Chapitre III : Langage PHP

1. Introduction

Le langage PHP (à l'origine : Personal Home Pages) est destiné à développer des applications Web. Il est appelé actuellement : PHP Hypertext Preprocessor. Ce langage est disponible pour différents systèmes d'exploitation. De plus, l'interpréteur PHP est un logiciel Open Source : Libre de droit.

Le PHP est un langage de scripts exécuté coté Serveur. On parle alors de scripts PHP ou programmes PHP. Les scripts PHP sont interprétés et permettent de générer du code HTML qui sera envoyé au client.

1.1 Caractéristiques du PHP

- Il est très proche du Langage C.
- Il s'intègre au cœur d'une page HTML. C'est à dire, on écrit une page HTML avec du code inclus. Par contre, dans les autres langages : on écrit le programme pour afficher (en sortie) une page HTML.
- Les instructions PHP n'apparaissent pas dans le résultat final. Ainsi, il n'y a aucun moyen d'accéder à partir du poste client au code source.

1.2 Que peut faire PHP ?

Le PHP a plusieurs fonctionnalités intéressantes. On cite :

- Il peut collecter les données.
- Il permet de générer dynamiquement des pages Web.
- Il envoie et reçoit des cookies.
- Il a des supports de services utilisant les protocoles : IMAP, SNMP, POP3, HTTP, …
- Il offre une interaction avec d'autres protocoles en ouvrant des connexions vers divers ports TCP/IP.
- Il a un ensemble d'outils : manipulation et génération d'images, traitement de fichiers, création de PDF, …
- Il peut supporter un grand nombre de Bases de Données : MySQL, dBase, FilePro, Informix, Interbase, mSQL, ODBC, Oracle, Unix dbm, …

1.3 Avantages de PHP

Parmi les avantages du langage PHP on cite :

- Il facilite le développement de sites Web dynamiques et interactifs.
- Il offre une réponse simple et claire.
- Le PHP est un langage souple avec une syntaxe classique et pratique : la plupart des instructions connues, il s'appuie sur des standards, il a des raccourcis d'utilisation, la partie déclaration n'est pas nécessaire, Il a un système de conversion assuré par le moteur PHP…
- Le PHP a une bibliothèque riche : l'écriture d'algorithmes complexes est possible en quelques lignes.
- Il a des fonctions de découpage de chaînes de caractères ainsi que le traitement des expressions régulières
- La tâche d'interprétation des commandes est déléguée à un composant appelé moteur. Le moteur est installé sur le serveur
- Quelque soit le système d'exploitation, le moteur PHP est capable de : lire et transformer une chaîne de caractères, créer et modifier des fichiers (txt ou bin.), donner informations systèmes…
- La maintenance des scripts PHP est facile. On dit que les scripts PHP sont réutilisables.

2. Minimum de PHP

2.1 Insertion du code PHP

On insère du code PHP au milieu du code HTML. On utilise : **`<?php`** pour marquer le début du bloc de script. La fin du bloc de script est marquée par **`?>`**.

Il existe d'autres façons qui ne sont pas reconnues par tous les interpréteurs PHP et qui nécessitent parfois de modifier le fichier de configuration *php.ini*. Ainsi, on peut aussi insérer des instructions PHP à l'aide :

```
<?   …   ?>
<script language="php">   …   </script>
<%   …   %>
```

2.2 Séparateur d'instructions

Chaque instruction doit se terminer par le point virgule «;». Pour la dernière instruction, le point virgule est facultatif car la marque de fin de script (**?>**) marque la fin de la dernière instruction.

2.3 Sortie

Pour générer en sortie du code qui est envoyé vers le client, le PHP a l'instruction **echo**. Cette instruction permet de faire sortir la chaîne de caractères qui suivra **echo**.

Exemple : `echo "chaîne de caractères";`

Remarque : On peut aussi utiliser la fonction **printf**.

2.4 Interprétation d'une page PHP

Tout bloc mis entre **<?php** et **?>** est exécuté puis un code HTML est généré en sortie (selon les instructions) à la place du bloc d'instructions puis envoyé au client. Tout autre bloc est envoyé sans retouches au client. En cas d'erreurs (lexicales et/ou syntaxiques), un message est intégré dans le code de sortie et l'exécution est interrompue.

2.5 Exemple

Soit le code suivant sauvegardé sur le serveur :

```
<html>
   <head>
      <title> test </title>
   </head>
   <body>
      <?php echo "3éme Li Acad"; ?>
   </body>
</html>
```

Si la page ci-dessus est demandée par un utilisateur (client) le code ci-dessous lui est envoyé après exécution :

```
<html>
   <head>
      <title> test </title>
   </head>
   <body>
      3éme Li Acad
   </body>
</html>
```

2.6 Commentaires PHP

Comme dans tout langage, il est possible d'insérer des commentaires en PHP. Il existe diverses façons pour le faire :

// : Pour un commentaire sur une ligne. Il ignore tous ce qui vient après jusqu'à la fin de la ligne.

Pour étaler le commentaire sur plusieurs lignes, on marque son début par **/*** et se termine par ***/**.

: Pour un commentaire comme dans le Shell Unix (sur une ligne).

Remarque : Il faut éviter d'imbriquer les commentaires.

3. Données PHP

3.1 Les variables

Le PHP n'est pas un langage fortement structuré : il n'existe pas de partie déclarative bien définie et aucune déclaration n'est nécessaire.

Une variable est reconnue à travers le symbole **$**. Ainsi, elle est toujours précédée par le symbole **$**. Par exemple : **$x** **$txt1** … Il faut savoir que PHP tient compte de la casse. Ainsi, **$a** et **$A** sont deux variables différentes.

Le PHP admet divers types de données :

- **Scalaires :** booléens, entiers, réels (nombres flottants), chaînes de caractères, …
- **Composés :** tableaux, objets, …
- **Spéciaux :** ressource (qu'on verra lors de la manipulation de fichiers ou bases de données) et **NULL**

Même si le PHP admet plusieurs types de données, le type n'est pas déclaré. Le moteur décide selon le contexte. Pour créer une variable, il suffit de lui affecter une valeur. Par la suite, elle devient accessible et continuera à exister jusqu'à la fin.

3.2 Types de données

3.2.1 Type Booléen

Il est définit par le mot clé **bool** ou **boolean**. Il prend deux valeurs possibles : **TRUE** ou **FALSE**. Le moteur convertit les variables en booléen comme suit :

- **0** ou **0.0** ou **""** ou **"0"** ou **NULL** sont considérés comme **FALSE**.
- Tableau vide, objet vide sont considérés comme **FALSE**.
- Toute autre valeur (même **-1**) est considérée comme **TRUE**.

3.2.2 Type Entier

Il est définit par le mot clé **integer** ou **int**. Il peut être :

- Décimal base 10. Par exemple : **$x=141; $y=-141;**
- Octal (base 8). Par exemple : **$x=0215;**
- Hexadécimal (base 16). Par exemple : **$x=0x8D;**

En cas de dépassement de capacité, l'entier est considéré comme réel. La valeur **FALSE** est considérée **0** et la valeur **TRUE** est considérée **1**.

3.2.3 Type Réel

Il est définit par le mot clé **float**, **double** ou **real**. C'est toute valeur contenant le point «.» pour séparer la partie entière de la partie décimale. Par exemple : **$pi = 3.14;**

3.2.4 Type Chaîne de caractères

Il est définit par le mot clé **string**. Par exemple : **$chn="3 Li Acad"; $c='Bienvenue';**

Plus détails seront donnés dans la section gestion des chaînes de caractères.

3.2.5 Tableaux en PHP

Les tableaux dans PHP sont identifiés par un nom qui doit respecter la syntaxe d'un identificateur. L'indice de chaque entrée (élément) dans le tableau est appelée clé. Chaque valeur associée à une entrée (élément) est appelée valeur.

On dit que les tableaux en PHP sont associatifs : clé, valeur. La clé d'un tableau peut être entière (positif) ou chaîne de caractères.

Exemple :

```
$tab['nom']="Mohamed";
$tab[3]='Ali';
```

tab est le nom du tableau. **nom** et **3** sont deux clés du tableau **tab**. La valeur associée à la clé **nom** est **Mohamed** et **Ali** est la valeur associée à la clé **3**.

Plus détails seront donnés dans la partie gestion des tableaux.

3.3 Les références

L'utilisation des références permet d'attribuer un autre nom à une variable à l'aide du symbole **&**.

Exemple : Soit le script suivant

```
$a=1; $b=&$a; // b et a représentent la même chose (même cellule en mémoire)
echo($a) // affiche 1
echo($b) // affiche aussi 1
$a = 2;
echo($b); // affiche 2
```

3.4 Variables dynamiques

En PHP, le nom d'une variable peut être à son tour variable. On sauvegarde le nom sous forme de chaîne de caractères dans une variable. On peut appeler la variable en doublant le symbole **$**. L'exemple ci-dessous illustre cette situation :

```
$x="y"; $$x=1; //c'est la variable y qui aura la valeur 1!!!
$z="tab"; ${$z}[1] =2; //c'est le 1er élément de tab qui aura la valeur 2
$vec[1] = "u"; $vec[2] = "v";
${$vec[1]} = 3; // c'est la variable u qui aura la valeur 3
```

3.5 Suppression et existence d'une variable

La suppression de variables permet la libération d'espace mémoire. Il se fait à l'aide de la fonction : **`unset($idf_var);`**

A tout moment, on peut vérifier l'existence d'une variable à l'aide de la fonction booléenne : **`isset($var);`**

3.6 Variables liées aux données des formulaires

Les valeurs des champs d'un formulaire envoyés au script peuvent être trouvées dans les tableaux :**`$_POST`** et **`$_GET`**. C'est selon la méthode d'envoi du formulaire. Chaque clé du tableau est une chaîne de caractères comportant le nom d'un champ. Chaque valeur associée à une clé est la valeur du champ telle qu'elle a été remplie, cochée, choisie ou sélectionnée par l'utilisateur.

Remarques :

- Pour les champs à choix multiples (cases à cocher ou boîte de liste de sélection), il est conseillé de leur attribuer dans le code HTML un nom se terminant par **`[]`**. Ceci permettra de récupérer les choix de l'utilisateur dans un tableau. Si aucun choix n'a été effectué par l'utilisateur le tableau sera vide. Par exemple, si le champs s'appelle **`ch[]`** et la méthode est *post* alors le $i^{ème}$ choix sera dans **`$_POST["ch"][i-1]`**.
- Il est possible de vérifier si le champ envoyé a été laissé vide par l'utilisateur. Pour les champs textuels, il suffit de les comparer avec la chaîne de caractère vide **`""`**. Pour les champs à cocher ou à sélectionner, on peut utiliser la fonction **`isset()`** pour vérifier l'existence ou non de ce champ. Par ailleurs, on peut utiliser la fonction booléenne **`empty()`** et qui retourne *vrai* si le champ est laissé vide par l'utilisateur.
- Il existe un autre tableau en PHP qui contient les données du formulaire. Il s'agit de **`$_REQUEST`** qui est une copie de **`$_POST`** et de **`$_GET`**. En d'autres termes, la modification dans **`$_REQUEST`** n'aura aucun effet sur **`$_POST`** et **`$_GET`** et vice-versa.
- Il existe d'autres façons mais qui ne sont pas reconnues par tous les interpréteurs et il faut vérifier que la variable *register_globals* est à *On* (n'est pas à *Off*) dans le fichier de configuration *php.ini*. Ainsi, à un champ du formulaire qui s'appelle *nom_champ*, PHP peut associer une variable **`$nom_champ`**. La valeur envoyée par l'utilisateur est stockée dans cette variable. Comme on peut les trouver aussi dans les tableaux :**`$HTTP_POST_VARS`** et **`$HTTP_GET_VARS`** de la même manière que pour **`$_POST`** et **`$_GET`**.

3.7 Les Constantes

3.7.1 Variables d'environnement

Le PHP dispose de nombreuses variables prédéfinies qui contiennent des informations sur le serveurs ou les informations communiquées entre le serveur et le client (formulaire, cookie, …). Ces variables ont la particularité de commencer par **`$_`**.

Exemple : **`$_SERVER["SERVER_NAME"]`** donne le nom du serveur.

3.7.2 Constantes définies par l'utilisateur

On peut définir une constante en suivant la syntaxe suivante : **`define("idf_cst", "val_cst");`**

Exemple :

```
define("PI", "3.14");
echo(PI); //affiche 3.14
```

4. Opérateurs PHP

4.1 L'affectation =

Soit l'exemple illustratif suivant :

```
$a = 2;
$b = "ceci est un exemple";
$c = ($b = 4) = + 5; //4 dans b et 9 dans c
$tab[$i] = 3.14;
```

4.2 Opérateurs Arithmétiques :

Le tableau ci-dessous liste les différents opérateurs arithmétiques du langage PHP.

Opérateur	Description	Opérateur	Description
+	Addition.	*	Multiplication.
-	Soustraction.	%	Modulo.
/	Division qui est par défaut réelle.		

4.3 Opérateur de chaînes

Pour les chaînes de caractères, le PHP offre un opérateur de concaténation qui est le point : **.**

4.4 Opérateurs de Comparaison

Le tableau ci-dessous liste les différents opérateurs de comparaison dans PHP.

Opérateur	Description	Opérateur	Description	Opérateur	Description
==	Egalité.	>	Supérieur strict.	>=	Supérieur ou égal.
===	Egalité et même type.	<	Inférieur strict.	<=	Inférieur ou égal.
!=	Différence.				

4.5 Opérateurs Logiques

Le tableau ci-dessous liste les différents opérateurs logiques dans PHP.

Opérateur	Description	Opérateur	Description
and ou **&&**	Et logique.	**xor**	Ou exclusif.
or ou **||**	Ou logique.	**!**	La négation.

4.6 Opérateurs de bits

Le tableau ci-dessous liste les différents opérateurs de bits dans PHP.

Op.	Descr.	Op.	Description	Op.	Description
&	Et.	~	Non (négation).	<<	Décalage à gauche d'1 bit (multiplication par 2)
|	Ou.	^	ou exclusif «xor».	>>	Décalage à droite d'1 bit (division entière par 2)

4.7 Expressions abrégées

Soit l'exemple suivant qui illustre quelques cas d'expressions abrégées qu'offre le PHP :

```
$x = $y = 2; //affecte 2 à x et y en même temps
$x++; ++$x; // incrémentation
$x--; --$x; // décrémentation
$y=$x++; //affectation de la valeur de x à y puis incrémentation de x
$y=--$x; // décrémentation de x puis affectation de la valeur de x à y
$x .= "Licence"; //concaténation de Licence à la chaîne dans x
```

5. Conversion de variables

Le moteur PHP assure automatiquement la conversion comme l'illustre l'exemple suivant :

```
$x = 1 + 2.5; // de type réel, x = 3.5
$x = 1 + "2.5"; // de type réel, x = 3.5
$x = 1 + "abcd"; // de type entier, x = 1
$x = 1 + "a2"; // de type entier, x = 1
$x = 1 + "10 cm"; // de type entier, x = 11
$x = 1 + "10.0 cm"; // de type entier, x = 11
```

5.1 Forcer un type

On peut forcer le type d'une variable de deux manières différentes:

```
settype($var, "type");
(type) $var;
```

Où **type** peut être :

- **int**, **integer** : pour les entiers.
- **bool**, **boolean** : pour les booléens.

- **real**, **float**, **double** : pour les réels.
- **array** : pour les tableaux.
- **object** : pour les objets.

Exemple :

```
$x = 3.6; settype($x, "int"); // x aura la valeur 3
$a = (int) (8/3); // met dans a la valeur de la division entière de 8 par 3
```

5.2 Tester le type d'une variable

On peut tester le type d'une variable à l'aide :

```
$t=gettype($var); //retourne dans $t le type de $var sous forme d'une chaîne de car
```

On peut aussi utiliser les fonctions booléennes suivantes :

```
is_long($x); is_double($x); is_string($x); is_array($x); is_object($x);
```

6. Chaînes de caractères et Expressions régulières

6.1 Ecriture d'une chaîne

6.1.1 Chaînes avec quottes

On peut écrire une chaîne de caractères entre guillemets simples : ' et '. Cette écriture permet d'afficher le texte tel qu'il est.

Exemple :

```
$x=2; echo'C\'est un exemple $x';
// Donne : C'est un exemple $x
```

On utilise le caractère Antislash (\) pour précéder les caractères ' et \ afin de les afficher et éviter d'éventuelles erreurs d'interprétation.

6.1.2 Chaînes avec guillemets

On peut écrire une chaîne de caractères entre guillemets doubles : " et ". Cette écriture permet d'afficher le texte ainsi que les caractères spéciaux et remplace les variables par leurs contenus.

Exemple :

```
$x=2;
echo"C'est un \"exemple \" :\n $x"; //Ceci donne :
// C'est un "exemple" :
// 2
```

\n représente le saut de ligne.

6.1.3 La syntaxe HereDoc

Elle obéit à la syntaxe suivante :

```
<<< identificateur
Chaîne sur une ou plusieurs lignes
identificateur;
```

Il est préférable de choisir un identificateur qui n'apparaît pas dans la chaîne de caractères.

Cette écriture se comporte de la même façon qu'une chaîne à guillemets doubles. Elle permet d'afficher le contenu des variables et d'écrire la chaîne sur plusieurs lignes.

Remarque : Il faut faire attention à cette écriture! Elle n'est pas reconnue par tous les interpréteurs.

6.2 Fonction de traitement des chaînes de caractères

Le PHP dispose d'une panoplie de fonctions de traitement des chaînes de caractères. Nous citons ici quelques-unes :

Fonction	Description
`$ln = strlen($ch);`	Pour récupérer la longueur d'une chaîne
`strcmp($ch1, ch2);`	Pour comparer deux chaînes.
`$c1 = $ch{$i-1};` `$c2 = $ch{0};`	Pour accéder au $i^{ème}$ caractère d'une chaîne **$ch**. La deuxième donne dans **$c2** le premier caractère de la chaîne **$ch**.
`str_replace($ch1, $ch2, $ch3);`	Remplace dans **$ch3** toute occurrence de **$ch1** par **$ch2**.

`$tab=explode($sep , $ch , $limite);`	Découpe **$ch** en plusieurs sous-chaînes par rapport à la chaine de caractères **$sep**. Chacune des sous chaînes est sauvegardée comme élément du tableau **$tab**. L'argument **$limite** est facultatif. Il s'agit d'un entier *n* qui permet de limiter le découpage à *n* sous-chaînes. Si *n*>0, il découpe les *n* premiers sous mots est retourne le reste de la chaine dans le dernier emplacement du tableau. Si *n*<0, il découpe le début de la chaîne sauf les -*n* dernières sous-chaînes. Si *n*=0, il se comporte comme si *n*=0.
`addslashes($ch);`	ajoute **** devant tout caractère spécial dans **$ch**.
`stripslashes($ch);`	Supprime les **** qui précèdent les caractères spéciaux.

6.3 Les expressions régulières

6.3.1 Ecriture d'une expression régulière

Pour exprimer la syntaxe d'un mot donné (adresse email, URL, date, identificateur, …), on utilise les expressions régulières qu'offre le langage PHP.

Pour écrire une expression régulière, on donne chaque caractère et le nombre d'occurrences de ce caractère comme suit :

Pour définir un caractère ou l'ensemble des caractères, nous utilisons les symboles :

Symbole	Description
`[]`	Permet de définir l'ensemble auquel appartient un caractère.
`|`	Ou. Par exemple : **[a|b|c]** veut dire soit le caractère *a* soit *b* soit *c*. Elle est équivalente à **[abc]**.
`-`	Une séquence ordonnée de caractères. Par exemple : **[a-c]** un caractère entre *a* et *c*. Elle est équivalente à **[abc]**.
`.`	N'importe quel caractère.
`^`	Indique le début obligatoire d'une chaîne. S'il est inséré entre [], cela veut dire que les caractères qui le suivent ne doivent pas exister dans la chaîne.
`$`	Indique la fin obligatoire d'une chaîne.

Nous avons les cas particuliers suivants :

Ecriture	Description
[a] ou **a**	S'il s'agit exactement du caractère *a*.
[0-9]	Caractère numérique.
[a-z]	Caractère alphabétique minuscules.
[A-Z]	Caractère alphabétique majuscules.
[a-zA-Z0-9]	Caractère alphanumérique.
[\x09]	Espacement ou tabulation.
[\t\v\f]	Tout type d'espace.

Il existe aussi des classes prédéfinies de caractères ou symboles qui sont données par le tableau ci-dessous :

Nom de la classe	Description
[:alpha:]	Caractère alphabétique.
[:upper:]	Caractère majuscule.
[:digit:]	Chiffre de 0 à 9.
[:alnum:]	Caractère alphanumérique.
[:punct:]	Caractère de ponctuation.
[:xdigit:]	Caractère hexadécimal.
[:blank:]	Caractère d'espacement et de tabulation.
[:space:]	Caractère d'espacement.
[:ctrl:]	Caractères de contrôle (les premiers du code ASCII)
[:print:]	Caractère imprimable (tous les caractères sauf les caractères de contrôle)
[:graph:]	Caractère d'imprimerie (imprimables à l'exception de l'espace et tabulation)

Pour définir le nombre d'occurrence :

Symbole	Description
`?`	0 ou exactement une occurrence.
`*`	0 ou plusieurs occurrences.
`+`	Au moins une occurrence.
`{}`	Permet de définir le nombre d'occurrences en chiffres du caractère précédent.
`{n}`	Exactement n occurrences.
`{n,m}`	Au moins *n* occurrences et au plus *m* occurrences.
`{n,}`	Au moins *n* occurrences.

Exemple :

Pour exprimer la syntaxe d'une date au format : *jj-mm-aaaa*, on écrit :

`^[0-9]{2}\-[0-9]{2}\-[0-9]{4}$` ou `^[[:digit:]]{2}\-[[:digit:]]{2}\-[[:digit:]]{4}*$`

Pour exprimer la syntaxe d'un identificateur du qui peut contenir des caractères alphanumériques et le tiret du bas (underscore) et qui commence obligatoirement par une lettre alphabétique, on écrit :

`^[[:alpha:]][[:alnum:]_]*$` ou `^[a-zA-Z][a-zA-Z0-9_]*$`

6.3.2 Gestion des expressions régulières

Il existe plusieurs fonctions de traitement des expressions régulières. Nous donnons ici juste quelques-unes :

Fonction	Description
`preg_match($exp_reg , $ch, $sortie, $flag, $debut);`	Vérifie la correspondance entre l'expression régulière **`$exp_reg`** (de type chaîne de caractères) et la chaîne **`$ch`**. Les trois derniers arguments sont facultatifs. **`$sortie`** qui est un tableau de chaînes de caractères contient les résultats de la recherche. Tel que le texte correspondant à toute l'expression régulière dans **`$sortie[0]`** et celui correspondant à la première parenthèse capturante dans **`$sortie[1]`** et ainsi de suite. **`$flag`** permet de définir la façon avec laquelle se fait la recherche. Si **`$flag`** est défini à **`PREG_OFFSET_CAPTURE`**, **`$sortie`** sera un tableau à deux dimensions **`$sortie[$i][0]`** donne la chaîne de caractère correspondante et **`$sortie[$i][1]`** donne la position du premier caractère dans **`$ch`**. **`$debut`** permet de définir la position dans **`$ch`** pour le début de recherche. Cette fonction retourne **`TRUE`** ou **`FALSE`**.
`preg_match_all($exp_reg , $ch, $sortie, $flag);`	Permet de rechercher dans un texte **`$ch`** toute sous chaîne correspondant à l'expression régulière **`$exp_reg`**. Les résultats sont ordonnés dans **`$sortie`** (tableau à 2 dimensions) selon la valeur de **`$flag`**. Cette fonction retourne **`TRUE`** si la recherche aboutit. Si **`$flag=PREG_PATTERN_ORDER`**, on a dans **`$sortie[0]`** toutes les sous chaîne de **`$ch`** qui satisfont **`$exp_reg`** et dans **`$sortie[$i]`** toutes les sous chaînes de **`$ch`** qui satisfont la **`$i`** ème parenthèse de **`$exp_reg`**. Si **`$flag=PREG_SET_ORDER`**, on a dans **`$sortie[$i][0]`** la **`$i`** ème sous chaîne de **`$ch`** qui satisfait **`$exp_reg`** et dans **`$sortie[$i][$j]`** la sous chaînes de **`$sortie[$i][0]`** qui satisfait la **`$j`** ème parenthèse de **`$exp_reg`**.
`$tab=preg_split($exp_reg , $ch, $limite, $flag);`	Découpe **`$ch`** en plusieurs sous-chaînes par rapport à l'expression régulière **`$exp_reg`**. Chacune des sous chaînes est sauvegardée comme élément du tableau **`$tab`**. L'argument **`$limite`** est facultatif. Il s'agit d'un entier *n* qui permet de limiter le découpage à *n* sous-chaînes. Si *n*=0, -1 ou *NULL*, aucune limite n'est définie.

Remarque : L'expression régulière doit être écrite sous forme de chaine de caractères. Il faut la délimiter (début et fin) par un symbole spécial comme le **#**. En effet, un délimiteur sert pour marquer le début et la fin de cette expression. On peut utiliser les caractères suivants comme délimiteurs : **#**, **/**, **§**, **!** ou **~**.On peut rajouter à la fin de l'expression le **i** pour que l'interpréteur ne tienne pas compte de la casse.

Exemple1 : Dans cet exemple, nous vérifions la validité d'un pseudonyme qui doit contenir entre 4 et 20 caractères alphanumériques ou tiret du bas (_) ou tiret (-) ou point (.). Les deux instructions ci-dessous retournent le même résultat.

```
preg_match("#^[A-Za-z0-9_\.\-]{4,20}$#", $pseudo);
preg_match("#^[a-z0-9_\.\-]{4,20}$#i", $pseudo); // le i pour ignorer la casse
```

Exemple2 : Dans cet exemple, nous cherchons tous les sous-noms qui composent un nom donné.

```
$nom="El Hadj-Moussa Moh Cherif Ameziane";
$r="#[^a-z]*([a-z]+)[^a-z]*#i";
preg_match_all($r , $nom, $sortie, PREG_PATTERN_ORDER);
foreach($sortie[1] as $trouve){
   echo $trouve . '***'; // Affiche à chaque itération un sous nom suivi de 3 étoile
} // Affiche à la fin El***Hadj***Moussa***Moh***Cherif***Ameziane***
```

Remarque : D'autres fonctions similaires existent telles que : **ereg**, **eregi**, **ereg_replace** et **split** mais qui sont actuellement dépréciées par les interpréteurs PHP.

7. Gestion des tableaux

7.1 Création d'un tableau

Un tableau peut être créé directement par affectation comme le précise la syntaxe suivante :

```
$tab[cle1] = valeur1; $tab[cle2] = valeur2; … $tab[clen] = valeurn;
```

La clé peut être vide :

```
$tab[]=val; // insère val dans le tableau à la première clé libre.
```

Un tableau peut être aussi créé par la fonction **array** comme le précise la syntaxe suivante :

```
$tab = array(cle1=>val1, cle2=>val2, …, clen=>valn);
```

Il est possible de créer le tableau sans préciser les clés : `$tab = array(val1, val2, …, valn);`

Ceci est équivalent à: `$tab[0] = val1; $tab[1] = val2; …; tab[n-1] = valn;`

7.2 Tableaux multidimensionnels

On peut aussi définir des tableaux à 2 (matrice) ou plusieurs dimensions. Il suffit de considérer chaque élément comme un autre tableau. Par exemple :

```
$tab['nom'][1] = "Mohamed";
$tab['nom'][2] = "Ali";
```

7.3 Fonctions de traitement des tableaux

Le PHP dispose de plusieurs fonctions de gestion et traitement des tableaux. Ci-dessous, nous citons quelques-unes :

Fonction	Description
`sort($tab);`	Trie un tableau par rapport aux valeurs dans l'ordre.
`rsort($tab);`	Trie un tableau par rapport aux valeurs dans un ordre inverse.
`ksort($tab);`	Trie un tableau par rapport aux clés dans l'ordre.
`krsort($tab);`	Trie un tableau par rapport aux clés dans un ordre inverse.
`reset($tab);`	Remet le pointeur au premier élément du tableau.
`end($tab);`	Positionne le pointeur sur le dernier élément du tableau.
`$nb=sizeof($tab);`	Retourne le nombre d'éléments du tableau **$tab**.
`$nb=count($tab);`	Retourne le nombre d'éléments du tableau **$tab**.
`$element= each($tab);`	Donne la combinaison clé, valeur de l'élément actuel de $tab puis pointe vers l'élément suivant. **$element[0]** contient la clé et **$element[1]** contient la valeur.
`list($cle, $valeur) = each($tab);`	**list** est associée à **each** afin de récupérer la clé actuelle et sa valeur dans deux variables différentes.

Remarque : A tout moment, on peut supprimer des éléments d'un tableau à l'aide de `unset($tab[cle]);`

8. Instructions PHP

8.1 Bloc d'instructions

Pour délimiter un bloc d'instructions, on utilise comme dans le langage *C* les accolades.

`{` : pour marquer le début du bloc.

`}` : pour marquer la fin du bloc.

Si le bloc ne contient qu'une seule instruction, on peut se passer des accolades.

8.2 Instructions conditionnelles

8.2.1 Instruction if

Syntaxe :

```
if (expression_logique) bloc_instructions
```

Exemple :

```
if ($x>0) $x--;
if ($x<=0)  {
   $x++;
   echo $x; }
```

8.2.2 Instruction if else

Syntaxe :

```
if (exp_logique) bloc_instructions_1 else bloc_instructions_2
```

Exemple :

```
If ($x>0) $x--; else $x++;
If ($x<=0)     { $x++; echo $x; }
          else  {$x--; echo $x; }
```

8.2.3 Instruction if … elseif

Pour éviter les **`if`** imbriqués on peut utiliser cette instruction Si **`cond_1`** est vérifiée, il exécute le bloc **`bloc_instructions_1`**. Si les conditions de **`cond_1`** à **`cond_i`** ne sont pas vérifiées et **`cond_i+1`** est vérifiée, il exécute les instructions de **`bloc_instructions_i+1`**. Si les *n* conditions ne sont pas vérifiées, il exécute le bloc d'instructions qui suit le **`else`**.

Syntaxe :

```
if (cond_1) { bloc_instruction_1;}
elseif(cond_2) {bloc_instructions_2;}
…
elseif(cond_n) {bloc_instructions_n;}
else {bloc_instructions_n+1;}
```

8.2.4 Instruction conditionnelle abrégée

Syntaxe :

```
$var = Expression ? Val1 : Val2;
```

Cette instruction est équivalente à :

```
If (Expression)   {$var = Val1;}
          else    {$var = Val2;}
```

8.2.5 Instruction switch case

Cette instruction se comporte de la même façon que le **`switch`** du langage C.

Syntaxe :

```
switch($var) {
   case val_1 : bloc_instr_1; break;
...
   case val_n : bloc_instr_n; break;
   default : bloc_instr_n+1;
}
```

8.3 Les boucles

8.3.1 Boucle while

Tant que la condition est vérifiée, il exécute les instructions. Avec une seule instruction, on peut enlever les accolades.

Syntaxe :

```
while(condition) {
   inst_1;
   inst_2;
…
   inst_n;
}
```

8.3.2 Boucle do while

Exécute les instructions puis évalue la condition. Si elle est vérifiée, il continue sinon il s'arrête. Avec une seule instruction, on peut enlever les accolades.

Syntaxe :

```
do {
   inst_1;
   inst_2;
…
   inst_n;
   } while(condition);
```

8.3.3 Boucle for

Exécute **expr1** avant de commencer la première itération de la boucle. Avant chaque itération, teste **expr2** puis exécute **expr3** à la fin de chaque itération. Avec une seule instruction, on peut enlever les accolades.

Syntaxe :

```
for (expr1; expr2; expr3) {
   inst1;
   inst2;
…
   instn;}
```

8.3.4 Boucle foreach

Cette boucle est utilisée pour parcourir les tableaux. Elle commence par pointer sur le premier élément **$idf_tab**. A chaque itération, elle sauvegarde la valeur de l'élément actuel du tableau dans la variable **$val**. A la fin de chaque itération, elle pointe sur l'élément suivant dans le tableau. La boucle se termine après avoir passé en revue tous les éléments du tableau.

Avec une seule instruction, on peut enlever les accolades.

Syntaxe :

```
foreach($idf_tab as $val) {
   Inst_1;
   Inst_2;
…
   Inst_n;}
```

Il existe une autre forme de **foreach**. Elle se comporte de la même chose que la première forme de **foreach**. La seule différence est qu'elle sauvegarde en plus la clé de l'élément actuel dans **$cle**.

Syntaxe :

```
foreach($idf_tab as $cle=>$val) {
   Inst_1;
   Inst_2;
…
   Inst_n;}
```

8.3.5 Les clauses break et continue

Comme dans le langage *C*, on peut utiliser les deux instructions **break** et **continue** dans les boucles.

```
break; //Pour sortir de la boucle sans vérification de la condition d'arrêt.
continue; //Pour abandonner l'itération courante et passer directement à l'itération suivante.
```

9. Les fonctions en PHP

9.1 Définition d'une fonction

Syntaxe :

```
function idf_fonct($arg1,$arg2,…,$argn){
   inst_1;
   inst_2;
…
   inst_n;
   return $var; }
```

9.1.1 L'instruction return

Elle permet à la fonction de retourner une valeur et quitter la fonction. **return** peut être insérée à n'importe quel endroit (selon le besoin) dans la fonction. On peut aussi avoir une fonction qui ne retourne pas de valeurs : c'est à dire procédure.

9.1.2 Retourner plusieurs valeurs

Pour pouvoir retourner un grand nombre de valeurs dans une fonction, on peut utiliser un tableau :

```
return $tab; //$tab doit être de type tableau.
```

Comme on peut créer ce tableau au moment de l'utilisation de **return** :

```
return array($cle_1=>$val_1, …, $cle_n=>$val_n);
return array($cle_1, …, $cle_n);
```

9.1.3 Arguments d'une fonction

On peut avoir une fonction sans arguments : `function idf_fct() { instr…}`

Un argument peut avoir une (ou plusieurs) valeur(s) par défaut lors de la définition de la fonction :

```
function f($x, $y, $z=1) { instructions…}
```

Les arguments définis par défaut doivent être mis en derniers dans la liste des arguments.

9.2 Nombre illimité d'arguments

On peut utiliser un tableau comme argument. Par exemple :

```
function f($tab) {
      …
      $nb_arg = count ($tab);
      // Pour récup. nbre d'argmts.
      …
      foreach ($tab as $arg) {
      … //Pour récup. tous les arguments
      }
}
```

L'autre façon pour avoir un nombre illimité de paramètres est de définir une fonction sans arguments. Par la suite, on utilise certaines fonctions PHP pour traiter les arguments quel que soit leur nombre. On appelle ces fonctions PHP à l'intérieur du corps de la fonction. Le tableau suivant décrit les fonctions qu'on peut utiliser :

Fonction	Description
`$nb_arg=func_num_args();`	Pour retourner le nombre d'arguments passés à cette fonction.
`$arg=func_get_arg($i);`	Pour retourner la valeur du paramètre numéro i+1.
`$tab_arg=func_get_args();`	Pour retourner toutes les valeurs des paramètres dans un tableau.

9.3 Appel à une fonction

L'appel à une fonction doit être toujours fait après sa définition comme suit :

```
$idf_var= idf_fonction($param_1, …, $param_n);
$idf_fonction($param_1, …, $param_n);// si la fonction ne retourne rien
$idf_var= idf_fonction();// Si la fonction n'admet pas d'arguments
```

Si certains arguments sont par défaut dans la définition de la fonction, lors de l'appel on peut ne pas mettre les paramètres correspondant à ces arguments et les valeurs par défaut seront appliquées. Dans le cas où on envoi des paramètres pour les arguments par défaut ce sont les valeurs envoyées qui seront appliquées au lieu des valeurs par défaut.

Lors de l'appel les arguments sont passés des manières suivantes : **Par valeurs** ou **Par références** (de façon **permanente** ou **non permanente**).

Le passage d'arguments par valeurs est un passage ordinaire. Il n'y a rien de spécial à rajouter lors de la définition. Lors de l'appel, on met des valeurs ou des variables comme paramètres. Dans ce cas, les variables transmises sont en entrée.

Le passage d'arguments par références rend les variables transmises en arguments en entrée/sortie.

Le passage par références non permanent se fait lors de l'appel à la fonction. Il n'y a rien de spécial à rajouter lors de la définition. Lors de l'appel, on fait référence à la variable transmise en argument en la précédant du symbole **&**. **Seulement cette façon est déconseillée** et peut générer un « *warning* ». **Elle n'existe plus dans les nouvelles versions de PHP**.

Le passage par références permanent se fait lors de la définition de la fonction. Lors de la définition, on précède la variable argument de **&**. Il n'y a rien de spécial à rajouter lors de l'appel.

Exemple :

```
function f1($x) { $x++;}
function f2(&$y) { $y++;} /*définition d'un passage (pour f2) qui se fait toujours par
références: c'est à dire permanent.*/
//passage par valeurs
$a=1; f1($a); echo $a; /*affiche 1. $a en entrée seulement*/
//passage par références non permanent qui est déconseillé et génère un warning ou une erreur
$a=1; f1(&$a); echo $a; /*affiche 2. $a est en entrée/sortie*/
//passage par reference permanent
$a=1; f2($a); echo $a; /*affiche 2. $a est en entrée et sortie*/
```

9.4 Variables globales/ locales

9.4.1 Variable locale

Par défaut la variable utilisée dans une fonction est locale. Sa durée de vie se limite au temps d'exécution de la fonction. On peut rendre une variable locale à l'aide de : **`local $var;`**

Exemple

```
$a = 1;
function f1()
   { echo $a;}
f1();
```

L'exécution de ce code n'affiche rien et/ou génère une erreur. Car il s'agit de la variable **`$a`** de la fonction **`f1`** qui ne contient aucune valeur et non pas de **`$a`** du code principal.

9.4.2 Variable globale

Elle est visible et reconnue dans l'intégralité du code PHP. On peut manipuler une variable **`$var`** comme étant variable globale à l'aide de : **`$GLOBALS["var"];`** En d'autre termes, on spécifie qu'il s'agit de la variable globale et non locale.

Dans les anciennes versions du PHP, il fallait déclarer dans la fonction la variable globale à l'aide de : **`global $var;`**

Exemple 1 :

```
$a = 1;
function f2() {
      global $a, $b;
      $b = $a++;
}
f2();
echo $a; //affiche 2
echo $b; //affiche 1
```

Exemple 2 :

```
$a = 1;
function f3() {
      $GLOBALS["b"] = $GLOBALS["a"]++;
}
f3();
echo $a; //affiche 2
echo $b; //affiche 1
```

9.4.3 Variable statique

Une variable dans une fonction est dite statique si elle est propre à la fonction et reconnue durant toute l'exécution du code PHP. C'est à dire elle garde la valeur lors du prochain appel. En d'autres termes, elle n'est pas réinitialisée à chaque appel comme c'est le cas pour la variable locale. On peut rendre une variable statique à l'aide de : **`static $var;`**

Exemple

```
$a = 1;
function f4() {
      static $a=2; // est exécutée juste lors du premier appel.
      $a++;
      echo $a;
}
f4(); // Affiche 3
echo $a ; // affiche 1 car il s'agit du $a du code principal
f4(); // Affiche 4
```

9.5 Fonctions dynamiques

Comme on a des variables dynamiques en PHP, on a aussi des fonctions dynamiques. On peut sauvegarder le nom d'une fonction dans une variable comme indiqué ci-dessous :

```
$ch="nom_fonction";
$ch(liste_arguments); // équivalente à nom_fonction(liste_arguments);
```

On peut même utiliser un tableau contenant les noms de différentes fonctions comme indiqué ci-dessous :

```
$tab_fct=array("nom_fonction_1" , …, "nom_fonction_n");
$tab_fct[$i-1](liste_arguments); //équivalentes à nom_fonction_i(liste_arguments);
```

9.6 Fonctions prédéfinies

Les fonctions prédéfinies ne sont pas sensibles à la casse. Ainsi, elles peuvent être écrites en majuscules ou minuscules. Nous nous intéresserons ici aux fonctions mathématiques qui sont les plus utilisées. Le tableau suivant illustre quelques unes. Le type nombre veut dire que le paramètre ou la variable peuvent être entiers ou réels.

La fonction	Description / Sortie de la fonction	Les types
`$y=abs($x);`	Elle retourne la valeur absolue de *$x*.	*$x* et *$y* : nombres.
`$y=ceil($x);`	Elle retourne l'entier immédiatement supérieur à *$x*.	*$x* et *$y* : réels.
`$y=floor($x);`	Elle retourne la partie entière de *$x* (arrondir)	*$x* et *$y* : réels.
`$y=round($x,$N);`	Elle arrondit *$x* avec *$N* chiffres après la virgule.	*$x* et *$y* : réels et *$N* : entier.
`$z=fmod($x,$y);`	Elle retourne le reste (réel) de la division de *$x/$y*	*$x*, *$y* et *$z* : réels.
`$y=sqrt($x);`	Elle retourne la racine carrée de *$x*	*$x* et *$y* : réels
`$y=pow($x,$n);`	Elle réalise le calcul la puissance : $\$x^{\$n}$.	*$n*, *$x* et *$y* : nombres
`$y=exp($x);`	Elle calcule l'exponentiel ($e^{\$x}$) d'un nombre donné.	*$x* et *$y* sont des réels.
`$y=log($x);`	Elle calcule le logarithme népérien de *$x*.	*$x* et *$y* sont des réels
`$y=log10($x);`	Elle retourne le logarithme base 10 de *$x*.	*$x* et *$y* sont des réels.
`$y=max($a, $b);`	Elle retourne le maximum.	*$a*, *$b* et *$y* sont des nombres.
`$y=min($a, $b);`	Elle retourne le minimum.	*$a*, *$b* et *$y* sont des nombres.
`$y=pi();`	Elle retourne la valeur exacte de π.	*$y* est un réel.
`$y=rand();`	Elle retourne un nombre aléatoire positif ou nul. On peut aussi utiliser `$y=mt_rand();`	*$y* est un entier.
`$y=rand($a, $b);`	Elle retourne un nombre aléatoire entre *$a* et *$b* inclus. On peut aussi utiliser `$y=mt_rand($a, $b);`	*$y*, *$a* et *$b* sont des entiers.
`$y=sin($x);`	Elle retourne le sinus de *$x*. *$x* en radians.	*$x* et *$y* sont des réels.
`$y=cos($x);`	Elle retourne le cosinus de *$x*. *$x* en radians.	*$x* et *$y* sont des réels.
`$y=tan($x);`	Elle retourne la tangente de *$x*. *$x* en radians.	*$x* et *$y* sont des réels.
`$y=asin($x);`	Elle retourne en radians l'arc sinus de *$x*. *$x* doit être un réel dans l'intervalle [-1, +1].	*$x* et *$y* sont des réels.
`$y=acos($x);`	Elle retourne en radians l'arc-cosinus de *$x*. *$x* doit être un réel dans l'intervalle [-1, +1].	*$x* et *$y* sont des réels.
`$y=atan($x);`	Elle retourne en radians l'arc-tangente de *$x*.	*$x* et *$y* sont des réels.

Par ailleurs, le PHP dispose d'une multitude de fonctions. Ces fonctions sont rangées en modules d'extensions. Il est possible de vérifier les modules installés sur le serveur en exécutant la fonction suivante :

```
$tab=get_loaded_extensions();
```

Cette fonction retourne un tableau, dont chaque entrée contient le nom d'un module disponible.

Il est possible aussi de lister les fonctions d'un module donné à l'aide de :

```
$tab=get_extensions_funcs($nom-module);
```

Cette fonction retourne aussi un tableau, dont chaque entrée est une chaîne de caractères contenant le nom d'une fonction du module.

10. Gestion de la date

Il existe un compteur du nombre de secondes entre la date d'origine qui est le 01/01/1970 à 00h00'00" et la date actuelle. Ce nombre de secondes écoulées est appelé « *timestamp* » ou « *instant Unix* ».

Dans le langage PHP, Il existe plusieurs fonctions qui permettent de traiter les dates. Ci-dessous quelques-unes :

La fonction	Description
`$t=time();`	Cette fonction retourne le timestamp actuel (`$t` est un entier).
`$d=getdate($t);`	Elle a comme argument un entier `$t` qui est un timestamp. Elle retourne la date (`$d`) sous forme de tableau comme suit : `$d['wday']` Le jour de semaine de 0 à 6 (0=dim.) `$d['weekday']` Le jour de semaine en anglais. `$d['mday']` Le jour du mois de 1 à 31. `$d['mon']` Le mois de 1 à 12 (La valeur 1 représente janvier.). `$d['month']` Le mois en anglais. `$d['year']` L'année (entier sur 4 positions). `$d['hours']` L'heure de 0 à 23. `$d['minutes']` Les minutes de 0 à 59. `$d[`seconds']` Les secondes de 0 à 59. `$d[`yday']` Le jour de l'an de 1 à 366.
`$c=date($f, $t);`	Elle a comme argument une chaîne de caractère `$f` qui représente la forme de date retournée et un entier `$t` qui est un timestamp. Le second argument est facultatif et s'il n'est pas précisé, elle se base sur le temps courant. Elle retourne la date (`$c`) sous forme de chaîne de caractères respectant le format précisé dans `$f` qui peut être formé des caractères : `l` Le jour de semaine en anglais. `D` Le jour de semaine en anglais sur trois (3) lettres. `d` Le jour du mois de 1 à 31. `m` Le mois de 1 à 12 (La valeur 1 représente janvier.). `M` Le mois en anglais. `Y` L'année (entier sur 4 positions). `y` L'année (entier sur 2 positions). `H` L'heure de 00 à 23. `h` L'heure de 01 à 12. `i` Les minutes de 00 à 59. `s` Les secondes de 00 à 59. `a` am ou pm. `A` AM ou PM. Exemple : `date("D d M Y", 1560873946)`; retourne Tue 18 Jun 2019 `date("H:i:s", 1560873946)`; retourne 18:05:46 Nous avons utilisé l'espace comme séparateur dans le premier exemple et les deux-points (:) dans le second. Nous aurions pu utiliser d'autres séparateurs pour préciser le format à afficher.
`$b =checkdate($m, $j, $a);`	Elle permet de vérifier la validité de la date en entrée. `$m`, `$j` et `$a` sont 3 entiers représentant respectivement le mois, le jour et l'année. Elle retourne un booléen `$b`. Par exemple : `checkdate(2,30,2011);` retourne `FALSE` car le 30 février 2011 n'est pas une date.

`$b = mktime($h, $mn, $s, $m, $j, $a);`	Cette fonction retourne le timestamp correspondant à la date définie par le jour `$j`, le mois `$m`, l'année `$a` et l'heure : `$h` heures, `$mn` minutes et `$s` secondes.

11. Gestion des fichiers

La gestion des fichiers en PHP se fait presque de la même manière qu'en langage *C*. Dans cette section, nous avons sélectionné une liste de fonctions de traitement de fichiers dans le langage PHP.

11.1 Vérifications et Création

Ci-dessous la description de quelques fonctions permettant de créer et de faire certaines vérifications sur un fichier dont le nom est sauvegardé dans la variable `$nom_fichier`.

La fonction	Description
`$b=file_exists($nom_fichier);`	Retourne un booléen, `TRUE` si le fichier existe.
`touch($nom_fichier, $t);`	Permet de créer un fichier dont le nom est `$nom_fichier` et la date de dernière modification est le timestamp `$t`.
`$b=is_file($nom_fichier);`	Retourne un booléen, `TRUE` si c'est un fichier.
`$b=is_readable($nom_fichier);`	Retourne `FALSE`, s'il est protégé en lecture.
`$b=is_writable($nom_fichier);`	Retourne `FALSE`, s'il est protégé en écriture.
`$b=is_uploaded_file($nom_fichier);`	Retourne `TRUE`, s'il a été envoyé par formulaire.
`$type=filetype($nom_fichier);`	Retourne le type du fichier sous forme de chaîne de caractères.

11.2 Ouverture d'un fichier

L'ouverture d'un fichier est nécessaire avant toute manipulation de ce dernier. Cette opération permet d'ouvrir un fichier selon le mode spécifié. La fonction est la suivante :

```
$f=fopen($nom_fichier, $mode, $path);
```

Elle retourne une variable (`$f`) de type ressource. Le type ressource est un type particulier. Cette variable ressource permettra d'effectuer toutes les opérations possibles sur le fichier en faisant référence seulement à `$f` et sans citer le nom du fichier.

L'argument `$mode` spécifie le mode d'ouverture du fichier. C'est une chaîne de caractères comme suit:

`r`	read, indique une ouverture en lecture.
`w`	write, indique une ouverture en écriture. Écrase le contenu.
`a`	append, indique une ouverture en écriture avec ajout du contenu à la fin du fichier.

Lorsque le mode est suivi du caractère `+`, le fichier est ouvert en lecture et écriture en même temps. Lorsque le mode est suivi par `b`, le fichier est traité de façon binaire. Les modes `a` et `w` permettent la création du fichier s'il n'existe pas.

L'argument `$path` est facultatif, s'il est à `1` ou `TRUE`, la recherche du fichier s'élargira aux sous-répertoires (sous-dossiers).

La fonction `fopen` permet aussi l'ouverture de fichiers se trouvant sur d'autres sites (un autre serveur). Il suffit de donner comme nom de fichier l'URL complète et avoir les droits d'accès.

11.3 Lecture / Ecriture

Plusieurs fonctions de lecture ou d'écriture sont disponibles. Nous avons sélectionné quelques unes dans le tableau ci-dessous. `$f` représente la variable ressource faisant référence au fichier.

Fonction	Description
`$ch=fgets($f, $nb_oct);`	Lit le fichier et retourne la chaîne de caractères `$ch` de taille maximale de `$nb_oct` octets. La lecture s'arrête à la rencontre d'un saut de ligne ou la fin de fichier et renvoie `0` en cas d'échec.
`$ch=fread($f, $nb_oct);`	Lit le fichier et retourne un chaîne de caractères `$ch` de taille égale à `$nb_oct` octets. La lecture s'arrête à la rencontre de la fin de fichier.
`$c=fgetc($f);`	Lit le fichier et retourne un caractère `$c`.
`fputs($f, $ch, $n);`	Ecrit dans le fichier le contenu de la chaîne de caractères `$ch`. Le parmètre `$n` est facultatif mais s'il est précisé, seuls les `$n` premiers caractères de `$ch` seront écrits.
`fwrite($f, $ch, $n);`	Ecrit dans le fichier le contenu de la chaîne de caractères `$ch`. Le paramètre `$n` est facultatif. S'il est précisé, seuls les `$n` premiers caractères de `$ch` seront écrits.

11.4 Verrouillage d'un fichier

Cette fonction est très importante dans le Web. Ceci est dû au risque d'accès multiples au fichier. La fonction est la suivante :

```
$b=flock($f, $n);
```

Elle retourne un booléen : **TRUE** si l'opération de verrouillage a réussi et **FALSE** sinon.

Cette fonction a deux arguments : **$f** est la variable ressource utilisée lors de l'ouverture et **$n** est un entier qui admet trois valeurs : **1** pour verrouiller l'écriture, **2** pour verrouiller la lecture et écriture et **3** pour déverrouiller.

11.5 Autres fonctions de traitement des fichiers

Dans le tableau suivant, **$f** représente la variable ressource faisant référence au fichier.

Fonction	Description
`$b=feof($f);`	Vérification de la fin de fichier. Retourne un booléen **$b**. **TRUE** : si la fin de fichier a été atteinte.
`$taille=filesize($f);`	Retourne la taille du fichier en octets.
`$b=fclose($f);`	Cette fonction est nécessaire avant de quitter le programme (script). Elle retourne un booléen **$b**. **TRUE** : si l'opération a réussi.

11.6 Télécharger des fichiers vers le serveur

Il est possible de donner la main à l'utilisateur pour uploader un ficher vers le serveur à partir de la machine cliente en utilisant un formulaire. Pour le réaliser, Il faut tout d'abord réaliser le formulaire avec un champ bien particulier qui est le champ **<input>** de **type="file"**. Pour plus de détails sur l'insertion de ce genre de champs, il faut se référer au chapitre I.

Le formulaire doit être obligatoirement envoyé avec la méthode **"post"**. Il faut rajouter dans la balise **<form>** l'attribut **enctype="multipart/form-data"**.

Après l'envoi du fichier à travers le formulaire, le fichier est sauvegardé sur le serveur dans un répertoire tampon. Ce répertoire est défini dans *php.ini* grâce à la directive **upload_tmp_dir**. Il est enregistré sous un nom différent généré aléatoirement. Si aucun script ne le manipule immédiatement, il sera perdu.

Pour retrouver le fichier, on a le tableau associatif **$_FILES**. C'est un tableau à deux (2) dimensions.

La 1ère clé : nom attribué au champ formulaire **"file"**.

La 2ème clé indique les informations nécessaires au traitement du fichier transféré.

Si par exemple le champ s'appelle **"f1"**, on aura :

L'élément du tableau	Description
`$_FILES["f1"]["name"]`	Nom (original) du fichier chez le client.
`$_FILES["f1"]["type"]`	Type MIME du fichier.
`$_FILES["f1"]["size"]`	Taille en octets
`$_FILES["f1"]["tmp_name"]`	Nom temporaire du fichier sur le serveur.
`$_FILES["f1"]["error"]`	Le code d'erreur (4 valeurs définies par des constantes)

Pour enregistrer le fichier et le renommer on fait appel à la fonction php suivante :

```
move_uploaded_file($_FILES["f1"]["tmp_name"],$nom_fich_final);
```

Cette fonction retourne un booléen : **TRUE** si l'opération a réussi, **FALSE** dans le cas contraire.

Remarques :

- Pour limiter la taille du fichier à transférer, on rajoute un champ caché dans le formulaire comme suit :

```
<input type="hidden" name="MAX_FILE_SIZE" value="nb_octets">
```

- On peut aussi le faire pour tous les scripts à travers le fichier *php.ini* en modifiant la directive **upload_max_filesize**. La taille est en octets.
- On peut limiter le transfert (selon l'extension et le type) en rajoutant l'attribut **accept** dans la balise **input** de **type="file"**. La valeur de cet attribut est le type MIME qui sera accepté : **text/html**, **image/jpeg**, **image/gif**…

12. Envoi d'email

Il est possible de déclencher l'envoi d'un email avec PHP et ce grâce à la fonction **mail** dont la syntaxe est la suivante :

```
mail($dest, $objet, $message, $entetes) ;
```

Le tableau ci-dessous décrit les différents arguments :

Argument	Description
$dest	Chaine de caractère décrivant l'adresse email du destinataire
$objet	Chaine de caractère pour définir l'objet (sujet) de l'email.
$message	Chaine de caractère pour insérer le contenu du message
$entetes	C'est un argument facultatif permettant de décrire les en-têtes nécessaires à l'envoi de l'email. Chaque en-tête se termine par un saut de ligne **\n** (**\r\n** pour le windows). Les en-têtes sont **From:** pour définir l'adresse de l'expéditeur, **cc:** pour l'adresse destinataire en copie, **bcc:** pour l'adresse destinataire en copie cachée, **Reply-To:** adresse de réponse si le destinataire répond en cliquant sur le bouton « répondre », **X-Mailer:** Nom de l'application d'envoi du courrier, **Date:** Date de l'email au format *JJ MM AAAA hh:mm:ss +0D00* où la valeur de *D* représente le décalage horaire par rapport à GMT. Pour insérer plusieurs adresses email destinataires, il faut les séparer par des virgules.

La fonction retourne un booléen.

13. Les cookies

Ce sont de petits fichiers qu'on peut insérer dans la machine du client. Leur taille ne dépasse pas 4 Ko. De plus, un serveur *X* ne peut pas écrire plus de 20 cookies sur un client *Y*. Il faut aussi savoir qu'ils peuvent être désactivés par le client. Les cookies peuvent avoir plusieurs intérêts : sauvegarder des données, s'assurer qu'il s'agit du même client, publicité…

13.1 Création / Suppression

En PHP, on a une seule fonction qui permet soit de créer soit supprimer un cookie. Il s'agit de **setcookie**. Sa syntaxe est la suivante :

```
setcookie($nom_c, $valeur_c, $date_fin, $chemin, $domaine, $securite, $httponly);
```

Seul le premier argument qui est obligatoire. Pour créer le cookie, il faut donner au moins un nom et une valeur. Pour le détruire, il faut juste donner le nom. La tableau ci-dessous définit les autres arguments :

Argument	Description
$date_fin	Définit la date (timestamp) à partir de laquelle le cookie ne sera plus valide (utilisable). Par exemple **time()+86400** pour lui donner une durée de vie de 24H.
$chemin	chemin vers les dossiers (répertoires) dont les scripts sont autorisés à utiliser ce cookie. Par exemple si on met **/**, tous les scripts des répertoires et sous répertoires du domaine sont autorisés et si on met **/d1/rep**, tous les scripts sont dans ce dossier et ses sous-dossiers sont autorisés.
$domaine	Pour préciser les sous-domaine qui sont autorisés à manipuler le cookie.
$securite	**TRUE** si le cookie se transmet par connexion sécurisée (https).
$httponly	**TRUE** pour dire que le cookie est accessible uniquement par le protocole http (interdit aux langages de script comme JavaScript).

La fonction **setcookie(…)** retourne un booléen.

Pour transmettre plusieurs valeurs dans le même cookie, on utilise la notation de tableau comme le montre l'exemple suivant :

```
setcookie("etudiant[nom]", "BENOMAR", time()+3600);
setcookie("etudiant[prenom]", "ALI", time()+3600);
setcookie("etudiant[faculte]", "FI", time()+3600);
```

13.2 Lecture des cookies

Pour lire les cookies, on accède au tableau prédéfini **$_COOKIE[nom]**.

Si on a transmis plusieurs valeurs dans le même cookie, on peut utiliser la boucle **foreach** sur le tableau **$_COOKIE[nom]**. On peut aussi y accéder directement comme par exemple : **$_COOKIE["etudiant"]["nom"]**

14. Les sessions

Le protocole HTTP est un protocole de transmission sans états. C'est-à-dire, rien ne lui permet de savoir si deux requêtes différentes émanent du même poste client ou non. Ceci peut être vu dans le cas de la conservation des informations d'une page vers une autre.

Le mécanisme de sessions qui est l'une des solutions à ce problème a été introduite depuis PHP4. Ainsi, on peut distinguer les sessions avec cookies et les sessions sans cookies.

Le mécanisme des sessions passe généralement par les étapes suivantes :

- Ouverture de la session.
- Attribution d'un identifiant unique : transmis d'une page à une autre (par cookie ou ajouté à l'URL). C'est cet identifiant qui permet au serveur de reconnaître s'il s'agit du même client ou non.
- Définition des variables liées à la session stockées dans un dossier du serveur.
- Lecture/écritures des variables de session.
- Fermeture session / Destruction variables de session.

14.1 Les sessions avec cookies

Il faut que chaque page commence par l'instruction suivante : **`session_start();`**

Par la suite, on définit une ou plusieurs variables de session (selon le besoin) comme suit :

```
$_SESSION['nom_var'] = $valeur;
```

L'accès dans toutes les pages à la variable de session se fait par le tableau : **`$_SESSION`**

14.2 Les sessions sans cookies

Si l'utilisateur désactive les cookies sur son navigateur, le mécanisme de sessions qu'on a vu dans la section précédente ne fonctionnera pas. Le mécanisme de sessions sans cookies est une solution dans ce genre de situations.

Ainsi, la valeur de la session est définie dans une constante **`SID`**. Cette valeur est attribuée après l'appel à la fonction **`session_start()`**. Il faut rajouter cette valeur dans tous les liens à la fin de l'URL après le symbole **`?`**. Comme le montre l'exemple suivant : **`<a href="http://www.adresse.ext/fichier.php? <?php echo SID;?>">`**

La définition et l'accès aux variables de session se fait toujours par le tableau : **`$_SESSION`**

14.3 Terminer une session

A la fin, il faut supprimer la session ouverte. Pour détruire toutes les variables session, nous avons deux fonctions :

```
session_unset(); //détruit toutes les variables session
session_destroy(); //met fin à la session
```

Il faut signaler qu'une session est fermée automatiquement après un certain délai défini dans *php.ini* à travers la directive **`session.gc_maxlifetime`**.

14.4 Remarque

Par mesure de sécurité, il est recommandé d'utiliser les sessions avec cookies. En effet, la transmission de l'identificateur de session **`SID`** par l'URL peut être interceptée par des personnes malveillantes qui peuvent par la suite l'utiliser pour une attaque. Il est aussi déconseillé de rallonger la durée de vie d'une session dans *php.ini* et ce par mesure de sécurité.

15. Insertion de scripts externes

Il est possible de sauvegarder des scripts (codes) qui sont utilisés par plusieurs pages dans un fichier (ou plusieurs fichiers) et l'inclure à fois qu'on a besoin de ces scripts. Ceci nous permet d'éviter de les réécrire à dans plusieurs pages le même code. Pour ce faire, nous avons deux fonctions :

```
include 'script.php';
require 'script.php';
```

Ces deux fonctions permettent d'inclure tout le contenu du fichier spécifié dans le fichier actuel. La première, en cas d'erreur, génère un simple avertissement (*warning*) alors que la seconde génère une erreur fatale.