

Programmation en PHP

Ivan Kurzweg

18 mai 2007

Programmation en PHP

by Ivan Kurzweg

Copyright © 2007 Ivan KURZWEG, *ik-r@wanadoo.fr*

Permission to use, copy, modify, and distribute this documentation for any purpose with or without fee is here by granted, provided that the above copyright notice and this permission notice appear in all copies.

Table des matières

1	Programmation Php - Notions de bases	3
1.1	Présentation de Php	3
1.2	Instructions	4
1.3	Variables	5
1.4	Opérateurs	6
1.4.1	Opérateurs arithmétiques	6
1.4.2	Opérateurs de comparaison	6
1.4.3	Opérateurs logiques	7
1.5	Tableaux	7
1.5.1	Tableau à une dimension	7
1.5.2	Tableau à plusieurs dimension	7
1.6	Exercices d'applications	8
1.6.1	Analyse de code	8
1.6.2	Instructions simples	8
1.6.3	Fiche stagiaire	9
2	Structures de contrôles	9
2.1	Conditionnelles	9
2.1.1	if	9
2.1.2	Le Switch	10
2.2	Répétitives	11
2.2.1	La boucle for	11
2.2.2	La boucle while	11
2.2.3	Sortie d'une boucle	12
2.2.4	La boucle foreach	13
2.2.5	Un petit mot sur les fonctions	14
2.3	Exercices d'applications	14
2.3.1	Boucles et conditionnelles	14
2.3.2	Tableaux	14
2.3.3	Interaction avec HTML	14
2.3.4	Carré magique	14
3	Passage de valeurs entre pages	15
3.1	Passages de variables dans l'url	15
3.2	Formulaires	15
3.3	Cookies	17
3.4	Exercices d'applications	17
4	Interaction avec Mysql	17
4.1	Accès au serveur MySQL	18
4.2	Exécution de code SQL	19
4.3	Traitement des résultats	20
4.3.1	Traitement des résultats d'une requête de type SELECT	20
4.3.2	Test des résultats d'une requête de type SELECT	21
4.4	Fermeture de connexion à la base MySQL	21
4.5	Exercices d'applications	22
5	Développement d'applications internet	22
5.1	Fonctions	22
5.1.1	Utilisation de variables locales dans une fonction	23
5.1.2	Passage de paramètres à une fonction	24
5.1.3	Retour d'une valeur depuis une fonction	25
5.1.4	Fonctions intégrées à PHP	26
5.2	Inclusion de fichiers	26
5.3	Variables de sessions	28
5.4	Éléments de sécurisation	29

6 Références	30
6.1 Bibliographie	30

Table des figures

1 Accès à une page Php	4
----------------------------------	---

Liste des tableaux

1 Opérateurs arithmétiques	6
2 Opérateurs de comparaison	7
3 Opérateurs logiques	7
4 Carré magique	14

Résumé

Ce document présente une introduction à la programmation PHP appliqué au développement Internet. Il ne se veut bien sûr pas exhaustif, et de nombreuses documentations peuvent être trouvées en complément sur Internet. Dans tous les cas, le site de référence sera www.php.net qui fournira l'ensemble de la documentation officielle.

Résumé

Ce document présente une introduction à la programmation PHP appliqué au développement Internet. Il ne se veut bien sûr pas exhaustif, et de nombreuses documentations peuvent être trouvées en complément sur Internet. Dans tous les cas, le site de référence sera www.php.net qui fournira l'ensemble de la documentation officielle.

1 Programmation Php - Notions de bases

1.1 Présentation de Php

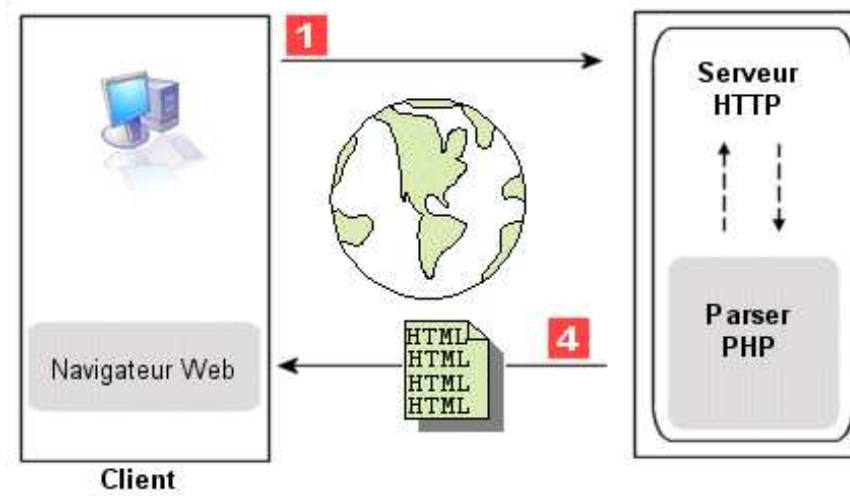
Php pour *PHP :Hypertext Preprocessor* est un langage de programmation de script, spécialement conçu pour permettre la création de pages Web dynamiques. Ce n'est pas le seul langage de programmation que l'on utilise sur le net (Ruby, Python, Perl, ..) mais c'est le plus répandu : plus de 13 millions de domaines l'utilisaient en 2004.

IL dispose en effet de plusieurs avantages :

- Rapidité
- Gratuité
- Facile à utiliser
- Multi OS
- Sources de documentations multiples
- Support des bases de données

Dans le cas de la programmation de pages Web dynamiques, le code PHP est inclus dans le code HTML. L'interpréteur PHP est appelé par le serveur Web, lors d'une demande d'une page `.php`. Le contenu du code PHP est ainsi remplacé par les *sorties* du programme.

FIG. 1 Accès à une page Php



1. Le client envoie une requête (URL) au serveur WEB `http://www.domaine.com/page1.php` par exemple.
2. Le serveur HTTP récupère le fichier sur le disque
3. Grâce à l'extension, (.PHP) du fichier, il le transmet au parser PHP pour être exécuté.
4. Le résultat (sauf erreur) est retourné au client, sous forme de page HTML

Exemple 1.1 Un premier programme

La ligne de code suivante

```
<?①echo "<p>Hello World</p>"; ?> ②
```

donnera l'interprétation suivante dans la page HTML

```
<p>Hello World</p>
```

Qui donnera dans la navigateur

```
Hello World
```

- ① Caractères spéciaux pour débiter l'inclusion de code PHP
- ② Caractères spéciaux pour terminer l'inclusion de code PHP

1.2 Instructions

Lors de l'interprétation d'une page PHP, le serveur Web *parse* les lignes de la page. Chaque fois qu'il rencontre les caractères `<?`, il demande alors à l'interpréteur Php d'exécuter le *bloc d'instructions*. Les *instructions* sont exécutées séquentiellement, du haut de la page vers le bas, jusqu'au caractère `>`. Chaque instruction est terminée par un `;`. Ainsi, les espaces et les tabulations, et même les retours à la ligne n'ont pas d'influence sur le code.

Les commentaires sont ajoutés aux codes en utilisant les caractères `//` ou `#` (ce qui indique à Php que tous les caractères qui suivent sur la ligne ne doivent pas être interprétés) ou en encadrant le texte par `/*` et `*/`.

Ainsi le premier exemple peut également s'écrire :

Exemple 1.2 Des commentaires et des espaces ..

```
<?
// Affichage texte
echo "<p>
    Hello World </p>";
#sorties code Php
?>
```

Certaines instructions peuvent contenir des *expressions*. Les instructions sont exécutées, les expressions sont évaluées.

Les instructions peuvent être regroupées dans des blocs, délimités par { }. Un bloc garanti que toutes les instructions sont exécutées de manière séquentielle de la première à la dernière.

Exemple 1.3 Les blocs d'instructions

```
if (le cyclone revient) ❶
{ ❷
    rentrer les animaux;
    faire des réserves;
} ❸
```

- ❶ Si la condition est respectée ...
- ❷ Début du bloc
- ❸ Fin du bloc

1.3 Variables

Une variable est un espace mémoire utilisé pour stocker une information. Elle est définie par un nom. On peut affecter une valeur à une variable, et utiliser cette valeur plus tard dans le programme. Un exemple d'utilisation des variables peut être de stocker les données entrées par un utilisateur.

Si les noms des variables sont au choix du programmeur, il est important de noter quelques règles de nommages :

- les noms de variables commencent par un \$
- les noms de variables sont de longueur quelconque
- les noms de variables peuvent contenir des chiffres, des lettres et le signe underscore
- les noms des variables sont sensibles à la casse.

Contrairement à beaucoup d'autres langages, la déclaration des variables n'est pas une obligation en Php. Lorsqu'elle est omise, cette déclaration est implicite au moment de la première affectation. *L'affectation* est l'instruction qui permet de stocker une valeur dans une variable, elle est définie par le signe =.

C'est donc la première affectation qui détermine le type de la variable. Le type d'une variable représente le type de données qu'elle va pouvoir contenir : numérique, caractère, tableau, ...

Exemple 1.4 Affectations et variables

```
<p>Hello World!
<? $age = 33;
    $nom = "Ivan"; ?>
<br />Hello <? echo $nom; ?>
<br />Tu as <? echo $age; ?> ans
</p>
```

Exemple 1.5 Variables et inclusion de code HTML

```
<p>Un exemple d'inclusion de code Php avec des variables
pour stocker par exemple une URL
<? $base_url = "http://www.kurzweg.info"; ?>
<br />La racine de ce site se trouve
<a href="<? echo $base_url?>"alt="racine">ici</a>
</p>
```

NOTE

Il est possible de supprimer une variable (et pas seulement sa valeur) en utilisant la fonction **unset(\$var)**.

1.4 Opérateurs

Nous avons vu comment déclarer des variables et leur affecter des valeurs. Voyons maintenant comment manipuler ces valeurs :

1.4.1 Opérateurs arithmétiques**TAB. 1** Opérateurs arithmétiques

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste de la division entière)

1.4.2 Opérateurs de comparaison

Les opérateurs de comparaison permettent d'obtenir une valeur booléenne VRAI ou FAUX selon le résultat de l'expression :

TAB. 2 Opérateurs de comparaison

Opérateur	Description
==	Egalité
>	Strictement supérieur
>=	Supérieur ou égal
<	Strictement inférieur
<=	Inférieur ou égale
!=	Non égalité
<>	Non égalité

1.4.3 Opérateurs logiques

Les opérateurs logiques permettent par exemple de combiner les résultats de comparaisons. Il renvoient la valeur booléenne VRAI ou FAUX.

TAB. 3 Opérateurs logiques

Opérateur	Vrai quand ...
OR	une des deux opérandes est VRAI
AND	les deux opérandes sont VRAI
XOR	Uniquement lorsqu'un opérande est VRAI, pas les deux

1.5 Tableaux

Les tableaux sont des structures de données permettant de stocker plusieurs valeurs dans une même variable. Très utilisés en PHP, on dispose de tableaux à une ou plusieurs dimensions.

1.5.1 Tableau à une dimension

Un tableau à une dimension permet donc de stocker plusieurs valeurs *de même type* dans une même variable. L'accès à chaque cellule du tableau se fait en utilisant un indice : une entier commençant à 0 ou une chaîne de caractère identifiant la cellule.

```
Exemple :
$prof[0] = "Assoune";
$prof[1] = "Obré";
$prof[2] = "Lausin";
$prenoms_enseignant['Assoune'] = "Maximin";
$prenoms_enseignant['Obré'] = "Philippe";
$prenoms_enseignant['Lausin'] = "Benoit";
```

Il est possible de déclarer et d'affecter des valeurs à un tableau en utilisant le constructeur `array()`.

```
Exemple :
$prof = array("Assoune", "Obré", "Lausin");
$prenoms_enseignant = array('Assoune' => "Maximin",
    'Obré' => "Philippe",
    'Lausin' => "Benoit");
```

1.5.2 Tableau à plusieurs dimension

Les tableaux à plusieurs dimensions permettent de stocker plusieurs valeurs dans chaque cellule du tableau. Il faut néanmoins respecter l'ordre et le type de chaque valeur.

Exemple :

```
$prof[0][0] = "Assoune";  
$prof[1][0] = "Obré";  
$prof[2][0] = "Lausin";  
$prof[0][1] = "Maximin";  
$prof[1][1] = "Philippe";  
$prof[2][1] = "Benoit";
```

1.6 Exercices d'applications

1.6.1 Analyse de code

Interprétez et corrigez si nécessaires les extraits de code suivants :

- ```
$string = 'c'est surement un probleme de guillemets';
echo $string
```
- ```
$age = 12;  
$result1 = "$age";  
$result2 = "$age";  
echo $result1;  
echo "<br />";  
echo $result2;
```
- ```
$nb = 10;
$ch1 = "Il y a '$nb' personnes connectées";
$ch2 = "Il y a \"$nb\" personnes connectées. '";
echo $ch1, "
\n";
echo $ch2;
```
- ```
$string1 = `Hello`;  
$string2 = `World!`;  
$stringall = $string1.$string2;  
echo $stringall;
```
- ```
$nix = "BSD";
echo "Free$nix, Open$nix, PC$nix sont tous des Unix libres ...";
```
- ```
$number = 2;  
$string = "Hello";  
$combined = $number + $string;  
$combined2 = $number.$string;  
echo $combined;  
echo <br />;  
echo $combined2;
```
- ```
$cpt=4;
$cpt+=2;
$cpt-=3;
$cpt*=2;
$cpt/=3;
echo $cpt;
```

### 1.6.2 Instructions simples

Ecrivez le code permettant d'inverser le contenu de deux variables \$x et \$y. (au départ, \$x contient 2 et \$y contient 4, et c'est l'inverse en fin de programme). Le programme devra afficher le contenu des variables avant et après le traitement.

Ecrivez une page PHP qui fournit le code HTML affichant sur la même page :

1. La surface et la circonférence d'un cercle de 2cm de rayon (PI et le rayon devront être stockés dans des variables)
2. Le résultat de l'évaluation de l'expression ((VRAI et FAUX) OU VRAI) ET FAUX
3. Le résultat de l'évaluation de l'expression ((VRAI et FAUX) OU (VRAI ET FAUX))
4. La phrase la date actuelle est :date et il est : heure
5. Le nom de la machine
6. Un lien vers les différentes informations sur les modules et les versions de PHP disponibles sur votre machine.

### 1.6.3 Fiche stagiaire

Reprenez la fiche stagiaire créée dans le cours HTML, et stockez toutes les valeurs (nom, prénom, etc ..) dans la première case d'un tableau (à plusieurs dimensions). Affichez ensuite la page en utilisant ce tableau.

## 2 Structures de contrôles

### 2.1 Conditionnelles

#### 2.1.1 if

Les structures conditionnelles permettent de n'exécuter un bloc d'instructions que si une condition est réalisée. Le synopsis de la conditionnelle est le suivant :

```
if (condition1 ...❶)
{
 Bloc d'instructions 1..❷
}
elseif (condition2 ...)❸
{
 Bloc d'instructions 2..❹
}
else❺
{
 Bloc d'instructions 3..
}
❻
```

- ❶ Une *expression* dont l'évaluation donne VRAI ou FAUX
- ❷ Si la condition 1 est vrai alors tout le bloc 2 est exécuté et le programme va en 6
- ❸ Si la condition 1 est FAUX alors la valeur de la condition 3 est testée
- ❹ Si la condition 2 est VRAI alors le bloc 4 est exécuté et le programme va en 6
- ❺ Si la condition 2 est FAUSSE alors le bloc 5 est exécuté et le programme va en 6
- ❻ Fin de la conditionnelle

Voici un exemple d'enchaînement de conditionnelles :

---

**Exemple 2.1** Emboitements de condition

```
if ($country == "Germany")
{
 $version = "German";
 $message = " Sie sehen unseren Katalog auf Deutsch";
}
elseif ($country == "France")
{
 $version = "French";
 $message = " Vous verrez notre catalogue en francais";
}
elseif ($country == "Italy")
{
 $version = "Italian";
 $message = " Vedrete il nostro catalogo in Italiano";
}
else
{
 $version = "English";
 $message = "You will see our catalog in English";
}
echo "$message
";
```

---

**2.1.2 Le Switch**

Nous avons vu dans l'exemple précédent la nécessité de renoter plusieurs fois le même test d'égalité. Dans le cas de multiples tests de la valeur d'une variable, l'écriture peut devenir vite fastidieuse. C'est là qu'intervient le **switch** :

```
switch ($variable)
{
 case value :
 bloc d'instructions
 break;
 case value :
 bloc d'instructions
 break;
 ...
 default:
 bloc d'instructions
 break;
}
```

Il faut bien noter ici que l'on ne peut tester que la valeur d'une seule variable !

Un exemple d'utilisation du switch :

---

**Exemple 2.2** Emboitements de condition

---

```
switch ($dept)
{
 case 974 :
 $dept_nom = "Réunion";
 break;
 case 972 :
 $dept_nom = "Guyane";
 default:
 $dept_nom = "France métropolitaine";
 break;
}
```

---

## 2.2 Répétitives

Les structure répétitives (itératives) permettent de répéter un certain nombre de fois un bloc d'instructions. Nous en étudierons deux, la boucle **for** et la boucle **while**

### 2.2.1 La boucle for

L'instruction **for** permet de répéter un bloc d'instructions pour un nombre de fois défini. Elle doit définir la variable qui sera modifiée à chaque itération, sa valeur de départ, la valeur de fin de boucle, et l'opération à effectuer à chaque itération :

---

**Exemple 2.3** Affichage des nombres inférieurs à 100

---

```
<?
for ($cpt=1;$cpt<100;$cpt++)
{
 echo "$cpt
";
}
?>
```

---

---

**Exemple 2.4** Calcul de la somme des nombres pairs inférieurs à 100

---

```
<?
$som=0;
$nb=100;
for ($cpt=1;$cpt<$nb+1;$cpt+=2)
{
 $som=$som+$nb;
}
echo "La somme des $nb premiers entiers est de $som";
?>
```

---

### 2.2.2 La boucle while

L'instruction **while** permet de répéter un bloc d'instructions tant qu'une condition n'est pas remplie. Elle suit le synopsis suivant :

```
while (condition❶)
{
 bloc d'instructions ... ;
}
```

❶ tant que cette condition est VRAI, on exécute le bloc et on réévalue la condition.

Bien sûr, il est à la charge du programmeur de bien s'assurer que la condition d'exécution de la boucle est modifiée et passe à un moment à FAUX .. sinon, on est dans le cas d'une boucle infinie.

---

#### Exemple 2.5 Affichage des nombres inférieurs à 100

---

```
<?
$cpt=1;
while ($cpt<100)
{
 echo "$cpt
";
 $cpt++;
}
?>
```

---

#### Exemple 2.6 Une boucle infinie !!

---

```
<?
$cpt=1;
while ($cpt<100)
{
 echo "$cpt
";
 // $cpt++; on commente la ligne, la valeur de $cpt ne changera plus,
 // la boucle ne s'arrêtera pas
}
?>
```

### 2.2.3 Sortie d'une boucle

Même si les boucles devraient être écrites pour s'arrêter d'elles mêmes, il peut être intéressant de forcer la sortie, avec l'instruction **break**. **break** permet de sortir complètement de la boucle et de reprendre le cours du programme après le bloc d'instructions.

```
$cpt = 0;
while ($cpt < 5)
{
 $cpt++;
 If ($cpt == 3)
 {
 echo "break
";
 break;
 }
 echo "Fin de boucle: cpt=$cpt
";
}
echo "Sortie de boucle<p>";
```

### 2.2.4 La boucle foreach

L'instruction **foreach** permet de faire des itérations sur les éléments d'un tableau. Elle suit le synopsis suivant :

```
foreach ($array❶ as $value❷);
{
 bloc d'instructions ... ;
}
```

- ❶ Le tableau qui va être parcouru
- ❷ La variable qui contiendra la valeur de chaque cellule à chaque itération

En reprenant l'exemple du tableau créé dans le premier chapitre, on peut ainsi parcourir le tableau formateur en récupérant la valeur de chaque cellule (exemple 1) mais également la clef de la cellule (exemple 2).

Exemple :

```
$prof = array("Assoune", "Obré", "Lausin");
$prenoms_enseignant = array('Assoune' => "Maximin",
 'Obré' => "Philippe",
 'Lausin' => "Benoit");

foreach ($prof as $nom) {
 echo "Professeur : $nom
";
}

foreach ($prenoms_enseignant as $n => $ $p) {
 echo "Professeur : $n $p
";
}
```

Une utilisation des tableaux à plusieurs dimensions est également fréquente. Dans ce cas, il peut être intéressant d'imbriquer les parcours de chaque dimensions :

```
//tableau à deux dimension : catégorie et produits
$prod['vetements']['T-shirt'] = 20.00;
$prod['vetement']['pantalon'] = 22.50;
$prod['literie']['taie'] = 25.00;
$prod['literie']['drap'] = 50.00;
$prod['fourniture']['lampe'] = 44.00;
$prod['fourniture']['table'] = 75.00;

//récupération du prix d'un T-Shirt
$prix = $prod['vetements']['T-shirt'];

//affichage du prix d'un pantalon
echo $prod['vetement']['pantalon'];

//affichage d'un texte
echo "le prix d'une lampe est \${$prod['fourniture']['lampe']}";

echo "<table border=1>";
foreach($prod as $categorie)
{
 foreach($categorie as $produit => $prix)
 {
 $ch_prix = sprintf("%01.2f", $prix);
 echo "<tr><td>$produit:</td><td>\${$ch_prix}</td></tr>";
 }
}
echo "</table>";
```

### 2.2.5 Un petit mot sur les fonctions

Dans l'exemple précédent, nous avons aperçu la fonction `sprintf`. Nous détaillerons dans le chapitre 4 ce qu'est une fonction, mais nous pouvons déjà dire qu'il s'agit d'un bloc d'instructions qui peut être appelé depuis un programme. S'il est possible au programmeur de concevoir lui-même des fonctions, il existe également nombre de fonctions prédéfinies, c'est-à-dire intégrées nativement dans le langage, ou incluses dans des modules.

Nous aurons bien sûr l'occasion de manipuler des fonctions liées à MySQL par exemple, mais le site [php.net](http://php.net) vous donne déjà une idée du nombre de fonctions disponibles. La documentation les classe par catégories.

## 2.3 Exercices d'applications

### 2.3.1 Boucles et conditionnelles

1. Ecrivez un code Php qui affiche "hello world" en titre de niveau 1, niveau 2 .. niveau 6.
2. Créer une balise de titre H1 : « Calcul sur les variables ». Affecter respectivement les valeurs 0.206, 150 et 10 aux variables TVA, prix et Nombre. Calculer le prix HT et le prix TTC pour les 10 articles et les afficher. On affichera également le type de chaque variable.
3. Affectez respectivement les valeurs 150, 50 et 10 aux variables prix\_table, prix\_armoire et Nombre. Calculez le prix HT total pour les 10 armoires. Comparez le prix de l'armoire et de la table et affichez quel est le prix le plus élevé.
4. Affectez une valeur à la variable `nbre` et affichez la somme des entiers de 1 à `nbre`. Donnez les versions du programme en utilisant l'instruction **FOR** puis avec l'instruction **WHILE**.

### 2.3.2 Tableaux

1. Initialisez un tableau de 4 cellules, et faites en la somme en utilisant deux méthodes différentes.
2. Initialisez un tableau de 4 cellules (contenant des nombres en francs) et en faire la conversion en euros. On affichera la somme totale des cellules du tableau en euros ainsi que chaque cellule du tableau.

### 2.3.3 Interaction avec HTML

1. Affichez sur une page HTML un tableau présentant le calendrier du mois en cours. Ajoutez du code PHP permettant de mettre en rouge la cellule correspondant au jour d'aujourd'hui.
2. Reprenez votre CV avec les variables PHP. Quels changements faudrait-il apporter pour afficher plusieurs CV dans la même page ?

### 2.3.4 Carré magique

Un tableau de nombres entiers est dit *magique* quand la somme des cases de chaque colonne est égale à la somme des cases de chaque ligne, et à la somme des cases de chaque diagonale.

Vous devez écrire une page PHP qui affiche le tableau suivant, et qui prouve que le carré est magique. Vérifiez en changeant une des valeurs. Toutes les valeurs du carré doivent être stockées dans une seule variable.

TAB. 4 Carré magique

|    |    |    |    |    |
|----|----|----|----|----|
| 17 | 24 | 1  | 8  | 15 |
| 23 | 5  | 7  | 14 | 16 |
| 4  | 6  | 13 | 20 | 22 |
| 10 | 12 | 19 | 21 | 3  |
| 11 | 18 | 25 | 2  | 9  |

## 3 Passage de valeurs entre pages

### Résumé

La navigation de page en page sur un site Internet est une opération naturelle pour l'utilisateur. Du point de vue du développeur, il est parfois nécessaire de conserver des informations tout au long de la *session* d'un internaute, pour par exemple se souvenir de son *login*, du nombre de page qu'il a visité, de son panier d'achat, etc .. Nous distinguerons 4 manières différentes d'implémenter cette notion dans les codes PHP :

1. l'ajout d'information dans un lien hypertexte (dans l'*url* cible)
2. le passage de données via un formulaire
3. l'utilisation de cookies
4. l'utilisation de variables de session

Nous examinerons les 3 premières solutions dans ce chapitre, et nous reviendrons dans le dernier chapitre sur la notion de variables de session.

### 3.1 Passages de variables dans l'url

Si les formulaires constituent un élément essentiel d'interaction avec l'utilisateur et les pages HTML, il est généralement nécessaire de passer des informations de page en page, sans utiliser de formulaires. Pour cela, nous utilisons la méthode GET du protocole HTTP, qui permet d'ajouter des valeurs à l'URL. Néanmoins, il faut considérer cette méthode comme un risque de sécurité.

Le principe consiste à ajouter le couple variables/valeur en fin d'URL, en séparant chaque couple par des &. L'ensemble des arguments est séparé de l'URL par un ?.

---

#### Exemple 3.1 Passage de paramètre par URL

---

```
cliquez ici pour vous enregistrer
<? $prenom="ivan" ?>
<? $nom="kurzweg" ?>
<a href="trait.php?nom=<?echo $nom;?>&prenom=<?echo $prenom;?>">
cliquez ici pour vous enregistrer
```

#### NOTE



Cette technique est limitée par la longueur maximale d'une URL, qui varie d'un navigateur à l'autre, et d'un serveur à l'autre. Elle présente également le défaut de montrer en clair les valeurs passées en paramètres, y compris par exemple des mots de passe ! A n'utiliser donc qu'avec parcimonie !

### 3.2 Formulaires

Nous avons vu dans le cours HTML l'utilisation des formulaires et des différents champs qui peuvent les composer. Nous avons également vu l'attribut HTML **action** qui permet de spécifier la page sur laquelle seront traitées les données.

PHP dispose bien sûr de caractéristiques intéressantes pour traiter les formulaires. Selon la méthode d'envoi du formulaire, **GET** ou **POST**, l'ensemble des valeurs sont stockées dans un tableau indicé sur le nom des champs. Ce tableau est `$_GET` ou `$_POST` selon le mode d'action.

Le traitement des données du formulaire est donc lié au traitement du tableau. L'exemple ci-dessous montre le traitement d'un formulaire simple :

**Exemple 3.2** Traitement d'un champ texte d'un formulaire

Exemple 1 : un formulaire simple, et le code PHP affichant le résultat :

```
<form method="POST" action="exemple.php" >
 <input name="mon_champ" type="text"/>
 <input name="valider" type="submit" value="OK"/>
</form>

<?
if (isset($_POST['mon_champ'])) {
?>
 Votre champ contenait :
 <?php echo $_POST['mon_champ']; ?>

<?php
}
?>
```

**Exemple 3.3** Traitement d'un champ radio d'un formulaire et ré-affichage du formulaire

```
<?php
$mon_champ = isset($_POST['mon_champ']) ? $_POST['mon_champ'] : '';

if ($mon_champ) {
?>
 Vous avez choisi :
 <?php echo $mon_champ; ?>

<?php
}
?>

<form method="POST">
 <input type="radio" name="mon_champ" value="Option 1"
 <?php if($mon_champ == "Option 1") { echo 'checked'; } ?>/>Option 1

 <input type="radio" name="mon_champ" value="Option 2"
 <?php if($mon_champ == "Option 2") { echo 'checked'; } ?>/>Option 2

 <input type="radio" name="mon_champ" value="Option 3"
 <?php if($mon_champ == "Option 3") { echo 'checked'; } ?>/>Option 3

 <input type="submit" value="OK"/>
</form>
```

**Exemple 3.4** Traitement d'un champ liste d'un formulaire et ré-affichage du formulaire

```

<?php
$mon_champ = isset($_POST['mon_champ']) ? $_POST['mon_champ'] : '';

if ($mon_champ) {
?>
 Votre champ contenait :
 <?php echo $mon_champ; ?>

<?php
}
?>

<form method="POST">
 <select name="mon_champ">
 <option <?php if($mon_champ == "Option 1") { echo 'selected'; } ?>>
 Option 1</option>
 <option <?php if($mon_champ == "Option 2") { echo 'selected'; } ?>>
 Option 2</option>
 <option <?php if($mon_champ == "Option 3") { echo 'selected'; } ?>>
 Option 3</option>
 </select>
 <input type="submit" value="OK"/>
</form>

```

### 3.3 Cookies

Les cookies sont des petites quantités d'informations contenant des paires *variables=valeur*, stockées sur le navigateur de l'utilisateur. Ce stockage déporté sur le client pose bien évidemment des problèmes : l'utilisateur peut refuser les cookies, ou encore les supprimer régulièrement, empêchant ainsi votre application de fonctionner correctement.

La fonction **setcookie("variable", "valeur")** permet de créer un cookie et d'y stocker la paire de valeur. L'utilisation des cookies étant problématique, nous ne nous attardons pas plus sur leur fonctionnement.

### 3.4 Exercices d'applications

Ecrire un formulaire qui permettent de rentrer les données d'une fiche stagiaire. Le formulaire envoie les valeurs stockées vers votre page "cv", modifier pour être remplie dynamiquement par les valeurs fournies.

## 4 Interaction avec Mysql

Nous avons vu dans le cours sur les bases de données la manière de stocker des valeurs et d'y accéder sur une base MySQL. Nous allons voir dans ce chapitre la façon d'interagir avec la base de données depuis des pages écrites en PHP. En particulier, nous détaillerons les étapes suivantes :

1. *Connexion au serveur MySQL (authentification)* : nous récupérons une ressource de connexion sur laquelle nous pourrions travailler.
2. *Sélection d'une base de données* : le serveur MySQL pouvant héberger plusieurs base de données, nous sélectionnons la cible de nos traitements
3. *Passage de la requête* : nous récupérons ici une ressource, qui contient (mais pas de manière immédiatement accessible) la réponse à notre requête.
4. *Vérification de la validité de la ressource récupérée* : éventuellement des tests sur le nombre de lignes renvoyées par la requêtes, ou sur la bonne exécution de cette requête.

5. *Extraction des données à partir de la ressource* : pour récupérer les informations dans un format exploitable par PHP.
6. *Fermeture de la connexion* : indiquer au système de libérer les ressources monopolisées.

Les différentes fonctions que nous allons voir sont bien sûr détaillées dans la documentation de php, sur <http://www.php.net>.

## 4.1 Accès au serveur MySQL

L'accès à un serveur MySQL depuis une page PHP se fait via la fonction `mysql_connect (serveur, login, password)`. Les informations à passer sont donc :

- `serveur` : l'adresse IP ou le nom DNS du serveur hébergeant le SGBDR MySQL
- `login` : un compte utilisateur de MySQL, ayant les droits nécessaires sur la base que vous souhaitez exploiter
- `password` : le mot de passe du compte

Ces valeurs varient selon les cas : en local, elles dépendront de votre configuration, chez un hébergeur internet elles sont généralement fournies au moment de l'abonnement.

La fonction `mysql_connect` renvoie un identifiant de ressource, qu'il est nécessaire de stocker dans une variable. C'est cette valeur qui permettra d'identifier le lien au serveur de la base de données pour les futures requêtes. En effet, rien n'interdit de se connecter simultanément à plusieurs serveurs !

---

### Exemple 4.1 Connexion à un serveur MySQL

---

Exemple 1:

```
$connexion = mysql_connect("localhost", "init11", "pass_init11");
```

Exemple 2 :

```
$host = "localhost";
$user = "init_11";
$password = "pass_init11";

if (!$connexion = mysql_connect($host, $user, $password))
{
 $message = mysql_error();
 echo "$message
";
 die();
}
```

---

Une fois la connexion avec le serveur établie, nous allons nous servir de ce lien pour sélectionner la base sur laquelle travailler. En effet, nous avons vu qu'il est possible d'héberger plusieurs bases de données sur un même serveur MySQL, mais nos futures requêtes à la base doivent s'appliquer à une seule base ! Pour cela, PHP dispose de la fonction `mysql_selectdb ("nom_bdd", connexion)`. Les paramètres à passer à la fonction sont les suivants :

- `nom_bdd` : le nom de la base de données telle qu'il apparaît sur le serveur MySQL
- `lien` : l'identifiant de la ressource que l'on a récupéré à la suite de l'exécution de `mysql_connect`

---

**Exemple 4.2** Sélection de la base de données

---

Exemple 1 :

```
$db= mysql_select_db("init_ll", $connexion) or die(mysql_error());
```

Exemple 2 :

```
$database = "init_ll";
$db = mysql_select_db($database,$connexion)
 or die ("Problème de sélection de la base de données");
```

Dans cet exemple, on sélectionne la base de données `init_ll` sur le serveur auquel on se connecte via l'identifiant `link`. Dans le cas où la fonction échoue, la fonction `mysql_error` sera exécutée, fournissant les informations nécessaires au débogage.

---

Nous verrons dans le dernier chapitre la manière de globaliser ces opérations en les incluant dans des fichiers séparés des codes PHP. Dans le cas contraire, en cas de changement d'hébergement du site, il faut reprendre toutes les pages PHP contenant des connexions à la base de données, et les modifier pour faire apparaître les nouvelles informations ...

## 4.2 Exécution de code SQL

Une fois la connexion au serveur de bases de données établie, et la base de données sélectionnée, il est alors possible de manipuler l'ensemble des données MySQL en écrivant les requêtes SQL, et en les faisant exécuter sur le serveur.

### NOTE



L'ensemble des requêtes SQL disponible sous MySQL est à notre disposition, qu'il s'agisse de sélectionner des données, d'en ajouter, d'en supprimer ou encore d'en modifier. Ces requêtes ont été vues dans le cours MySQL, et nous n'allons pas les reprendre toutes. Rappelez vous cependant que si vous avez des soucis avec la rédaction des requêtes, il est possible de laisser à des outils tels que PhpMyAdmin le soin de les concevoir pour vous..

La première étape consiste à concevoir la requête, qui n'est pas forcément une requête statique : il peut être nécessaire de la construire en utilisant des données provenant du code PHP, par exemple issues d'un formulaire. Il est ainsi préférable de construire la requête en la stockant dans une chaîne de caractères avant de l'exécuter. Une fois la requête conçue, l'exécution se fait en appelant la fonction `mysql_query($query, $connexion)`. `$query` contiendra la chaîne de caractère de la requête SQL, et `$connexion` l'identifiant obtenu à l'exécution de la fonction `mysql_selectdb()`.

---

**Exemple 4.3** Exécution d'une requête SQL

---

Exemple 1 :

```
$query = "SELECT * FROM formateurs";
$result = mysql_query($query,$connexion)
 or die ("Erreur d'exécution de la requête SQL !!");
```

Exemple 2 :

```
$query = "SELECT * FROM formateurs WHERE FORM_NOM LIKE 'KURZWEG'";
$result = mysql_query($query,$connexion)
 or die ("Erreur d'exécution de la requête SQL !!");
```

Exemple 3 :

```
$query = sprintf("SELECT * FROM formateurs WHERE FORM_NOM LIKE '%s'",$_POST["nom ←
"]);
$result = mysql_query($query,$connexion)
 or die ("Erreur d'exécution de la requête SQL !!");
```

Vous remarquerez l'emploi des apostrophes à l'intérieur de la requête SQL, alors que ce sont des guillemets qui encadrent la chaîne PHP. Dans le troisième exemple, la requête est construite à partir d'informations récupérées depuis un formulaire, en utilisant la fonction `sprintf()` qui permet de *formater* une chaîne de caractères.

---

La variable `$result` utilisée dans les exemples suivants possède un comportement différent selon que :

- la requête SQL renvoie un résultat (une requête de sélection) : `$result` identifie le tableau qui contiendra toutes les informations issues de la requête
- la requête SQL ne renvoie pas de résultat (insertion, modification, suppression d'enregistrement) : `$result` est alors un booléen qui contiendra `FALSE` ou `TRUE` selon que la requête se soit bien passée ou non.

## 4.3 Traitement des résultats

### 4.3.1 Traitement des résultats d'une requête de type SELECT

Les fonctions MySQL permettent d'extraire les résultats d'une requête sous différentes formes, mais la plus commune est le tableau. Ce tableau extrait un enregistrement et un seul, chaque cellule du tableau étant identifié par le nom du champ de la table. Parcourir tous les résultats de la requête nécessite ainsi de recharger le tableau pour chacune des lignes. Une méthode commune est l'emploi d'une boucle `while`. La fonction `mysql_fetch_array` stocke une ligne résultat de la requête dans un tableau, et pointe sur la ligne suivante. Quand tous les résultats sont traités, la fonction renvoie `FALSE`.

**Exemple 4.4** Traitement des résultats d'une requête

```

$query = "SELECT * FROM stagiaires";
$result = mysql_query($query,$connexion)
 or die ("Erreur d'exécution de la requête SQL !!");

echo "";
while ($ligne = mysql_fetch_array($result)) {
 echo "" . $ligne['STAG_NUM'] . " : "
 . $ligne['STAG_NOM'] . " - "
 . $ligne['STAG_PRENOM'] . "" ;
 echo "
";
}
echo ""

```

Cet exemple va sélectionner tous les champs de toutes les lignes dans la table `stagiaires`. Ensuite, la fonction `mysql_fetch_array` va stocker ligne par ligne les enregistrements. La boucle PHP va afficher une liste à puces, chaque item contenant le numéro, le nom et le prénom du stagiaire.

**4.3.2 Test des résultats d'une requête de type SELECT**

Dans certains cas, il peut être nécessaire de connaître des informations sur les résultats de la requête. La plus courante est le nombre de lignes renvoyées par la requête, ou encore le nombre de champs, respectivement connus via les fonctions `mysql_num_row($result)` et `mysql_num_fields($result)`.

**Exemple 4.5** Test des résultats d'une requête

```

$prenom = "ivan";
$query = sprintf("SELECT * FROM stagiaires WHERE STAG_PRENOM LIKE '%s'", $prenom);
$result = mysql_query($query,$connexion)
 or die ("Erreur d'exécution de la requête SQL !!");

if (mysql_num_rows($result) == 0)
{
 echo "<p>Pas de stagiaires avec $prenom comme prenom !</p>";
}
else
{
 echo "";
 while ($ligne = mysql_fetch_array($result)) {
 echo "" . $ligne['STAG_NUM'] . " : "
 . $ligne['STAG_NOM'] . " - "
 . $ligne['STAG_PRENOM'] . "" ;
 echo "
";
 }
 echo ""
}

```

Dans cet exemple, on vérifie que la requête a renvoyé au moins une ligne avant d'afficher les résultats. Si elle n'a rien retourné, on affiche un message d'avertissement.

**4.4 Fermeture de connexion à la base MySQL**

Il existe des configurations particulières pour lesquelles les connexions à la base de données initiées dans une page PHP peuvent être laissées ouvertes, mais dans la grande majorité des cas, il est

largement préférable de fermer la connexion au serveur de base de données, juste après les traitements. Ainsi, même si de nombreux utilisateurs sont connectés sur le site simultanément, le risque d'arriver au maximum des utilisateurs accédant à MySQL s'en trouve réduit. Il faut donc penser à tout le temps fermer la connexion à MySQL en fin de page PHP en utilisant la fonction `mysql_close($connexion)`

---

#### Exemple 4.6 Exemple de traitement complet

---

```
$host = "localhost";
$user = "init_ll";
$password = "pass_initll";

if (!$connexion = mysql_connect($host,$user,$password))
{
 $message = mysql_error();
 echo "$message
";
 die();
}

$prenom = "ivan";
$query = sprintf("SELECT * FROM stagiaires WHERE STAG_PRENOM LIKE '%s'", $prenom);
$result = mysql_query($query,$connexion)
 or die ("Erreur d'exécution de la requête SQL !!");

if (mysql_num_rows($result) == 0)
{
 echo "<p>Pas de stagiaires avec $prenom comme prenom !</p>";
}
else
{
 echo "";
 while ($ligne = mysql_fetch_array($result)) {
 echo "" . $ligne['STAG_NUM']
 . " : " . $ligne['STAG_NOM']
 . " - " . $ligne['STAG_PRENOM'] . "";

 echo "
";
 }
 echo "";
}

mysql_close($connexion);
```

---

## 4.5 Exercices d'applications

Travailler à partir de la base `stagiaires`. Faire une page où s'affiche la liste des stagiaires (Nom, prenom, date de naissance) sous forme de liste ordonnée, la liste des formateurs, la liste des modules.

Travailler à partir de la base `ordinateurs`. Modifier la pages PHP "index.html", de manière à indiquer dans la première le nombre total d'ordinateurs dans la base et le nombre d'ordinateurs dans chaque catégorie. Modifier ensuite la page `portable` de manière à en obtenir la liste depuis la base de données.

## 5 Développement d'applications internet

### 5.1 Fonctions

Une application (Internet ou autre), doit souvent exécuter le même code à plusieurs endroits du programme, ou dans différents programmes. Un exemple pourrait être par exemple l'affichage d'un logo dans toutes vos pages web :

**Exemple 5.1** Affichage d'un logo en PHP

```

echo '<p><hr width="50" align="left" />', "\n";
echo '
', "\n ←
";
echo '<hr width="50" align="left" /></p>', "\n";

```

Un fonction `affiche_logo` pourrait reprendre toutes les instructions précédentes. L'affichage du logo se ferait alors par simple appel à cette fonction :

**Exemple 5.2** Affichage d'un logo en PHP avec appel d'une fonction

```
display_logo();
```

Cette notion de fonction a plusieurs avantage :

- moins de code à écrire
- une meilleure lisibilité
- moins d'erreurs au développement
- une meilleure évolutivité

La déclaration d'une fonction se fait de la manière suivante :

```

function nom_fonction()
{
 bloc d'instructions;
 return;
}

```

Ce qui donnerait pour l'exemple précédent la déclaration suivante :

**Exemple 5.3** Fonction d'affichage d'un logo en PHP

```

function affiche_logo()
{
 echo '<p><hr width="50" align="left" />', "\n";
 echo '
 ←
 />', "\n";
 echo '<hr width="50" align="left" /></p>', "\n";
 return;
}

```

Le mot clef **return** interrompt l'exécution de la fonction, et provoque le retour au programme appelant. Toute fonction devrait comporter au moins une instruction **return**.

Il est également possible de manipuler des variables dans les fonctions, et nous distinguerons alors les variables locales (celles qui ne sont visibles qu'à l'intérieur du corps de la fonction), et les paramètres de la fonction, variables qui sont affectées lors de l'appel à la fonction.

**5.1.1 Utilisation de variables locales dans une fonction**

La déclaration d'une variable dans le corps d'une fonction a la particularité de ne la rendre visible que dans la fonction. On parle alors de *portée* de variable, limitée à une *portée locale* dans ce cas. L'exemple suivant ne produit pas de sortie :

---

**Exemple 5.4** Fonction d'affichage d'un logo en PHP

---

```
//déclaration de la fonction
function format_nom()
{
 $prenom = "Ivan"; //déclaration de la variable en local
 $nom = "Kurzweg";
 $nom_complet = $prenom.", ".$nom; //pas d'erreurs à l'exécution
}

//appel de la fonction dans la page PHP
format_name();

//mais aucune sortie à l'exécution de echo, la variable name n'est pas connue
echo "$name";
```

---

Il existe la possibilité d'augmenter la portée d'une variable à toute la page PHP en utilisant le mot clef **global**. Mais cette technique est souvent source d'erreurs et de confusion, nous ne nous y attarderons pas.

### 5.1.2 Passage de paramètres à une fonction

Il est possible de passer des valeurs entre une fonction et le programme appelant, et ce dans les deux sens : du programme appelant vers la fonction en utilisant les paramètres de la fonction, et de la fonction vers le programme appelant en utilisant l'instruction **return**.

Pour passer des paramètres (des valeurs) à une fonction, il faut au préalable avoir déclaré la fonction de manière à ce qu'elle attende ces valeurs :

```
function nom_fonction($var1, $var2, ...)
{
 bloc d'instructions;
 return;
}
```

L'appel de la fonction se fait alors en renseignant les valeurs :

```
nom_fonction(valeur, valeur, ...);
```

**Exemple 5.5** Fonction de calcul d'un montant TTC

```
//déclaration de la fonction
function calcul_ttc($montantHT,$tva)
{
 switch ($tva)
 {
 case "HT" :
 $taxes = 0;
 break;
 case "DOM" :
 $taxes = 1.086;
 break;
 default:
 $taxes = 1.186;
 break;
 }
 $montantTTC = $montantHT * $taxes;
 echo "$montantTTC
";
}

//initialisation des variables
$prix = 2000.00;
$tauxTaxes = "DOM";

//appel de la fonction
calcul_ttc($prix,$tauxTaxes);
```

**5.1.3 Retour d'une valeur depuis une fonction**

S'il est possible de passer des valeurs en paramètres à une fonction, il est également possible d'obtenir une valeur de la fonction. C'est le rôle de l'instruction **return** de renvoyer la valeur selon le modèle suivant :

```
function nom_fonction($var1, $var2, ...)
{
 bloc d'instructions;
 return valeur;
}
```

Il est alors possible d'affecter la valeur de retour de la fonction à une variable du programme appelant :

```
$variable = nom_fonction(valeur,valeur,...);
```

**Exemple 5.6** Fonction de calcul d'un montant TTC

```
//déclaration de la fonction
function calcul_ttc($montantHT,$tva)
{
 switch ($tva)
 {
 case "HT" :
 $taxes = 0;
 break;
 case "DOM" :
 $taxes = 1.086;
 break;
 default:
 $taxes = 1.186;
 break;
 }
 $montantTTC = $montantHT * $taxes;
 //on ne fait plus l'écho, mais on va renvoyer la valeur obtenue
 //echo "$=montantTTC
";
 return $montantTTC;
}

//initialisation des variables
$prix = 2000.00;
$tauxTaxes = "DOM";

//appel de la fonction, et affectation du résultat à une variable
$montant = calcul_ttc($prix,$tauxTaxes);
```

**5.1.4 Fonctions intégrées à PHP**

Il existe un certains nombres de fonctions intégrées au langage PHP, ou accessibles après l'ajout d'un module complémentaire. Nous avons par exemple vu des fonctions propres à MySQL, disponibles puisque nous avons installé le *paquet* PHP5-mysql.

Ainsi, nous pouvons disposer de fonctions prédéfinies, qui permettent par exemple de manipuler des images, de générer des fichiers au format PDF, etc.. Le site [www.php.net](http://www.php.net) vous en présente de nombreuses.

**5.2 Inclusion de fichiers**

Php permet de stocker des instructions dans des fichiers externes - un fichier séparé du programme - et l'ajouter dans toutes les pages nécessaires. L'instruction **include** permet de réaliser cette "importation". Cette méthode se révèle pratique pour stocker des instructions, des fonctions ou des variables utilisées fréquemment. Le format d'une inclusion est le suivant :

```
include("chemin/nom_fichier");
```

Le fichier peut avoir n'importe quel nom, mais vous pouvez par exemple le suffixer de `.inc`. Si le fichier contient du code PHP, ne pas oublier les caractères signalant le début et la fin de ce code ..

De manière à rendre l'application Internet que vous développez facilement maintenable et lisible, voici la liste d'un certain nombre de données à inclure dans des fichiers séparés, appelés depuis les pages PHP :

- *Du code HTML* : il peut être intéressant de mettre beaucoup de code HTML dans des fichiers d'inclusion. Par exemple, la partie déclarative (doctype et entête de la page), les parties du code HTML

fixes sur le site (bandeau de navigation, pied de page, etc ..), voir même des fomulaires .. De manière générale, il faut essayer de mettre toutes les parties de codes HTML répétitives dans es fichiers à inclure ...

- *Les informations nécessaires pour accéder à la base de données.* En effet, lors de la migration du site depuis l'environnement de développement vers l'environnement de production, il n'y aura que ce fichier à modifier :

```
<?
$host = "localhost";
$user = "init_ll";
$password = "pass_initll";

if (!$connexion = mysql_connect($host,$user,$password))
{
 $message = mysql_error();
 echo "$message
";
 die();
}
?>
```

- *Les fonctions PHP* : nous avons vu dans les paragraphes précédents la manière de concevoir des fonctions. Il peut être intéressant de les mutualiser dans des fichiers d'inclusion.

Au final, une page PHP qui fournira une page HTML complète pourrait s'écrire :

---

**Exemple 5.7** Inclusion de fichiers HTML et PHP

---

```
<?
//-----
// inclusions des entêtes et des codes PHP
//-----

//inclusion des entêtes HTML
include("entete.inc");

//inclusion des fonctions PHP
include("fonctions.inc");

//inclusion du bandeau de navigation
include("bandeau.inc");

//inclusion de la connexion à la base
include("connect.inc");

?>

<?
//-----
//traitements PHP et affichage du corps de la page
//des requêtes SQL, du code HTML , etc
//-----

...
...
...
?>

<?
//-----
// inclusions des codes HTML de base de page
//-----

//inclusion des pieds de page HTML
include("pied_page.inc");

?>
```

---

### 5.3 Variables de sessions

On parle de *session* comme du temps passé par un utilisateur sur l'application développée. Durant cette session, il peut être nécessaire de stocker des informations de bout en bout, comme le numéro de l'utilisateur par exemple. Php propose ainsi les *variables de session*, attribuée à chaque utilisateur pour sa session.

Pour permettre de distinguer les utilisateurs entre eux, PHP assigne un identifiant de session, unique. Ce numéro est stocké sur le serveur, dans un fichier, et chez le client soit via des cookies, soit de page en page en utilisant la méthode POST si les cookies sont désactivés. Les variables de session sont obtenues en utilisant le tableau `$_SESSION`.

Les sessions doivent être *démarrées* depuis les pages PHP, en utilisant la fonction `session_start`. L'exemple suivant propose deux pages utilisant des variables de session :

**Exemple 5.8** Démarrer une session

```

<?
 session_start();
?>
<html>
<head><title>Premiere page</title></head>
<body>
<?
 $_SESSION['var_session'] = "testing";
?>
 <p>Un test ...
 <form action="sessionTest2.php" method="POST">
 <input type="hidden" name="var_form" value="testing" />
 <input type="submit" value="Page suivante" />
 </form>
 </p>
</body></html>

```

**Exemple 5.9** Récupérer la variable de session

```

<?
 session_start();
?>
<html>
<head><title>Seconde page</title></head>
<body>

<?php
 echo "variable_session = {"$_SESSION["var_session"]}
\n";
 echo "var_form = {"$_POST["var_form"]}
\n";
?>
</body></html>

```

**5.4 Éléments de sécurisation**

La sécurité a été longtemps quelque peu ignorée par les développeurs, mais un "défaçage" n'arrive pas qu'aux autres ! Les risques principaux d'une mauvaise sécurisation sont :

- *Vol d'information* : revente des adresses mails stockées, réutilisation des numéros de cartes bleues, etc ...
- *Défaçage* : effacement complet du site et de la base de données
- *Fishing* : utilisation de votre site à des fins frauduleuses
- ...

S'il n'est pas possible de donner la liste exhaustive des risques, des attaques et des moyens de défense, voici quelques items minimums sur lesquels s'attarder :

- *Ne pas laisser des répertoires en accès direct* : quand une url pointe vers un répertoire plutôt qu'un fichier - <http://www.kurweg.info/InitLL2007/>, le serveur renvoie automatiquement la page `index.html`, `index.htm` ou `index.php`. Si cette page n'est pas présente dans le répertoire, le serveur peut alors lister tous les fichiers du répertoire, sauf si l'option **Indexes** est en place. A ce moment là, le serveur refuse le listage du répertoire.
- *Sécuriser les mots de passe* : nous avons vu qu'il était envisageable de stocker les informations de connexion à la base de données dans un fichier include. Bien évidemment, il faudra protéger ce fichier (utiliser les fichiers `.htpasswd`).

- *Sécuriser les formulaires* : même si PHP propose maintenant d'office des mécanisme de protection contre les injections de codes, il peut être judicieux de bien vérifier l'emploi des données entrées dans les formulaires.

## 6 Références

### 6.1 Bibliographie

- [1] Welling Luke et Thomson Laura, *PHP and MySQL - Web Development*, SAMS Edition .
- [2] Ratschiller Tobias et Gerken Till, *Web Application Development with Php 4.0*, New Riders .
- [3] Valade Janet, *PHP and MySQL for Dummies*, Wiley Publishing, Inc .
- [4] Lane David, *Web database applications*, O'Reilly .
- [5] Rasmus Lerdorf, *Programming PHP*, O'Reilly .