PHP_Part2

(Selon les versions 4.x et 5.x)

1 Introduction

1.1 Définition

PHP est un langage de script HTML exécuté du côté du **serveur**. Il signifie « PHP : **Hypertext PreProcessor** ». Sa syntaxe est très inspirée du langage C, de Java et de Perl, avec des améliorations spécifiques.

Le but du langage est d'écrire rapidement des pages HTML dynamiques.

1.2 Historique

L'origine de PHP remonte à 1995 quand Rasmus Lerdorf a créé PHP/FI, une librairie de scripts Perl utilisés pour la publication de son CV sur son site personnel. Au fur et à mesure des évolutions, la librairie a été portée en C et agrémentée de nouvelles fonctionnalités pour créer des pages dynamiques simples pour le web et accéder à quelques sources de données. PHP/FI signifie Personal Home /Forms Interpreter.

PHP/FI 2.0 sort en 1997, toujours développé par une seule personne.
PHP 3.0 sort en juin 1998, c'est la première version développée conjointement par Rasmus Lerdorf, Andi Gutmans et Zeev Suraski et entièrement réécrite (les versions précédentes étaient trop lentes pour une application commerciale).

Le moteur de PHP 4 a été repensé afin d'en améliorer les performances pour des applications complexes et d'augmenter encore la modularité. PHP 4 sort officiellement en mai 2000. C'est actuellement la dernière version stable (sans compter les 4.1, 4.2 et 4.3).

Et PHP 5, son but est une amélioration des performances du moteur Zend (Zend Engine 2), un modèle objet étendu et très proche du C++.

Aujourd'hui PHP 5.6

1.3 Utilisation pratique

- Forums et Messageries
- Commerce électronique
- Banque / Comptes en ligne
- Publication en ligne
- Moteurs de recherche
- Tout ce que vous voulez, (sauf les jeux)

Quelques sigles :

- **HTML** : Hypertext Markup Language
- **PHP** : Hypertext PreProcessor
- **SQL** : Structured Query Language
- **MySQL** : serveur de base de données et les outils pour y accéder
- **LAMP** : Linux – Apache – MySQL – PHP, le quatuor gagnant des serveurs web.

1.4 Pages statiques vs Pages dynamiques

Une page statique et une page écrite directement en HTML. Elle peut éventuellement incorporer du code JavaScript lui donnant un semblant de 'dynamisme' mais uniquement du côté du navigateur et des données locales.

Pour des traitements plus lourds nécessitant l'accès à une base de données, un formatage de tableau en fonction de résultats, une recherche poussée, du graphisme, il faut passer par des pages dynamiques et par un langage qui sera exécuté du côté du serveur

: ASP sur les serveurs Microsoft/IIS, Perl, PHP...

1.5 Scripts CGI

PHP peut être utilisé comme langage CGI. Pour rappel, un script ou programme CGI est un programme comme un autre mais dont la sortie est dédiée à un navigateur, par exemple un programme Shell, C ou Perl qui sort du HTML. PHP étant un langage script comme un autre on peut l'utiliser dans ce cas, mais ce n'est pas forcément le mieux.

1.6 Pages dynamiques et PHP

PHP est un langage Server Side ou côté serveur. Lors du chargement d'une PHP, c'est le serveur qui va lire, interpréter et exécuter le code. Puis il renvoie le résultat, généralement sous la forme de code HTML au navigateur. Ainsi le navigateur et l'utilisateur ne voient jamais le véritable code PHP exécuté. De plus le résultat étant une page web classique en HTML, pas besoin d'installer sur le client des composants spécifiques (java, ...). Il n'y a donc pas de notion de vitesse d'exécution du côté du client, mais celle du serveur est prépondérante.

1.7 Ce que vous devez déjà connaître

Vous devez connaître HTML ainsi qu'un langage de programmation procédural (C par exemple). Une connaissance basique du SQL peut être nécessaire.

1.8 Le nécessaire serveur

PHP ne sert pas seulement à faire des pages dynamiques. C'est un langage interprété qui au même titre que Perl, Python ou TCL est capable de lancer des scripts intéractifs ou non. On peut même utiliser PHP pour créer des interfaces graphiques (extension GTK). Le minimum nécessaire et vital pour apprendre PHP est donc l'interpréteur PHP lui-même sur un environnement supporté (Unix, Windows, Mac, ...).

1.9 Le nécessaire client

Pour développer il suffit d'un simple éditeur mais il vaut mieux préférer un éditeur plus évolué supportant la coloration syntaxique et quelques fonctions évoluées. L'éditeur HAPEdit 3 est parfait pour développer en PHP. L'accès

aux bases MySQL peut se faire soit depuis un navigateur (phpMyAdmin) soit depuis une interface MySQLCC ou encore DBDesigner pour la conception.

Pour les tests : un simple navigateur respectant les standards du web.

2 Bases : Présentation

PHP est langage très souple prenant ses sources dans divers langages comme le C, le Perl, le C++. Il est donc possible d'avoir plusieurs styles de scripts (programmation classique dite procédurale ou programmation objet). Cette souplesse permet une très grande liberté, un peu comme en Perl. L'inconvénient est qu'on peut très vite obtenir du code illisible, même si ça marche très bien. Prenez donc l'habitude de commenter votre code, d'espacer votre texte et de ne placer qu'une instruction par ligne.

2.0.1 Syntaxe de base

2.1 Intégration à HTML

Une page php porte l'extension « `.php` ». Une page PHP peut être entièrement programmée en PHP ou mélangée avec du code html. PHP est un langage « Embedded HTML », c'est à dire qu'il apparaît à n'importe quel endroit de la page HTML. Pour ça on le place dans des balises particulières : `<?php` et `?>`. On peut aussi utiliser les balises `<script language="php">` et `</script>`. La première forme est préférable pour plus de simplicité et une compatibilité XHTML. On écrit donc une page HTML dans laquelle on intègre du code PHP.

```
<html>
<head>
<title>Titre</title>
</head>
<body>
<?php
echo "Hello
World !"; ?>
</body>
</html>
```

Le code HTML généré sera le suivant

```
<html>
<head>
<title> Titre </title>
</head>
<body>
Hello
World !
</body>
</html>
```

L'utilisation de balises pour l'intégration de code dans une page web est très souple et permet de jongler facilement avec du code PHP et du code HTML :

```
<?php
  if ( expression ) {
?>
  <strong> Ceci est
vrai.</strong> <?php
  } else {
?>
<strong> Ceci est
faux.</strong>
<?php
  }
?>
```

2.2 Séparateur d'instructions

Comme en C une instruction se termine par un point-virgule « ; ». Notez que la balise fermante ?> implique la fin d'une instruction.

```
<?php
   echo "Ceci est un test";
?>
<?php echo "Ceci est un test" ?>
```

2.3 Bloc d'instructions

Un bloc d'instructions se place entre accolades { }. Un bloc d'instructions peut contenir du code de n'importe quelle longueur et est considéré dans le reste du code comme une instruction unique. Si c'est une expression (qui a une valeur) on peut donc assigner le bloc, faire des calculs, ...

2.4 Commentaires

Les commentaires s'utilisent comme en C et en C++ avec `/*.. */` et `//`. Notez qu'une

balise fermante ferme le commentaire en cours.

```
<?php
/* echo "salut !" */
// Commentaire sur

cette ligne ?>
```

3 Les variables

3.1 Déclarer une variable

Une variable commence par un dollar « $ » suivi d'un nom de variable. Les variables ne sont pas typées au moment de leur création. Attention PHP est sensible à la casse : var et Var ne sont pas les mêmes variables ! Voici les règles à respecter :

- Une variable peut commencer par une lettre
- Une variable peut commencer par un souligné (underscore) « _ »
- **Une variable ne doit pas commencer par un chiffre.**

```
// Déclaration et
règles $var=1; //
$var   est   à   1
$Var=2;  //  $  Var
est      à      2
$_toto='Salut';
// Ok
$3petitsmoutons=5; // Invalide : commence par un chiffre
```

Leur type dépend de leur valeur et de leur contexte d'utilisation. Mais on peut forcer (cast) ponctuellement une variable à un type de données, ce qui s'appelle le transtypage. De même comme le type de variable peut changer en fonction de son utilisation ou du contexte, PHP effectue automatiquement un transtypage, ce qui peut parfois fournir des résultats surprenants. On affecte une valeur à une variable avec le signe égal « = » avec ou sans espace avant ou après.

```
// Déclaration et
transty   $var='2';
// Une chaîne 2
$var+=1; // $var est maintenant un entier 3
$var=$var+0.3; // $var est maintenant un réel

de type double 3.3 $var=5 + "3 petits

canards"; // $var est un entier qui vaut 8
```

Par défaut les variables sont assignées par valeur : la valeur assignée à la variable est recopiée dans la variable. PHP peut aussi travailler par référence. Une variable peut donc référencer une autre variable. On dit alors que la variable devient un alias, ou pointe sur une autre variable. On assigne par référence en utilisant le signe « & » devant la variable assignée

```
$var=2;
$ref=&$var; // $ref devient une
référence de $var echo $ref; //
affiche 2
$var=3;
echo $ref; //
affiche 3
$ref=4;
echo $var; // affiche 4
```

3.2 Portée des variables

La portée d'une variable dépend du contexte. Une variable déclarée dans un script et hors d'une fonction est globale mais par défaut sa portée est limitée au script courant, ainsi qu'au code éventuellement inclus (include, require) et n'est pas accessible dans les fonctions ou d'autres scripts.

```
$a=1; // globale par défaut function foo() {
 echo $a; // c'est une variable locale à la fonction :
 n'affiche rien
}
```

Pour accéder à une variable globale dans une fonction, il faut utiliser le mot-clé `global`.

```
$a=1; // globale
par défaut $b=2; //
idem
function foo() {
 global $a,$b; // on récupère les
 variables globales $b=$a+$b;
}
echo $b; // affiche 3
```

PHP accepte les variables statiques. Comme en C une variable statique ne perd pas sa valeur quand on sort d'une fonction.

```
function
 test_static() {
 static $a=0;
 echo $a; // +1 à chaque passage
 dans la fonction $a++;
}
```

3.3 Variables prédéfinies

PHP dispose d'un grand nombre de variables prédéfinies. Ces variables sont généralement de type scalaires ou des tableaux. Elles sont souvent de type superglobales, c'est à dire accessible depuis n'importe où sans notion de portée. Voici quelques tableaux prédéfinis (voir au point Tableaux pour comprendre leur utilisation).

- $_GLOBALS : tableau des variables globales. La clé est le nom de la variable.
- $_SERVER : variables fournies par le serveur web, par exemple 'SERVER_NAME'
- $_GET : variables fournies par HTTP par la méthode GET (formulaires)
- $_POST : idem mais pour la méthode POST
- $_COOKIE : les variables fournies par un cookie
- $_FILES : variables sur le téléchargement d'un fichier (upload)
- $_ENV : accès aux variables d'environnement du serveur
- $_SESSION : les variables de session.

Note : VERIFIEZ TOUJOURS VOTRE VERSION PHP, DES DEFINITIONS PEUVENT CHANGER D'UNE VERSION A UNE AUTRE !

3.4 Variables dynamiques

Une variable dynamique utilise la valeur d'une variable comme nom d'une autre variable. On utilise les variables dynamiques en rajoutant un « $ » devant le nom de la première variable.

```
$a="var";
$$a=1; // $$a=1 equivaut en fait
à $var=1 echo $a; // affiche var
echo $$a; //

affiche 1 echo

$var; // affiche 1
```

Attention avec les tableaux ! Pour éviter toute ambiguïté, il est préférable de placer la variable entre accolades.

3.5 Types de variables

3.5.1 booléens

Un booléen peut prendre deux valeurs `TRUE` ou `FALSE`. Les deux constantes `TRUE` et `FALSE` peuvent être utilisées sans aucune distinction de casse (pas de différences entre les majuscules et les minuscules).

```
$var=FALSE; // FALSE, False, false, ...
$var2=True; // TRUE, True, true, ...
```

Tous les types peuvent être convertis en booléens. Voici les cas où une variable retournera FALSE en booléen suivant le type :

- Booléen : FALSE
- Entier : 0 (zéro)
- Nombre flottant : 0.0 (zéro)
- Chaîne : chaîne vide "" ou "0" (zéro)
- Tableau : tableau vide sans aucun élément
- Objet : objet vide sans aucun élément
- Constante spéciale NULL

Dans tous les autres cas, la valeur retournée est TRUE. Attention : -1 est considéré comme vrai donc TRUE. Comme en C, les tests de conditions dans les structures de contrôles effectuent une convertion booléenne du la condition.

```
if($var==true) echo "ok";

if($var) echo "ok"; //

Identique
```

3.5.2 Entiers

Un entier est l'ensemble des nombres naturels, c'est à dire dans virgule, positifs ou négatifs. Les entiers sont généralement codés sur 32 bits mais cela dépend de l'architecture. Si on affecte un nombre entier qui dépasse la capacité de la variable, celle-ci sera transformée en réel (float). Enfin il n'y a pas de notion d'entier non signé.

Lors de la conversion d'un booléen en entier, FALSE devient 0 et TRUE devient 1. Lors de la conversion d'un nombre à virgule flottante, le nombre sera arrondi à la valeur inférieure s'il est positif, ou supérieure s'il est négatif (conversion vers zéro). Pour la conversion depuis les chaînes, voir à ce type.

3.5.3 Virgule flottante

On parte ici des nombres réels, double ou float, c'est à dire les nombres à virgules. La virgule est spécifiée par le point « . ». La puissance de 10 s'exprime avec le « e » ou le « E ».

```
$var=1.234;
$var2=1.1e4; // 1.1 * 10^4 : 11000
```

3.5.4 Chaînes de caractères

Une chaîne est une séquence de caractères. PHP travaille en ASCII soit 256 caractères, mais ne supporte pas encore le format Unicode, prévu dans la version 5. Il n'y a pas de limite théorique pour la taille de la chaîne.

On distingue trois syntaxes pour utiliser un chaîne

- Les guillemets simples '...' (apostrophes) : Comme en shell, tous les caractères inclus dans la chaîne sont sortis tels quels sans interprétation. Si vous devez afficher un guillemet simple, il faudra l'échapper : \'

- Les guillemets doubles "..." : Certaines séquences de caractères sont interprétées et les variables sont substituées (remplacées par leur valeur).
- HereDoc : Identique aux HereScripts en Shell. Le texte saisi jusqu'à un délimiteur spécifié est placé dans la variable.

```
$var\n"; $tata= <<<FIN
Salut les amis Comment ça va ? FIN;
echo $tata;
```

Si vous devez utiliser une chaîne de caractères comme valeur numérique (dans une addition par exemple, attention à son contenu. La chaîne sera de type double (réel) si elle contient un 'e' ou un 'E'. Sinon ce sera un entier. La valeur numérique est ensuite définie par le début de la chaîne. Si la chaîne commence par une valeur numérique, elle sera utilisée, sinon elle sera égale à 0 zéro. Si la première expression est une chaîne, le type de variable dépend de la seconde expression.

```
$val=10+"2.55"; // float,
12.55 $val=1+"tata2"; //
1 + 0 = 1
$val=2+"3 petits moutons"; // 2 + 3 = 5 (le 3 est en premier dans
la chaîne)
```

3.5.5 Les tableaux

Un tableau PHP est une association ordonnée. Une association fait correspondre des valeurs à des clés. Les tableaux sont très souples, ils peuvent avoir de multiples dimensions, être indexés par une clé numérique ou texte, être utilisés comme table de hachage, une pile, une queue, ... Une valeur de tableau peut être elle-même un tableau, pour créer des arbres par exemple.

Un tableau est créé avec la fonction `array()` qui prend comme arguments des paires « `key => value` » séparées par des virgules. La clé peut être soit un entier soit du texte. Attention, 8 est un entier, 08 une chaîne ! Si la clé est absente alors c'est la dernière clé entière plus 1 qui est choisie. Si c'est la première, c'est 0 zéro.

On accède aux éléments d'un tableau à l'aide des crochets « [et] ». On place entre ces crochets la clé entière ou la chaîne.

```
$var=array(10,15,17,2
3,9); echo $var[0];
// 10
echo $var[3]; // 17
$tab=array("a"=>12,"nom"=>"tata","pipo",
17,4=>5); echo $tab[0]; // pipo
echo $tab[1]; //
17 echo
$tab['a']; // 12
echo $tab['nom']; // toto
```

L'utilisation de la fonction array n'est pas obligatoire et on peut déclarer un tableau à la volée.

```
$tab2[1]=2;
$tab2[]=6; // equivaut

$tab2[2]=6
```

```
$tab2['test']='Ma chaîne';
```

On peut aussi créer des tableaux multidimensionnels à l'aide des méthodes précédentes.

```
$tab=array("un"=>array("riri",1=>"fifi",2=>'loulou'),2=>array(1,2,
3),array('un', 'deux','trois'));
echo $tab['un'][0]; //
riri echo $tab[2][1];
// 2
echo $tab[3][2]; // trois
$tab2['un']['deux']='test'; // créé un tableau à deux dimensions
```

Il existe une fonction très pratique pour lister le contenu d'un tableau, ou pour modifier les éléments : `foreach()`.

```
$tab=array(1=>'un',2=>'deux',3=>'t
rois'); foreach($tab as $valeur) {
 echo "$valeur \n"; // affiche un deux trois
}
foreach($tab as $cle => $valeur) {
 echo "$cle => $valeur\n"; // affiche 1 => un, 2 => deux, 3 => trois
}
```

3.5.6 La variable objet

Les objets se créent avec l'instruction `class`. Pour créer une instance de l'objet il faut utiliser le mot clé `new`.

```
class test  {
 function
  affiche_hello() {
  echo "Hello !";
 }
}
$obj=new test;
$obj-
>affiche_hello();
```

3.6 Les constantes

Les constantes est un nom qui prend une valeur ne pouvant pas être modifiée une fois fixée. Une constante n'est accessible qu'en lecture seule. Elles sont sensibles à la casse et doivent par convention être écrites en majuscules.

On définit une constante avec la fonction `define()` et doit respecter certaines règles :

- une constante ne commence pas par un $
- une constante est accessible depuis n'importe quel endroit du code
- une constante ne peut pas être redéfinie
- une constante ne peut contenir d'un scalaire (entier, booléen, chaîne, double).

```
define(CONSTANTE,"valeur");
echo CONSTANTE; // affiche "valeur"
```

3.7 Obtenir le type d'une variable

Pour obtenir le type d'une variable, on utilise la fonction « **gettype** » qui retourne une chaîne de texte indiquant le type. **Mais attention rien ne garantit que le résultat soit le même d'une version PHP à une autre.**

Les types retournés sont "boolean", "integer", "double" (pour des raisons historiques, "double" est retournée lorsqu'une valeur de type float est fournie, au lieu de la chaîne "float"), "string", "array", "object", "resource" (depuis PHP 4), "NULL" (depuis PHP 4), "unknown type"

Si vous souhaitez réellement tester le type d'une variable, il est préférable d'utiliser les fonctions de type « **is_*** » : **is_array, is_bool, is_double, is_float, is_int, is_integer, is_long, is_null, is_numeric, is_object, is_real, is_resource, is_string, is_callable (est-ce une fonction).**

3.8 Définir et supprimer une variable

Si vous souhaitez savoir si une variable est définie, c'est à dire si elle est affectée, on utilise « **isset** ».

Enfin si vous souhaitez supprimer une variable, il faut utiliser « **unset** ».

4 Les opérateurs

4.1 La précédence des opérateurs

C'est l'ordre dans lequel les valeurs doivent être analysées. Ainsi l'expression 4 + 3 * 7 n'est pas lue de gauche à droite et ne retourne pas 49 mais 25. Voici le tableau des priorités des opérateurs par ordre croissant de priorité :

Associativité	Opérateurs
gauche	,
gauche	or
gauche	xor
gauche	and
Droite	print
gauche	= += -= *= /= .= %= &= |= ^= ~= <<=>>=
gauche	? :
gauche	||
gauche	&&
gauche	|
gauche	^
gauche	&
non-associative	== != === !==
non-associative	< <= > >=

gauche	<< >>
gauche	+ - .
gauche	* / %
Droite	! ~ ++ -- (int) (double) (string) (array) (object) @
Droite	[
non-associative	new

4.2 Opérateurs arithmétiques

Les opérateurs +, -, *, / et %. Le « % » est le modulo : le reste de la division.

4.3 Opérateurs d'assignation

Le principal est le = mais on a aussi comme en C des opérateurs combinés +=, -=, *=, /=, %=, .= ...

4.4 Opérateurs sur les bits

Les opérateurs sont le & (AND), | (OR), ^ (XOR), ~ (NOT, ~$a), >> ($a>>$b décalage de $b bits sur la gauche) et << ($a << $b décalage de $b bits sur la droite). Un décalage de bits sur la gauche équivaut à une multiplication par deux, un décalage sur la droite à une division par deux.

4.5 Opérateurs de comparaison

Les opérateurs sont == ($a==$b, même valeur), === ($a===$b, même valeur et même type), != ou <> (différent), <, >, <=, >=.

Il y a aussi l'opérateur ternaire « ?: » expr1?expr2:expr3 Si expr1 est vrai alors expr2 sinon expr3.

4.6 Opérateur d'erreur

On dispose d'un opérateur spécial @ qui appliqué à une expression empêche la sortie d'un message d'erreur en cas de problème. On peut toujours récupérer le message d'erreur éventuel à l'aide de la variable $php_errormsg mais uniquement si l'option « track_errors » est à « On » dans le php.ini.

```
$retour=@$tab['toto']; // ne retourne pas d'erreurs si l'index
toto n'existe pas
```

4.7 Opérateur d'exécution

On peut exécuter des commandes externes au PHP comme en Shell avec les opérateurs « guillemts inverses « ` » (altgr+6). Attention l'option « safe_mode » doit être à « On » dans le php.ini. On peut aussi utiliser la fonction « shell_exec » qui fait exécuter une commande par le shell.

4.8 Opérateurs d'incrémentation/décrémentation

On dispose comme en C des opérateurs ++ et --, à utiliser avant ou après le nom de variable.

```
$a++; // retourne $a puis
l'incrémente de 1 ++$a; //
incrémente $a de 1 puis retourne $a
$a--; // retourne $a puis décrémente
de 1 --$a; // décrémente $a de 1
puis retourne $a
```

Attention ! Les opérateurs réagissent aux types de variables. Le PHP réagit comme en PERL. Ainsi :

```
$a='Z
';
$a++;
echo $a; //
retourne AA $a++;
echo $a; // retourne AB
```

4.9 Opérateurs logiques

Les opérateurs logiques sont :

```
« and » ou « && » ($a and $b, $a && $b) vrai si $a et $b sont vrais
« or » ou « || » ($a or $b, $a || $b) vrai si $a ou $b sont vrais
« xor » ($a xor $b) vrai si $a ou $b sont vrais mais pas les deux
  en même temps
« ! » (!$a) vrai si $a est faux.
```

Attention, and et or n'ont pas la même priorité (priorité plus faible) que && et || !

4.10 Opérateurs de chaînes

Il y a deux opérateurs de chaînes : le « . » qui concatène deux chaînes entre elles et le « .= » déjà vu qui est l'opérateur d'assignation.

```
$a="Bonjour";
$b=$a." les amis"; // $b contient Bonjour les

amis $b.="! Salut."; // $b contient Bonjour

les amis! Salut.
```

4.11 Opérateur de tableaux

On peut « additionner » deux tableaux entre eux avec le « + » : le tableau de droite est ajouté au tableau de gauche.

5 La notion d'expression

En PHP, une expression peut être résumée en « tout ce qui a une valeur ». Ceci dit, on remarque vite qu'en PHP tout ou presque est une expression. Une variable ou une constante se voient affectés des valeurs. Cette valeur est donc l'expression de la variable ou de la constante.

Nous pouvons résumer en disant qu'une expression représente tout ce qui peut être évalué. On ne peut évaluer que les valeurs…

6 Les structures de contrôle

6.1 if

```
if(expression) commande ou { bloc de
commandes } else commande ou { bloc de
commandes }
```

Il y a aussi le « elseif », combinaison du if et du else. Le elseif en un mot peut aussi s'écrire en deux mots : le résultat est le même. On peut écrire des elseif en chaîne. Le premier dont l'expression est vrai est exécuté.

```
If(expression) commande ou { bloc de
commandes } elsif(expression) commande ou
{ bloc de commandes } elsif(expression)
commande ou { bloc de commandes }
...
```

On peut placer du HTML comme commande ou dans le bloc de commande.

```
<?php if ($a == 5)
{ ?> A = 5
<?php } ?>
```

On peut aussi utiliser une syntaxe alternative : on ouvre le bloc (juste après le if, le else ou le elseif) avec les « : » deux points, et on ferme l'instruction avec « endif ».

```
<?php
if ($a == 5):
    print "a = 5";
    print "...";
elseif ($a == 6):
    print "a = 6";
    print "!!!";
else:
    print "a ne vaut ni 5 ni
6"; endif;
?>
```

6.2 while

6.2.1 while classique

C'est la boucle « tant que » simple : tant que la condition n'est pas vraie, on continue la boucle. L'expression est placée en début de boucle : si l'expression est fausse avant de rentrer dans la boucle, la boucle n'est pas exécutée.

```
While(expression) commande ou { bloc de commandes }
```

On peut aussi utiliser la syntaxe alternative :

```
while(expression): commande ou { bloc de
commandes } endwhile
```

6.2.2 do ... while

C'est la seconde possibilité. Dans ce cas la commande ou le bloc de commande est exécutée au moins une fois, car l'expression conditionnelle est testée en fin de boucle.

```
do { bloc de commandes } while(expression)
```

6.3 for

Le « for » du PHP est identique au « for » du C.

```
for(expr1;expr2;expr3) commande ou { bloc de commandes }
```

« expr1 » est exécutée à la première entrée dans la boucle. « expr2 » est exécutée à chaque début d'itération jusqu'à ce que l'expression soit fausse auquel cas on sort de la boucle. « expr3 » est exécutée à la fin de l'itération.

L'usage habituel d'une telle boucle est de placer l'état initial en expr1, la condition de sortie en expr2 et le calcul en expr3. Mais on peu effectuer toutes sortes de choses.

```
// de 1 à 10
for ($i = 1; $i <= 10; print $i, $i++)
// infini
for(;;)
// de 1 à 10
for ($i = 1; $i <= 10; print $i, $i++) ;
```

On peut employer une syntaxe alternative avec le « : » et « endfor ».

```
for(expr1;expr2;expr3): commande ou { bloc de commandes
} endfor
```

6.4 foreach

La boucle « foreach » est peut-être l'une des plus intéressantes pour la manipulation de tableaux ou de résultats de requêtes SQL. Elle permet de lister les tableaux. Elle dispose de deux syntaxes.

```
foreach(array_expression as $value) commandes
foreach(array_expression as $key => $value) commandes
```

La première syntaxe récupère les éléments du tableau un par un, séquentiellement. La valeur de l'élément courant du tableau est placée dans $value.

La seconde syntaxe est presque identique, sauf qu'en plus la clé (l'index) de l'élément actuel est placée dans $key.

Attention : modifier la valeur de $value (ou de $key) ne modifie pas le tableau car cette boucle travaille sur une copie, pas une référence. Par contre dans le second cas, comme on dispose de la clé, rien n'empêche d'assigner quoi que ce soit à l'élément courant.

Remarque : un appel à foreach « rembobine » automatiquement le tableau à son premier élément. Mais pas dans les autres boucles, il faut alors utiliser « reset ».

```
reset($arr);
while (list(, $value) = each ($arr))
 { echo "Valeur: $value<br>\n";
}
```

```
foreach ($arr as $value) {
 echo "Valeur: $value<br>\n";
}
$a = array ( "un"
     => 1, "deux"
     => 2, "trois"
     => 3,
     "dix-sept" => 17
);

foreach($a as $k => $v) {
     print "\$a[$k] => $v.\n";
}
```

6.5 break et continue

L'instruction « break » permet de sortir d'un for, while, foreach ou switch. On peut lui indiquer de combien de structures on souhaite sortir si elles sont emboîtées.

L'instruction « continue » permet de passer à l'itération suivante. Attention PHP conside le switch comme une boucle, et dans ce cas, réévalue le switch. On peut indiquer à continue combien de structures emboîtées relancer.

6.6 switch

Le « switch » est équivalent à une série de if et permet de comparer avec un grand nombre de valeurs.

```
switch ($i) {
     case 0:
          print "i egale
          0"; break;
     case 1:
          print "i egale
          1"; break;
     case 2:
          print "i egale
          2"; break;
     default:
          print "i est inférieur à 0 ou supérieur à 2 »;
}
```

Le switch s'arrête à la première expression case vraie puis exécute le code suivant dans l'ordre indiqué, jusqu'à la première instruction break. S'il n'y a pas de break, tout le code jusqu'à la fin du switch est exécuté. Dans l'exemple suivant, si $i vaut 0, tous les print seront affichés !

```
switch ($i) {
     case 0:
          print "i egale
     0"; case 1:
          print "i egale
     1"; case 2:
          print "i egale 2";
}
```

Notez aussi que le default doit intervenir en dernier, sinon il n'a aucun intérêt.

Enfin on peut employer une syntaxe alternative avec « : » et « endswitch ».

```
switch ($i):
    case 0:
        print "i egale
        0"; break;
    case 1:
        print "i egale
        1"; break;
endswitch
```

6.7 return

Contrairement à d'autres langages, « return » n'est pas une fonction mais une instruction. Dans une fonction, return sert à sortir de celle-ci et à retourner une valeur. Dans un script, elle sort de celui-ci. Attention cependant dans les scripts inclus (voir require et include) : le return dans ce type de code considère qu'il sort de la fonction « require » ou « include » et donc ne sort pas du script ou de la fonction dans lesquels ce code a été inclus !

Comme return est un élément du langage et pas une fonction il y a pas besoin d'utiliser les parenthèses.

6.8 require et include (_once)

« require » et « include » incluent à l'endroit actuel et exécutent le fichier PHP. Ils sont identiques dans leur fonctionnement à une exception : le traitement des erreurs. Un include produit un « warning » (le code continue en principe à s'exécuter) tandis qu'un require produit une « erreur fatale » (l'exécution s'arrête).

Comme require et include sont des éléments du langage et pas des fonctions il y a pas besoin d'utiliser les parenthèses.

7 Gestion des formulaires en PHP

7.1 GET et POST

Le but est de récupérer le contenu des champs d'un formulaire HTML dans notre code PHP pour pouvoir le traiter. Lorsqu'un formulaire est envoyé à un script PHP, toutes les variables seront disponibles automatiquement dans le script.

Les formulaires peuvent être de type GET ou POST. Pour rappel, dans un formulaire de type GET, les informations sont passées directement par l'URL en clair, ce qui peut poser des problèmes de limitations suivant le serveur (de 256 à 8192 octets selon le cas). La méthode POST n'a pas ce genre de limitation, car les informations sont transmises par le conteneur de variables globales (dans l'entête) et sont de plus cachées. PHP peut gérer les deux méthodes de manière transparente.

7.2 Récupération par tableau

Chaque champ de formulaire en PHP est défini par un nom et une valeur. Dans un script, PHP va récupérer ces noms et ces valeurs dans des tableaux spéciaux dit superglobaux (accessibles depuis partout). Pour la méthode GET, le tableau est $_GET, pour la méthode

POST le tableau est $_POST. Si vous ne souhaitez pas vous soucier de la méthode, vous pouvez utiliser le tableau $_REQUEST. En index on aura le nom du champ de formulaire (ou de la variable passée en URL) et en valeur la valeur du champ. Par exemple :

```
<form action="foo.php" method="post">
Name: <input type="text" name="username"><br>
Email: <input type="text" name="email"><br>
<input type="submit" name="submit" value="Submit
me!"> </form>
```

Dans la PHP foo.php on aura :

```
<?php
echo  $_POST['username'];
echo  $_REQUEST['email'];
?>
```

Imaginons l'appel d'une page test.php par une URL comme ceci :

http://www.monsite.com/test.php?id=1

Ici on transmet une variable via une URL et donc la méthode implicite GET. Pour récupérer « id » dans un code PHP on peut donc faire :

```
<?php
echo $_GET['id'];
echo
$_REQUEST['id']; ?>
```

$_GET ne contiendra que les variables de type GET. $_POST ne contiendra que les variables de type POST. $_REQUEST contient les variables de type POST et GET mais aussi les variables de cookies. Il est aussi possible de procéder autrement en récupérant le nom du champ directement en tant que variable sans passer par un tableau. **Pour cela il faut vérifier dans le fichier php.ini que la valeur register_globals est à on**. Dans ce cas les noms des champs de formulaire seront les noms des variables :

```
<?php
echo $username;
echo $email;
?>
```

8 MySQL

8.1 Présentation

MySQL est un SGBDR : « Système de Gestion de base de Données Relationnel » qui se définit lui-même comme étant « La base de données Open Source la plus populaire au monde ». Edité par la société MySQL AB, MySQL est un produit Open Source libre d'utilisation sous licence GPL pour les projets libres.
La version de production actuelle de MySQL est la version 4 (4.0.17 à l'écriture de ce support), mais la grande majorité des serveurs des hébergeurs sont encore dans les dernières versions de MySQL 3.23 (3.23.58). La future version actuellement en développement est la 5.0.0 et n'est surtout pas à utiliser en production.

Les principales qualités de MySQL sont sa simplicité et sa rapidité. Son principale défaut est le

manque de fonctionnalités dites avancées (dans les versions précédentes) : clé étrangères, procédures stockées, triggers et selects imbriqués notamment. Mais cela ne doit pas occulter sa puissance avec l'ajout de fonctionnalités avancées comme une syntaxe SQL étendue (replace, limit, delete), les index de recherche « fulltext » permettant de créer des moteurs de recherche, ...

La prochaine version stable (5.0) comblera les lacunes des précédentes versions avec le support complet de la syntaxe SQL ANSI-99.

8.2 Outils

8.2.1 PhpMyAdmin

L'outil phpMyAdmin est une interface web à MySQL permettant d'effectuer la plupart des tâches de maintenance et d'utilisation. Cette solution fonctionne depuis n'importe quel navigateur et est indépendante de la machine. On accède à phpMyAdmin généralement par l'URL http://server_name/mysql.

8.2.2 MysqlCC

MysqlCC (MySQL Control Center) est le front-end graphique officiel de MySQL, développé par la même société. Basé sur e toolkit Qt, il est disponible sous Windows, Unix (linux) et bientôt sur MacOS. Il permet l'administration du serveur, la gestion des bases et tables, l'exécution de requêtes SQL interactives avec coloration syntaxique...

8.3 Créer une base

A partir de l'écran d'accueil de phpMyAdmin, on saisit le nom de la base dans « Créer une base de données ». Il faut de préférence choisir un nom simple et intuitif. Puis on clique sur « Créer ». Après la création une nouvelle s'affiche : c'est la principale d'administration de la base. En haut seront toujours présents après l'exécution d'une commande les résultats de celle-ci. Cet écran permet notamment l'exécution de commandes SQL, et le travail sur les tables.

8.4 MySQL et PHP

8.4.1 Connexion à une base de données

Deux étapes sont généralement nécessaires pour plus de simplicité. La seconde n'est pas obligatoire mais bien pratique dans le cas où on travaille sur une seule base de données dans toute la .

8.4.1.1 Connexion au serveur

On utilise la fonction **mysql_connect()**. Cette fonction prend (au minimum) trois paramètres : le serveur (hostname) l'utilisateur et son mot de passe.

```
$b_host="sql.tf-data.net"
$b_user="toto";
$b_pass="CreT1";
$cnx=mysql_connect($b_host, $b_user, $b_pass); if(!$cnx) die ("erreur de connexion à MySQL");
```

8.4.1.2 Choix d'une base

On choisit une base de données par défaut avec la fonction **mysql_select_db()**. Elle prend au

minimum un paramètre, le nom de la base. Le second paramètre optionnel est une ressource retournée par mysql_connect, en cas de connexion sur plusieurs serveurs MySQL. Par défaut, la fonction prend la dernière connexion ouverte (celle du dernier mysql_connect exécuté).

```
$b_base="slyunix";
$db=msql_select_db($b_base);
if(!$db) die("Erreur de connexion à la base $b_base");
```

8.4.1.3 Fermeture d'une connexion

On ferme une connexion à MySQL avec la fonction **mysql_close()**. Cependant dans le cas de connexion non persistante, cette fonction n'est pas obligatoire car PHP ferme automatiquement les connexions à la fin d'un script.

```
mysql_close($cnx);
```

8.4.1.4 Séquence complète pour une base

```
$b_host="sql.tf-data.net"
$b_user="toto";
$b_pass="CreT1";
$b_base="slyunix";
mysql_connect($b_host, $b_user, $b_pass) or die("erreur de connexion à MySQL");
mysql_select_db($b_base) or die("erreur à la selection de $b_base);
mysql_close();
```

8.4.2 Les requêtes

8.4.2.1 Exécuter une requête

On exécute une requête SQL avec la fonction **mysql_query()**. Cette fonction prend au moins un paramètre : une requête SQL sous forme de chaîne. La fonction retourne FALSE en cas d'échec (colonne ou table invalide, droits insuffisants, pas de connexion, etc).

ATTENTION : Ce n'est pas parce que l'appel à mysql_query() n'a pas retourné d'erreur que la fonction retourne des lignes de résultats dans le cas d'un SELECT par exemple. Enfin, la requête SQL ne doit pas finir par un point-virgule.

La requête peut être de n'importe quel type (selection, mise à jour, destruction, etc). Dans le cas d'un SELECT, SHOW, EXPLAIN ou DESCRIBE, mysql_query() retourne une ressource qui sera ensuite utilisée pour lire le résultat.

```
$result=mysql_query("select id_message, sujet from f_message");
```

8.4.2.2 Nombre de lignes affectées

Dans le cas d'un DELETE, INSERT, REPLACE ou UPDATE, on peut connaître le nombre de lignes affectées (modifiées) par la requête à l'aide de la fonction **mysql_affected_rows()**.

```
$result=mysql_query("delete from f_message where login='toto'");
if($result) echo mysql_affected_rows()." Enregistrements supprimés";
```

8.4.2.3 Nombre de lignes retournées

Dans le cas d'un SELECT, le nombre d'enregistrements (lignes) retourné est obtenu avec la fonction **mysql_num_rows()**.

```
$result=mysql_query("select * from f_message where id_message > 10");
if($result) echo mysql_num_rows().' enregistrements retournés');
```

8.4.2.4 Récupérer les résultats

La fonction la plus sympathique pour récupérer les enregistrements après l'exécution d'une sélection est **mysql_fetch_array()**. Elle prend au minimum un paramètre : une ressource résultat (résultat de mysql_query). Elle retourne une ligne de résultat sous forme d'un tableau associatif, d'un tableau indexé ou des deux. Par défaut, le tableau retourné est à la fois associatif et indexé.

Dans un tableau associatif, l'index du tableau est le nom du champ correspondant à la colonne. Dans un tableau indexé, les colonnes sont numérotées à partir de zéro.

Notez que mysql_fetch_array() ne retourne qu'une seule ligne de résultat. Pour passer à la suivante, il faut exécuter la fonction à nouveau. Elle retournera FALSE quand il n'y aura plus de lignes à lire.

```
$result=mysql_query("select sujet,texte from f_message where login='toto'");
$tab=mysql_fetch_array($result); // Première ligne du résultat
echo $tab['sujet'].', '.$tab['texte']; // affiche les champs sujet et
texte echo $tab[0].', '.$tab[1]; // idem
```

Si plusieurs colonnes portent le même nom, la dernière colonne sera prioritaire. Dans une requête affichant des noms de colonnes identiques, le mieux est de les renommer :

```
SELECT t1.nom as col1, t2_nom as col2 FROM t1, t2 ...
```

8.4.3 Récupération des erreurs

En cas d'erreur lors d'une manipulation sur MySQL, on peut récupérer le numéro de l'erreur MySQL par la fonction **mysql_errno()**. De même on peut récupérer le message d'erreur par la fonction **mysql_error()**.

On peut trouver la liste des codes d'erreurs ici :

http://www.mysql.com/doc/en/Error-returns.html

9 Le système de fichiers

9.1 Travail sur les fichiers

9.1.1 Ouverture

La fonction **fopen()** permet d'ouvrir ou de créer un fichier selon divers modes. Elle prend en premier paramètre le nom du fichier avec son chemin. Ce nom de fichier peut être une URL (il faut pour cela que la directive **allow_url_fopen** soit activée dans le php.ini). Le second paramètre est le mode d'ouverture :

- **r** :lecture seule, en début de fichier
- **r+**: lecture/écriture, en début de fichier
- **w** : écriture seule , taille du fichier à 0 (ancien contenu effacé), créé s'il n'existe pas
- **w+** : lecture/écriture, taille du fichier à 0, créé s'il n'existe pas
- **a** : écriture seule, en fin de fichier, créé s'il n'existe pas
- **a+** : lecture/écriture, en fin de fichier, créé s'il n'existe pas.

On peut sous Windows rajouter la lettre 'b' pour le type binaire, inutile sous Unix.

Enfin, fopen() admet un troisième paramètre : 1. Dans ce cas le fichier à ouvrir sera recherché dans le **include_path** (voir php.ini).

La valeur retournée est une ressource de fichier (identifiant de fichier) ou la valeur FALSE en cas d'échec.

```
$file=fopen('server.log','a+',1);
```

9.1.2 Lecture

9.1.2.1 fgets()

La fonction **fgets()** permet de lire une ligne d'un fichier en mode texte. La fonction s'arrête lorsqu'elle arrive à la fin du fichier ou à un retour chariot. On peut préciser en second paramètre une longueur de texte en octet qui est par défaut 1024 et ce paramètre est obligatoire avant les versions 4.2. Le premier paramètre est l'identifiant du fichier. FALSE sera retourné en cas d'erreur.
ATTENTION : fgets() ne retourne pas FALSE à la fin du fichier.

```
$ligne=fgets($file);
```

9.1.2.2 fread();

La fonction **fread()** permet de lire un fichier en mode binaire. Le second paramètre est la longueur souhaitée. Sous Windows il faut penser à placer 'b' dans fopen(). La lecture s'arrête lorsque les n octets (longueur) ont été lus, la fin du fichier a été atteinte ou qu'une erreur est survenue.

```
$file=fopen('monimage.gif','rb');
$img=fread($file,
$filesize('nomimage.gif')); fclose($file);
```

9.1.2.3 fscanf()

La fonction **fscanf()** lit des lignes d'un fichier en fonction d'un formatage particulier.

```
$file=fopen('users.txt','r');
while($ligne=fscanf($file,"%s\t%s\t%s\n"))
{ echo $ligne;
}
```

9.1.3 Ecriture

Les deux fonctions **fwrite()** et **fputs()** sont identiques 100% car **fputs()** est un alias de **fwrite()**. Elles écrivent une chaîne dans un fichier. On peut limiter la longueur de la chaîne en troisième paramètre mais par défaut toute la chaîne est écrite. La valeur retournée est le nombre d'octets écrits, ou FALSE en cas d'erreur.

```
$file=fopen('monfic.txt','a);
$ret=fwrite($file,'Une ligne dans mon
fichier'); fclose($file);
```

En cas de travail sur un fichier binaire, il ne faut pas oublier sous Windows de rajouter l'option 'b'.

9.1.4 Déplacement

9.1.4.1 fseek()

La fonction **fseek()** permet de se déplacer dans un fichier. Le second paramètre (offset) est le nombre d'octets de déplacement. Il peut être négatif. Par défaut le déplacement est calculé à partir du début du fichier mais un troisième paramètre peut modifier cet état. La valeur retournée est 0 (zéro) en cas de réussite, -1 sinon. Un positionnement après la fin du fichier n'est pas une erreur.

- SEEK_SET : Calcul à partir du début du fichier. Position finale : Début+offset
- SEEK_CUR : Calcul à partir de l'emplacement actuel. Position finale : Actuelle+offset
- SEEK_END : Calcul à partir de la fin du fichier. Position finale : Fin+Offset.

```
$ret=fseek($file,0, SEEK_END); // Fin du fichier
```

9.1.4.2 ftell()

La fonction **fseek()** renvoie la position actuelle du pointeur dans le fichier.

```
$pos=ftell($file);
```

9.1.4.3 rewind()

La fonction **rewind()** permet de retourner au début du fichier. Elle retourne TRUE en cas de réussite, FALSE sinon.

9.1.5 Fin de fichier

La fonction **feof()** indique si on est à la fin du fichier ou non. Elle retourne TRUE si la fin de fichier a été atteinte, FALSE sinon.

```
$fd = fopen ("/tmp/inputfile.txt",
"r"); while (!feof ($fd)) {
 $buffer = fgets($fd,
 4096); echo $buffer;
}
fclose ($fd);
```

9.1.6 Fermeture

On ferme un fichier avec la fonction **fclose()**.Elle retourne TRUE en cas de réussite, FALSE sinon.

10 Sessions et cookies

10.1 Les cookies

10.1.1 Création

Un cookie s'envoie avec la fonction **setcookie()**. Les arguments sont les suivants :

- Nom
- Valeur
- date d'expiration (un timestamp unix)
- chemin (validité du cookie suivant le chemin dans l'url)
- domaine (pare défaut, le vôtre)
- secure : flag de sécurité : le cookie n'est accessible que via une connexion sécurisée.

Seul le premier argument est obligatoire. Pour ne pas spécifier une valeur, il suffit de ne rien mettre. Attention à quelques limitations :

- Un cookie étant envoyé avec les entêtes HTTP, il ne peut pas être envoyé si une sortie a déjà eu lieu (html, texte, ...)
- Un cookie n'est pas disponible dans la qui l'a créé. Il faut soit recharger la , soit pointer sur une autre.
- Un cookie ne peut être supprimé qu'avec les mêmes paramètres qui ont servi à sa création. C'est le navigateur qui supprime le cookie.

```
setcookie("testcookie",session_id(),time()+3600);
if(isset($_COOKIE['testcookie'])) echo $_COOKIE['testcookie'];
```

Dans l'exemple précédent, au premier chargement rien ne sera affiché car le contenu du cookie n'est pas encore accessible. Au deuxième chargement la valeur sera réellement affichée, mais attention au décalage entre la mise à jour et l'accès.

```
if(!isset($_COOKIE['testcookie'])) {
 setcookie("testcookie",'toto',time()+3600);
 header("Location: http://".$_SERVER['HTTP_HOST'].$_SERVER['REQUEST_URI']);
}
```

Dans cet exemple, on regarde si le cookie existe, sinon on le créé et on rappelle la .

10.1.2 Accès

On accède au cookie grâce à la variable globale $_COOKIE qui est un tableau. L'index du tableau est le nom du cookie.

```
$valeur=$_COOKIE['testcookie'];
echo $valeur; // par rapport à l'exemple précédent : toto
```

Note : on peut placer les tableaux avec les cookies. Il suffit de nommer son cookie avec une notation par crochets.

```
setcookie("testcookie[1]",'toto',time()+3600);
setcookie("testcookie[2]",'titi',time()+3600);
setcookie("testcookie[3]",'tata',time()+3600);

... (   suivante)

foreach($_COOKIE as $tab) {
 if(is_array($tab)) foreach($tab as $key => $value) echo "$key => $value";
}
```

10.1.3 Suppression

Pour supprimer un cookie, il suffit de donner une date antérieure à la date actuelle à celui-ci.

```
if(!isset($_COOKIE['testcookie'])) {
setcookie("testcookie",'toto',time()+3600);
} else { setcookie("testcookie",'',time()-
3600);
}
if(isset($_COOKIE['testcookie'])) echo
"OK"; else echo "Pas de cookie";
```

Dans l'exemple précédent, les appels au script vont provoquer successivement l'affichage de « pas de cookie » et « OK ».

10.2 Identification HTTP

10.2.1 Connexion

Cette possibilité n'est disponible que si PHP est utilisé comme module Apache, et pas en tant que script CGI. L'identification HTTP permet via l'envoi d'un HEADER HTTP de demander une identification au client. Dans ce cas le navigateur va ouvrir une fenêtre de connexion demandant une saisie de login et de mot de passe. PHP récupère bien entendu ces valeurs. Voici l'exemple issu de la documentation PHP

```
if (!isset($_SERVER['PHP_AUTH_USER'])) { header('WWW-
 Authenticate: Basic realm="Slyunix"');
 header('HTTP/1.0 401 Unauthorized');
 echo 'Texte utilisé si le visiteur utilise le bouton
 d\'annulation'; exit;
} else {
 echo "<p>Bonjour, {$_SERVER['PHP_AUTH_USER']}.</p>";
 echo "<p>Votre mot de passe est {$_SERVER['PHP_AUTH_PW']}.</p>";
}
```

Dans cet exemple, on passe en entête une demande d'authentification. En cas d'échec (bouton annuler) un message d'annulation est affiché. Sinon on peut récupérer les valeurs via les variables :

- $_SERVER['PHP_AUTH_USER'] : login
- $_SERVER['PHP_AUTH_PW'] : mot de passe

10.2.2 Déconnexion

La déconnexion n'est pas si évidente que ça car une fois connecté, les identifiants sont placés dans la session. La méthode la plus simple consiste à fermer le navigateur et à le relancer.

10.3 Sessions PHP

10.3.1 Principe

Les sessions permettent de préserver des données lors de la visite d'un site. Chaque personne se voit attribué un identifiant unique appelé identifiant de session, ou SID. Ce SID est soit stocké dans un cookie, soit passé par URL. On peut ainsi définir un nombre infini de variables qui seront accessibles durant toute la durée de la session.

Notez que si vous fermez et relancez votre navigateur, vous changez d'identifiant, et donc la précédente session est perdue, même si elle était nommée.

On peut connaître son SID grâce à la constante de même nom, ou par la fonction **session_id()**.

10.3.2 Utilisation

10.3.2.1 Ouverture

Si dans PHP.INI la valeur « session.auto_start » est activée, les sessions sont démarrées de manière automatique lors de l'accès à une page. Sinon il faut utiliser explicitement la fonction **session_start()**. Son appel créé une session ou restaure une session précédemment stockée sur le serveur.

ATTENTION : L'ouverture d'une session doit obligatoirement apparaître avant toute sortie/affichage du script PHP ou de balises HTML, même une ligne vide !

On peut créer ou récupérer une session nommée en précisant un **session_name()** qui prend comme paramètre le nom de la session qui doit être créée (premier appel) ou accédée (appels suivants). La session par défaut se nomme « **PHPSESSID** » et c'est celle-là qui est chargée si session_name() n'est pas appelée. Il faut donc appeler session_name() AVANT session_start() pour accéder à une session nommée.

Une session peut aussi être indirectement créée ou ouverte avec l'utilisation de la fonction **session_register()**.

10.3.2.2 Variables de session

Il y a deux moyens de créer des variables de session, qui seront accessibles depuis n'importe quelle page visitée à partir du moment ou session_start() a été appelée. Le premier moyen ne fonctionne ne fonctionne pas si **register_globals** est à **off** dans PHP.INI :

Méthode 1 (déconseillée) :
La fonction **session_register()** prend comme paramètre un nom de variable et la définit comme variable de session. Cette variable peut ensuite être accédée par son nom.

```
$txt='Bonjour les amis';
session_register('txt'); // txt devient une variable de session.
```

Pour savoir si une variable est enregistrée dans la session courante, on utilise **session_is_registered()**.

```
if(session_is_registered('txt')) echo $txt;
```

Pour supprimer une variable de session, il faut utiliser **session_unregister()**. Par contre il ne faut pas oublier d'utiliser **unset()** pour supprimer la variable globale, car la précédente fonction ne la supprime pas, elle empêche seulement sa sauvegarde à la fin du script.

```
if(session_is_registered('txt'))
 { session_unregister('txt');
 unset($txt);
}
```

Méthode 2 (La meilleure à utiliser) :

On utilise tout simplement le tableau global **$_SESSION**. Ainsi :

```
$_SESSION['txt']="Bonjour les amis";
```

crée la variable de session txt,

```
echo $_SESSION['txt']
```

l'affiche et

```
unset($_SESSION['txt']);
```

l'efface. Pour effacer toutes les variables de sessions, il suffit de faire

```
$_SESSION=array();
```

10.3.3 Expiration de la session

La durée de vie d'une session PHP dépend de trois paramètres de configuration du fichier PHP.INI : **session.gc_maxlifetime, session.cookie_lifetime** et **session.cache_expire**. Le premier concerne la durée de vie des données sur le serveur, en nombre de secondes. Il est par défaut de 1440 secondes, soit 24 minutes. Au delà, les données sont supprimées et elles seront rechargées via le cookie de session lors du prochain appel.

Le second est la durée de vie du cookie de session, exprimé en secondes. Par défaut la valeur est zéro (0) ce qui signifie : jusqu'à ce que le navigateur soit éteint.

Le troisième est la durée de vie des données de session, en minutes. Sa valeur par défaut est de 180 minutes, soit trois heures. Ce qui veut dire que dans une même session, on peut rester trois heures sur la même page, les valeurs ne sont pas perdues. Au delà, il faut rouvrir une nouvelle session et restaurer de nouvelles valeurs. A chaque chargement de page (ouverture de session), on recommence le décompte.

On peut modifier le délai d'expiration de la session grâce à la fonction **session**

_**cache_expire()** qui prend comme valeur la nouvelle durée en minutes. Sans paramètre elle retourne la valeur actuelle.

10.3.4 Changer l'accès par défaut

PHP sait très bien se débrouiller tout seul mais on peut remplacer la prise en charge par défaut des sessions par ses propres fonctions, dans le but de gérer l'ouverture et la fermeture ou encore la sauvegarde des données de session dans un fichier ou une base de données.

Une autre utilisation peut être le traçage via PHP de toutes les ouvertures/fermetures des sessions.
Pour ça on utilise la fonction **session_set_save_handler()**. Son étude dépasse le cadre du cours.