Cours introductif au PHP

Riad MOKADEM

Qu'est-ce que PHP?

PHP est un langage interprété (un langage de script) exécuté du côté serveur (comme les scripts CGI, ASP, ...) et non du côté client (un script écrit en Javascript ou une applet Java s'exécute sur votre ordinateur...). La syntaxe du langage provient de celles du langage C, du Perl et de Java. Ses principaux atouts sont :

- Une grande communauté de développeurs partageant des centaines de milliers d'exemples de script PHP ;
- La gratuité et la disponibilité du code source (PHP est distribué sous licence GNU GPL) ;
- La simplicité d'écriture de scripts ;
- La possibilité d'inclure le script PHP au sein d'une page HTML (contrairement aux scripts CGi, pour lesquels il faut écrire des lignes de code pour afficher chaque ligne en langage HTML) ;
- La simplicité d'interfaçage avec des bases de données (de nombreux SGBD sont supportés, mais le plus utilisé avec ce langage est *MySQL*, un SGBD gratuit disponible sur de nombreuses plateformes : Unix, Linux, Windows, MacOs X, Solaris, etc...) ;
- L'intégration au sein de nombreux serveurs web (Apache, Microsoft IIS, etc.).

Origines de PHP

Le langage PHP a été mis au point au début d'automne 1994 par Rasmus Lerdorf. Ce langage de script lui permettait de conserver la trace des utilisateurs venant consulter son CV sur son site, grâce à l'accès à une base de données par l'intermédiaire de requêtes SQL. Ainsi, étant donné que de nombreux internautes lui demandèrent ce programme, Rasmus Lerdorf mit en ligne en 1995 la première version de ce programme qu'il baptisa *Personal Sommaire Page Tools*, puis *Personal Home Page v1.0* (traduisez *page personnelle version 1.0*).

Etant donné le succès de PHP 1.0, Rasmus Lerdorf décida d'améliorer ce langage en y intégrant des structures plus avancées telles que des boucles, des structures conditionnelles, et y intégra un package permettant d'interpréter les formulaires qu'il avait développé (*FI*, Form Interpreter) ainsi que le support de mSQL. C'est de cette façon que la version 2 du langage, baptisée pour l'occasion *PHP/FI version 2*, vit le jour durant l'été 1995. Il fut rapidement utilisé sur de nombreux sites (15000 fin 1996, puis 50000 en milieu d'année 1997).

A partir de 1997, Zeev Suraski et Andi Gurmans rejoignirent Rasmus pour former une équipe de programmeurs afin de mettre au point PHP 3 (Stig Bakken, Shane Caraveo et Jim Winstead les rejoignirent par la suite). C'est ainsi que la version 3.0 de PHP fut disponible le 6 juin 1998.

A la fin de l'année 1999 la version 4.0 de PHP, baptisée PHP4, est apparue. PHP en est aujourd'hui à sa cinquième version.

SGBD supportés par PHP

PHP permet un interfaçage simple avec de nombreux systèmes de gestion de bases de données (SGBD), parmi lesquels :

- Adabas D
- dBase
- Empress
- FilePro
- Informix
- Interbase
- mSQL
- MySQL
- Oracle
- PostgreSQL

- Solid
- Sybase
- Velocis
- Unix dbm

L'interprétation du code par le serveur

Un script PHP est un simple fichier texte contenant des instructions écrites à l'aide de caractères ASCII 7 bits (des caractères non accentués) incluses dans un code HTML à l'aide de balises spéciales et stocké sur le serveur. Ce fichier doit avoir l'extension « .php » pour pouvoir être interprété par le serveur.
Ainsi, lorsqu'un navigateur (le client) désire accéder à une page dynamique réalisé en PHP :

- le serveur reconnait l'extension d'un fichier PHP et le transmet à l'interpréteur PHP
- Dès que l'interpréteur rencontre une balise indiquant que les lignes suivantes sont du code PHP, il ne lit plus les instructions: il les exécute!
- L'interpréteur exécute l'instruction puis envoie les sorties éventuelles au serveur
- A la fin du script, le serveur transmet le résultat au client (le navigateur).

Un script PHP est interprété par le serveur, les utilisateurs ne peuvent donc pas voir le code source!

Le code PHP stocké sur le serveur n'est donc **jamais** visible directement par le client puisque dès qu'il en demande l'accès, le serveur l'interprète!
De cette façon aucune modification n'est à apporter sur les navigateurs.

Implantation au sein du code HTML

Pour que le script soit interprété par le serveur deux conditions sont nécessaires :

- Le fichier contenant le code doit avoir l'extension telle que *.php* et non *.html*
- Le code PHP contenu dans le code HTML doit être délimité par des balises du type *<?* et *?>*

Un script PHP doit :
- comporter l'extension *.php*
- être imbriqué entre les délimiteurs *<?* et *?>*

Pour des raisons de conformité avec certaines normes (XML et ASP par exemple), plusieurs balises peuvent être utilisées pour délimiter un code PHP :

1. *<?* et *?>*
2. *<?php* et *?>*
3. *<script language="php">* et *</script>*
4. *<%php* et *%>*

Un exemple de script simple

Voici ci-dessous l'exemple classique de script PHP :

```
<html>
<head><title>Exemple</title></head>
<body>
<?php
        echo "Hello world";
?>
</body>
</html>
```

On notera bien évidemment que la fonction *echo* permet d'afficher sur le navigateur la chaine délimitée par les guillemets.

L'interprétation du code

Un code PHP (celui compris entre les délimiteurs <?php et ?>) est un ensemble d'instructions se terminant chacune par un point-virgule (comme en langage C). Lorsque le code est interprété, les espaces, retours

chariot et tabulation ne sont pas pris en compte par le serveur. Il est tout de même conseillé d'en mettre (ce n'est pas parce qu'ils ne sont pas interprétés que l'on ne peut pas les utiliser) afin de rendre le code plus lisible (pour vous, puisque les utilisateurs ne peuvent lire le code source: il est interprété).

Les commentaires

Une autre façon de rendre le code plus compréhensible consiste à insérer des commentaires, des lignes qui seront tout simplement ignorées par le serveur lors de l'interprétation.

Pour ce faire, il est possible, comme en langage C, d'utiliser des balises qui vont permettre de délimiter les explications afin que l'interpréteur les ignore et passe directement à la suite du fichier.
Ces délimiteurs sont /* et */. Un commentaire sera donc noté de la façon suivante :

```
/* Voici un commentaire! */
```

Il y a toutefois quelques règles à respecter :

- Les commentaires peuvent être placés n'importe où à l'intérieur des délimiteurs de script PHP
- Les commentaires ne peuvent contenir le délimiteur de fin de commentaire (*/)
- Les commentaires ne peuvent être imbriqués
- Les commentaires peuvent être écrits sur plusieurs lignes
- Les commentaires ne peuvent pas couper un mot du code en deux

Il est possible aussi d'utiliser un type de commentaire permettant de mettre toute la fin d'une ligne en commentaire en utilisant le double *slash* (//). Tout ce qui se situe à droite de ce symbole sera mis en commentaire.

Typologie

La manière d'écrire les choses en langage PHP a son importance. Le langage PHP est par exemple sensible à la casse (en anglais *case sensitive*), cela signifie qu'un nom contenant des majuscules est différent du même nom écrit en minuscules. Toutefois, cette règle ne s'applique pas aux fonctions, les spécifications du langage PHP précisent que la fonction *print* peut être appelée *print()*, *Print()* ou *PRINT()*.
Enfin, toute instruction se termine par un point-virgule.

Introduction

Un serveur web est un logiciel permettant de rendre accessibles à de nombreux ordinateurs (les clients) des pages web stockées sur le disque. Cette fiche pratique explique comment installer le serveur web Apache sur un système de type UNIX (typiquement une distribution de Linux telle que **RedHat**, **Mandrake** ou n'importe quelle autre).

Pour cela quelques connaissances sur Linux ou bien Unix sont nécessaires. Le but de cette fiche va être d'être capable de récupérer les sources des différents éléments nécessaires et de les compiler (un compilateur C est donc nécessaire, il est généralement installé par défaut sur la plupart des distributions Linux) afin d'avoir un système opérationnel.

L'installation suivante comprend l'installation de l'interpréteur PHP, un langage de programmation permettant de créer des pages créées dynamiquement, ainsi que le SGBD MySQL, un système de gestion de bases de données relationnelles puissant fonctionnant sous Linux (et gratuit!).

Télécharger les sources

- Les sources de PHP peuvent être téléchargées sur le site http://www.php.net
- Les sources de Apache peuvent être téléchargées sur le site http://www.apache.org
- Les sources de MySQL peuvent être téléchargées sur le site http://www.mysql.org

installer Apache et PHP

1. Décompresser les archives :
2. ```
   tar zxvf apache_1.3.x.tar.gz
   tar zxvf php-3.0.x.tar
   ```
3. Configurer Apache
   ```

4. `cd apache_1.3.x`
   `./configure --prefix=/www`
5. Configurer PHP
6. `cd ../php-3.0.x`
   `./configure --with-mysql --with-apache=../apache_1.3.x --enable-track-vars`
   Si vous préférez installer PHP dans un autre répertoire, il faut utiliser l'option de configuration --with-config-file-path=/path
7. Compiler PHP
8. `make`
   `make install`
9. Installer Apache
10. `cd ../apache_1.3.x`
11. `./configure --prefix=/www --activate-module=src/modules/php3/libphp3.a`
12. `make`
    `make install`
13. Modifier le fichier de configuration de PHP
14. `cd ../php-3.0.x`
    `cp php3.ini-dist /usr/local/lib/php3.ini`
    Vous pouvez désormais éditer le fichier de configuration /usr/local/lib/php3.ini.
15. Editez le fichier de configuration du serveur apache (généralement *httpd.conf* ou *srm.conf* et ajoutez la ligne suivante :
    `AddType application/x-httpd-php3 .php3`

    Il s'agit de choisir l'extension associée aux scripts PHP. Par souci d'homogénéité, il est courant de choisir l'extension **.php3**
16. Démarrez le serveur Apache. (Il est essentiel d'arrêter et redémarrer le serveur, et non uniquement de le relancer. Il suffit généralement de taper *apachectl stop*, puis *apachectl start*).

Premier lancement

Pour vérifier si l'installation a bien fonctionnée, il vous suffit de créer un petit fichier dans la racine des documents du serveur web (appelée *DocumentRoot* dans le fichier de configuration *httpd.conf*). Nommez ce fichier *toto.php3*, et mettez le code suivant dans ce fichier :

```
<html>

<head><title>Exemple</title></head>

<body>

<?php
echo "PHP fonctionne!";

?>

</body>

</html>
```

Lancez un navigateur sur cette machine et entrez l'URL suivante :

`http://localhost/toto.php3`

localhost désigne la machine sur laquelle vous vous trouvez...

Vous devriez logiquement voir apparaître la phrase *"PHP fonctionne!"* sur votre navigateur !

Introduction à EasyPHP

Afin de faire fonctionner PHP, il est nécessaire à la base d'en télécharger les sources depuis un site spécialisé (par exemple PHP.net), puis de compiler celui-ci (ainsi que d'éditer les liens) afin de créer un fichier exécutable.

Ce processus demande des notions avancées en informatique, c'est pourquoi trois adeptes de PHP (Emmanuel Faivre, Laurent Abbal et Thierry Murail) ont mis au point un package (appelé *EasyPHP*) contenant 3 produits incontournables de la scène PHP :

- Le serveur Web Apache
- Le moteur de scripts PHP4
- La base de données MySQL
- Un outil de gestion de base de donnée graphique, Phpmyadmin

EasyPHP est ainsi un pack fonctionnant sous Windows permettant d'installer en un clin d'oeil les éléments nécessaires au fonctionnement d'un site web dynamique développé en PHP

Récupérer EasyPHP

Le pack EasyPHP est disponible sur les sites suivants :

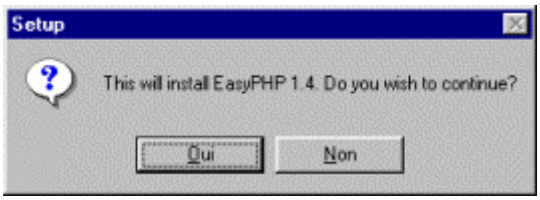
- www.manucorp.com
- www.easyphp.org

Il vous suffit dans un premier temps de télécharger la version la plus récente de EasyPHP. Vous pouvez la télécharger à cette adresse :
Page de téléchargement de EasyPhP

Installer EasyPHP

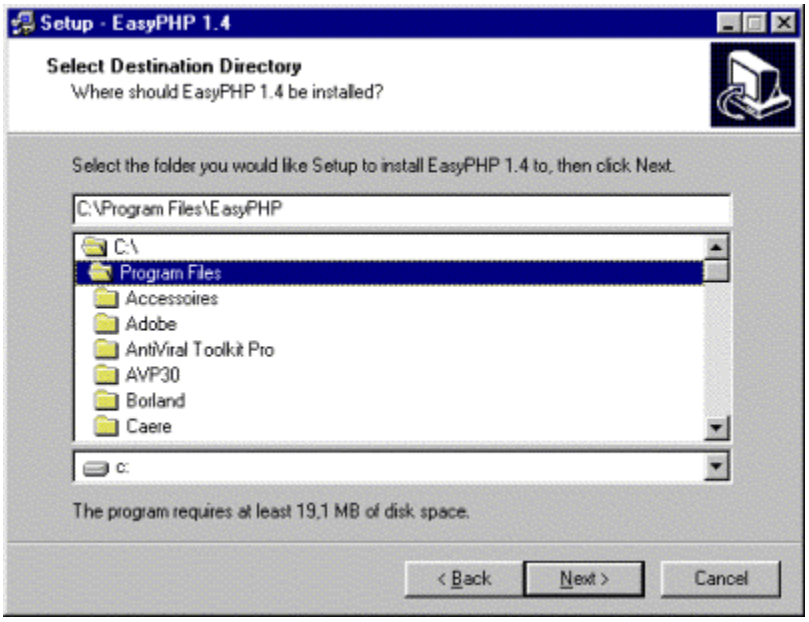
L'installation de EasyPHP est très simple, notamment avec l'apparition de la version 1.4 comportant un installeur automatique.

Pour installer EasyPHP, il vous suffit dans un premier temps de double-cliquer sur le fichier téléchargé précédemment :

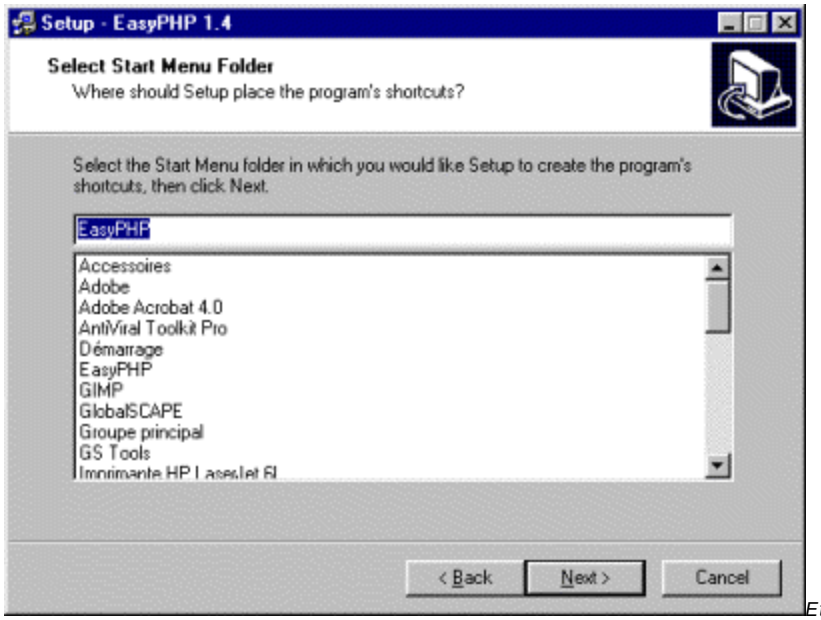


L'écran d'installation de EasyPHP suivant devrait apparaître, cliquez sur *Next* (*Suivant*</) :

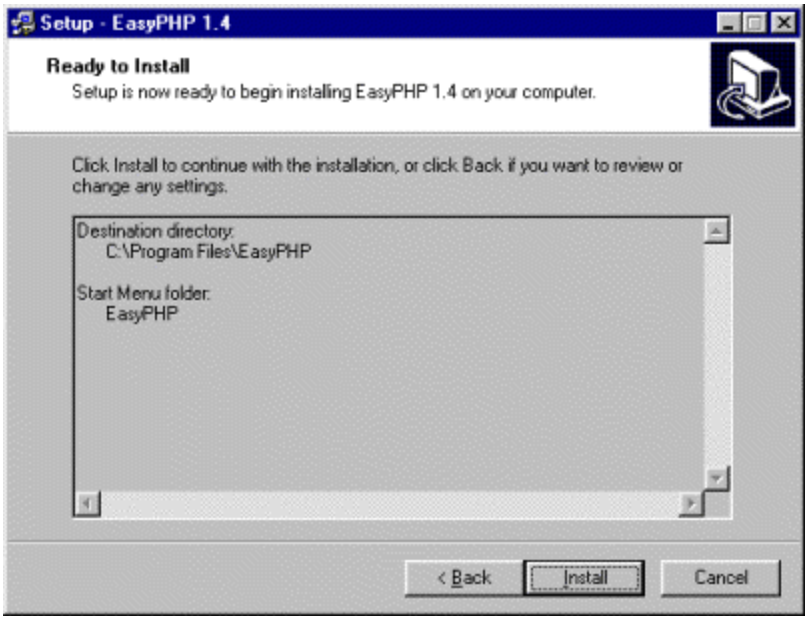
L'installeur va ensuite vous demander de préciser le répertoire d'installation :



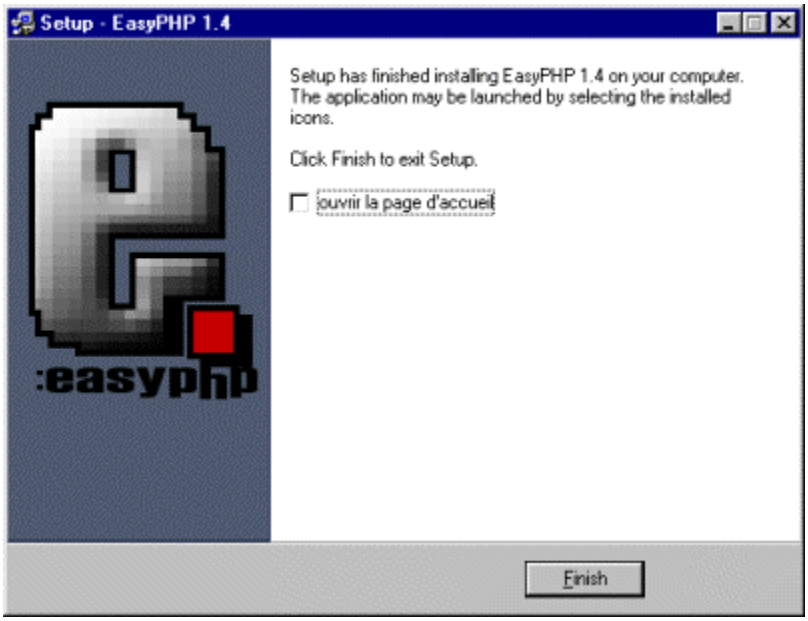
Puis il va demander la création d'un groupe dans le menu démarrer



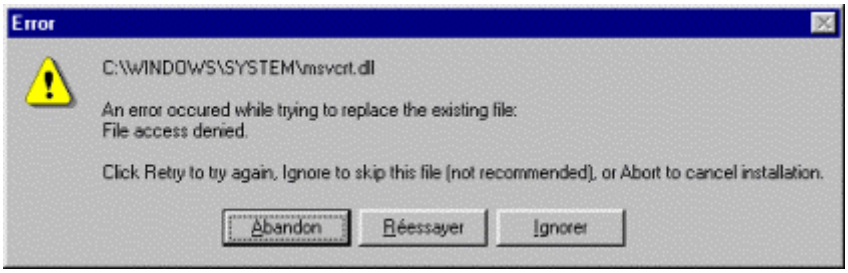
Et enfin il va vous récapituler les éléments de l'installation avant de procéder à la copie des fichiers.



Après la copie des fichiers, EasyPHP vous présente l'écran suivant indiquant que l'installation s'est déroulée correctement



Il se peut lors de l'installation que l'installeur vous indique l'erreur suivante :



Cette erreur indique que la librairie msvcrt.dll n'a pû être copiée. La raison de cette erreur provient du fait que votre système Windows est actuellement en train d'utiliser cette libraire et ne peut donc

l'écraser.

Pour y remédier, copiez cette librairie (par exemple dans c:\) sur votre disque dur (cliquez ici pour télécharger la librairie pour Windows 9x), puis redémarrez en mode MS-DOS, puis tapez *copy c:\msvcrt.dll c:\windows\system*. Le système va vous demander de confirmer cet écrasement répondez "Oui" (Y ou O), redémarrez Windows et EasyPHP devrait fonctionner !

Démarrage de EasyPHP

Pour démarrer Apache, MySQL et PHP, il vous suffit de lancer EasyPHP à partir du groupe créé dans le menu démarrer :

Pour vérifier si EasyPHP fonctionne, il vous suffit de taper dans votre navigateur préféré :

- *http://localhost*
- *ou http://127.0.0.1*

Les deux adresses ci-dessus représentant votre machine locale.

Editer votre site

Pour créer votre site web dynamique avec EasyPHP, il vous suffit de déposer vos créations dans le sous-répertoire /www de EasyPHP.

Par exemple créez un fichier texte contenant le texte suivant :

```
<?
phpinfo();

?>
```

Puis renommez ce fichier en phpinfo.php3 et déposez-le dans le sous-répertoire /www.
Vous pouvez désormais visualiser le résultat à l'adresse suivante :
http://localhost/phpinfo.php3

Plus d'informations

Pour plus d'informations ou en cas de problème avec EasyPHP, allez sur EasyPHP.org. En cas de problème, commencez par consulter la FAQ, puis consultez le Forum

Concept de variable avec PHP

Une variable est un objet repéré par son nom, pouvant contenir des données, qui pourront être modifiées lors de l'exécution du programme. Les variables en langage PHP peuvent être de trois types :

- scalaires
- tableaux
- tableaux associatifs

Quelque soit le type de variable, son nom doit obligatoirement être précédé du caractère dollar ($).
Contrairement à de nombreux langages de programmation, comme le langage C, les variables en PHP n'ont pas besoin d'être déclarées, c'est-à-dire que l'on peut commencer à les utiliser sans en avoir averti l'interpréteur précédemment, ainsi si la variable existait précédemment, son contenu est utilisé, sinon l'interpréteur lui affectera la valeur en lui assignant 0 par défaut. De cette façon si vous ajoutez 3 à une nouvelle variable (non définie plus haut dans le code), sa valeur sera 3...

Nommage des variables

Avec PHP, les noms de variables doivent répondre à certains critères :

- un nom de variable doit commencer par une lettre (majuscule ou minuscule) ou un "_" (pas par un chiffre)
- un nom de variable peut comporter des lettres, des chiffres et le caractère _ (les espaces ne sont pas autorisés!)

Nom de variable correct	Nom de variable incorrect	Raison
$Variable	$Nom de Variable	comporte des espaces
$Nom_De_Variable	$123Nom_De_Variable	commence par un chiffre
$nom_de_variable	$toto@mailcity.com	caractère spécial @
$nom_de_variable_123	$Nom-de-variable	signe - interdit
$nom_de_variable	nom_de_variable	ne commence pas par $

Les noms de variables sont sensibles à la casse (le langage PHP fait la différence entre un nom en majuscule et un nom en minuscules), il faut donc veiller à utiliser des noms comportant la même casse! Toutefois, les noms de fonctions font exception à cette règle...

Variables scalaires

Le langage PHP propose trois types de variables scalaires :

- entiers: nombres naturels sans décimale (sans virgule)
- réels: nombres décimaux (on parle généralement de type *double*, car il s'agit de nombre décimaux à double précision)
- chaines de caractères: ensembles de caractères

Il n'est pas nécessaire en PHP de typer les variables, c'est-à-dire de définir leur type, il suffit de leur assigner une valeur pour en définir le type :

- entiers: nombre sans virgule
- réels: nombres avec une virgule (en réalité un point)
- chaines de caractères: ensembles de caractères entre guillemets simples ou doubles

Instruction	Type de la variable
$Variable = 0;	type entier
$Variable = 12;	type entier
$Variable = 0.0;	type réel
$Variable = 12.0;	type réel
$Variable = "0.0";	type chaîne
$Variable = "Bonjour tout le monde";	type chaîne

Il existe des caractères repérés par un code ASCII spécial permettant d'effectuer des opérations particulières. Ces caractères peuvent être représentés plus simplement en langage PHP grâce au caractère '\' suivi d'une lettre, qui précise qu'il s'agit d'un caractère de contrôle :

Caractère	Description
\"	guillemet
\\	barre oblique inverse (antislash)
\r	retour chariot
\n	retour à la ligne
\t	tabulation

En effet, certains de ces caractères ne pourraient pas être représentés autrement (un retour à la ligne ne peut pas être représenté à l'écran). D'autre part, les caractères \ et " ne peuvent pas faire partie en tant que tel d'une chaîne de caractère, pour des raisons évidente d'ambiguité...

Variables tableaux

Les variables, telles que nous les avons vues, ne permettent de stocker qu'une seule donnée à la fois. Or, pour de nombreuses données, comme cela est souvent le cas, des variables distinctes seraient beaucoup trop lourdes à gérer. Heureusement, PHP propose des structures de données permettant de stocker l'ensemble de ces données dans une "variable commune". Ainsi, pour accéder à ces valeurs il suffit de parcourir la variable de type complexe composée de « variables » de type simple.

Les tableaux stockent des données sous forme de liste. Les données contenues dans la liste sont accessibles grâce à un index (un numéro représentant l'élément de la liste). Contrairement à des langages tels que le langage C, il est possible de stocker des éléments de types différents dans un même tableau.

Ainsi, pour désigner un élément de tableau, il suffit de faire suivre au nom du tableau l'indice de l'élément entre crochets :

```
$Tableau[0] = 12;
$Tableau[1] = "CCM";
```

Avec PHP, il n'est pas nécessaire de préciser la valeur de l'index lorsque l'on veut remplir un tableau, car il assigne la valeur 0 au premier élément (si le tableau est vide) et incrémente les indices suivants. De cette façon, il est facile de remplir un tableau avec des valeurs. Le code précédent est équivalent à:

```
$Tableau[] = 12;
$Tableau[] = "CCM";
```

- Les indices de tableau commencent à zéro
- tous les types de variables peuvent être contenus dans un tableau

Lorsqu'un tableau contient d'autres tableaux, on parle de tableaux multidimensionnels. Il est possible de créer directement des tableaux multidimensionnels en utilisant plusieurs paires de crochets pour les index (autant de paires de crochets que la dimension voulue). Par exemple, un tableau à deux dimensions pourra être déclaré comme suit :

```
$Tableau[0][0] = 12;
$Tableau[0][1] = "CCM";
$Tableau[1][0] = 1245.652;
$Tableau[1][1] = "Au revoir";
```

Variables tableaux associatifs

PHP permet l'utilisation de chaînes de caractères au lieu de simples entiers pour définir les indices d'un tableau, on parle alors de *tableaux associatifs*. Cette façon de nommer les indices peut parfois être plus agréable à utiliser :

```
$Toto["Age"] = 12;
$Toto["Adresse"] = "22 rue des bois fleuris";
$Toto["Nom"] = "Ah, vous auriez bien aimé
connaître le nom de famille de Toto...";
```

Portée (visibilité) des variables

Selon l'endroit où on déclare une variable, celle-ci pourra être accessible (visible) de partout dans le code ou bien que dans une portion confinée de celui-ci (à l'intérieur d'une fonction par exemple), on parle de *portée* (ou *visibilité*) d'une variable.

Lorsqu'une variable est déclarée dans le code même, c'est-à-dire à l'extérieur de toute fonction ou de tout bloc d'instructions, elle est accessible de partout dans le code (n'importe quelle fonction du programme peut faire appel à cette variable). On parle alors de **variable globale**

Lorsque l'on déclare une variable à l'intérieur d'un bloc d'instructions (entre des accolades), sa portée se confine à l'intérieur du bloc dans lequel elle est déclarée.

- Une variable déclarée au début du code, c'est-à-dire avant tout bloc de donnée, sera globale, on pourra alors les utiliser à partir de n'importe quel bloc d'instructions
- Une variable déclarée à l'intérieur d'un bloc d'instructions (dans une fonction ou une boucle par exemple) aura une portée limitée à ce seul bloc d'instructions, c'est-à-dire qu'elle est inutilisable ailleurs, on parle alors de variable locale

D'une manière générale il est préférable de donner des noms différents aux variables locales et globales pour des raisons de lisibilité et de compréhension du code.

Définition de constantes

Une constante est une variable dont la valeur est inchangeable lors de l'exécution d'un programme. Avec PHP, les constantes sont définies grâce à la fonction *define()*. la syntaxe de la fonction *define()* est la suivante :

```
define("Nom_de_la_variable", Valeur);
```

Le nom d'une constante définie à l'aide de la fonction *define()* ne doit pas commencer par le caractère $ (de cette façon aucune affectation n'est possible).

Qu'est-ce qu'un opérateur?

Les opérateurs sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations, les évaluer, ...
On distingue plusieurs types d'opérateurs :

- les opérateurs de calcul
- les opérateurs d'assignation
- les opérateurs d'incrémentation
- les opérateurs de comparaison
- les opérateurs logiques
- (les opérateurs bit-à-bit)
- (les opérateurs de rotation de bit)

Les opérateurs de calcul

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable

Opérateur	Dénomination	Effet	Exemple	Résultat (pour *x=7*)
+	opérateur d'addition	Ajoute deux valeurs	$x+3	10
-	opérateur de soustraction	Soustrait deux valeurs	$x-3	4
*	opérateur de multiplication	Multiplie deux valeurs	$x*3	21
/	plus: opérateur de division	Divise deux valeurs	$x/3	2.3333333
=	opérateur d'affectation	Affecte une valeur à une variable	$x=3	Met la valeur 3 dans la variable $x
%	opérateur modulo	Donne le reste de la division entière entre 2 nombres	$x%3	1

Les opérateurs d'assignation

Ces opérateurs permettent de simplifier des opérations telles que *ajouter une valeur dans une variable et stocker le résultat dans la variable*. Une telle opération s'écrirait habituellement de la façon suivante par exemple: *$x=$x+2*
Avec les opérateurs d'assignation il est possible d'écrire cette opération sous la forme suivante: *$x+=2*
Ainsi, si la valeur de x était 7 avant opération, elle sera de 9 après...

Les autres opérateurs du même type sont les suivants :

Opérateur	Effet
+ =	addition deux valeurs et stocke le résultat dans la variable (à gauche)
- =	soustrait deux valeurs et stocke le résultat dans la variable
*** =**	multiplie deux valeurs et stocke le résultat dans la variable
/ =	divise deux valeurs et stocke le résultat dans la variable
% =	donne le reste de la division deux valeurs et stocke le résultat dans la variable
| =	Effectue un OU logique entre deux valeurs et stocke le résultat dans la variable
^ =	Effectue un OU exclusif entre deux valeurs et stocke le résultat dans la variable
& =	Effectue un Et logique entre deux valeurs et stocke le résultat dans la variable
. =	Concatène deux chaînes et stocke le résultat dans la variable

Les opérateurs d'incrémentation

Ce type d'opérateur permet de facilement augmenter ou diminuer d'une unité une variable. Ces opérateurs sont très utiles pour des structures telles que des boucles, qui ont besoin d'un compteur (variable qui augmente de un en un).

Un opérateur de type *$x++* permet de remplacer des notations lourdes telles que *$x=$x+1* ou bien *$x+=1*

Opérateur	Dénomination	Effet	Syntaxe	Résultat (avec x valant 7)
++	Incrémentation	Augmente d'une unité la variable	$x++	8
--	Décrémentation	Diminue d'une unité la variable	$x--	6

Les opérateurs de comparaison

Opérateur	Dénomination	Effet	Exemple	Résultat
== **A ne pas confondre avec le signe d'affectation (=)!!**	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité	$x==3	Retourne 1 si $x est égal à 3, sinon 0
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur	$x<3	Retourne 1 si $x est inférieur à 3, sinon 0
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur	$x<=3	Retourne 1 si $x est inférieur ou égale à 3, sinon 0
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur	$x>3	Retourne 1 si $x est supérieur à 3, sinon 0
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur	$x>=3	Retourne 1 si $x est supérieur ou égal à 3, sinon 0
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur	$x!=3	Retourne 1 si $x est différent de 3, sinon 0

Les opérateurs logiques (booléens)

Ce type d'opérateur permet de vérifier si plusieurs conditions sont vraies :

Opérateur	Dénomination	Effet	Syntaxe
|| ou **OR**	OU logique	Vérifie qu'une des conditions est réalisée	((condition1)||(condition2))
&& ou **AND**	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&(condition2))
XOR	OU exclusif	Vérifie qu'une et une seule des conditions est réalisée	((condition1)XOR(condition2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

(Les opérateurs bit-à-bit)

Si vous ne comprenez pas ces opérateurs cela n'est pas important, vous n'en aurez probablement pas l'utilité. Pour ceux qui voudraient comprendre, rendez- vous aux chapitres suivants :

- compréhension du binaire
- représentation des données
- Instructions arithmétiques et logiques en assembleur

Ce type d'opérateur traite ses opérandes comme des données binaires, plutôt que des données décimales, hexadécimales ou octales. Ces opérateurs traitent ces données selon leur représentation binaire mais retournent des valeurs numériques standard dans leur format d'origine.

Les opérateurs suivants effectuent des opérations bit-à-bit, c'est-à-dire avec des bits de même poids.

Opérateur	Dénomination	Effet	Syntaxe	Résultat
&	ET bit-à-bit	Retourne 1 si les deux bits de même poids sont à 1	9 & 12 (1001 & 1100)	8 (1000)

|	OU bit-à-bit	Retourne 1 si l'un ou l'autre des deux bits de même poids est à 1 (ou les deux)	9 | 12 (1001 | 1100)	13 (1101)
^	OU exclusif	Retourne 1 si l'un des deux bits de même poids est à 1 (mais pas les deux)	9 ^ 12 (1001 ^ 1100)	5 (0101)
~	Complément (NON)	Retourne 1 si le bit est à 0 (et inversement)	~9 (~1001)	6 (0110)

(Les opérateurs de rotation de bit)

Si vous ne comprenez pas ces opérateurs cela n'est pas important, vous n'en aurez probablement pas l'utilité. Pour ceux qui voudraient comprendre, rendez- vous aux chapitres suivants :

- compréhension du binaire
- représentation des données
- Instructions arithmétiques et logiques en assembleur

Ce type d'opérateur traite ses opérandes comme des données binaires d'une longueur de 32 bits, plutôt que des données décimales, hexadécimales ou octales. Ces opérateurs traitent ces données selon leur représentation binaire mais retournent des valeurs numériques standards dans leur format d'origine.

Les opérateurs suivants effectuent des rotations sur les bits, c'est-à-dire qu'il décale chacun des bits d'un nombre de bits vers la gauche ou vers la droite. La première opérande désigne la donnée sur laquelle on va faire le décalage, la seconde désigne le nombre de bits duquel elle va être décalée.

Opérateur	Dénomination	Effet	Syntaxe	Résultat
<<	Rotation à gauche	Décale les bits vers la gauche (multiplie par 2 à chaque décalage). Les zéros qui sortent à gauche sont perdus, tandis que des zéros sont insérés à droite	6 << 1 (110 << 1)	12 (1100)
>>	Rotation à droite avec conservation du signe	Décale les bits vers la droite (divise par 2 à chaque décalage). Les zéros qui sortent à droite sont perdus, tandis que le bit non-nul de poids plus fort est recopié à gauche	6 >> 1 (0110 >> 1)	3 (0011)

Autres opérateurs

Les opérateurs ne peuvent pas être classés dans une catégorie spécifique mais ils ont tout de même chacun leur importance!

Opérateur	Dénomination	Effet	Syntaxe	Résultat
.	Concaténation	Joint deux chaînes bout à bout	"Bonjour"."Au revoir"	"BonjourAu revoir"
$	Référencement de variable	Permet de définir une variable	$MaVariable = 2;	
->	Propriété d'un objet	Permet d'accéder aux données membres d'une classe	$MonObjet->Propriete	

Les priorités

Lorsque l'on associe plusieurs opérateurs, il faut que l'interpréteur PHP sache dans quel ordre les traiter, voici donc dans **l'ordre décroissant** les priorités de tous les opérateurs :

Priorité des opérateurs												
+ + + + + + + + + + + + + + +	()	[]										
+ + + + + + + + + + + + + +	--	+ +	!	~	-							
+ + + + + + + + + + + + +	*	/	%									
+ + + + + + + + + + + +	+	-										
+ + + + + + + + + + +	<	< =	> =	>								
+ + + + + + + + + +	= =	! =										
+ + + + + + + + +	&											
+ + + + + + + +	^											
+ + + + + + +	|											

+ + + + + +	&&											
+ + + + +	||											
+ + + +	?	:										
+ + +	=	+ =	- =	* =	/ =	% =	< < =	> > =	> > > =	& =	^ =	| =
+ +	AND											
+	XOR											

Qu'est-ce qu'une structure conditionnelle?

On appelle ***structure conditionnelle*** les instructions qui permettent de tester si une condition est vraie ou non, c'est-à-dire si la valeur de son expression vaut 0 ou 1 (le PHP associe le mot clé *true* à 1 et false à 0). Ces structures conditionnelles peuvent être associées à des structures qui se répètent suivant la réalisation de la condition, on appelle ces structures des ***structures de boucle***

La notion de bloc

Une expression suivie d'un point-virgule est appelée instruction. Par exemple *a++;* est une instruction. Lorsque l'on veut regrouper plusieurs instructions, on peut créer ce que l'on appelle un **bloc**, c'est-à-dire un ensemble d'instructions (suivies respectivement par des point-virgules) et comprises entre les accolades { et }.

Les instructions *if*, *while* et *for* peuvent par exemple être suivies d'un bloc d'instructions à exécuter...

L'instruction if

L'instruction if est la structure de test la plus basique, on la retrouve dans tous les langages (avec une syntaxe différente...). Elle permet d'exécuter une série d'instruction si jamais une condition est réalisée.

La syntaxe de cette expression est la suivante :

```
if (condition réalisée) {
   liste d'instructions
}
```

Remarques:

- la condition doit être entre des parenthèses
- il est possible de définir plusieurs conditions à remplir avec les opérateurs ET et OU (&& et ||) par exemple l'instruction suivante teste si les deux conditions sont vraies :
  `if ((condition1)&&(condition2))`
  L'instruction ci-dessous exécutera les instructions si l'une ou l'autre des deux conditions est vraie :
  `if ((condition1)||(condition2))`
- s'il n'y a qu'une instruction, les accolades ne sont pas nécessaires...

L'instruction if ... else

L'instruction *if* dans sa forme basique ne permet de tester qu'une condition, or la plupart du temps on aimerait pouvoir choisir les instructions à exécuter **en cas de non réalisation de la condition**...
L'expression *if ... else* permet d'exécuter une autre série d'instruction en cas de non-réalisation de la condition.

La syntaxe de cette expression est la suivante :

```
if (condition réalisée) {
   liste d'instructions
}

else {
     autre série d'instructions
}
```

L'instruction if ... elseif ... else

L'instruction *if ... else* ne permet de tester qu'une condition, or il est parfois nécessaire de tester plusieurs conditions de façon exclusive, c'est-à-dire que sur toutes les conditions une seule sera réalisée ...
L'expression *if ... elseif ... else* permet d'enchaîner une série d'instructions et évite d'avoir à imbriquer des instructions *if*.

La syntaxe de cette expression est la suivante :

```
if (condition réalisée) {
   liste d'instructions
}

elseif (autre condition réalisée) {
       autre série d'instructions
}

 ...
else (dernière condition réalisée) {
     série d'instructions
}
```

une façon plus courte de faire un test (opérateur ternaire)

Il est possible de faire un test avec une structure beaucoup moins lourde grâce à la structure suivante, appelée opérateur ternaire :

```
(condition) ? instruction si vrai : instruction si faux
```

Remarques:

- la condition doit être entre des parenthèses
- Lorsque la condition est vraie, l'instruction de gauche est exécutée
- Lorsque la condition est fausse, l'instruction de droite est exécutée

L'instruction switch

L'instruction *switch* permet de faire plusieurs tests de valeurs sur le contenu d'une même variable. Ce branchement conditionnel simplifie beaucoup le test de plusieurs valeurs d'une variable, car cette opération aurait été compliquée (mais possible) avec des *if* imbriqués. Sa syntaxe est la suivante :

```
switch (Variable) {

case Valeur1 :
          Liste d'instructions
          break;

case Valeur2 :
          Liste d'instructions
          break;

case Valeurs... :
          Liste d'instructions
          break;

default:

          Liste d'instructions
          break;

}
```

Les parenthèses qui suivent le mot clé *switch* indiquent une expression dont la valeur est testée successivement par chacun des *case*. Lorsque l'expression testée est égale à une des valeurs suivant un *case*, la liste d'instructions qui suit celui-ci est exécutée. Le mot clé *break* indique la sortie de la structure conditionnelle. Le mot clé *default* précède la liste d'instructions qui sera exécutée si l'expression n'est jamais égale à une des valeurs.

N'oubliez pas d'insérer des instructions *break* entre chaque test, ce genre d'oubli est difficile à détecter car aucune erreur n'est signalée...

Les boucles

Les boucles sont des structures qui permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce qu'une condition ne soit plus réalisée...
On appelle parfois ces structures *instructions répétitives* ou bien *itérations*.
La façon la plus commune de faire une boucle, est de créer un compteur (une variable qui s'incrémente, c'est-à-dire qui augmente de 1 à chaque tour de boucle) et de faire arrêter la boucle lorsque le compteur dépasse une certaine valeur.

La boucle for

L'instruction *for* permet d'exécuter plusieurs fois la même série d'instructions: c'est une boucle!

Dans sa syntaxe, il suffit de préciser le nom de la variable qui sert de compteur (et éventuellement sa valeur de départ, la condition sur la variable pour laquelle la boucle s'arrête (basiquement une condition qui teste si la valeur du compteur dépasse une limite) et enfin une instruction qui incrémente (ou décrémente) le compteur.

La syntaxe de cette expression est la suivante :

```
for (compteur; condition; modification du compteur) {
    liste d'instructions
}
```

Par exemple :

```
for ($i=1; $i<6; $i++) {
    echo "$i<br>";

}
```

Cette boucle affiche 5 fois la valeur de *$i*, c'est-à-dire 1,2,3,4,5
Elle commence à $i=1, vérifie que $i est bien inférieur à 6, etc... jusqu'à atteindre la valeur $i=6, pour laquelle la condition ne sera plus réalisée, la boucle s'interrompra et le programme continuera son cours.

D'autre part, le langage PHP autorise la déclaration de la variable de boucle dans l'instruction *for* elle-même!

Par exemple :

```
for ($i=0; $i<10; $i++) {
    echo "$i<br>";

}
```

- il faudra toujours vérifier que la boucle a bien une condition de sortie (i.e le compteur s'incrémente correctement)
- une instruction *echo* dans votre boucle est un bon moyen pour vérifier la valeur du compteur pas à pas en l'affichant!
- il faut bien compter le nombre de fois que l'on veut faire exécuter la boucle:
  - *for($i=0;$i<10;$i++)* exécute 10 fois la boucle ($i de 0 à 9)
  - *for($i=0;$i<=10;$i++)* exécute 11 fois la boucle ($i de 0 à 10)
  - *for($i=1;$i<10;$i++)* exécute 9 fois la boucle ($i de 1 à 9)
  - *for($i=1;$i<=10;$i++)* exécute 10 fois la boucle ($i de 1 à 10)

L'instruction while

L'instruction *while* représente un autre moyen d'exécuter plusieurs fois la même série d'instructions.

La syntaxe de cette expression est la suivante :

```
while (condition réalisée) {
      liste d'instructions
```

```
}
```

Cette instruction exécute la liste d'instructions **tant que** (*while* est un mot anglais qui signifie *tant que*) la condition est réalisée.

La condition de sortie pouvant être n'importe quelle structure conditionnelle, les risques de boucle infinie (boucle dont la condition est toujours vraie) sont grands, c'est-à-dire qu'elle risque de provoquer un plantage du navigateur!

Saut inconditionnel

Il peut être nécessaire de faire sauter à la boucle une ou plusieurs valeurs sans pour autant mettre fin à celle-ci.

La syntaxe de cette expression est "*continue;*" (cette instruction se place dans une boucle!), on l'associe généralement à une structure conditionnelle, sinon les lignes situées entre cette instruction et la fin de la boucle seraient obsolètes.

Exemple: Imaginons que l'on veuille imprimer pour $x allant de 1 à 10 la valeur de 1/($x-7) ... il est évident que pour $x=7 il y aura une erreur. Heureusement, grâce à l'instruction *continue* il est possible de traiter cette valeur à part puis de continuer la boucle!

```
$x=1;

while ($x<=10) {
      if ($x == 7) {
         echo "Division par zéro!";

         continue;

      }
$a = 1/($x-7);

echo "$a<br>";

$x++;

}
```

Il y avait une erreur dans ce programme... peut-être ne l'avez-vous pas vue :
Lorsque $x est égal à 7, le compteur ne s'incrémente plus, il reste constamment à la valeur 7, il aurait fallu écrire :

```
$x=1;

while ($x<=10) {
      if ($x == 7) {
         echo "division par 0";

         $x++;

         continue;

      }
      $a = 1/($x-7);

      echo "$a<br>";

     $x++;

}
```

Arrêt inconditionnel

A l'inverse, il peut être voulu d'arrêter prématurément la boucle, pour une autre condition que celle précisé dans l'en-tête de la boucle. L'instruction *break* permet d'arrêter une boucle (*for* ou bien *while*). Il s'agit, tout comme *continue*, de l'associer à une structure conditionnelle, sans laquelle la boucle ne ferait jamais plus d'un tour!

Dans l'exemple de tout à l'heure, par exemple si l'on ne savait pas à quel moment le dénominateur ($x-7) s'annule (bon...OK...pour des équations plus compliquées par exemple) il serait possible de faire arrêter la boucle en cas d'annulation du dénominateur, pour éviter une division par zéro!

```
for ($x=1; $x<=10; $x++) {
    $a = $x-7;

    if ($a == 0) {
       echo "division par 0";

       break;

    }
    echo "1/$a<br>";

}
```

Arrêt d'exécution du script

PHP autorise l'utilisation de la commande *exit*, qui permet d'interrompre totalement l'interprétation du script, ce qui signifie que le serveur n'envoie plus d'informations au navigateur: le script est figé dans son état actuel. cette instruction est particulièrement utile lors de l'apparition d'erreur

La notion de fonction

On appelle *fonction* un sous-programme qui permet d'effectuer un ensemble d'instructions par simple appel de la fonction dans le **corps** du programme principal. Les fonctions permettent d'exécuter dans plusieurs parties du programme une série d'instructions, cela permet une simplicité du code et donc une taille de programme minimale. D'autre part, une fonction peut faire appel à elle-même, on parle alors de fonction récursive (il ne faut pas oublier de mettre une condition de sortie au risque sinon de ne pas pouvoir arrêter le programme...).

La déclaration d'une fonction

PHP recèle de nombreuses fonctions intégrées permettant d'effectuer des actions courantes. Toutefois, il est possible de définir des fonctions, dites *fonctions utilisateurs* afin de simplifier l'exécution de séries d'instructions répétitives. Contrairement à de nombreux autres langages, PHP nécessite que l'on définisse une fonction avant que celle-ci puisse être utilisée, car pour l'appeler dans le corps du programme il faut que l'interpréteur la connaisse, c'est-à-dire qu'il connaisse son nom, ses arguments et les instructions qu'elle contient. La définition d'une fonction s'appelle "*déclaration"* et peut se faire n'importe où dans le code. La déclaration d'une fonction se fait grâce au mot-clé *function*, selon la syntaxe suivante :

```
function Nom_De_La_Fonction(argument1, argument2, ...) {
         liste d'instructions
}
```

Remarques:

- le nom de la fonction suit les mêmes règles que les noms de variables :
  - le nom doit commencer par une lettre
  - un nom de fonction peut comporter des lettres, des chiffres et les caractères _ et & (les espaces ne sont pas autorisés!)
  - le nom de la fonction, comme celui des variables est sensible à la casse (différenciation entre les minuscules et majuscules)
- Les arguments sont facultatifs, mais s'il n'y a pas d'arguments, les parenthèses doivent rester présentes
- Il ne faut pas oublier de refermer les accolades

- Le nombre d'accolades ouvertes (fonction, boucles et autres structures) doit être égal au nombre d'accolades fermées!
- La même chose s'applique pour les parenthèses, les crochets ou les guillemets!

Une fois cette étape franchie, votre fonction ne s'exécutera pas tant que l'on ne fait pas appel à elle quelque part dans la page!

Appel de fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom (une fois de plus en respectant la casse) suivie d'une parenthèse ouverte (éventuellement des arguments) puis d'une parenthèse fermée :

```
Nom_De_La_Fonction();
```

Remarques:

- le point virgule signifie la fin d'une instruction et permet à l'interpréteur de distinguer les différents blocs d'instructions
- si jamais vous avez défini des arguments dans la déclaration de la fonction, il faudra veiller à les inclure lors de l'appel de la fonction (le même nombre d'arguments séparés par des virgules!)
  `Nom_De_La_Fonction(argument1, argument2);`

Renvoi d'une valeur par une fonction

La fonction peut renvoyer une valeur (et donc se terminer) grâce au mot-clé *return*. Lorsque l'instruction *return* est rencontrée, la fonction évalue la valeur qui la suit, puis la renvoie au programme appelant (programme à partir duquel la fonction a été appelée).

Une fonction peut contenir plusieurs instructions *return*, ce sera toutefois la première instruction *return* rencontrée qui provoquera la fin de la fonction et le renvoi de la valeur qui la suit.

La syntaxe de l'instruction return est simple :

```
return valeur_ou_variable;
```

Les arguments d'une fonction

Il est possible de passer des arguments à une fonction, c'est-à-dire lui fournir une valeur ou le nom d'une variable afin que la fonction puisse effectuer des opérations sur ces arguments ou bien grâce à ces arguments.
Le passage d'arguments à une fonction se fait au moyen d'une liste d'arguments (séparés par des virgules) entre parenthèses suivant immédiatement le nom de la fonction. Les arguments peuvent être de simple variables, mais aussi des tableaux ou des objets. A noter qu'il est possible de donner une valeur par défaut à ces arguments en faisant suivre le nom de la variable par le signe "=" puis la valeur que l'on affecte par défaut à la variable.

Lorsque vous voulez utiliser un argument dans le corps de la fonction en tant que variable, celui-ci doit être précédé par le signe *$*.

```
<?
function dire_texte($qui, $texte = 'Bonjour')
{

 if(empty($qui)){ // $qui est vide, on retourne faux
  return false;

 }else{
  echo "$texte $qui"; // on affiche le texte
  return true; // fonction exécutée avec succès
 }
}

?>
```

Ainsi cette fonction peut être appelée de deux façons différentes :

```
<?
// Passage des deux paramètres
dire_texte("cher phpeur", "Bienvenue"); // affiche "Bienvenue cher phpeur"
// Utilisation de la valeur par défaut du deuxième paramètre
dire_texte("cher phpeur"); // affiche "Bonjour cher phpeur"
?>
```

Travailler sur des variables dans les fonctions

Lorsque vous manipulerez des variables dans des fonctions, il vous arrivera de constater que vous avez beau modifier la variable dans la fonction celle-ci retrouve sa valeur d'origine dès que l'on sort de la fonction...

Cela est du à la portée des variables, c'est-à-dire si elles ont été définies comme **variables globales ou locales**.
Il existe plusieurs niveaux de définition de variables :

- Une variable précédée du mot clé **global** sera visible dans l'ensemble du code, c'est-à-dire que sa portée ne sera pas limitée à la fonction seulement. Ainsi, toutes les fonctions pourront utiliser et modifier cette même variable
- Le niveau **static** permet de définir une variable locale à la fonction, qui persiste durant tout le temps d'exécution du script
- Par défaut, la variable possède le niveau **local**, c'est-à-dire que la variable ne sera modifiée qu'à l'intérieur de la fonction et retrouvera la valeur qu'elle avait juste avant l'appel de fonction à la sortie de celle-ci

```
<?
$chaine = "Nombre de camions : ";

function ajoute_camion($mode='')
{

 global $chaine;

 static $nb=0;

  $nb++; // on incrémente le nombre de camions
 if($mode == "affiche"){
  echo $chaine.$nb; // on affiche le nombre de camions
 }
}

ajoute_camion(); // nb == 1
ajoute_camion(); // nb == 2
ajoute_camion(); // nb == 3
ajoute_camion("affiche"); // affiche Nombre de camions : 4
?>
```

Passage de paramètre par référence

Une autre méthode pour modifier une variable consiste à la faire précéder du caractère *&*, précisant qu'il s'agit alors d'un alias: la valeur de la variable est modifiée à la sortie de la fonction. On parle alors de passage par référence. Dans ce cas on passe la référence (adresse mémoire) de la variable à la fonction, ce qui permet de modifier sa valeur.

```
<?
function dire_texte($qui, &$texte)
{

 $texte = "Bienvenue $qui";

}

$chaine = "Bonjour ";

dire_texte("cher phpeur",$chaine);

echo $chaine; // affiche "Bienvenue cher phpeur"
?>
```

Retourner plusieurs variables

Lorsque vous souhaitez qu'une fonction retourne plusieurs valeurs, le plus simple est d'utiliser un tableau.

```
<?
function nom_fonction()
{

.....
return array( $variable1, $variable2, $variable3 );

// on retourne les valeurs voulues dans un tableau
}

$retour = nom_fonction();
```

```
echo "$retour[0] - $retour[1] - $retour[2]";

?>
```

La récursivité

Les fonctions récursives sont des fonctions qui s'appellent elles-mêmes. Ce type de fonction se révéle indispensable pour parcourir une arborescence par exemple.
Voici un exemple simple.

```
<?
function fonction_recursive($n=0)
{

 $n++;

 echo "$n <br>";

 if($n < 10){ // si n est inférieur à 10 on continue
  fonction_recursive($n);

 }
}

fonction_recursive(); // affiche les nb de 1 à 10
?>
```

La notion de classe

Php3 intègre un soupçon de caractéristiques empruntées aux langages orientés objet, c'est-à-dire la possibilité d'utiliser des objets, entités regroupant des données et des fonctions au sein d'une structure et rendant la programmation plus simple qu'en programmation habituelle (appelée *programmation procédurale* par opposition à la programmation orientée objet).

On appelle classe la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composeront un objet. Un objet est donc "issu" d'une classe, c'est le produit qui sort d'un moule. En réalité on dit qu'un objet est une **instanciation** d'une classe, c'est la raison pour laquelle on pourra parler indifféremment d'*objet* ou d'*instance* (éventuellement d'*occurrence*).

Une classe est composée de deux parties :

- **Les attributs** (parfois appelés *données membres*): il s'agit des données représentant l'état de l'objet
- **Les méthodes** (parfois appelées *fonctions membres*): il s'agit des opérations applicables aux objets

déclaration d'une classe

Pour pouvoir manipuler des objets, il est essentiel de définir des classes, c'est-à-dire définir la structure d'un objet. Avec Php, cette définition se fait de la manière suivante :

```
class Nom_de_la_classe {
          // Déclarations des données membres
          var $Donnee_Membre_1;

          var $Donnee_Membre_2;

          var $...
          // Déclarations des méthodes
          function Nom_de_la_fonction_membre1(parametres) {
          liste d'instructions;

          }
}
```

Nom_de_la_classe représente bien évidemment le type d'objet désigné par la classe ou du moins le nom que vous leur attribuez.

Contrairement aux langages orientés objet comme le C++, Php n'inclut pas dans sa version 3 de *niveaux de visibilité* des éléments de la classe, il n'y a donc pas de concept d'encapsulation, un des concepts majeurs de la programmation orientée objet.

Contrairement à la déclaration de classes en C++, la déclaration de la classe ne se finit pas par un point-virgule!

Instanciation de la classe

Après avoir déclaré une classe, il faut instancier des objets pour pouvoir l'exploiter. Cette opération se fait à l'aide du mot clé *new* permettant de faire des objets découlant d'une classe. La syntaxe du mot clé new est la suivante :

```
$Nom_de_l_objet = new Nom_de_la_classe;
```

A partir du moment où l'objet est instancié, il possède des propriétés qui lui sont propres, cela signifie que si vous instanciez un nouvel objet, la modification des propriétés de l'un n'influera aucunement sur celles de l'autre.

Il existe une méthode spéciale (portant le même nom que la classe) s'exécutant automatiquement lors de l'instanciation de l'objet. Cette méthode, appelée **constructeur** est très utile pour initialiser les données membres lors de l'instanciation.

Accéder aux propriétés d'un objet

L'accès aux propriétés d'un objet se fait grâce au nom de l'objet, suivi d'une flêche (->) représentée par un moins (-) et un signe supérieur (>), puis du nom de la donnée membre (sans le signe $). Par exemple :

```
$Nom_de_l_objet->Nom_de_la_donnee_membre = Valeur;
```

Accéder aux méthodes d'un objet

L'accès aux méthodes d'un objet se fait comme pour l'accès aux propriétés, c'est-à-dire par le nom de l'objet, suivi d'une flêche et du nom de la méthode. La méthode est suivie de parenthèses, contenant les paramètres, si il y'en a. L'accès à une méthode se fait donc de la façon suivante :

```
$Nom_de_l_objet->Nom_de_la_fonction_membre(parametre1,parametre2,...);
```

La variable courante *$this*

Le mot clé *$this* permet de désigner l'objet dans lequel on se trouve, c'est-à-dire que lorsque l'on désire faire référence dans une fonction membre à l'objet dans lequel elle se trouve, on utilise *this*.

Grâce à cette variable spéciale, il est possible dans une fonction membre de faire référence aux propriétés situées dans le même objet que la fonction membre.

Ainsi, lorsque l'on désire accéder à une propriété d'un objet à partir d'une méthode du même objet, il suffit de faire précéder le nom de la donnée membre par *$this->*. Par exemple :

```
class Toto{
          var $age;

          var $sexe;

          var $adresse;

          function DefineTotoAge($Age){
          $this->age = $Age;

          }
}

$toto_test = new Toto;

$toto_test->DefineTotoAge(10);

echo "L'age de TOTO : " . $toto_test->age . "<br/>";
```

Les limitations de l'utilisation de classes avec Php

PHP, dans sa version 3, reste assez limité du point de vue de la programmation objet. La plupart des aspects marquants de la programmation objet ne sont pas présents dans le langage :

- l'encapsulation
- l'héritage
- le polymorphisme

Le but de PHP est de permettre la création de pages web dynamiques, ainsi son but premier est de pouvoir envoyer des données au navigateur.

Les trois fonctions standards

PHP fournit 3 fonctions permettant d'envoyer du texte au navigateur. Ces fonctions ont la particularité de pouvoir insérer dans les données envoyées des valeurs variables, pouvant être fonction d'une valeur récupérée par exemple, c'est ce qui rend possible la création de pages dynamiques. Les 3 fonctions sont les suivantes :

- echo
- print
- printf

La fonction *echo*

La fonction **echo** permet d'envoyer au navigateur la chaîne de caractères (délimitée par des guillemets) qui la suit. La syntaxe de cette fonction est la suivante :

```
echo Expression;
```

L'expression peut être une chaîne de caractères ou une expression que l'interpréteur évalue

```
echo "Chaine de caracteres";

echo (1+2)*87;
```

Ainsi, étant donné que la chaîne de caractères est délimitée par des guillemets, l'insertion de guillemets doubles dans la chaîne provoquerait une erreur. C'est la raison pour laquelle les guillemets doubles, ainsi que tous les caractères spéciaux, doivent être précédés d'un antislash. Voici un récapitulatif des caractères spéciaux nécessitant l'ajout d'un antislash :

Caractère	Description
<TD\"< td><TDGUILLEMET< td>	
<TD\$< td><TDCARACT&EGRAVE;RE td $<>	
<TD\\< td>< inverse oblique>	
<TD\R< td><>	
<TD\N< td>< la à>	
<TD\T< td><TDTABULATION< td>	

Le caractère $ a un rôle particulier dans la mesure où l'interpréteur le comprend comme une variable, ce qui signifie que lorsque le caractère $ est rencontré dans la chaîne qui suit la fonction *echo*, l'interpréteur récupère le nom de la variable qui suit le caractère $ et le remplace par sa valeur. Dans l'exemple suivant par exemple, on assigne la date actuelle à une variable appelée *$MaDate*, puis on l'affiche sur le navigateur :

```
<HTML>

<HEAD>

<TITLE>Affichage de l'heure</TITLE>

</HEAD>

<BODY>

<?
// Récupération de la date
// et stockage dans une variable
$MaDate = date("Y");

echo "Nous sommes en $MaDate";

?>
```

```
</BODY>

</HTML>
```

La fonction *print*

La fonction *print* est similaire à la fonction *echo* à la différence près que l'expression à afficher est entre parenthèses. La syntaxe de la fonction *print* est la suivante :

```
print(expression);
```
L'expression peut, comme pour la fonction *echo* être une chaîne de caractères ou une expression que l'interpréteur évalue :
```
print("Chaine de caracteres");

print ((1+2)*87);
```

La fonction *printf*

La fonction *printf()* (empruntée au langage C) est rarement utilisée car sa syntaxe est plus lourde. Toutefois, contrairement aux deux fonctions précédentes, elle permet un formatage des données, cela signifie que l'on peut choisir le format dans lequel une variable sera affichée à l'écran.
La syntaxe de *printf()* est la suivante :

```
printf (chaîne formattée);
```

Une chaîne formattée est une chaîne contenant des codes spéciaux permettant de repérer l'emplacement d'une valeur à insérer et son format, c'est-à-dire sa représentation. A chaque code rencontré doit être associé une valeur ou une variable, que l'on retrouve en paramètre à la fin de la fonction *printf*. Les valeurs à insérer dans la chaîne formattées sont séparées par des virgules et doivent apparaître dans l'ordre où les codes apparaissent dans la chaîne formattée Les codes de formatage des types de données sont les suivants :

Code	Type de format
%b	Entier en notation binaire
%c	Caractère codé par son code ASCII
%d	Entier en notation décimale
%e	Type double (nombre à virgule) au format scientifique (1.76e+3)
%f	Type double (nombre à virgule)
%o	Entier en notation octale
%s	Chaîne de caractères
%x	Entier en notation hexadécimale (lettres en minuscules)
%X	Entier en notation hexadécimale (lettres en majuscules)
%%	Caractère %

Imaginons que l'on définisse une variable en virgule flottante, afin d'obtenir une précision de calcul plus grande qu'avec un entier, mais qu'on désire l'afficher en tant qu'entier. Dans ce cas la fonction *printf* prend toute son importance :

```
$Pi = 3.1415927;

$R = 24.546;

$Perimetre = 2 * $Pi * $R;

printf ("Le périmètre du cercle est %d",$Perimetre);
```

L'importance de l'implantation du code php au sein du code HTML

Le code PHP peut être implanté au sein du code HTML. Cette caractéristique n'est pas à négliger car le fait d'écrire uniquement du code PHP là où il est nécessaire rend la programmation plus simple (il est plus simple d'écrire du code HTML que des fonctions *echo* ou *print*, dans lesquelles les caractères spéciaux doivent être précédés d'un antislash sous peine de voir des erreurs lors de l'exécution). L'exemple le plus simple concerne les pages dynamiques dont l'en-tête est toujours le même: dans ce cas, le code PHP peut ne commencer

qu'à partir de la balise <*BODY*>, au moment où la page peut s'afficher différemment selon une variable par exemple.

Mieux, il est possible d'écrire plusieurs portions de script en PHP, séparées par du code HTML statique car les variables/fonctions déclarées dans une portion de script seront accessibles dans les portions de scripts inférieures.

Notion de variable d'environnement

Les variables d'environnement sont, comme leur nom l'indique, des données stockées dans des variables permettant au programme d'avoir des informations sur son environnement. L'environnement, dans le cas du script PHP est :

- Le serveur
- Le client

Ces variables sont créées par le serveur à chaque fois que le script PHP est appelé, le serveur les lui fournit en paramètres cachés lors de l'exécution de l'interpréteur.

Elles permettent notamment d'avoir des informations sur le type de serveur, son administrateur, la date à laquelle le script a été appelé, l'adresse IP et le type de navigateur du client,...

Les variables d'environnement

On peut donc classer les variables d'environnement en deux catégories :

- Les variables d'environnement dépendant du client
- Les variables d'environnement dépendant du serveur

Les variables d'environnement dépendant du client

Variable d'environnement	Description
$AUTH_TYPE	Il s'agit de la méthode d'authentification qui a été utilisée par le client pour accéder au script PHP
$COMSPEC	Location de l'interpréteur de commandes sur la machine (Sous Windows)
$CONTENT_TYPE	Type de données contenu présent dans le corps de la requête. Il s'agit du type MIME des données
$DOCUMENT_ROOT	Racine des documents sur le serveur
$DOCUMENT_URI	Adresse du script PHP en relatif (à partir de la racine du serveur)
$HTTP_ACCEPT	Types MIME reconnus par le serveur (séparés par des virgules)
$HTTP_ACCEPT_ENCODING	Types d'encodage que le serveur peut réaliser (gzip,deflate)
$HTTP_ACCEPT_LANGUAGE	Langue utilisée par le serveur (par défaut *en-us*)
$HTTP_CONNECTION	Type de connexion ouverte entre le client et le serveur (par exemple *Keep-Alive*)
$HTTP_HOST	Nom d'hôte de la machine du client (associée à l'adresse IP)
$HTTP_REFERER	URL de la page qui a appelé le script PHP
$HTTP_USER_AGENT	Cette variable permet d'avoir des informations sur le type de navigateur utilisé par le client, ainsi que son système d'exploitation. Voici quelques exemples de User-Agents : • `Mozilla/4.0 (compatible;` • `MSIE 5.01;` • `Windows NT;` • `TUCOWS Network)` • `Mozilla/4.7 [en] (X11;` • `I;` • `Linux 2.2.14-15mdk i686)`
$LAST_MODIFIED	Date et heure de dernière modification du fichier
$PATH	Il s'agit du chemin d'accès aux différents répertoires sur le serveur
$PATH_INFO	Il s'agit du chemin d'accès au script PHP en relatif (de la racine du serveur jusqu'au script PHP)

$PHP_SELF	Nom du script PHP
$REDIRECT_STATUS	Il s'agit de l'état de la redirection (echec ou succès)
$REDIRECT_URL	Il s'agit de l'URL vers laquelle le navigateur du client a été redirigé
$QUERY_STRING	Il s'agit de la partie de l'URL (ayant servi à accéder au script PHP) située après le point d'interrogation. C'est de cette manière que sont transmises les données d'un formulaire dans le cas de la méthode GET
$REMOTE_ADDR	Cette variable contient l'adresse IP du client appelant le script CGI
$REMOTE_PORT	Cette variable permet de savoir le port sur lequel la requête HTTP a été envoyée au serveur
$SCRIPT_FILENAME	Chemin d'accès complet au script PHP • Sous windows, il sera de la forme : c:/php/php.exe
$SCRIPT_NAME	Chemin d'accès relatif (par rapport au chemin d'accès à la racine web ($DOCUMENT_ROOT)) au script PHP

Les variables d'environnement dépendant du serveur

Variable d'environnement	Description
$DATE_GMT	Date actuelle au format GMT
$DATE_LOCAL	Date actuelle au format local
$DOCUMENT_ROOT	Racine des documents Web sur le serveur
$GATEWAY_INTERFACE	Version des spécifications CGI utilisées par le serveur
$HTTP_HOST	Nom de domaine du serveur
$SERVER_ADDR	Adresse IP du serveur
$SERVER_ADMIN	Adresse de l'administrateur du serveur
$SERVER_NAME	Nom donné au serveur en local
$SERVER_PORT	Numéro de port associé au protocole HTTP sur le serveur
$SERVER_PROTOCOL	Nom et version du protocole utilisé pour envoyer la requête au script PHP
$SERVER_SOFTWARE	Type (logiciel) du serveur web • Pour un serveur Apache sous Unix : `Apache/1.3.2 (Unix) PHP/3.0.5` • Pour un serveur Apache sous Windows : `Apache/1.3.2 (Win32) PHP/3.0.5`

Affichage des variables d'environnement

Il est possible de créer un script permettant de visualiser l'ensemble des variables d'environnement.

La première façon consiste à utiliser la fonction *phpinfo()* qui affiche toute seule un tableau récapitulatif des paramètres du serveur et de l'intepréteur PHP, ainsi qu'un tableau des variables d'environnement

```
<?
phpinfo();

?>
```

PHP fournit la fonction *getenv()* permettant de retourner la valeur de la variable d'environnement passée en paramètre :

```
<?
echo getenv("HTTP_USER_AGENT");

?>
```

Enfin il est possible de définir des variables d'environnement :

```
<?
echo putenv("MA_VARIABLE=mavaleur
```

La gestion des fichiers avec PHP

Avec PHP, la création ou la lecture de fichiers est, une fois de plus, assez simple. Il existe une multitude de fonctions dédiées à l'utilisation des fichiers. La communication entre le script PHP et le fichier est repérée par une variable, indiquant l'état du fichier et que l'on peut passer en paramètre aux fonctions spécialisées pour le manipuler.

La fonction *fopen()*

La fonction de base est la fonction fopen(). C'est elle qui permet d'ouvrir un fichier, que ce soit pour le lire, le créer, ou y écrire. Voilà sa syntaxe :

```
 entier fopen(chaine nomdufichier, chaine mode);
```

Le mode indique le type d'opération qu'il sera possible d'effectuer sur le fichier après ouverture. Il s'agit d'une lettre (en réalité une chaîne de caractères) indiquant l'opération possible :

- **r** (comme *read*) indique une ouverture en lecture seulement
- **w** (comme *write*) indique une ouverture en écriture seulement (la fonction crée le fichier s'il n'existe pas)
- **a** (comme *append*) indique une ouverture en écriture seulement avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas)

Lorsque le mode est suivi du caractère **+** celui-ci peut être lu et écrit. Enfin, le fait de faire suivre le mode par la lettre **b** entre crochets indique que le fichier est traité de façon binaire.
Voici un tableau récapitulant l'ensemble des modes de fichiers possibles :

Mode	Description
r	ouverture en lecture seulement
w	ouverture en écriture seulement (la fonction crée le fichier s'il n'existe pas)
a	ouverture en écriture seulement avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas)
r+	ouverture en lecture et écriture
w+	ouverture en lecture et écriture (la fonction crée le fichier s'il n'existe pas)
a+	ouverture en lecture et écriture avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas)

Voici des exemples d'utilisations possibles de cette fonction :

```
$fp = fopen("../fichier.txt","r"); //lecture
//écriture depuis début du fichier
$fp = fopen("ftp://phpfrance.com/pub/fichier.txt","w");

//écriture depuis fin du fichier
$fp = fopen("http://igalaxie.com/fichier.txt","a");
```

De plus, la fonction fopen permet d'ouvrir des fichiers présents sur le web grâce à leur URL. Voici un script permettant de récupérer le contenu d'une page d'un site web :

```
<?
$fp = fopen("http://www.commentcamarche.net","r"); //lecture du fichier
while (!feof($fp)) { //on parcourt toutes les lignes
  $page .= fgets($fp, 4096); // lecture du contenu de la ligne
}

?>
```

Il est généralement utile de tester si l'ouverture de fichier s'est bien déroulée ainsi que d'éventuellement stopper le script PHP si cela n'est pas le cas :

```
<?
if (!$fp = fopen("http://www.commentcamarche.net","r")) {
echo "Echec de l'ouverture du fichier";

exit;

}

else {
// votre code;
```

```
}

?>
```

Un fichier ouvert avec la fonction *fopen()* doit être fermé, à la fin de son utilisation, par la fonction ***fclose()*** en lui passant en paramètre l'entier retourné par la fonction *fopen()*

Lecture et écriture

Une fois que le fichier a été ouvert avec le mode désiré, il est possible de lire son contenu et d'y écrire des informations grâce aux fonctions :

- **fputs()** (aussi parfois appelée **fwrite()**, les deux noms sont équivalents, on parle d'alias) permettant d'écrire une chaîne de caractères dans le fichier
  `entier fputs(entier Etat_du_fichier, chaine Sortie);`
  La fonction *fputs()* renvoie le nombre de caractères effectivement écrits dans le fichier
- **fgets()** permettant de récupérer une ligne du fichier
  `chaîne fgets(entier Etat_du_fichier, entier Longueur);`
  Le paramètre *Longueur* désigne le nombre de caractères maximum que la fonction est sensée récupérer sur la ligne. La fonction *fgets()* renvoie 0 en cas d'échec, la chaîne dans le cas contraire

Etant donné que la fonction *fgets()* récupère à chaque appel une nouvelle ligne du fichier, il est essentiel, pour récupérer l'intégralité du contenu d'un fichier de l'insérer dans une boucle *while*.

Ainsi, on utilise la fonction ***feof()***, fonction testant si la fin du fichier n'a pas été atteinte, en tant que test de la boucle *while*. De cette façon, tant que la fin du fichier n'a pas été atteinte, on lit la ligne suivante du fichier...

```
<?
if (!$fp = fopen("fichier.txt","r")) {
echo "Echec de l'ouverture du fichier";

exit;

}

else {
          while(!feof($fp)) {
          // On récupère une ligne
                    $Ligne = fgets($fp,255);

          // On affiche la ligne
                    echo $Ligne;

          // On stocke l'ensemble des lignes dans une variable
                    $Fichier .= $Ligne;

          }
          fclose($fp); // On ferme le fichier
}

?>
```

Pour stocker des infos dans le fichier, il faut dans un premier temps ouvrir le fichier en écriture en le créant si il n'existe pas. On a donc le choix entre le mode 'w' et le mode 'a'. On préférera le second puisque le pointeur se trouve en fin de fichier (autrement dit on écrit à la suite de ce qui se trouve dans le fichier au lieu d'écraser le contenu existant éventuellement déjà).

```
<?
$fp = fopen("php_8_fichier.txt","a"); // ouverture du fichier en écriture
fputs($fp, "\n"); // on va a la ligne
fputs($fp, "$nom|$email"); // on écrit le nom et email dans le fichier
fclose($fp);

?>
```

Voici un petit script permettant de récupérer le titre d'une page Web (le texte compris entre les balises <TITLE> et </TITLE>). Il utilise les expressions régulières pour localiser le texte.

```
<?
$fp = fopen("http://www.commentcamarche.net","r"); //lecture du fichier
```

```
while (!feof($fp)) { //on parcourt toutes les lignes
  $page .= fgets($fp, 4096); // lecture du contenu de la ligne
}

$titre = eregi("<title>(.*)</title>",$page,$regs); //on isole le titre
echo $regs[1];

fclose($fp);

?>
```

Les tests de fichiers

PHP fournit de nombreuses fonctions permettant de faire des tests sur les fichiers pour connaître leurs propriétés. Voici la liste des fonctions des tests :

- **is_dir()** permet de savoir si le fichier dont le nom est passé en paramètre correspond à un répertoire
  ```
  booléen is_dir(chaine Nom_du_fichier);
  ```
  La fonction *is_dir()* renvoie 1 si il s'agit d'un répertoire, 0 dans le cas contraire
  ```
  <?
  if (!is_dir("install")) {
  echo "Il ne s'agit pas d'un répertoire
  ";

  }

  else {
  echo "Il s'agit bien d'un répertoire
  ";

  }

  ?>
  ```
- **is_executable()** permet de savoir si le fichier dont le nom est passé en paramètre est exécutable
  ```
  booléen is_executable(chaine Nom_du_fichier);
  ```
  La fonction *is_executable()* renvoie 1 si le fichier est exécutable, 0 dans le cas contraire
- **is_file()** permet de savoir si le fichier dont le nom est passé en paramètre ne correspond ni à un répertoire, ni à un lien symbolique
  ```
  booléen is_file(chaine Nom_du_fichier);
  ```
  La fonction *is_file()* renvoie 1 si il s'agit d'un fichier, 0 dans le cas contraire
- **is_link()** permet de savoir si le fichier dont le nom est passé en paramètre correpond à un lien symbolique
  ```
  booléen is_link(chaine Nom_du_fichier);
  ```
  La fonction *is_link()* renvoie 1 si il s'agit d'un lien symbolique, 0 dans le cas contraire

D'autres façons de lire et écrire

Dans certains cas, il peut être rébarbatif de devoir mettre en oeuvre les fonctions *fopen()* et *fgets* pour lire l'intégralité du contenu d'un fichier. Pour cette raison PHP fournit des fonctions supplémentaires permettant de faire directement certaines opérations.

La fonction **file()** permet de retourner dans un tableau l'intégralité d'un fichier en mettant chacune de ces lignes dans un élément du tableau (rappel : le premier élément d'un tableau est repéré par l'indice 0). Voilà sa syntaxe :

```
 Tableau file(chaine nomdufichier);
```

L'exemple suivant montre comment parcourir l'ensemble du tableau afin d'afficher le fichier.

```
<?
$Fichier = "fichier.txt";

if (is_file($Fichier)) {
          if ($TabFich = file($Fichier)) {
                    for($i = 0; $i < count($TabFich); $i++)
                              echo $TabFich[$i];

          }
          else {
          echo "Le fichier ne peut être lu...<br>";
```

```
        }
}

else {
echo "Désolé le fichier n'est pas valide<br>";

}

?>
```

La fonction **fpassthru()** permet d'envoyer le contenu du fichier dans la fenêtre du navigateur. La syntaxe de cette fonction est la suivante :

```
booléen fpassthru(entier etat);
```

Cette fonction permet en réalité d'envoyer le contenu du fichier à partir de la position courante dans le fichier, c'est-à-dire qu'il est possible par exemple de lire quelques lignes avec *fgets()*, puis d'envoyer le reste au navigateur… Le script suivant permet de parcourir tous les fichiers HTML contenus dans votre site et d'en afficher l'arborescence :

```
<HTML>
<HEAD>
<TITLE>Affichage de l'arborescence</TITLE>
</HEAD>
<BODY>

<?
function ScanDir($Directory){
if (is_dir($Directory) && is_readable($Directory)) {
          if($MyDirectory = opendir($Directory)) {
          while($Entry = readdir($MyDirectory)) {
                    if (is_dir($Directory."/".$Entry)) {
                              if (($Entry != ".") && ($Entry != "..")) {
                    echo "<li><b>Repertoire</b>: $Directory/$Entry</li>\n";
                    echo "<ul>";
                    ScanDir($Directory."/".$Entry);
                    echo "</ul>";

                              }
                    }
                    else {
                    echo "<li><b>Fichier</b>: $Directory/$Entry </li>\n";

                              if (eregi("(\.html)|(\.htm)",$Entry)){
                    $MetaTags = get_meta_tags($Directory."/".$Entry);

                                        }
                              }
                    }
          closedir($MyDirectory);

}

}

}

$open_basedir=".";

echo "<ul>";
ScanDir(".");
echo "</ul>";

?>
</BODY>
</HTML>
```