes

4.7.1. Séquences d'échappement

```
\n
```

- retour à la ligne

```
\t
```

- tabulation

```
\\
```

- \

```
\$
```

- $

```
\"
```

- "

4.7.2. Fonctions pour les chaînes

```
eval(chaine)
```

- evalue la chaine 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)

```
chaine strtok(ch, sep)
```

- separe ch en fonction de sep

```
chaine quotemeta(ch)
```

- retourne la chaîne avec un \ devant les caractères . \\ + * ? [^] ($)

```
chaine str_repeat (ch, n)
```

- répète n fois la chaîne (PHP4)

NB : l'argument ch ne doit être précisé qu'au premier appel de strtok

```
<?php
$phrase = "ceci est une phrase de test pour strtok";
echo "<BR>test de strtok<BR>\n";
for($mot = strtok($phrase, " ") ; $mot != ""; $mot = strtok(" ")){
    echo "$mot.";
}
?>
```

```
test de strtok
ceci.est.une.phrase.de.test.pour.strtok.
```

4.7.3. Conversion

```
chaine sprintf(format, arg)
```

- retourne une chaine formatée

```
chaine chr(ascii)
```

- donne le caractere qui correspond au code ascii

```
int ord(car)
```

- donne le code ascii qui correspond au caractère

```
chaine pack(format,arg)
```

- $ch = pack("ccc", 66, 79, 78); affichera bon

```
chaine sql_regcase(exp)
```

- convertit une exp. rationnelle. en une exp rationnelle non sensible à la casse

4.7.4. position, sous-chaîne

```
entier strpos(ch, ssch)
```

- retourne la position de la 1ère occurrence de ssch dans ch

```
entier strrpos(ch, car)
```

- retourne la position de la dernière occurrence du caractère

```
entier strspn(ch, enscar)
```

- longueur de la sous-chaîne dont les caractères sont entièrement dans enscar

```
chaine strstr(ch, ssch)
```

- retourne la portion de ch à partir de la 1ère occurrence de ssch et jusqu'à la fin

```
chaine stristr(ch, ssch)
```

- strstr non sensible à la casse

```
chaine substr(ch, deb, l)
```

- renvoie la sous-chaîne de taille l qui commence à l'indice deb

4.7.5. comparaison

```
entier strcasecmp(ch1, ch2)
```

- comparaison sans tenir compte de la casse

```
entier strcmp(ch1, ch2)
```

- retourne 0 si ch1 = ch2

4.7.6. casse

```
chaine strtolower(ch)
```

- convertit en minuscules

```
chaine strtoupper(ch)
```

- convertit en majuscules

```
chaine ucfirst(ch)
```

- met la première lettre en majuscule

```
chaine ucwords(ch)
```

- met tous les mots de ch en majuscule

4.7.7. url, html

```
chaine htmlentities(ch)
```

- convertit les car de la chaine au format HTML

```
chaine htmlspecialchars (ch)
```

- convertit les caractères spéciaux en entités html (< -> <)

```
chaine 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)

```
tableau parse_url(requete)
```

- stocke une URL dans un tableau associatif (scheme, host, path, query).

```
chaine rawurldecode(URL)
```

- convertit en texte la chaine au format URL (%)

```
chaine rawurlencode(ch)
```

- convertit ch au format URL (%)

```
chaine urldecode(URL)
```

- conversion en texte de l'URL (pas pour les données binaires)

```
chaine urlencode(ch)
```

- conversion de la chaine au format URL (par pour bin)

4.7.8. espaces

```
chaine chop(ch)
```

- retourne la chaine sans les espaces

```
chaine trim(ch)
```

- supprime les espaces de début et fin de chaîne

4.7.9. Expressions rationnelles

```
|
```

- ou

```
*
```

- 0 ou +

```
+
```

- au moins une fois

```
?
```

- 0 ou 1 fois

```
{n}
```

- n fois

```
{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 ^ .[]()|?{}\ s'obtiennent en ajoutant un \ devant.

4.7.10. Fonctions pour les expressions rationnelles

```
booleen 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, les autres cases les sous-occurrences. Retourne 0 si pas de correspondance

```
chaine 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

```
 ereg_replace(10, "<BR>", ch)
```

- remplace les \n par des

```
booleen eregi(exp, ch, tabocc)
```

- comme ereg sans tenir compte de la casse

```
booleen 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

```
<?
echo "<BR><HR>\nTest http_user_agent<BR>";
ereg("^([A-Za-z]+)/([0-9]+\.[0-9]+) (.*)$", $HTTP_USER_AGENT, $tab);
$nomnav = $tab[1];
$version = $tab[2];
$info = $tab[3];
echo "nomnav = $nomnav ; version = $version ; info = $info";
?>
```

```
Test http_user_agent
nomnav = Mozilla ; version = 4.7 ; info = [en] (X11; I; Linux
2.2.15pre3 ppc)
```

4.8. Fichiers

4.8.1. Ouverture :

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

- Mode :
  - `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
- NB : pour un fichier binaire il faut ajouter après le mode un b

4.8.2. vérification de l'ouverture

```
if(!($POINTEUR)){
    print("erreur ouverture fichier");
    exit;
}
/* inutile de mettre un else pour la suite */
```

4.8.3. ouverture et stockage dans un tableau

```
$fichier = file("nomfich");
for($i = 0; $i < count($fichier); $i++)
{
    print($fichier[$i]);
}
```

4.8.4. Fermeture

```
booleen fclose($POINTEUR);
```

4.8.5. Lecture jusqu'à la fin du fichier

```
while(!feof($POINTEUR)){
```

```
    $ligne = fgets($POINTEUR,255);
}
```

4.8.6. lecture d'un caractère

```
$car = fgetc($POINTEUR);
```

4.8.7. lecture de n caractères

```
$chaine = fgets($POINTEUR, 255);
```

4.8.8. se replacer au début d'un fichier

```
rewind($POINTEUR);
```

4.8.9. écriture

```
fputs($POINTEUR, "chaine");
```

4.8.10. exemple d'ouverture et lecture de fichier

```
<HTML>
<HEAD>
<title>essai fichier</title>
</HEAD>
<BODY>
<?
/* ouverture en lecture */
echo "<BR><HR>ouvert en lecture<BR><HR><BR>";
$POINTEUR = fopen("fichier.txt","r");
if(!($POINTEUR)){
    echo "erreur ouverture fichier.txt";
    exit;
}
while(!feof($POINTEUR)){
   $ligne = fgets($POINTEUR,255);
   echo "$ligne<BR>\n";
}
if( !(fclose($POINTEUR)) ){
    echo "erreur fermeture fichier.txt";
    exit;
}
/* ouverture avec stockage dans un tableau */
echo "<BR><HR>fichier dans un tableau<BR><HR><BR>";
$fichier = file("fichier.txt");
for($i = 0; $i < count($fichier); $i++)
```

```
{
    echo "$fichier[$i]<BR>";
}
?>
</BODY>
</HTML>
```

4.8.11. Manipulation de fichiers et répertoires

```
booleen chgrp("nomfich","nomgroupe")
```

- *change le groupe d'appartenance*

```
booleen chmod("nomfich",mode)
```

- *change les droits d'accès*

```
booleen chown("nomfich","owner")
```

- *change le propriétaire.*

```
int rename("nomfich", "newnomfich")
```

- *change le nom du fichier.*

```
chaine basename(chemin)
```

- *retourne le nom de fichier d'un chemin*

```
entier filectime("nomfich")
```

- *temps écoulé depuis la dernière modification du fichier.*

```
entier filesize("nomfich")
```

- *taille en octets du fichier.*

```
booleen file_exists("nomfich")
```

- *1 si le fichier existe.*

```
booleen is_file("nomfich")
```

- *1 si nomfich est un fichier.*

```
booleen is_executable("nomfich")
```

- *1 si fichier existe et est en mode exécutable.*

```
booleen is_readable("nomfich")
```

- *1 si fichier existe et est en mode lecture.*

```
booleen is_writeable("nomfich")
```

- *1 si fichier existe et est en mode écriture.*

```
chaine tempnam(rep, ch)
```

- *création d'un fichier temporaire unique dans le répertoire rep. Le préfixe est optionnel, retourne le nom du fichier temporaire.*

```
chaine dirname(chemin)
```

- *retourne le répertoire d'un chemin*

```
booleen is_dir("nom")
```

- *1 si nom est un répertoire.*

```
booleen chdir("repertoire")
```

- *change de repertoire, renvoie 1 si ok.*

```
closedir(pointeur)
```

- *ferme un répertoire ouvert par opendir*

```
booleen copy(fichsrce, repdest)
```

- *copie d'un fichier vers un répertoire.*

```
booleen mkdir("nomrep")
```

- *crée un nouveau répertoire*

```
entier opendir("nomrep")
```

- *ouvre un répertoire.*

```
booleen rmdir("nomrep")
```

- *supprime le répertoire.*

```
chaine readdir(pointeur)
```

- *pour lire un élément du répertoire (retourne un nom de fichier ou de répertoire).*

```
rewinddir(pointeur)
```

- *revenir au début du répertoire.*

```
<?php
// affichage du contenu du repertoire courant
$REP = opendir(".");
while($entree = readdir($REP))
     echo "$entree<BR>";
closedir($REP);
?>
```

4.8.12. Envoi de données vers le navigateur

```
booleen fpassthru($POINTEUR)
```

- *affiche le contenu d'un fichier dans la fenêtre du navigateur.*

```
<HTML>
<HEAD>
<title>essai fichier</title>
</HEAD>
<BODY>
Je vous envoie le contenu du fichier :<BR><HR>
<?
     if( !($POINTEUR = fopen("fichier.txt","r")) ){
```

```
        echo "pb ouverture";
        exit;
    }
    fpassthru($POINTEUR);
?>
<BR><HR>
</BODY>
</HTML>
```

```
entier readfile("nomfich")
```

- *lit le fichier et l'envoie au navigateur, retourne le nb d'octets lus.*

4.9. Shell et Pipe

4.9.1. Pipe

```
<?php
$pipe = popen("ls -l", "r");
while(!feof($pipe)){
  $ligne = fgets($pipe, 255);
  echo $ligne."<BR>";
}
pclose($pipe);
?>
```

- *affichera le contenu du répertoire*

4.9.2. Commandes du Shell

```
chaine exec(commande, sortie, retour)
```

- *Exécute la commande, la dernière ligne du résultat de la commande est retournée, n'envoie rien au navigateur. Argument sortie optionnel, s'il y est, chaque ligne du résultat est ajoutée à sortie comme un élément de tableau.*

```
chaine getenv(variable_env)
```

- *Retourne la valeur de la variable d'environnement.*

```
putenv(variable_env)
```

- *Pour paramétrer une variable d'environnement.*

```
chaine system(commande, retour)
```

- *Exécute la commande passée, envoie le résultat vers le navigateur, renvoie la dernière ligne de ce résultat.*

4.10. Fonctions mathématiques

4.10.1. 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(angle)
```

- tangente de l'angle

4.10.2. conversion

```
chaine decbin(entier)
```

- retourne la représentation binaire de entier

```
int bindec(chaine)
```

- retourne la représentation décimale de la chaîne binaire

```
chaine dechex(entier)
```

- retourne la représentation hexadécimale de entier

```
chaine decoct(entier)
```

- retourne la représentation octale de entier

```
décimal deg2rad(angle)
```

- donne le radian pour l'angle en degré

```
entier hexdec(chainehex)
```

- conversion chaine hexadécimale en un entier

```
décimal octdec(chaineoct)
```

- conversion chaine octale en un entier

```
décimal rad2deg(angle)
```

- retourne les degrés pour l'angle en radian

4.10.3. arrondi

```
 nb 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*

4.10.4. autres

```
nb abs(x)
```

- *valeur absolue*

```
décimal exp(x)
```

- *exponentielle*

```
décimal log(x)
```

- *logarithme*

```
décimal pi()
```

- *retourne la valeur de pi*

```
décimal pow(base, puissance)
```

- *élève la base à la puissance*

```
décimal sqrt(x)
```

- *racine carrée de x*

4.10.5. nombres aléatoires

```
 entier getrandmax()
```

- *retourne le nb aléatoire max pour rand*

```
entier rand(inf, sup)
```

- *retourne un nb aléatoire compris entre les entiers inf et sup*

```
entier srand(entier)
```

- *pour initialiser le générateur de nb aléatoires*

4.11. Fonctions de temps et de date

```
booleen checkdate(m, j, a)
```

- *1 si la date passée est valide, 0 sinon*

```
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*

```
I
```

- *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 unités*

```
Y
```

- *année sur 4 unités*

```
z
```

- *jour de l'année*

```
tableau getdate()
```

- *retourne un tableau associatif*

```
hours
```

- *heure format 24*

```
minutes
```

- *heure en minutes*

```
month
```

- *nom du moi*

```
mday
```

- jour du mois

```
mon
```

- mois

```
wsay
```

- jour (0..6)

```
weekday
```

- nom du jour

```
yday
```

- jour de l'année

```
year
```

- année

```
seconds
```

- minutes en secondes

```
0
```

- Timestamp (nb de secondes depuis le 01/01/70)

```
tableau gmdate(format)
```

- comme date mais au format Greenwich

```
entier gmmktime(h, m, s, mois, j, a)
```

- comme mktime mais Greenwich

```
entier mktime(h, m, s, mois, j, a)
```

- retourne la valeur Timestamp qui correspond aux arguments

```
entier time()
```

- retourne la valeur de Timestamp

5. Pratique

5.1. Etape 1 :

Présentation des outils
Offline sur machine
Online via midl.free.fr
Protocole FTP
Organisation par étudiant -> répertoire individuel
Page PHP : index.php présentation projet + lien vers parser.php
Page PHP : parser.php formulaire entrée texte auto-envoyé

5.2. Etape 2 :

Exercice du Damier

5.3. Etape 3 :

Exercice de la Boule Magique

5.4. Etape 4 :

Introduction de sql
Exercice de Zipf en tableaux

5.5. Etape 5 :

Introduction des graphiques
Exercice de Zipf en graphique

5.6. Etape 6 :

Introduction des fichiers
Exercice de blog