

# SOMMAIRE

<b>1.GÉNÉRALITÉS</b>	<b>5</b>
1.ORIGINES DE PHP	5
2.BIBLIOTHÈQUES INTÉGRÉES	5
3.SGBD SUPPORTÉS PAR PHP	5
4.L'INTERPRÉTATION DU CODE PAR LE SERVEUR	6
5.IMPLANTATION AU SEIN DU CODE HTML	6
6.UN EXEMPLE DE SCRIPT SIMPLE	6
7.L'INTERPRÉTATION DU CODE	7
8.LES COMMENTAIRES	7
9.TYPOLOGIE	7
<b>2.LES VARIABLES</b>	<b>8</b>
10.NOMMAGE DES VARIABLES	8
11.LES VARIABLES SCALAIRES	8
12.LA PORTÉE (VISIBILITÉ) DES VARIABLES	9
13.LES VARIABLES ISSUES DE FORMULAIRES	10
14.LES CONSTANTES DÉFINIES	10
<b>3.LES OPÉRATEURS</b>	<b>11</b>
15.LES OPÉRATEURS DE CALCUL	11
16.LES OPÉRATEURS D'ASSIGNATION	11
17.LES OPÉRATEURS D'INCRÉMENTATION	12
18.LES OPÉRATEURS DE COMPARAISON	12
19.LES OPÉRATEURS LOGIQUES (BOOLÉENS)	12
20.LES OPÉRATEURS BITWISE	13
21.AUTRES OPÉRATEURS	13
22.LES PRIORITÉS DES OPÉRATEURS	14
<b>4.LES STRUCTURES DE CONTRÔLES</b>	<b>15</b>
23.LA NOTION DE BLOC	15
24.LA STRUCTURE CONDITIONNELLE	15
24.1.L'instruction if	15
24.2.L'instruction if ... else	16
24.3.L'instruction if ... elseif ... else	16
24.4.L'opérateur ternaire (une façon plus courte de faire un test)	16
24.5.L'instruction switch	17
25.LES BOUCLES	17
25.1.L'instruction while	17
25.2.L'instruction do..while	18
25.3.L'instruction for	18
26.SAUT INCONDITIONNEL CONTINUE	19
27.ARRÊT INCONDITIONNEL BREAK	20
28.ARRÊT D'EXÉCUTION DU SCRIPT PAR EXIT	20
<b>5.LES TABLEAUX</b>	<b>21</b>
29.LES VARIABLES TABLEAUX	21
30.LES VARIABLES DES TABLEAUX ASSOCIATIFS	21
<b>6.LES FONCTIONS</b>	<b>22</b>
31.LA NOTION DE FONCTION	22
32.LA DÉCLARATION D'UNE FONCTION	22
33.APPEL DE FONCTION	23
34.LES ARGUMENTS D'UNE FONCTION	23
35.TRAVAILLER SUR DES VARIABLES DANS LES FONCTIONS	23
36.RENVOI D'UNE VALEUR PAR UNE FONCTION	23
37.PASSAGE D' ARGUMENT PAR RÉFÉRENCE	24
38.VALEUR PAR DÉFAUT DES ARGUMENTS	24
39.LA FONCTION INCLUDE	25
40.LA FONCTION EMPTY	25
41.LA FONCTION ISSET	25
42.LA FONCTION UNSET	25

<b>7.LA GESTION DES TABLEAUX</b>	<b>26</b>
43.ARRAY.....	26
44.IS_ARRAY.....	26
45.EACH.....	26
46.LIST.....	27
47.COUNT.....	28
48.SIZEOF.....	28
49.CURRENT.....	28
50.POS.....	28
51.NEXT.....	28
52.PREV.....	28
53.RESET.....	28
54.END.....	29
55.KEY.....	29
56.SORT.....	29
57.ASORT.....	30
58.ARSORT.....	30
59.RSORT.....	31
60.KSORT.....	31
61.USORT, UKSORT ET UASORT.....	32
62.EXTRACT.....	33
63.RANGE.....	33
64.SHUFFLE.....	34
65.COMPACT.....	34
66.AUTRES FONCTIONS SUR LES TABLEAUX SOUS PHP4.....	34
<b>8.LES CHÂÎNES DE CARACTÈRES</b>	<b>35</b>
67.AFFICHAGE D'UNE CHÂÎNE DE CARACTÈRES.....	35
68.INITIALISATION ET CONCATÉNATION.....	35
69.TRAITEMENT DES CHÂÎNES DE CARACTÈRES.....	36
69.1.addslashes.....	39
69.2.chr.....	39
69.3.crypt.....	39
69.4.ereg.....	40
69.5.ereg_replace.....	40
69.6.explode.....	40
69.7.implode.....	41
69.8.nl2br.....	41
69.9.ord.....	41
69.10.parse_str.....	41
69.11.printf.....	42
69.12.rawurldecode.....	44
69.13.rawurlencode.....	44
69.14.strchr.....	44
69.15.strcspn.....	44
69.16.strip_tags.....	45
69.17.strpos.....	45
69.18.strrchr.....	45
69.19.strrpos.....	45
69.20.strspn.....	46
69.21.strstr.....	46
69.22.strtok.....	46
69.23.strtr.....	47
69.24.substr.....	47
69.25.substr_replace.....	47
<b>9.LA GESTION DES DATES</b>	<b>48</b>
70.CALCULS DE DURÉES.....	48
71.OPÉRATIONS SUR LES DATES.....	48
71.1.time.....	48
71.2.mktime.....	48
71.3.gmmktime.....	48
71.4.checkdate.....	48
72.CONVERSIONS DE DATES.....	49

72.1.date.....	49
72.2.getdate.....	50
72.3.strftime.....	50
73.DÉTERMINATION DE LA DATE DE DERNIÈRE MODIFICATION D'UN SCRIPT.....	50
74.CONVERSION DES DATES AUX FORMATS MySQL.....	51
75.EXERCICE PRATIQUE.....	52
<b>10.LES OBJETS-----</b>	<b>55</b>
76.CRÉATION D'UN OBJET.....	55
77.LES PROPRIÉTÉS D'UN OBJET.....	56
78.LES MÉTHODES D'UN OBJET.....	57
79.L'HÉRITAGE.....	59
80.EXERCICE PRATIQUE.....	61
<b>11.LES FORMULAIRES-----</b>	<b>62</b>
81.LES VARIABLES PHP.....	62
82.LES ÉLÉMENTS SELECT MULTIPLES.....	64
83.LES CHAMPS D'UN FORMULAIRE DANS UN TABLEAU ASSOCIATIF.....	65
84.ECHANGES CLIENT/SERVEUR.....	67
85.COMBINAISON DE CODES HTML ET PHP.....	67
<b>12.LES FICHIERS-----</b>	<b>69</b>
86.STRUCTURE D'UN FICHIER POUR PHP.....	69
87.LIRE ET ÉCRIRE DANS UN FICHIER .....	69
88.OUVERTURE D'UN FICHIER .....	69
89.FERMETURE D'UN FICHIER .....	70
90.ACCÈS AUX FICHIERS.....	70
91.ACCÈS AUX RÉPERTOIRES.....	71
92.LIRE UN FICHIER DÉFINI PAR UNE URL.....	73
93.EXEMPLES DE TRAITEMENTS SUR LES FICHIERS.....	74
94.LA FONCTION POPEN.....	75
95.PROBLÈMES POSÉS PAR L'USAGE DE FICHIERS PLATS.....	75
<b>13.LES BASES DE DONNÉES-----</b>	<b>76</b>
96.ARCHITECTURE D'UNE BASE DE DONNÉES WEB.....	76
97.STRUCTURE DE L'INSTALLATION DE MySQL.....	77
98.LES PRINCIPALES INSTRUCTIONS SQL.....	77
98.1.SELECT.....	77
98.2.INSERT.....	79
98.3.UPDATE.....	79
98.4.DELETE.....	79
98.5.ALTER TABLE.....	79
98.6.DROP TABLE.....	80
98.7.DROP DATABASE.....	80
99.COMPATIBILITÉ AVEC LES STANDARDS.....	81
99.1.Extensions MySQL à la norme ANSI SQL 92.....	81
99.2.Fonctionnalités manquantes.....	83
100.PROGRAMMATION MySQL.....	87
100.1.Syntaxe des chaînes et nombres.....	87
100.2.Types des colonnes et espaces mémoire requis.....	87
101.CONNEXION ET ACCÈS À LA BASE DE DONNÉES MySQL.....	89
102.CRÉER ET SÉLECTIONNER UNE BASE DE DONNÉES MySQL.....	89
103.FONCTIONS D'ACCÈS À MySQL.....	90
103.1.mysql_connect.....	93
103.2.mysql_close.....	93
103.3.mysql_select_db.....	94
103.4.mysql_error.....	94
103.5.mysql_query.....	95
103.6.mysql_fetch_array.....	96
104.EXERCICE PRATIQUE.....	97
105.MÉTHODES D'ACCÈS À MySQL PAR PROGRAMMATION OBJET.....	100
<b>14.FONCTIONNALITÉS INTERNET-----</b>	<b>107</b>
106.LE COURRIER ÉLECTRONIQUE.....	107

106.1.mail.....	107
107.LE « FILE UPLOAD ».....	108
108.LE TRAITEMENT DES URL.....	109
108.1.Codage des URL.....	109
108.2.urlencode.....	109
108.3.urldecode.....	110
108.4.parse_url.....	110
109.LES MÉTHODES DE GESTION DU CONTEXTE APPLICATIF.....	111
109.1.Le contexte applicatif.....	111
109.2.Les différentes méthodes de gestion.....	111
109.2.1.URL longue.....	111
109.2.2.Variables cachées.....	112
109.2.3.Cookies.....	112
109.2.4.Variables de session.....	114
109.2.5.Base de données.....	117
<b>15.IMPLÉMENTER UNE AUTHENTIFICATION AVEC PHP ET MYSQL-----</b>	<b>118</b>
110.IMPLÉMENTER UN CONTRÔLE D'ACCÈS.....	118
110.1.Enregistrement des mots de passe.....	119
110.2.Exercice pratique.....	121
110.3.Protéger plusieurs pages.....	122
110.4.Utiliser l'authentification de base dans PHP.....	122
110.5.Utiliser l'authentification de base avec les fichiers .htaccess d'Apache.....	124
110.6.Utiliser l'authentification mod_auth_mysql.....	126
111.NOTIONS DE TRANSACTIONS SÉCURISÉES SSL.....	127
111.1.Utilisation de SSL.....	128
111.2.Processus.....	129
111.3.Fonctionnement.....	129
111.4.Certificats.....	130
111.5.Installation de SSL.....	130
<b>16.AUTRES FONCTIONNALITÉS OFFERTES PAR PHP-----</b>	<b>131</b>
112.LA SÉRIALISATION.....	131
113.CHARGEMENT DYNAMIQUE D'EXTENSIONS SOUS WINDOWS.....	132
113.1.Liste des extensions chargées.....	134
<b>17.INSTALLATION D'APACHE, PHP ET MYSQL-----</b>	<b>135</b>
114.TÉLÉCHARGER LES SOURCES.....	135
115.INSTALLER APACHE ET PHP.....	135
116.PREMIER LANCEMENT.....	136
<b>18.BIBLIOGRAPHIE-----</b>	<b>137</b>

# 1. Généralités

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

- La gratuité et la disponibilité du code source (PHP3 est distribué sous licence GNU GPL).
- La simplicité d'écriture de scripts.
- la possibilité d'inclure le script PHP au sein d'une page HTML (contrairement aux scripts CGI, pour lesquels il faut écrire des lignes de code pour afficher chaque ligne en langage HTML).
- La simplicité d'interfaçage avec des bases de données (de nombreux SGBD sont supportés, mais le plus utilisé avec ce langage est MySQL, un SGBD gratuit sur les plates-formes Unix et Linux, mais payant sous Windows).
- L'intégration au sein de nombreux serveurs Web (Apache, Microsoft IIS, ...).

PHP est très efficace. Les tests de performances publiés par Zend Technologies (<http://www.zend.com>) montrent que PHP dépasse tous ses concurrents.

## 1. Origines de PHP

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

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

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

A la fin de l'année 1999, une version bêta de PHP, baptisée PHP4 est apparue...

## 2. Bibliothèques intégrées

Comme PHP a été conçu pour être utilisé sur le Web, il possède plusieurs fonctions intégrées permettant d'effectuer la plupart des tâches en rapport avec le Web. Vous pouvez ainsi générer des images GIF en temps réel, vous connecter à d'autres services réseaux, envoyer des e-mails, travailler avec les cookies, et générer des documents PDF.

Le code source de PHP est disponible gratuitement. Contrairement aux produits commerciaux, dont les sources ne sont pas distribuées, vous avez tout à fait la possibilité de modifier ce langage, ou d'y ajouter de nouvelles caractéristiques. Des patches correctifs sont distribués par les concepteurs, il est donc inutile de s'inquiéter de la pérennité du support ou du développement de PHP.

## 3. SGBD supportés par Php

Php permet un interfaçage simple avec de nombreux SGBD. La version 3 du langage supporte les SGBD suivants : Adabas D ; Dbase ; MS SQL Server ; FilePro ; Informix ; InterBase ; mSQL ; Openlink ODBC ; iODBC ; MySQL ; Oracle Database ; PostgreSQL ; Solid ; Sybase ; Sybase-DB ; Velocis ; Unix dbm ; IBM DB2 ; Empress. La version 4 supporte en plus : Ingres II ; Oracle-oci7 ; Oracle-oci8 ; unixODBC ; DBMaker ; Berkeley DB3.

## 4. L'interprétation du code par le serveur

Un script PHP est un simple fichier texte contenant des instructions écrites à l'aide de caractères ASCII 7 bits (des caractères non accentués) incluses dans un code HTML à l'aide de balises spéciales et stocké sur le serveur. Ce fichier doit avoir l'extension ".php3" pour pouvoir être interprété par le serveur!

Ainsi, lorsqu'un navigateur (le client) désire accéder à une page dynamique réalisé en php3:

- le serveur reconnaît qu'il s'agit d'un fichier php3.
- il lit le fichier php3.
- Dès que le serveur rencontre une balise indiquant que les lignes suivantes sont du code php3, il "passe" en mode php3, ce qui signifie qu'il ne lit plus les instructions: il les exécute!
- Lorsque le serveur rencontre une instruction, il la transmet à l'interpréteur.
- L'interpréteur exécute l'instruction puis envoie les sorties éventuelles vers le navigateur.
- A la fin du script, le serveur transmet le résultat au client (le navigateur).

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

Le code php3 stocké sur le serveur n'est donc **jamais** visible directement par le client puisque dès qu'il en demande l'accès, le serveur l'interprète!; de cette façon aucune modification n'est à apporter sur les navigateurs...

## 5. Implantation au sein du code HTML

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

- Le fichier contenant le code doit avoir l'extension *.php3* et non *.html*
- Le code php3 contenu dans le code HTML doit être délimité par les balises *<?php* et *?>*

Un script PHP doit:

- comporter l'extension *.php3*
- être imbriqué entre les délimiteurs *<?php* et *?>*

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

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

## 6. Un exemple de script simple

Voici un exemple de script php:

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

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

## 7. L'interprétation du code

Un code PHP (celui compris entre les délimiteurs `<?php` et `?>`) est un ensemble d'instructions se terminant chacune par un point-virgule (comme en langage C). Lorsque le code est interprété, les espaces, retours chariot et tabulations ne sont pas pris en compte par le serveur. Il est tout de même conseillé d'en mettre (ce n'est pas parce qu'ils ne sont pas interprétés que l'on ne peut pas les utiliser) afin de rendre le code plus lisible (pour vous, puisque les utilisateurs ne peuvent lire le code source: il est interprété).

## 8. Les commentaires

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

Pour ce faire, il est possible, comme en langage C, d'utiliser des balises qui vont permettre de délimiter les explications afin que l'interpréteur les ignore et passe directement à la suite du fichier.

Ces délimiteurs sont `/*` et `*/`

Un commentaire sera donc noté de la façon suivante:

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

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

```
// Voici un autre commentaire!
```

Il y a toutefois quelques règles à respecter :

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

## 9. Typologie

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

Enfin, toute instruction se termine par un **point-virgule**.

## 2. Les variables

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

- scalaires
- tableaux
- tableaux associatifs

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

## 10. Nommage des variables

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

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

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

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

## 11. Les variables scalaires

Le langage PHP propose trois types de variables scalaires:

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

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

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



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

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

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

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

## 12. La portée (visibilité) des variables

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

Il existe trois niveaux de définition de variables :

- Lorsqu'une variable est déclarée à l'extérieur de toute fonction ou de tout bloc d'instructions, elle est accessible de partout dans le code (n'importe quelle fonction du programme peut faire appel à cette variable). On parle alors de **variable globale**.  
Afin d'utiliser une variable globale à l'intérieur d'une fonction, il est nécessaire de la référencer à l'aide du mot clé *global* : *global \$var;*
- Lorsque l'on déclare une variable à l'intérieur d'une fonction ou d'un bloc d'instructions (entre des accolades), sa portée se confine à l'intérieur de la fonction ou du bloc dans lequel elle est déclarée. On parle alors de **variable locale**.
- Il existe enfin les variables locales à une fonction qui persistent pendant le temps d'exécution de l'intégralité du code de la page PHP. Elles gardent donc leur valeur d'un appel à l'autre de la fonction. On parle alors de **variable statique**. Elles sont déclarées à l'aide du mot clé *static* : *static \$var;*

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

## 13. Les variables issues de formulaires

Ces variables sont issues de formulaires HTML, c'est-à-dire qu'elles correspondent aux différents champs positionnés entre les balises `<FORM>` et `</FORM>` de ce formulaire. La page PHP qui reçoit ces variables est celle qui est pointée par l'attribut `ACTION` de la balise `<FORM>`.

PHP crée automatiquement des variables du même nom que le contrôle du formulaire HTML.

Toutefois, en insérant la clause `< ? php_track_vars ?>`, il est également possible de récupérer ces variables dans le tableau `$HTTP_POST_VARS` si la méthode employée dans le formulaire est `POST`, ou dans le tableau `$HTTP_GET_VARS` si la méthode employée dans le formulaire est `GET`.

Exemple :

```
...
<FORM action = 'formul1.php3' method = POST>
<table>
  <tr>
    <td>NOM : </td>
    <td><input type = text length = 30 name = 'nom' /></td>
  </tr>
  <tr>
    <td>PRENOM : </td>
    <td><input type = text length = 30 name = 'prenom' /></td>
  </tr>
  <tr>
    <td colspan = 2 align = center>< input type = submit value = 'Valider' /></td>
  </tr>
</table>
</FORM>
```

A l'issue de la validation du formulaire, les données saisies dans les champs de type *text* nommés *nom* et *prenom* seront stockées respectivement dans les variables PHP nommées automatiquement *\$nom* et *\$prenom*. La valeur de ces variables pourra donc être récupérée dans la page PHP *formul1.php3*.

## 14. Les constantes définies

Une constante est un objet dont la valeur est inchangeable lors de l'exécution d'un programme.

Avec PHP, les constantes sont définies grâce à la fonction *define()*.

La syntaxe de la fonction *define()* est la suivante:

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

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

### 3. Les opérateurs

Les opérateurs sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations, les évaluer, ...

On distingue plusieurs types d'opérateurs:

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

### 15. Les opérateurs de calcul

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

Opérateur	Dénomination	Effet	Exemple	Résultat (avec x valant 7)
+	opérateur d'addition	Ajoute deux valeurs	$\$x+3$	10
-	opérateur de soustraction	Soustrait deux valeurs	$\$x-3$	4
*	opérateur de multiplication	Multiplie deux valeurs	$\$x*3$	21
/	plus: opérateur de division	Divise deux valeurs	$\$x/3$	2.3333333
=	opérateur d'affectation	Affecte une valeur à une variable	$\$x=3$	Met la valeur 3 dans la variable \$x

### 16. Les opérateurs d'assignation

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

Ainsi, si la valeur de x était 7 avant opération, elle sera de 9 après...

Les autres opérateurs du même type sont les suivants :

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

## 17. Les opérateurs d'incrémentation

Ce type d'opérateur permet de facilement augmenter ou diminuer d'une unité une variable. Ces opérateurs sont très utiles pour des structures telles que des boucles, qui ont besoin d'un compteur (variable qui augmente de un en un). Un opérateur de type `$x++` permet de remplacer des notations lourdes telles que `$x=$x+1` ou bien `$x+=1`

Opérateur	Dénomination	Effet	Syntaxe	Résultat (avec x valant 7)
++	Post ou pré-incrémentation	Augmente d'une unité la variable	<code>\$x++</code> ou <code>++\$x</code>	8
--	Post ou pré-décrémentation	Diminue d'une unité la variable	<code>\$x--</code> ou <code>--\$x</code>	6

## 18. Les opérateurs de comparaison

Ce type d'opérateur permet de comparer la valeur d'une variable à celle d'une autre variable ou à une simple valeur. L'opérateur ternaire (`? :`) fonctionne comme en langage C.

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

## 19. Les opérateurs logiques (booléens)

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

Opérateur	Dénomination	Effet	Syntaxe
<code>  </code> ou <b>OR</b>	OU logique	Vérifie qu'au moins une des conditions est réalisée	<code>((condition1)  condition2))</code>
<code>&amp;&amp;</code> ou <b>AND</b>	ET logique	Vérifie que toutes les conditions sont réalisées	<code>((condition1)&amp;&amp;condition2))</code>
<b>XOR</b>	OU exclusif	Opposé du OU logique (une des deux conditions est réalisée mais pas les deux)	<code>((condition1)XORcondition2))</code>
<b>!</b>	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	<code>(!condition)</code>

## 20. Les opérateurs bitwise

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

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

Opérateur	Dénomination	Effet	Syntaxe	Résultat
&	AND bitwise	Retourne 1 si les deux bits de même poids sont à 1	9 & 12 (1001 & 1100)	8 (1000)
	OR bitwise	Retourne 1 si l'un ou l'autre des deux bits de même poids est à 1 (ou les deux)	9   12 (1001   1100)	13 (1101)
^	XOR bitwise	Retourne 1 si l'un des deux bits de même poids est à 1 (mais pas les deux)	9 ^ 12 (1001 ^ 1100)	5 (0101)
~	Complément (NON)	Retourne 1 si le bit est à 0 (et inversement)	~9 (1001)	6 (0110)

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

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

## 21. Autres opérateurs

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

Opérateur	Dénomination	Effet	Syntaxe	Résultat
.	Concaténation	Joint deux chaînes bout à bout	"Bonjour"." Au revoir"	"Bonjour Au revoir"
\$	Référencement de variable	Permet de définir une variable Permet de créer une <b>variable dynamique</b> (ex : \$var = "bon" ;)	\$MaVariable = 2; \$\$var = "jour" ;	La variable contenant "jour" est \$\$var ou \$bon
->	Propriété d'un objet	Permet d'accéder aux données membres d'une classe	\$MonObjet->Propriete	

## 22. Les priorités des opérateurs

Lorsque l'on associe plusieurs opérateurs dans une même expression, il faut que le navigateur sache dans quel ordre les traiter, voici donc dans **l'ordre croissant** les priorités de tous les opérateurs:

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. Les structures de contrôles

### 23. La notion de bloc

Une expression suivie d'un point-virgule est appelée instruction. Par exemple `$a++;` est une instruction.

Lorsque l'on veut regrouper plusieurs instructions, on peut créer ce que l'on appelle un **bloc**, c'est-à-dire un ensemble d'instructions (suivies respectivement par des points-virgules) et comprises entre les accolades { et }.

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

### 24. La structure conditionnelle

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

#### 24.1. L'instruction if

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

La syntaxe de cette expression est la suivante :

```
if (condition)
{
    liste d'instructions           // ces instructions seront exécutées si la condition est réalisée
}
```

#### Remarques:

- la condition doit être entre parenthèses,
- il est possible de définir plusieurs conditions à remplir avec les opérateurs ET et OU (&& et ||)  
par exemple:  
*if ( (condition1) && (condition2) )*      teste si les deux conditions sont vraies  
*if ( (condition1) || (condition2) )*      teste si au moins une des deux conditions est vraie
- s'il n'y a qu'une instruction à exécuter, les accolades ne sont pas nécessaires.

## 24.2. L'instruction if ... else

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

L'expression *if ... else* permet d'exécuter une autre série d'instructions en cas de non-réalisation de la condition.

La syntaxe de cette expression est la suivante :

```
if (condition)
{
    liste d'instructions      // ces instructions seront exécutées si la condition est réalisée
}
else
{
    autre série d'instructions // ces instructions seront exécutées si la condition n'est pas réalisée
}
```

## 24.3. L'instruction if ... elseif ... else

L'instruction *if ... else* ne permet de tester qu'une condition, or il est parfois nécessaire de tester plusieurs conditions de façon exclusive, c'est-à-dire que sur toutes les conditions une seule sera réalisée ...

L'expression *if ... elseif ... else* permet d'enchaîner une série d'instructions et évite d'avoir à imbriquer des instructions *if*.

La syntaxe de cette expression est la suivante :

```
if (condition1)
{
    liste d'instructions // ces instructions seront exécutées si la condition 1 est réalisée
}
elseif (condition2)
{
    autre liste d'instructions // ces instructions seront exécutées si la condition 1 n'est pas réalisée
                                // et si la condition 2 est réalisée
}
else
{
    autre liste d'instructions // ces instructions seront exécutées si les conditions 1 et 2 ne sont pas réalisées
}
```

## 24.4. L'opérateur ternaire (une façon plus courte de faire un test)

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

*(condition) ? instruction si la condition est réalisée : instruction si la condition n'est pas réalisée ;*

### Remarques:

- la condition doit être entre parenthèses
- Lorsque la condition est vraie, l'instruction à gauche des ':' est exécutée
- Lorsque la condition est fausse, c'est l'instruction placée à droite des ':' qui est exécutée



## 24.5. L'instruction switch

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

```
switch (Variable)
{
    case Valeur1 :
        Liste d'instructions
        break ;
    case Valeur2 :
        Liste d'instructions
        break ;
    case Valeurn :
        Liste d'instructions
        break ;
    default:
        Liste d'instructions
        break ;
}
```

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

Si l'instruction **break ;** est omise à la suite de la liste d'instructions d'un *case*, PHP continuera à exécuter les instructions du ou des *case* suivants jusqu'à la rencontre d'un **break ;** ou la fin de l'instruction **switch**.

## 25. Les boucles

Les boucles sont des structures qui permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce que la condition ne soit plus réalisée...

On appelle parfois ces structures *répétitives* ou bien *itérations*.

Les trois instructions qui permettent d'effectuer une boucle sont : **while**, **do..while** et **for**.

### 25.1. L'instruction while

L'instruction *while* permet d'exécuter plusieurs fois une même série d'instructions.

La syntaxe de cette instruction est la suivante :

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

Cette instruction exécute la liste d'instructions **tant que** la condition est réalisée.

La condition doit **obligatoirement** contenir un *variant* comparé à un *invariant*.

Le *variant* est une variable dont la valeur varie au cours des différents passages dans la boucle.

L'*invariant* est une valeur ou une variable dont la valeur ne varie pas au cours des différents passages dans la boucle.

Si la condition n'est pas réalisée lors de sa première évaluation, aucune instruction de la liste n'est exécutée.

## 25.2. L'instruction do..while

L'instruction *do..while* permet également d'exécuter plusieurs fois une même série d'instructions.

La syntaxe de cette instruction est la suivante :

```
do
{
    liste d'instructions
} while (condition) ;
```

Cette instruction exécute la liste d'instructions **tant que** la condition est réalisée.

La condition doit **obligatoirement** contenir un *variant* comparé à un *invariant*.

Le *variant* est une variable dont la valeur varie au cours des différents passages dans la boucle.

L'*invariant* est une valeur ou une variable dont la valeur ne varie pas au cours des différents passages dans la boucle.

La principale différence par rapport à l'instruction *while* est que la **première** itération de la boucle est toujours exécutée car la condition d'arrêt n'est testée qu'à la fin de l'itération..

## 25.3. L'instruction for

L'instruction *for* est l'instruction la plus complexe permettant d'exécuter plusieurs fois la même série d'instructions.

La syntaxe de cette expression est la suivante :

```
for (expression1; condition ; expression2)
{
    liste d'instructions
}
```

- *expression1* est constituée d'aucune, une ou plusieurs instructions séparées par une virgule. Cette ou ces instructions sont exécutées **une seule fois avant** la première évaluation de la *condition* et donc avant la première itération. En règle générale cette expression contient l'**initialisation** de la variable dite *variant*.
- *condition* est la condition d'arrêt de la boucle. Elle équivaut au **tant que** du *while* et est évaluée avant chaque itération, elle doit donc **obligatoirement** contenir le *variant* (généralement initialisé dans *expression1*) comparé à un *invariant*.
- *expression2* est constituée d'aucune, une ou plusieurs instructions séparées par une virgule. Elle est exécutée systématiquement à chaque itération (**après** l'exécution de la liste des instructions et **avant** l'évaluation de la *condition* pour l'itération suivante). En règle générale cette expression contient l'**incrément** (ou la décrémentation) de la variable dite *variant*.

Les points-virgules doivent toujours être présents, même si *expression1* et/ou *expression2* ne contiennent aucune instruction.

Si la liste d'instructions est inexistante, l'instruction *for* doit se terminer par un point-virgule,

la syntaxe est alors la suivante : *for (expression1; condition ; expression2) ;*

Exemple:

```
for ($i=1 ; $i<6 ; $ i++)
{
    echo $i . ' ';
}
```

Cette boucle affiche 5 fois la valeur de *\$i*, c'est-à-dire 1,2,3,4,5,

Autre exemple restituant le même résultat :

```
for ($i=1 ; $i<6 ; echo $ i++ . ' ' ) ;
```

## 26. Saut inconditionnel continue

Il peut être nécessaire de ne pas exécuter certaines instructions dans la boucle pour une ou plusieurs valeurs sans pour autant mettre fin à celle-ci.

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

Exemple :

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

```
$x=1 ;
while ($x<=10)
{
    if ($x == 7)
    {
        echo("Attention division par zéro!");
        continue; // saut à l'itération suivante
    }
    $a = 1/($x-7);
    echo $a;
    $x++ ;
}
```

Il y a une erreur dans ce programme... peut-être l'avez-vous vue ! :

Lorsque x est égal à 7, le compteur ne s'incrémente plus, il reste constamment à la valeur 7, il aurait fallu écrire:

```
$x=1 ;
while ($x<=10)
{
    if ($x == 7)
    {
        echo("Attention division par zéro!");
        $x++ ; // incrémentation de $x
        continue; // saut à l'itération suivante avec $x=8
    }
    $a = 1/($x-7);
    echo $a;
    $x++ ;
}
```

## 27. Arrêt inconditionnel break

A l'inverse, il peut être voulu d'arrêter prématurément la boucle, pour une autre condition que celle précisée dans l'en-tête de la boucle. L'instruction *break* permet d'arrêter une boucle. Il s'agit, tout comme *continue*, de l'associer à une structure conditionnelle.

Dans l'exemple précédent, par exemple si l'on ne savait pas à quel moment le dénominateur (x-y) s'annule, il serait possible de faire arrêter la boucle en cas d'annulation du dénominateur, pour éviter une division par zéro!

```
.....
for ($x=1 ; $x<=10 ; $x++)
{
    $a = $x-$y; // on considère que $y a été initialisé ou calculé avant la boucle for
    if ($a == 0)
    {
        echo("Attention division par zéro !");
        break; // arrêt de la boucle et exécution de l'instructionx située après la boucle
    }

    echo 1/$a;
}

instructionx ;
```

## 28. Arrêt d'exécution du script par exit

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

## 5. Les tableaux

### 29. Les variables tableaux

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

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

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

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

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

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

Les indices de tableau commencent à zéro.

Tous les types de variables peuvent être contenus dans un tableau.

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

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

### 30. Les variables des tableaux associatifs

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

```
$Toto["Age"] = 12;  
$Toto["Adresse"] = "22 rue des bois fleuris";  
$Toto["Nom"] = "AVOUSDETROUVER";
```

```
$organisme = "ESAT";  
$tab_organismes[$organisme][0] = $organisme;           // nom de l'organisme  
$tab_organismes[$organisme][1] = $nbpers;              // nombre de personnes  
$tab_organismes[$organisme][2] = $nbabon;              // nombre d'abonnés
```

## 6. Les fonctions

### 31. La notion de fonction

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

### 32. La déclaration d'une fonction

PHP recèle de nombreuses fonctions intégrées permettant d'effectuer des actions courantes. Toutefois, il est possible de définir des fonctions, dites *fonctions utilisateurs* afin de simplifier l'exécution de séries d'instructions répétitives. Contrairement à de nombreux autres langages, PHP3 nécessite que l'on définisse une fonction avant que celle-ci puisse être utilisée, car pour l'appeler dans le corps du programme il faut que l'interpréteur la connaisse, c'est-à-dire qu'il connaisse son nom, ses arguments et les instructions qu'elle contient (PHP4 a supprimé cette restriction). La définition d'une fonction s'appelle "*déclaration*" et peut se faire n'importe où dans le code. La déclaration d'une fonction se fait grâce au mot clé *function*, selon la *syntaxe suivante* :

```
function Nom_De_La_Fonction(type1 argument1, type2 argument2, ...)  
{  
    liste d'instructions  
}
```

#### Remarques:

le nom de la fonction suit les mêmes règles que les noms de variables, c'est-à-dire :

- le nom doit commencer par une lettre
- un nom de fonction peut comporter des lettres, des chiffres et les caractères \_ et & (les espaces ne sont pas autorisés!)
- le nom de la fonction, comme celui des variables est sensible à la casse (différenciation entre les minuscules et majuscules)

Les arguments sont facultatifs, mais s'il n'y a pas d'argument, les parenthèses doivent rester présentes.

Il ne faut pas oublier de refermer les accolades.

Le nombre d'accolades ouvertes (fonction, boucles et autres structures) doit être égal au nombre d'accolades fermées!
---

La même chose s'applique pour les parenthèses, les crochets ou les guillemets!
--

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

### 33. Appel de fonction

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

Exemples :

```
Nom_De_La_Fonction();           // appel de fonction sans argument
Nom_De_La_Fonction(argument1, argument2); // appel de fonction avec 2 arguments
```

Remarques:

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

```
Nom_De_La_Fonction(argument1, argument2);
```

### 34. Les arguments d'une fonction

Il est possible de passer des arguments à une fonction, c'est-à-dire lui fournir des valeurs ou le nom de variables afin que la fonction puisse effectuer des opérations sur ces arguments ou bien grâce à ces arguments.

Le passage d'arguments à une fonction se fait au moyen d'une liste d'arguments (séparés par des virgules) entre parenthèses suivant immédiatement le nom de la fonction.

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

### 35. Travailler sur des variables dans les fonctions

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

Cela est dû à la portée des variables, c'est-à-dire si elles ont été définies comme **variables globales ou locales**.

- Une variable précédée du mot clé *global* sera visible dans l'ensemble du code, c'est-à-dire que sa portée ne sera pas limitée à la fonction seulement. Ainsi, toutes les fonctions pourront utiliser et modifier cette même variable
- dans le cas contraire, la variable ne pourra être modifiée qu'à l'intérieur de la fonction et retrouvera, à la sortie de celle-ci, la valeur qu'elle avait juste avant l'appel de fonction
- une autre méthode pour modifier une variable consiste à la faire précéder du caractère *&*, précisant qu'il s'agit alors d'un passage d'argument par référence.

### 36. Renvoi d'une valeur par une fonction

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

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

La syntaxe de l'instruction return est simple :

```
return valeur_ou_variable_ou_expression;
```

Exemple :

```
return $x + $y ;
```

## 37. Passage d'argument par référence

Par défaut, les arguments sont passés à la fonction par *valeur* : dans ce cas il y a **recopie** de la valeur de l'argument précisé lors de l'appel à la fonction dans l'argument mentionné dans la description de la fonction et qui est alors une variable locale à cette fonction.

Si vous voulez que la fonction puisse modifier la valeur de l'argument passé lors de l'appel, vous devez passer cet argument par référence. PHP vous permet ce passage par référence selon 2 méthodes :

1. Si vous voulez qu'un argument soit **toujours** passé par référence, vous pouvez ajouter un '&' devant l'argument dans **la déclaration** de la fonction :

```
Function carre(&$xlocal)
{
    $xlocal = $xlocal * $xlocal ;
}

$x = 4 ;
carre($x) ;
echo $x;    // affichera 16
```

2. Si vous voulez passer **ponctuellement** un argument par référence, vous pouvez ajouter un '&' devant l'argument dans **l'appel** de la fonction :

```
Function carre($xlocal)
{
    $xlocal = $xlocal * $xlocal ;
}

$x = 4 ;
carre($x) ;
echo $x;    // affichera 4
carre(&$x) ;
echo $x;    // affichera 16
```

## 38. Valeur par défaut des arguments

Vous pouvez définir des valeurs par défaut pour les arguments de type scalaire.

Exemple1 :

```
Function connecter($periph = "clavier")
{
    return "connexion du $periph réalisée !\n" ;
}

echo connecter() ;           // affichera 'connexion du clavier réalisée !'
echo connecter("scanner") ; // affichera 'connexion du scanner réalisée !'
```

Exemple2 :

```
Function connecter($type, $periph = "clavier")
{
    return "connexion du $periph avec fiche $type réalisée !\n" ;
}

echo connecter("USB") ;      // affichera 'connexion du clavier avec fiche USB réalisée !'
```

La valeur par défaut d'un argument doit **obligatoirement** être une constante !



## 39. La fonction include

La fonction *include* '*nom\_de\_fichier*'; **interprète** le fichier spécifié en argument à chaque fois que la fonction est rencontrée.

Vous pouvez donc utiliser la fonction *include* dans une structure de boucle afin d'interpréter plusieurs fichiers.

Exemple :

```
$tab_files = array('fic1.inc', 'fic2.inc', 'fic3.inc', 'fic4.inc') ;  
for( $i = 0 ; $i < sizeof($tab_files) ; $i++)  
{  
    include $tab_files[$i] ;  
}
```

*include* diffère de la fonction **require** '*nom\_de\_fichier*'; dans le sens où la fonction *include* interprète le fichier spécifié en argument à chaque fois que la fonction est rencontrée, alors que la fonction *require* '*nom\_de\_fichier*'; est **remplacée** systématiquement par le fichier spécifié en argument, que la fonction soit rencontrée ou non. *require* est identique au *#include* du langage C.

Vous devez inclure la fonction *include* dans un bloc {} lorsque celle-ci est placée dans une structure conditionnelle.

## 40. La fonction empty

Détermine si une variable est affectée.

Syntaxe :

```
bool empty(mixed var);
```

Elle retourne la valeur FALSE "faux" si la variable *var* est définie et a une valeur différente de 0. Elle retourne la valeur TRUE "vrai" si elle n'est pas définie ou si elle est définie et contient la valeur 0.

## 41. La fonction isset

Détermine si une variable est affectée.

Syntaxe :

```
bool isset(mixed var);
```

Elle retourne la valeur TRUE "vrai" si la variable *var* est définie et contient une valeur ; FALSE "faux" sinon.

## 42. La fonction unset

Détruit une variable.

Syntaxe :

```
bool unset(mixed var);
```

Détruit la variable *var* et renvoie la valeur TRUE "vrai".

## 7. La gestion des tableaux

### 43. array

`Array(...)`; retourne un tableau créé à partir des arguments fournis. Les arguments peuvent être sous la forme *indice* => *valeur*.

`Array()` n'est pas une fonction standard, elle existe simplement pour représenter littéralement des tableaux.

L'exemple suivant montre la construction d'un tableau bi-dimensionnel, l'assignation d'indices pour les tableaux associatifs, et comment écarter certains intervalles d'indices numériques.

```
$stab_fruits = array(
    "fruits" => array("a"=>"orange", "b"=>"banane", "c"=>"pomme"),
    "nombres" => array(1, 2, 3, 4, 5, 6),
    "trous" => array("premier", 5 => "deuxième", "troisième"));

echo $stab_fruits["fruits"]["a"];      // affichera 'orange'
echo $stab_fruits["nombres"][0];      // affichera '1'
echo $stab_fruits["trous"][0];        // affichera 'premier'
echo $stab_fruits["trous"][6];        // affichera 'troisième'
```

### 44. is\_array

Renvoie la valeur TRUE si la variable `$var` est un tableau, FALSE sinon.

La syntaxe est la suivante :

```
is_array($var)
```

### 45. each

La fonction `each()` permet de parcourir tous les éléments d'un tableau sans se soucier de ses bornes, ce qui en fait un outil extrêmement pratique.

Son fonctionnement est simple : elle retourne la combinaison *indice-valeur* courante du tableau passé en argument, puis se positionne sur l'élément suivant de ce tableau, et cela, du premier au dernier indice.

Lorsque la fin du tableau est atteinte, `each()` retourne la valeur FALSE.

Exemple :

```
$var = array();
$stab = array('val1', 'val2', 'val3', 'val4');
$var = each($stab);    // affectation du premier couple indice-valeur du tableau $stab à $var
while($var)           // tant que $var est vrai
{
    echo "$var[0] : $var[1]<br>";    // affichage de l'indice et de la valeur
    $var = each($stab);           // affectation du couple indice-valeur suivant du tableau $stab à $var
}
```

Résultat :

```
0 : val1
1 : val2
2 : val3
3 : val4
```

## 46. list

La fonction `list()` est très souvent associée à la fonction `each()`. Elle permet d'affecter les éléments d'un tableau à des variables distinctes.

En reprenant l'exemple précédent, on pourra utiliser `list()` de la manière suivante :

```
$tab = array('val1','val2','val3','val4') ;
while( list($indice, $valeur) = each($tab) ) // tant que la fonction each() renvoie une valeur non nulle (vraie)
{
    echo "$indice : $valeur <br>" ;           // affichage de chaque couple indice et valeur
}
```

Nous obtenons le même résultat que précédemment.

Le tableau `$var` est remplacé par deux variables distinctes.

- Les utilisations les plus pertinentes de la fonction interviennent notamment lors de requêtes effectuées sur des bases de données relationnelles qui renvoient les informations sous forme de tableaux.
- Dans les applications Intranet/Internet, il est courant de devoir construire des pages qui comportent un nombre indéterminé de champs de saisie, comme une page de saisie d'options. La solution est d'utiliser des champs de type 'text' qui portent le même nom. PHP va automatiquement créer un tableau dans la page cible, pour lequel les fonctions `list()` et `each()` sont particulièrement adaptées.

Exemple :

```
<FORM action = 'formul1.php3' method = POST>
<table>
  <tr>
    <td>Valeur 1 : </td>
    <td><input type = text length = 10 name = 'tabval[]' /></td>
  </tr>
  <tr>
    <td>Valeur 2 : </td>
    <td><input type = text length = 10 name = 'tabval[]' /></td>
  </tr>
  <tr>
    <td>Valeur 3 : </td>
    <td><input type = text length = 10 name = 'tabval[]' /></td>
  </tr>
  <tr>
    <td>Valeur 4 : </td>
    <td><input type = text length = 10 name = 'tabval[]' /></td>
  </tr>
  <tr>
    <td colspan = 2 align = center>< input type = submit value = 'Valider'></td>
  </tr>
</table>
</FORM>
```

Dans le formulaire `formul1.php3`, la série de valeurs est reçue sous forme d'un tableau portant le nom du champ texte. Il ne reste plus qu'à le parcourir de la façon suivante :

```
while( list($indice, $valeur) = each($tabval) ) // tant que la fonction each() renvoie une valeur non nulle
{
    echo "$indice : $valeur <br>" ;           // affichage de chaque couple indice et valeur
}
```

## 47. count

*count(\$var)* retourne le nombre d'éléments dans *var*, qui est généralement un tableau (et tout le reste n'aura qu'un élément).

Retourne 1 si la variable n'est pas un tableau.

Retourne 0 si la variable n'est pas créée, mais il peut aussi retourner 0 pour un tableau vide. Utilisez plutôt la commande *isset()* pour savoir si une variable existe ou pas.

## 48. sizeof

*sizeof(\$tab)* retourne également le nombre d'éléments du tableau *\$tab*.

## 49. current

Chaque tableau entretient un pointeur interne, qui est initialisé lorsque le premier élément est inséré dans le tableau. La fonction *current(\$tab)* ne fait que retourner l'élément courant pointé par le pointeur interne. *current()* ne déplace pas le pointeur. Si le pointeur est au-delà du dernier élément de la liste, *current()* retourne faux. Si le tableau contient des éléments vides ou des zéros (0 ou "", la chaîne vide) alors cette fonction retournera false pour ces éléments. Il est donc impossible de déterminer si vous êtes réellement à la fin de la liste en utilisant la fonction *current()*.

Pour passer en revue proprement un tableau qui peut contenir des éléments vides ou des zéros, utilisez la fonction *each()*.

## 50. pos

*pos(\$tab)* retourne l'élément courant du tableau *\$tab*. C'est un alias de *current()*.

## 51. next

*next(\$tab)* retourne l'élément suivant du tableau *\$tab*, ou false s'il n'y a plus d'élément. Le pointeur interne du tableau est avancé d'un élément.

*next()* se comporte comme *current()*, mais avec une différence : Il avance le pointeur interne de tableau d'un élément avant de retourner la valeur sur lequel il pointe. Lorsque le pointeur dépasse le dernier élément, *next()* retourne false. Si le tableau contient des éléments vides ou des zéros, cette fonction retournera false pour ces éléments. Pour passer proprement en revue un tableau, il faut utiliser *each()*.

## 52. prev

*prev(\$tab)* repositionne le pointeur interne du tableau *\$tab* à la dernière place qu'il occupait et retourne l'élément, ou bien retourne faux s'il ne reste plus d'élément. Si le tableau contient des éléments vides, cette fonction retournera faux pour ces éléments aussi. Pour passer en revue tous les éléments, utilisez plutôt *each()*.

*prev()* se comporte exactement comme *next()*, mais il fait reculer le pointeur plutôt que de l'avancer.

## 53. reset

*reset(\$tab)* replace le pointeur du tableau *\$tab* au premier élément.

*reset()* retourne la valeur du premier élément.

## 54. end

`end($tab)` déplace le pointeur interne du tableau `$tab` jusqu'au dernier élément.  
`end()` retourne la valeur du dernier élément.

## 55. key

`key($tab)` retourne l'indice courant du pointeur interne du tableau `$tab`.

Exemple :

```
<?
$tab = array('val1','val2','val3','val4');
echo "premier = ".reset($tab)." indice = ".key($tab)."<br>";
echo "dernier = ".end($tab)." indice = ".key($tab)."<br>";
?>
```

Résultat :

```
premier = val1 indice = 0
dernier = val4 indice = 3
```

## 56. sort

`sort($tab)` trie les éléments du tableau `$tab` du plus petit au plus grand

Exemple :

```
<html>
<head>
<title>Page d'essai PHP</title>
</head>
<body>
<?
echo "<form method='post' action='#'>";
$tab_fruits = array("papaye","orange","banane","ananas");
// tri du tableau en ordre croissant
sort($tab_fruits);
// affichage du tableau trié
for($key = 0; $key < sizeof($tab_fruits); $key++)
{
    echo "tab_fruits[$key] = ".$tab_fruits[$key]."<br>";
}
echo "</form>";
?>
</body>
</html>
```

Cet exemple va afficher :

```
tab_fruits[0] = ananas
tab_fruits[1] = banane
tab_fruits[2] = orange
tab_fruits[3] = papaye
```

Les fruits ont été classés dans l'ordre alphabétique.

## 57. asort

La fonction `asort($tab)` trie le tableau `$tab` en ordre croissant et de telle manière que la corrélation entre les indices et les valeurs soit conservée. L'usage principal est lors de tri de tableaux associatifs où l'ordre des éléments est important.

Exemple :

```
$tab_fruits = array("d"=>"papaye","a"=>"orange","b"=>"banane","c"=>"ananas");
// tri du tableau en ordre croissant
asort($tab_fruits);
// affichage du tableau trié
for($key = 0, reset($tab_fruits); $key < sizeof($tab_fruits); $key++, next($tab_fruits))
{
    $indice = key($tab_fruits);
    echo "tab_fruits[$indice] = ".$tab_fruits[$indice]."<br>";
}
```

Cet exemple va afficher :

```
tab_fruits[c] = ananas
tab_fruits[b] = banane
tab_fruits[a] = orange
tab_fruits[d] = papaye
```

Les fruits ont été classés dans l'ordre alphabétique et leur indice respectif a été conservé.

## 58. arsort

La fonction `arsort($tab)` trie le tableau `$tab` en ordre décroissant et de telle manière que la corrélation entre les indices et les valeurs soit conservée. L'usage principal est lors de tri de tableaux associatifs où l'ordre des éléments est important.

Exemple :

```
$tab_fruits = array ("a"=>"orange","c"=>"ananas","b"=>"banane","d"=>"papaye" );
// tri du tableau en ordre décroissant
arsort($tab_fruits);
// affichage du tableau trié
for($key = 0, reset($tab_fruits); $key < sizeof($tab_fruits); $key++, next($tab_fruits))
{
    $indice = key($tab_fruits);
    echo "tab_fruits[$indice] = ".$tab_fruits[$indice]."<br>";
}
```

Cet exemple va afficher :

```
tab_fruits[d] = papaye
tab_fruits[a] = orange
tab_fruits[b] = banane
tab_fruits[c] = ananas
```

Les fruits ont été classés dans l'ordre alphabétique inverse et leur indice respectif a été conservé.

## 59. rsort

La fonction *rsort(\$tab)* trie le tableau *\$tab* en ordre décroissant mais ne conserve pas la corrélation entre les indices et les valeurs.

L’affichage du tableau *\$tab\_fruits* de l’exemple précédent et trié à l’aide de la fonction *rsort(\$tab\_fruits)* donnerait :

```
tab_fruits[0] = papaye  
tab_fruits[1] = orange  
tab_fruits[2] = banane  
tab_fruits[3] = ananas
```

## 60. ksort

La fonction *ksort(\$tab)* trie le tableau *\$tab* en ordre croissant sur les indices et conserve la corrélation entre les indices et les valeurs.

L’affichage du tableau *\$tab\_fruits* de l’exemple précédent et trié à l’aide de la fonction *ksort(\$tab\_fruits)* donnerait:

```
tab_fruits[a] = orange  
tab_fruits[b] = banane  
tab_fruits[c] = ananas  
tab_fruits[d] = papaye
```

## 61. usort, uksort et uasort

Pour trier les éléments d'un tableau selon plusieurs critères, il faut utiliser des fonctions de comparaison personnalisées. En effet, les fonctions intégrées de PHP ne réalisent que des tris basiques. Par conséquent, si l'on désire utiliser des critères de tri plus fins, il est nécessaire de développer une fonction sur mesure.

Cette fonction sera ensuite passée en paramètre aux fonctions *uasort*, *uksort* et *usort* qui réalisent respectivement des tris semblables à ceux effectués par les fonctions *asort*, *ksort* et *sort*.

Voici un exemple qui classe une série de nombres en fonction de leur écart avec la valeur 50, de l'écart le plus faible au plus important :

*// fonction de comparaison qui retourne 0 si les 2 nombres comparés sont égaux, 1 si le premier est plus éloigné de 50 du deuxième, et -1 dans le cas contraire.*

*Function cmp(\$nb1, \$nb2)*

```
{
    if( $nb1 == $nb2 )
        return 0;
    if( abs(50 - $nb1) > abs(50 - $nb2) )
        return 1;
    else
        return -1;
}
```

*// construction du tableau*

*\$tab = array(10,25,5,62,118);*

*// tri du tableau avec la fonction cmp()*

*usort(\$tab, cmp);*

*// affichage du tableau trié*

```
while( list($indice,$valeur) = each($tab) )
    echo "$indice : $valeur (écart = " . abs(50 - $valeur) . ")<br>";
```

Résultat de l'affichage du tableau trié :

*0 : 62 (écart = 12)*

*1 : 25 (écart = 25)*

*2 : 10 (écart = 40)*

*3 : 5 (écart = 45)*

*4 : 118 (écart = 68)*

Le tableau est trié en ordre croissant sur l'écart, mais *usort* n'a pas conservé la corrélation entre les indices et les valeurs. En utilisant *uasort*, on obtient un tableau dans lequel la corrélation entre les indices et les valeurs est conservée.

Résultat de l'affichage du tableau trié avec *uasort(\$tab, cmp)* :

*3 : 62 (écart = 12)*

*1 : 25 (écart = 25)*

*0 : 10 (écart = 40)*

*2 : 5 (écart = 45)*

*4 : 118 (écart = 68)*



## 62. extract

*void extract(array var\_array, int extract\_type, string prefix)*

Cette fonction sert à exporter un tableau vers la table des symboles. Elle prend un tableau associatif *var\_array* et crée les variables dont les noms sont les indices de ce tableau, et leur affecte la valeur associée. Pour chaque paire indice/valeur, cette fonction crée une variable, avec les paramètres *extract\_type* et *prefix*.

*extract()* vérifie l'existence de la variable avant de la créer. La manière de traiter les collisions est déterminée par *extract\_type*. Ce paramètre peut prendre une des valeurs suivantes :

- EXTR\_OVERWRITE : Lors d'une collision, réécrire la variable existante.
- EXTR\_SKIP : Lors d'une collision, ne pas réécrire la variable existante
- EXTR\_PREFIX\_SAME : Lors d'une collision, ajouter le préfixe *prefix*, et créer une nouvelle variable.
- EXTR\_PREFIX\_ALL : Ajouter le préfixe *prefix*, et créer une nouvelle variable.

Si *extract\_type* est omis, *extract()* utilise EXTR\_OVERWRITE par défaut.

Notez que *prefix* n'est nécessaire que pour les valeurs de *extract\_type* suivantes : EXTR\_PREFIX\_SAME et EXTR\_PREFIX\_ALL.

*extract()* vérifie que les indices constituent un nom de variable valide, et si c'est le cas, procède à son exportation.

Exemple :

```
<php?
$taille = "grand";
$var_array = array( "couleur" => "bleu",
                  "taille" => "moyen",
                  "forme" => "sphere");
extract($var_array, EXTR_PREFIX_SAME, "wddx");
echo "$couleur, $taille, $forme, $wddx_taille<br>";
?>
```

L'exemple ci dessus va afficher :

*bleu, grand, sphere, moyen*

La variable *\$taille* n'a pas été réécrite, car on avait spécifié le paramètre EXTR\_PREFIX\_SAME, qui a permis la création *\$wddx\_taille*. Si EXTR\_SKIP avait été utilisé, alors *\$wddx\_taille* n'aurait pas été créé. Avec EXTR\_OVERWRITE, *\$taille* aurait pris la valeur "moyen", et avec EXTR\_PREFIX\_ALL, les variables créées seraient *\$wddx\_couleur*, *\$wddx\_taille*, et *\$wddx\_forme*.

## 63. range

La fonction *range(int low, int high)* retourne un tableau contenant tous les entiers depuis *low* jusqu'à *high*, inclus.

Exemple :

```
$tab = array();
$tab = range(0,10);
for( $i = 0 ; $i < sizeof($tab) ; $i++)
    echo "$tab[$i] ";
```

Cet exemple affichera :

*0 1 2 3 4 5 6 7 8 9 10*

## 64. shuffle

La fonction *shuffle(\$tab)* mélange les éléments du tableau *\$tab*.

Exemple :

```
$tab = array();
$tab = range(0,10);
shuffle($tab) ;
for( $i = 0 ; $i < sizeof($tab) ; $i++)
    echo "$tab[$i] ";
```

Cet exemple affichera de façon aléatoire :

*6 1 0 5 9 8 7 3 10 4 2*

## 65. compact

*compact()* accepte différents paramètres. Les paramètres peuvent être des variables contenant des chaînes, ou un tableau de chaînes, qui peut contenir d'autres tableaux de noms, que *compact()* traitera récursivement.

Pour chacun des arguments, *compact()* recherche une variable avec une variable de même nom dans la table courante des symboles, et l'ajoute dans le tableau, de manière à avoir la relation nom => 'valeur de variable'. En bref, c'est le contraire de la fonction *extract()*.

*compact()* retourne le tableau ainsi créé.

Exemple :

```
$ville = "San Francisco";
$etat = "CA";
$evenement = "SIGGRAPH";
$location_vars = array("ville", "etat");
$result = compact("evenement", $location_vars);
for( $i = 0, reset($result) ; $i < sizeof($result) ; $i++, next($result))
{
    $indice = key($result);
    echo "$indice => $result[$indice]<br>";
}
```

Après cette opération, *\$result* sera le tableau suivant :

```
evenement => SIGGRAPH
ville => San Francisco
etat => CA
```

## 66. Autres fonctions sur les tableaux sous Php4

<b>Array_push</b>	Empile un ou plusieurs éléments à la fin d'un tableau
<b>Array_pop</b>	Dépille un élément de la fin d'un tableau
<b>Array_unshift</b>	Empile un ou plusieurs éléments au début d'un tableau
<b>Array_shift</b>	Dépille un élément au début d'un tableau
<b>Array_slice</b>	Extrait une portion de tableau
<b>Array_splice</b>	Efface une portion de tableau et la remplace
<b>Array_merge</b>	Rassemble deux ou plusieurs tableaux en un seul
<b>Array_keys</b>	Retourne les indices d'un tableau
<b>Array_values</b>	Retourne les valeurs d'un tableau
<b>Array_walk</b>	Exécute une fonction sur chacun des membres d'un tableau
<b>In_array</b>	Retourne vrai si une valeur appartient à un tableau

## 8. Les chaînes de caractères

Une chaîne de caractères peut être comparée à un tableau de caractères.

PHP propose un ensemble de fonctionnalités qui permettent de réaliser diverses opérations sur les chaînes de caractères, selon des principes de position analogues aux indices d'un tableau. Toutefois, il n'est pas possible d'accéder directement aux différents caractères qui composent la chaîne par leur indice. On devra toujours passer par une fonction du SDK (kit de développement PHP).

PHP reprend la plus grande partie des fonctions de traitement de chaîne de caractères du langage C. Il propose, d'autre part, beaucoup d'autres fonctions très utiles, telles que les expressions régulières, divers encodages et parsers spécifiques au protocole HTTP et au HTML, ainsi que des fonctions de cryptage.

### 67. Affichage d'une chaîne de caractères

Il existe deux catégories de fonctions d'affichage d'une chaîne en PHP : les fonctions d'affichage simple et les fonctions d'affichage formaté.

Les deux fonctions utilisées pour afficher une chaîne de caractères non formatée sont : *echo* et *print*.

*echo* est une instruction, elle s'écrit **sans** parenthèse. Il est possible d'intégrer des variables dans l'affichage :

```
$var = 10 ;  
echo "la variable var vaut $var" ;      // affichera 'la variable var vaut 10'
```

La fonction *print* est similaire mais nécessite des parenthèses :

```
print("la variable var vaut $var");      // affichera 'la variable var vaut 10'
```

Pour afficher une chaîne de caractères formatée, PHP propose deux fonctions identiques à celles du langage C : *printf* et *sprintf*. Elles possèdent toutes deux une syntaxe voisine :

```
printf("format", variables) ;  
variable_resultat = sprintf("format", variables) ;
```

La différence entre ces deux fonctions réside dans le fait que *printf* produit un affichage immédiat, alors qu'il faut affecter le résultat de *sprintf* à une variable, puis si besoin, afficher cette variable à l'aide de *echo* ou *print*.

### 68. Initialisation et concaténation

En PHP, les chaînes de caractères s'initialisent grâce à l'opérateur d'affectation '=' :

```
$chaine = "bonjour" ;      // $chaine est initialisée à 'bonjour'
```

La concaténation de deux chaînes se fait au moyen de l'opérateur '.' (point) :

```
$chaine2 = $chaine . " monsieur" ;      // $chaine2 contient 'bonjour monsieur'  
$chaine .= " monsieur" ;      // $chaine contient maintenant 'bonjour monsieur'
```

## 69. Traitement des chaînes de caractères

Pour toutes les opérations de gestion classique de chaînes, PHP propose les fonctions issues du langage C et beaucoup d'autres, qui facilitent davantage l'exploitation de ces chaînes de caractères.

Parmi la liste des fonctions énumérées ci-dessous, nous nous attacherons à donner des exemples de fonctions classiques, pour les autres nous vous conseillons de consulter la documentation Nexen.net « Manuel PHP traduit en français par Nexen » :

**string addslashes(string str)** : Ajoute un slash devant les caractères spéciaux : ' " \ et NUL contenu dans la chaîne *str*.

**string bin2hex(string)** : Convertit une valeur binaire en hexadécimal

**string chop(string str)** : Enlève les caractères blancs de la fin de la chaîne *str* et retourne la chaîne nettoyée. Les caractères blancs sont : "\n", "\r", "\t", "\v", "\0", et " " (espace).

**string chr(int code\_ascii)** : Retourne le caractère correspondant à la valeur ASCII *code\_ascii*.

**string chunk\_split(string str, [int chunklen], [string end])** : Scinde la chaîne *str* en plus petits morceaux. Cette fonction insert une fin de chaîne *end* (par défaut "\r\n"), tous les *chunklen* (par défaut 76) caractères. La chaîne retournée est une nouvelle chaîne, et l'original n'est pas modifié.

**string convert\_cyr\_string(string str, string from, string to)** : Convertit la chaîne *str* d'un alphabet cyrillique vers un autre. Les arguments *from* et *to* sont des caractères qui représentent la source et la destination.

**string crypt(string str, [string salt])** : Crypte la chaîne *str* avec un DES (Data Encryption Standard).

**int ereg(string mask, string str, [array regs])** : Recherche dans la chaîne *str* les séquences de caractères qui correspondent au masque donné *mask*.

**int eregi(string mask, string str, [array regs])** : Recherche dans la chaîne *str* les séquences de caractères qui correspondent au masque donné *mask*. Cette fonction est identique à *ereg()*, hormis le fait qu'elle ignore la casse des caractères lors de la recherche sur les caractères alphabétiques.

**string ereg\_replace(string needle, string replace, string str)** : Remplace toutes les occurrences de *needle* dans *str* par la chaîne *replace*.

**string eregi\_replace(string needle, string replace, string str)** : Remplace toutes les occurrences de *needle* dans *str* par la chaîne *replace*. Cette fonction est identique à *ereg\_replace()*, en dehors du fait qu'elle ne tient pas compte de la casse des caractères alphabétiques.

**array explode(string separator, string str)** : Scinde la chaîne *str* en morceaux, grâce au délimiteur *separator*.

**void flush(void)** : Vide les buffers de sorties que PHP utilisait (CGI, serveur Web, etc.)

**array get\_meta\_tags(string filename, [int use\_include\_path])** : Extrait toutes les balises meta du fichier *filename*, et les retourne sous forme d'un tableau.

**string htmlspecialchars(string str)** : Convertit tous les caractères spéciaux de la chaîne *str* en équivalent HTML. PHP remplace les valeurs suivantes : '&' (et commercial) devient '&amp;'; '"' (guillemets doubles) devient '&quot;'; '<' (inférieur à) devient '&lt;'; '>' (supérieur à) devient '&gt;'.

**string htmlentities(string str)** : Convertit tous les caractères de la chaîne *str* qui ont une séquence d'échappement en équivalent HTML

**string implode(string glue, array tab)** : Regroupe tous les éléments du tableau *tab* dans une chaîne, avec la chaîne de jointure *glue*.

**string join(string glue, array tab)**: Regroupe tous les éléments du tableau *tab* dans une chaîne, avec la chaîne de jointure *glue*. *Join()* est un alias de *implode()*.

**string ltrim(string str)**: Enlève les caractères blancs du début de la chaîne *str* et retourne la chaîne nettoyée. Les caractères blancs sont : `"\n"`, `"\r"`, `"\t"`, `"\v"`, `"\0"`, et `" "` (espace).

**string md5(string str)**: Crypte la chaîne *str* en utilisant la méthode MD5.

**string nl2br(string str)**: Convertit les retours à la ligne `'\n'` en balise HTML `'<BR>'`.

**int ord(string str)**: Retourne la valeur ASCII du premier caractère de la chaîne *str*.

**void parse\_str(string str)**: Analyse la chaîne *str* comme si c'était une chaîne passée par URL, et affecte les variables qu'elle y trouve.

**int printf(string format, [mixed args]...)**: Affiche une chaîne formatée

**string quoted\_printable\_decode(string str)**: Convertit la chaîne *str* contenant des slash en une chaîne sans slash.

**string quotemeta(string str)**: Ajoute un backslash devant tous les caractères méta suivants : `\ + * ? [ ^ ] ( $ )`

**string rawurldecode(string str)**: Décode la chaîne URL *str*.

**string rawurlencode(string str)**: Encode la chaîne *str* en chaîne URL, selon la RFC1738.

**string setlocale(string category, string locale)**: Change les informations locales.

**int similar\_text(string first, string second, [double percent])**: Calcule la similarité des deux chaînes *first* et *second*. En passant une référence comme troisième argument, *similar\_text()* va calculer le pourcentage de similarité. Il retourne le nombre de caractères correspondant l'un à l'autre, d'une chaîne à l'autre.

**string soundex(string str)**: Calcule la valeur soundex de la chaîne *str*. Une valeur Soundex est telle que deux mots prononcés de la même façon auront des valeurs Soundex identiques, et cela permet d'effectuer des recherches dans les bases de données, même si vous connaissez la prononciation, mais pas l'orthographe. Cette fonction retourne une chaîne de 4 caractères, commençant par une lettre.

**string sprintf(string format, [mixed args]...)**: Retourne une chaîne formatée. *Sprintf()* fonctionne comme *printf()*, mais la chaîne formatée est retournée au lieu d'être affichée.

**string strchr(string str, string needle)**: Retourne toute la chaîne *str* à partir de la première occurrence de *needle*, jusqu'à la fin. Cette fonction est un alias de la fonction *strstr()*.

**int strcmp(string str1, string str2)**: Comparaison des deux chaînes *str1* et *str2* sensible à la casse. Retourne `< 0` si *str1* est plus petite que *str2*; `> 0` si *str1* est plus grande que *str2*, et `0` si elles sont égales.

**int strcspn(string str, string mask)**: Retourne la longueur du premier segment de la chaîne *str* qui ne corresponde à aucun des caractères de la chaîne *mask*.

**string strip\_tags(string str, [string allowable\_tags])**: Enlève les balises HTML et PHP de la chaîne *str*. Cela empêche les utilisateurs malicieux de planter vos scripts lorsque le navigateur affiche les données saisies par les utilisateurs. En cas de balises non fermées, ou de balises mal formées, elle génère une erreur. Vous pouvez utiliser l'option *allowable\_tags* pour spécifier les balises qui seront ignorées.

**string stripslashes(string str)**: Enlève les slash ajoutés par la fonction *addslashes()*, les doubles backslashes sont remplacés par des simples.

**int strlen(string str)**: Retourne la longueur de la chaîne *str*.

**int strpos(string str, string needle, [int offset])** : Retourne la position numérique de la première occurrence de *needle* dans la chaîne *str*. L'option *offset* vous permet de spécifier le caractère de début de recherche dans *str*. Cette position est toujours relative au début de la chaîne.

**string strrchr(string str, string needle)** : Cette fonction retourne la partie de la chaîne *str* qui commence à la dernière occurrence de *needle* et va jusqu'à la fin de la chaîne *str*.

**string strrev(string str)** : Retourne la chaîne *str* inversée.

**int strrpos(string str, char needle)** : Retourne la position numérique de la dernière occurrence du caractère *needle* dans la chaîne *str*.

**int strspn(string str, string mask)** : Retourne la longueur du premier segment de la chaîne *str* qui corresponde au masque donné *mask*.

**string strstr(string str, string needle)** : Retourne la chaîne *str* à partir de la première occurrence de *needle*, jusqu'à la fin.

**string strtok(string str, string delimit)** : Morcelle la chaîne *str* grâce au délimiteur *delimit*. Notez que seul le premier appel à *strtok()* nécessite les deux arguments. Tous les appels ultérieurs à *strtok()* ne nécessitent que le délimiteur *delimit*. Pour initialiser à nouveau *strtok()*, ou pour recommencer, fournissez à nouveau le paramètre *str*. La chaîne *str* sera découpée dès que l'un des caractères de *delimit* est trouvé. Exemple :

```
<?    echo "<pre>";
      $chaîne = "Ceci est    un exemple\ninteressant"; // les séparateurs sont : l'espace, la tabulation et le \n
      echo "$chaîne<br>----- la même chaîne, mais morcelée -----<br>";
      $tok = strtok($chaîne," \n");
      while ($tok)
      {
          echo "Mot = $tok<br>";
          $tok = strtok(" \n");
      }
      echo "</pre>";
?>
```

Résultat à l'affichage :

```
Ceci est    un exemple
interessant
----- la même chaîne, mais morcelée -----
Mot = Ceci
Mot = est
Mot = un
Mot = exemple
Mot = interessant
```

**string strtolower(string str)** : Retourne la chaîne *str* avec tous les caractères alphabétiques en minuscule.

**string strtoupper (string str)** : Retourne la chaîne *str* avec tous les caractères alphabétiques en majuscule.

**string str\_replace(string needle, string replace, string str)** : Remplace toutes les occurrences de *needle* dans *str* par la chaîne *replace*. Cette fonction est identique à la fonction *ereg\_replace()*.

**string strtr(string str, string from, string to)** : Cette fonction travaille sur *str*, remplaçant chaque occurrence de chaque caractère de la chaîne *from* correspondant à la chaîne *to* et retourne le résultat.

**string substr(string str, int start, [int length])** : Retourne une portion de la chaîne *str* spécifiée avec le début *start* et la longueur *length* facultative.

**string substr\_replace(string str, string replace, int start, [int length])** : Fonctionne exactement comme *substr()* et permet de remplacer la sous-chaîne extraite par la sous-chaîne *replace*.

**string trim(string str)** : Enlève les caractères blancs de début et de fin de la chaîne *str* et retourne la chaîne nettoyée. Les caractères blancs sont : "\n", "\r", "\t", "\v", "\0", et " " (espace).

**string ucfirst(string str)** : Met le premier caractère de la chaîne *str* en majuscule, si ce caractère est alphabétique. La chaîne modifiée est retournée.

**string ucwords(string str)** : Met le premier caractère de chaque mot de la chaîne *str* en majuscule si ce caractère est alphabétique. La chaîne modifiée est retournée.

## 69.1. addslashes

Retourne une chaîne avec un backslash devant chaque caractère qui en a besoin pour être inséré dans une requête de base de données. Ces caractères sont guillemets simples ('), guillemets doubles ("), backslash (\) et NUL (l'octet nul).

Dans l'exemple suivant, la balise HTML 'input' vous permet de saisir le nom d'un organisme (ex : Ecole Supérieure et d'Application des Transmissions), celui-ci sera stocké dans la variable PHP nommée *\$organisme* :

```
<input type = 'text' size = 50 name = 'organisme'>
```

La variable *\$organisme* contient dans cet exemple un guillemet simple qui marque, comme un guillemet double, le début et la fin d'une chaîne de caractères. Lorsque cette chaîne sera interprétée, le guillemet simple sera considéré comme marqueur de fin de chaîne, la partie de la chaîne qui suit ce marqueur (Application des Transmissions) ne sera donc pas pris en compte.

Pour éviter cela, il faut utiliser *addslashes* à toutes les variables chaînes de caractères qui sont susceptibles de contenir un des caractères mentionnés ci-dessus :

```
$organisme = addslashes($organisme) ;
```

```
// $organisme contient désormais "Ecole Supérieure et d'Application des Transmissions"
```

Si vous faites appel 2 fois consécutivement à la fonction *addslashes* sur une même variable, les caractères nécessitant le masquage par un backslash seront précédés de 2 backslashes.

## 69.2. chr

Retourne le caractère correspondant au code ASCII passé en argument.

Syntaxe :

```
string chr(int code_ascii) ;
```

Exemple :

```
$str .= chr(13); // ajoute un retour chariot à la fin de la chaîne $str
```

## 69.3. crypt

*crypt()* va coder une chaîne en utilisant la méthode de cryptage du DES standard.

Syntaxe :

```
string crypt(string str, [string salt]);
```

Les arguments sont : la chaîne à crypter, et un grain de sel qui servira de base pour le cryptage.

Si le grain de sel n'est pas fourni, il sera automatiquement généré par PHP.

Certains systèmes d'exploitation acceptent plus d'un type de cryptage. En fait, le DES standard est parfois remplacé par un cryptage MD5. Le type de cryptage est alors choisi en fonction du grain de sel. A l'installation, PHP détermine les possibilités de cryptage et décidera d'accepter d'autres grains de sel pour d'autres types de cryptage. Si le grain de sel n'est pas fourni, PHP générera alors un grain de 2 caractères, pour le DES standard, à moins que le système ne dispose de MD5 : dans ce cas, PHP générera un grain de sel pour MD5, par défaut. PHP affecte la variable d'environnement CRYPT\_SALT\_LENGTH à 2 s'il utilise le DES standard, et à 12 s'il utilise le MD5.

Le cryptage standard fournit le grain de sel dans les deux premiers octets du résultat de la fonction *crypt()*.

Sur les systèmes qui supportent plusieurs méthodes de cryptage, les variables d'environnement suivantes sont mises à 0 ou à 1, en fonction de la disponibilité de la méthode :

- CRYPT\_STD\_DES : DES standard avec 2 octets de SALT
- CRYPT\_EXT\_DES : DES étendu avec 9 octets de SALT
- CRYPT\_MD5 : MD5 avec 12 octets de SALT commençant à \$1\$
- CRYPT\_BLOWFISH : DES étendu avec 16 octets de SALT commençant à \$2\$

Il n'y a pas d'algorithme de décryptage, étant donné que *crypt()* est injective.

## 69.4. ereg

Recherche dans la chaîne *str* les séquences de caractères qui correspondent au masque donné *mask*.

Syntaxe :

```
int ereg(string mask, string str, [array regs]);
```

Si au moins une séquence est trouvée (éventuellement dans les parenthèses de *mask*), et que la fonction est appelée avec un troisième argument *regs*, les résultats seront enregistrés dans *regs*.

*\$regs[0]* contient une copie de la chaîne *str*.

*\$regs[1]* contiendra la capture désignée par la première parenthèse.

*\$regs[2]* contiendra la capture désignée par la deuxième parenthèse, et ainsi de suite.

La recherche est sensible à la casse.

*ereg()* retourne TRUE si une occurrence a été trouvée dans la chaîne, et FALSE dans le cas contraire, ou si une erreur est survenue.

Exemple :

```
$date = "2002-09-18";           // date sous forme 'année-mois-jour'
if ( ereg( "[0-9]{4})-([0-9]{1,2})-([0-9]{1,2})", $date, $regs ) )
// [0-9]{4} signifie les chiffres de 0 à 9 sur une longueur de 4
// [0-9]{1,2} signifie les chiffres de 0 à 9 sur une longueur de 1 ou 2
{
    echo "$regs[0]<br>";
    echo "$regs[3].$regs[2].$regs[1]";    //affichage de la date sous forme 'jour.mois.année'
}
else
    echo "Format de date invalide : $date";
```

Résultat à l'affichage :

2002-09-18

18.09.2002

## 69.5. ereg\_replace

Remplace toutes les occurrences de *needle* dans *str* par la chaîne *replace*.

Syntaxe : *string ereg\_replace(string needle, string replace, string str);*

Si aucune occurrence n'est trouvée, la chaîne *str* sera retournée intacte.

Exemple :

```
$chaine = "ceci est une chaîne exemple<br>voici ci démonstration d'un rempcicement";
echo ereg_replace("ci","la",$chaine);
```

Résultat à l'affichage :

cela est une chaîne exemple

voila la démonstration d'un remplacement

## 69.6. explode

Retourne un tableau qui contient les éléments de la chaîne, séparés par un séparateur.

Syntaxe :

```
array explode(string separator, string chaine);
```

Exemple :

```
$adresse_internet = "prenom.nom@nom_de_domaine ";
$tab = explode ('@',$adresse_internet) ;
echo $tab[0] ;           // affichera 'prenom.nom'
echo $tab[1] ;           // affichera 'nom_de_domaine'
```



## 69.7. implode

Regroupe tous les éléments du tableau *tab* dans une chaîne, avec la chaîne de jointure *glue*.

Syntaxe :

```
string implode(string glue, array tab);
```

Exemple :

```
$tab = array("un","deux","trois","quatre");  
$chaine = implode(" ; ", $tab);  
echo $chaine ; // affichera : 'un ; deux ; trois ; quatre'
```

## 69.8. nl2br

Convertit les retours à la ligne '\n' en balise HTML '<BR>'.

Syntaxe :

```
string nl2br(string str) ;
```

Exemple :

```
$chaine = "première ligne\ndeuxième ligne";  
echo $chaine ; // affichera 'première ligne deuxième ligne'  
echo nl2br($chaine) ;  
/* affichera :  
première ligne  
deuxième ligne  
*/
```

## 69.9. ord

Retourne la valeur ASCII du premier caractère de la chaîne *str*.

Syntaxe :

```
int ord(string str);
```

Exemple :

```
$chaine = "0123456";  
echo ord($chaine) ; // affichera '48'
```

## 69.10. parse\_str

Analyse la chaîne *str* comme si c'était une chaîne passée par URL, et affecte les variables qu'elle y trouve.

Syntaxe :

```
void parse_str(string str);
```

Exemple :

```
$str = "premier=valeur&second[]=Ceci+fonctionne&second[]=aussi";  
parse_str($str);  
echo $premier; // affichera 'valeur'  
echo $second[0]; // affichera 'Ceci fonctionne'  
echo $second[1]; // affichera 'aussi'
```

## 69.11. printf

Affiche les arguments en fonction du format.

Syntaxe :

`printf(string format, [mixed arguments]...);`

La chaîne de format est composée de 0 ou plusieurs directives : ce sont généralement des caractères qui sont recopiés tels quel (hormis %), et des spécifications de conversion, chacune d'elles disposant de son propre argument.

Chaque spécification de conversion débute avec le symbole '%' et comprend, dans l'ordre:

1. Une option de remplissage, qui indique quel caractère sera utilisé pour le remplissage, et la taille finale de la chaîne. Le caractère de remplissage peut être un espace ou le caractère zéro (0). La valeur par défaut est l'espace. Une autre valeur peut être spécifiée en la préfixant par un guillemet simple (''). Voir les exemples plus loin.
2. Un argument optionnel *alignment specifier* qui indique que le résultat doit être justifié à droite ou à gauche. Par défaut, il est justifié à gauche. Le caractère - signifie : justification à droite.
3. Argument optionnel, *width specifier* indique le nombre minimum de caractères que la conversion devrait retourner.
4. Argument optionnel, *precision specifier* indique le nombre de chiffres utilisés pour afficher un nombre à virgule flottante. Cette option n'a d'effet que sur les nombres à virgule, type *double*. (Une autre fonction pratique pour formater les nombres est : `number_format()`.)
5. *type specifier* indique le type de données passées en argument : Les types possibles sont :
  - % - un signe pourcentage : aucun argument nécessaire.
  - b - l'argument est traité comme un entier, et représenté comme un nombre binaire.
  - c - l'argument est traité comme un entier, et représenté comme un nombre ascii.
  - d - l'argument est traité comme un entier, et représenté comme un nombre décimal.
  - f - l'argument est traité comme un double, et représenté comme un nombre à virgule flottante.
  - o - l'argument est traité comme un entier, et représenté comme un nombre octal.
  - s - l'argument est traité tel quel, et représenté comme une chaîne.
  - x - l'argument est traité comme un entier, et représenté comme un nombre hexadécimal (en minuscule).
  - X - l'argument est traité comme un entier, et représenté comme un nombre hexadécimal (en majuscule).

Exemples :

Entiers :        \$i = 12        (b/ représente un espace)

Format	Affichage
<code>printf("i = %d", \$i);</code>	i = 12
<code>printf("i = %-5d", \$i);</code>	i = 12b/b/b/
<code>printf("i = %5d", \$i);</code>	i = b/b/b/12
<code>printf("i = %05d", \$i);</code>	i = 00012
<code>printf("i = %o", \$i);</code>	i = 14
<code>printf("i = %u", \$i);</code>	i = 12
<code>printf("i = %x", \$i);</code>	i = c
<code>printf("i = %X", \$i);</code>	i = C

\$i = -12

Format	Affichage
printf("i = %d", \$i);	i = -12
printf("i = %o", \$i);	i = 177764
printf("i = %u", \$i);	i = 65524
printf("i = %x", \$i);	i = fff4
printf("i = %X", \$i);	i = FFF4

Réels :        \$i = 1.414

Format	Affichage
printf("i = %f", \$i);	i = 1.414000
printf("i = %3.2f", \$i);	i = 1.41
printf("i = %3f", \$i);	i = 1.414000
printf("i = %.2f", \$i);	i = 1.41
printf("i = %e", \$i);	i = 1.41400e+00
printf("i = %E", \$i);	i = 1.41400E+00
printf("i = %g", \$i);	i = 1.414

Caractères :    \$i = 'A'

Format	Affichage
printf("i = %c", \$i);	i = A
printf("i = %3c", \$i);	i = b/b/A
printf("i = %-3c", \$i);	i = Ab/b/
printf("i = %x", \$i);	i = 41

Divers :

```
printf("carre de a = %d", $a*$a);
printf("le résultat de fonction1 = %d", fonction1(10));
printf("a>b renvoie %d", $a>$b);
```

Pour afficher un backslash '\' ou des guillemets '"', il faut les masquer par un backslash.

Exemple:

```
printf("ce caractère '\\' est un \"backslash\"\\n"); // affichera : ce caractère '\\' est un "backslash"
```

Pour afficher un pourcent '%' dans une chaîne format contenant au moins un format il faut le masquer par un pourcent '%'

## 69.12. rawurldecode

Retourne une chaîne dont les séquences de caractères %xy, avec xy deux valeurs hexadécimales, auront été remplacées par le caractère ascii correspondant.

Syntaxe :

```
string rawurldecode(string str);
```

Exemple :

```
$chaîne = "adresse%20Internet%20%3A%20nom.prenom%40nom_de_domaine";  
echo rawurldecode($chaîne); // affichera 'adresse Internet : nom.prenom@nom_de_domaine'
```

## 69.13. rawurlencode

Retourne une chaîne dont tous les caractères non-alpha-numériques (hormis – et \_) auront été remplacés par des séquences %xy, avec xy deux valeurs hexadécimales. Ce codage est conforme à la RFC1738 qui évite que les caractères spéciaux soient interprétés comme des délimiteurs.

Cette fonction est pratique pour transmettre des informations via une URL. C'est aussi un moyen de passer des informations d'une page à l'autre.

Syntaxe :

```
string rawurlencode(string str);
```

Exemple :

```
$chaîne = "adresse Internet : nom.prenom@nom_de_domaine";  
echo rawurlencode($chaîne); // affichera 'adresse%20Internet%20%3A%20nom.prenom%40nom_de_domaine'
```

## 69.14. strchr

Retourne toute la chaîne *str* à partir de la première occurrence de *needle*, jusqu'à la fin. Cette fonction est un alias de la fonction *strstr()*.

Syntaxe :

```
string strchr(string str, string needle);
```

Exemple :

```
$chaîne = "adresse Internet : nom.prenom@nom_de_domaine";  
echo "Email ".strchr($chaîne,":")."<br>"; // affichera 'Email : nom.prenom@nom_de_domaine'
```

## 69.15. strcspn

Retourne la longueur du premier segment de la chaîne *str* qui ne corresponde à aucun des caractères de la chaîne *masque*.

Syntaxe :

```
int strcspn(string str, string masque);
```

Exemple :

```
$chaîne = "adresse Internet : nom.prenom@nom_de_domaine";  
echo strcspn($chaîne,":")."<br>"; // affichera '17'
```

## 69.16. strip\_tags

Enlève les balises HTML et PHP de la chaîne *str*. En cas de balises non fermées, ou de balises mal formées, elle génère une erreur. Vous pouvez utiliser l'option *allowable\_tags* pour spécifier les balises qui seront ignorées.

Syntaxe : *string strip\_tags(string str, [string allowable\_tags]);*

Exemple :

```
$chaine = "adresse Internet : nom.prenom@nom_de_domaine<br>";  
echo $chaine;  
echo strip_tags($chaine);  
echo "----<br>";  
echo strip_tags($chaine,"<br>");      // enlève les balises HTML et PHP sauf la balise <br>  
echo "----";
```

Résultat à l'affichage :

```
adresse Internet : nom.prenom@nom_de_domaine  
adresse Internet : nom.prenom@nom_de_domaine----  
adresse Internet : nom.prenom@nom_de_domaine  
----
```

## 69.17. strpos

Retourne la position numérique de la première occurrence du caractère *needle* dans la chaîne *str*.

Syntaxe : *int strpos(string str, char needle, [int offset]) ;*

Contrairement à la fonction *strrpos()*, cette fonction peut prendre une chaîne complète comme paramètre.

Si *needle* n'est pas trouvé, elle retourne faux (false).

L'option *offset* vous permet de spécifier le caractère de début de recherche dans *str* . Cette position est toujours relative au début de la chaîne.

Exemple :

```
$chaine = "nom.prenom@nom_de_domaine<br>";  
echo strpos($chaine,"nom",3);          // affichera '7'
```

## 69.18. strrchr

Cette fonction retourne la partie de la chaîne *str* qui commence à la dernière occurrence de *needle* et va jusqu'à la fin de la chaîne *str*.

Syntaxe : *string strrchr(string str, char needle);*

La fonction retourne faux (false) si *needle* n'est pas trouvé.

Si *needle* contient plus d'un caractère, les autres sont ignorés.

Exemple :

```
$chaine = "nom.prenom@nom_de_domaine<br>";  
echo strrchr($chaine,"@");           // affichera '@nom_de_domaine'
```

## 69.19. strrpos

Retourne la position numérique de la dernière occurrence du caractère *needle* dans la chaîne *str*.

Syntaxe : *int strrpos(string str, char needle) ;*

Cette fonction ne peut accepter qu'un seul caractère.

Si *needle* n'est pas trouvé, elle retourne faux (false).

Exemple :

```
$chaine = "nom.prenom@nom_de_domaine<br>";  
echo strrpos($chaine,'o');          // affichera '19'
```

## 69.20. strstr

Retourne la longueur du premier segment de chaîne qui corresponde au masque donné.

Syntaxe :

```
int strstr(string str, string masque);
```

Exemple :

```
$chaine = "nom.prenom@nom_de_domaine<br>";  
echo strstr($chaine,"nom_");           // affichera '3'
```

## 69.21. strstr

Retourne la chaîne *str* à partir de la première occurrence de *needle*, jusqu'à la fin.

Syntaxe :

```
string strstr(string str, string needle);
```

Si *needle* n'est pas trouvé, la fonction retourne faux (FALSE).

Exemple :

```
$chaine = "nom.prenom@nom_de_domaine<br>";  
echo strstr($chaine,"nom_");           // affichera 'nom_de_domaine'
```

## 69.22. strtok

Morcelle la chaîne *str* grâce au délimiteur *delimit*.

Syntaxe :

```
string strtok(string str, string delimit);
```

Seul le premier appel à *strtok()* utilise l'argument chaîne *str*. Après, chaque appel à *strtok()* ne requiert que le délimiteur à utiliser.

Pour recommencer un morcellement, vous devez appeler *strtok()* avec un nouvel argument *str* et un délimiteur. *strtok()* accepte des délimiteurs multiples. La chaîne sera morcelée dès qu'elle rencontrera un des délimiteurs.

Un délimiteur égal à "0" sera confondu avec FALSE.

Exemple :

```
$chaine = "ceci est une chaîne exemple";  
$tok = strtok($chaine," ");  
while($tok)  
{  
    echo "mot = $tok<br>";  
    $tok = strtok(" ");  
}
```

Résultat à l'affichage :

```
mot = ceci  
mot = est  
mot = une  
mot = chaîne  
mot = exemple
```

## 69.23. strstr

Cette fonction travaille sur *str*, remplaçant chaque occurrence de chaque caractère de la chaîne *from* correspondant à la chaîne *to* et retourne le résultat.

Syntaxe :

```
string strstr(string str, string from, string to);
```

Si *from* et *to* sont de longueur différentes, les caractères en trop sont ignorés.

Exemple :

```
$chaine = "Loïc fête son anniversaire au château où régna fantômeI. " ;  
echo strstr($chaine, "ääääëêïïoù", "aaeeeiiou");
```

Résultat à l'affichage :

```
Loïc fête son anniversaire au chateau ou regna fantomeI.
```

Depuis PHP4, *strstr()* peut aussi être appelée avec deux arguments. Dans ce cas, elle se comporte différemment : *from* doit être un tableau associatif contenant des paires de chaînes, qui seront remplacées dans la chaîne source. *strstr()* recherchera toujours la chaîne la plus longue, et la remplacera en premier. Elle ne remplacera jamais une portion de chaîne qu'elle a déjà remplacée.

## 69.24. substr

Retourne une portion de la chaîne *str* spécifiée avec le début *start* et la longueur *length* facultative.

Syntaxe :

```
string substr(string str, int start, [int length]);
```

Si *start* est positif, la chaîne retournée commencera au caractère *start* de la chaîne *str*.

Le début de la chaîne est obtenu avec *start* = 0. Exemple :

```
$rest = substr("abcdef", 1); // retourne "bcdef"
```

Si *start* est négatif, la chaîne retournée commencera au caractère *start* de la chaîne *str*, en partant de la fin.

Exemples :

```
$rest = substr("abcdef", -1); // retourne "f"  
$rest = substr("abcdef", -2); // retourne "ef"
```

Si *length* est donné, et positif, la chaîne retournée aura la longueur *length*. Exemples :

```
$rest = substr("abcdef", 1, 3); // retourne "bcd"  
$rest = substr("abcdef", -3, 1); // retourne "d"
```

Si *length* est donné, et négatif, la chaîne retournée aura la longueur de la chaîne à partir de *start* et jusqu'à *length* caractères de la fin de la chaîne. Exemple : *\$rest = substr("abcdef", 1, -1); // retourne "bcde"*

## 69.25. substr\_replace

Fonctionne exactement comme *substr()* et permet de remplacer la sous-chaîne extraite par la sous-chaîne *replace*.

Syntaxe :

```
string substr_replace(string str, string replace, int start, [int length]);
```

Exemple :

```
$num_adherent = "0001" ;  
$chaine = "ESAT0000CSA" ;  
echo substr_replace($chaine,$num_adherent,4,4) ; // affichera 'ESAT0001CSA'
```

## 9. La gestion des dates

PHP gère les dates sous la forme d'un entier long appelé horodatage Unix, qui représente le nombre de secondes écoulées depuis le 1<sup>er</sup> janvier 1970 à minuit. Il s'agit de l'unique format de date reconnu par PHP.

Tous les stockages, les opérations et les conversions qui portent sur les dates doivent donc se conformer à ce format.

### 70. Calculs de durées

Tous les calculs de durées se font à partir de ce nombre de secondes écoulées depuis le 1<sup>er</sup> janvier 1970. C'est un entier, et il se manipule comme tel. Ainsi, pour calculer une durée, il suffit de retrancher l'entier long qui représente la date la plus ancienne de celui qui représente la date la plus récente.

PHP est pourvu de tous les outils nécessaires à la conversion de ces entiers longs en dates lisibles. En revanche, il n'existe aucune fonction intégrée qui permette de convertir un nombre de secondes en une durée découpée en années, mois, jours, heures, minutes et secondes.

### 71. Opérations sur les dates

Les opérations spécifiques aux dates consistent à générer un entier long, qui représente une date définie, ainsi qu'une fonction de contrôle de validité.

PHP fournit les fonctions *time()*, *mktime()* et *gmmktime()* pour générer l'entier long, et la fonction *checkdate()* pour vérifier la validité des dates.

#### 71.1. time

*time()* retourne l'entier long qui représente la date courante.

Exemple :

```
echo time(); // affichera 1032784180 le 23 septembre 2002 à 14H29 et quelques secondes
```

#### 71.2. mktime

*mktime()* retourne l'entier long qui correspond à une date donnée passée en paramètre sous la forme : heures,minutes,secondes,mois,jour,année[, heure d'été (0) d'hiver (1) inconnu (-1)].

Si l'heure n'a pas d'importance, vous pouvez passer la valeur 0 pour chacun des trois premiers paramètres.

Si vous ne passez aucun paramètre, la fonction renvoie l'horodatage Unix correspondant à la date et l'heure courantes.

Exemple :

```
echo mktime(14,30,0,9,23,2002); // affichera 1032784200
```

#### 71.3. gmmktime

*gmmktime()* est semblable à *mktime()*, mais retourne l'entier qui correspond à l'heure GMT.

#### 71.4. checkdate

*checkdate()* retourne un booléen qui indique si la date passée en paramètre sous la forme mois,jour,année est valide ou non.

Les années bissextiles sont prises en compte. Ainsi on pourra facilement générer un calendrier perpétuel à l'aide de cette fonction. Exemple :

```
checkdate(2,29,1999); // retournera faux car 1999 n'est pas bissextile
```

```
checkdate(2,29,2000); // retournera vrai car 2000 est bissextile
```



## 72. Conversions de dates

Les dates sont converties, du format entier long vers un format lisible, principalement par la fonction `date()`.

### 72.1. date

`date()` convertit une date du format horodatage Unix (entier long) vers un format lisible. Elle retourne une chaîne de caractères mise en forme représentant une date. Si l'horodatage Unix (entier long) est omis, la date renvoyée est la date du jour.

**Syntaxe** : `string date(format,[ horodatage Unix]) ;`

Vous pouvez contrôler de façon précise la chaîne renvoyée par `date()` à l'aide d'un argument chaîne (le format) que vous devez lui transmettre.

La chaîne format est composée de codes et d'autres données qui seront recopiées dans le résultat renvoyé (ces données servent principalement de séparateurs).

La liste des codes pouvant être contenus dans une chaîne format est la suivante :

Code format	Description du résultat restitué	Exemple restitué
a	"am" ou "pm" en minuscules	pm
A	"AM" ou "PM" en majuscules	PM
d	jour dans le mois précédé d'un zéro (01-31)	09
D	jour de la semaine en trois lettres	Mon
F	nom du mois de l'année	September
g	heure (sur 12 heures non précédée d'un zéro)	3
G	heure (sur 24 heures non précédée d'un zéro)	15
h	heure (sur 12 heures précédée d'un zéro)	03
H	heure (sur 24 heures précédée d'un zéro)	15
i	minutes précédées d'un zéro (00-59)	05
I	booléen ('1' si l'heure d'hiver est activée, '0' sinon)	1
j	jour dans le mois non précédé d'un zéro (1-31)	9
l	jour de la semaine	Monday
L	année bissextile ('1' pour vrai, '0' pour faux)	0
m	mois de l'année précédé d'un zéro (01-12)	09
M	mois de l'année en trois lettres	Sep
n	mois de l'année non précédé d'un zéro (1-12)	9
s	secondes précédées d'un zéro (00-59)	07
t	nombre de jours dans le mois donné (28-31)	30
T	fuseau horaire du serveur	Paris, Madrid
U	horodatage Unix (entier long)	1032790097
w	jour de la semaine (0 pour dimanche – 6 pour samedi)	1
y	année sur deux chiffres	02
Y	année sur quatre chiffres	2002
z	jour de l'année (quantième)	251
Z	temps en secondes écoulé depuis GMT (décalage horaire -43200 - 43200)	7200

Lorsque des dates doivent être communiquées de PHP vers MySQL, la fonction `date()` permet facilement d'obtenir le format approprié. Il faut simplement utiliser les formats de jour et de mois sur deux chiffres (préfixés d'un zéro).

**Exemple** :

```
echo "nous sommes le ".date("d/m/Y").", et il est ".date("H : i : s");
// affichera : nous sommes le 23/09/2002, et il est 16 : 25 : 56
```

## 72.2. getdate

`getdate()` convertit également la date avant de la présenter à l'utilisateur. Cette fonction accepte en option un horodatage Unix (entier long) et renvoie un tableau associatif contenant les informations relatives à la date. Si vous exécutez cette fonction sans argument, elle travaille avec l'horodatage Unix courant, tel qu'il serait renvoyé par `time()`.

Syntaxe :

`array getdate([long horodatage Unix]) ;`

Le tableau associatif renvoyé par `getdate()` est le suivant :

Clé	Description	Exemple
seconds	secondes ayant dépassées la minute (0-59)	7
minutes	minutes ayant dépassées l'heure (0-59)	5
hours	heures de la journée (0-23)	16
mday	jour dans le mois (1-31)	23
wday	jour de la semaine (0 pour dimanche – 6 pour samedi)	1
mon	mois de l'année (1-12)	9
year	année sur quatre chiffres	2002
yday	jour de l'année (quantième : 0-365)	265
weekday	nom du jour de la semaine	Monday
month	nom du mois de l'année	September
0	horodatage Unix	1032790097

Exemple :

```
$tab_date = array();
$tab_date = getdate();           // aucun argument n'a été transmis, la date du jour sera donc utilisée
echo "Nous sommes le : $tab_date[mday]/$tab_date[mon]/$tab_date[year] " ;
// affichera : Nous sommes le : 23/9/2002
```

## 72.3. strftime

La fonction `strftime()` fonctionne comme la fonction `date()`. Les codes de la chaîne format sont cependant composés du caractère '%' suivi d'une lettre.

Les 4 codes qui nous restituent les noms du jour et du mois dans la langue donnée par `setlocale()` sont les suivants :

- %a - nom abrégé du jour de la semaine (local).
- %A - nom complet du jour de la semaine (local).
- %b - nom abrégé du mois (local).
- %B - nom complet du mois (local).

## 73. Détermination de la date de dernière modification d'un script

Il est généralement très apprécié d'ajouter une date de dernière modification sur chacune des pages d'un site Web.

La date de dernière modification d'un script peut être connue à l'aide de la fonction `getlastmod()`.

Syntaxe :

`int getlastmod(void) ;`

La fonction `getlastmod()` renvoie un horodatage Unix, qui peut être mis en forme au moyen de la fonction `date()` pour obtenir une date plus compréhensible.

Exemple :

```
echo "Date de dernière modification : ".date("g:i a, j M Y",getlastmod());
```

## 74. Conversion des dates aux formats MySQL

Lorsque des dates doivent être communiquées de PHP vers MySQL, la fonction `date()` vue plus haut permet facilement d'obtenir le format approprié. Il faut simplement utiliser les formats de jour et de mois pour lesquels des zéros sont placés en préfixe, de manière à éviter des confusions au niveau de MySQL.

Dans MySQL, deux fonctions permettent de formater les dates : `DATE_FORMAT()` et `UNIX_TIMESTAMP()`.

La fonction `DATE_FORMAT()` est semblable à la fonction `date()` de PHP, si ce n'est qu'elle utilise des codes de format différents. La conversion la plus courante consiste à exprimer une date sous le format MM-JJ-AAAA, plutôt que sous le format AAAA-MM-JJ en vigueur dans MySQL. Cette conversion s'effectue simplement au moyen de la requête suivante :

```
SELECT DATE_FORMAT(date_column, '%m %d %Y') FROM tablename ;
```

Le code de format `%m` correspond à une représentation du mois sous la forme d'un nombre à deux chiffres, `%d` représente le jour sous la forme d'un nombre à deux chiffres, et `%Y` représente l'année sous la forme d'un nombre à quatre chiffres. Les codes de format MySQL les plus utiles pour ce type de conversion sont les suivants :

Code	Description
<code>%M</code>	Mois, plein texte
<code>%W</code>	Nom du jour de la semaine, plein texte
<code>%D</code>	Jour du mois, numérique, avec un préfixe textuel (par exemple Ist)
<code>%Y</code>	Année, numérique, sur 4 chiffres
<code>%y</code>	Année, numérique, sur 2 chiffres
<code>%a</code>	Nom du jour de la semaine, sur 3 lettres
<code>%d</code>	Jour du mois, numérique, zéro en préfixe
<code>%e</code>	Jour du mois, numérique, pas de zéro en préfixe
<code>%m</code>	Mois, numérique, zéro en préfixe
<code>%c</code>	Mois, numérique, pas de zéro en préfixe
<code>%b</code>	Mois, texte, sur 3 lettres
<code>%j</code>	Jour de l'année, numérique
<code>%H</code>	Heure, système à 24 heures, zéro en préfixe
<code>%k</code>	Heure, système à 24 heures, pas de zéro en préfixe
<code>%h</code> ou <code>%I</code>	Heure, système à 12 heures, zéro en préfixe
<code>%l</code>	Heure, système à 12 heures, pas de zéro en préfixe
<code>%i</code>	Minutes, numérique, zéro en préfixe
<code>%r</code>	Heure, système à 12 heures (hh:mm:ss[AM PM])
<code>%T</code>	Heure, système à 24 heures (hh:mm:ss)
<code>%S</code> ou <code>%s</code>	Secondes, numérique, zéro en préfixe
<code>%p</code>	AM ou PM
<code>%w</code>	Jour de la semaine, numérique, de 0 (dimanche) à 6 (samedi)

La fonction `UNIX_TIMESTAMP()` est comparable, si ce n'est qu'elle convertit une colonne en un horodatage Unix.

Exemple :

```
SELECT UNIX_TIMESTAMP(date_column) FROM tablename ;
```

Cette requête renvoie la date exprimée sous la forme d'un horodatage Unix. Vous pouvez ensuite la manipuler à votre gré dans un script PHP.

En règle générale, l'usage d'horodatages UNIX est préférable dans les calculs de dates, tandis que le format de date standard s'emploie lorsqu'il s'agit simplement de stocker ou d'afficher des dates. L'implémentation des calculs et des comparaisons de dates se révèle plus facile avec les horodatages Unix.

## 75. Exercice pratique

Affichez le nom du jour en français, à partir d'une date sélectionnée.

Dans un premier formulaire nommé 'selection\_date.php3', vous sélectionnerez une date à l'aide de listes déroulantes pour le jour (1-31), le mois (1-12) et l'année (année courante - 10 - année courante + 10). La date du jour sera affichée implicitement.

Vous créerez trois boutons 'Valider', 'Annuler' et 'Quitter'.

- Le bouton ‘Valider’ fera appel à un deuxième formulaire nommé ‘affiche\_jour.php3’, qui contrôlera la validité de la date sélectionnée, et affichera le nom du jour. Dans ce deuxième formulaire, un bouton ‘Ressaisir’ renverra l'utilisateur au premier formulaire, un bouton ‘Quitter’ entraînera la fermeture de la fenêtre et du navigateur.
- Le bouton ‘Annuler’ annulera la sélection et réaffichera la date du jour.
- Le bouton ‘Quitter’ entraînera la fermeture de la fenêtre et du navigateur.

Entrez la date 25 9 2002 Le 25/9/2002 est un mercredi

Valider Annuler Quitter Ressaisir Quitter

Formulaire 'selection\_date.php3' :

```
<html>  
<head>  
<title>Sélection d'une date</title>  
</head>  
<body>  
<?php  
    echo "<form method='post' action='affiche_jour.php3'>";  
    $date = getdate();  
    $annee_en_cours = $date['year'];  
    $mois_en_cours = $date['mon'];  
    $jour_en_cours = $date['mday'];  
  
    echo "<br><br>";  
    echo "<div align='center'>";  
    echo "<b>Entrez la date</b>";  
    echo "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~";  
  
    echo "<select name = 'jour'>";  
    echo "<option SELECTED>" . $jour_en_cours . "</option>";  
  
    for( $jj = 1 ; $jj <= 31 ; $jj++ )  
        echo "<option>" . $jj . "</option>";  
  
    echo "</select>";  
    echo "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~";  
  
    echo "<select name = 'mois'>";  
    echo "<option SELECTED>" . $mois_en_cours . "</option>";  
  
    for( $mm = 1 ; $mm <= 12 ; $mm++ )  
        echo "<option>" . $mm . "</option>";  
  
    echo "</select>";  
    echo "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~";  
  
    echo "<select name = 'annee'>";  
    echo "<option SELECTED>" . $annee_en_cours . "</option>";
```

```
for( $aaaa = $annee_en_cours - 10 ; $aaaa <= $annee_en_cours + 10 ; $aaaa++ )  
    echo "<option>". $aaaa . "</option>";  
  
echo "</select>";  
echo "<br><br>";  
echo "<input type='submit' name = 'valider' value='Valider'>";  
echo "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~";  
echo "<input type='reset' name = 'annuler' value='Annuler'>";  
echo "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~";  
echo "<input type='button' name='quitter' value='Quitter' OnClick='parent.close();'>";  
echo "</div>";  
echo "</form>";  
  
?>  
</body>  
</html>
```

Formulaire 'affiche\_jour.php3' :

```
<html>
<head>
<title>Affichage du jour</title>
</head>
<body>
<?
    echo "<font method='post' action='#>";
    $jour = $_HTTP_POST_VARS["jour"];
    $mois = $_HTTP_POST_VARS["mois"];
    $annee = $_HTTP_POST_VARS["annee"];

    $tab_jours = array("0"=> "dimanche", "1"=> "lundi", "2"=> "mardi", "3"=> "mercredi",
        "4"=> "jeudi", "5"=> "vendredi", "6"=> "samedi");
    echo "<br><br>";
    echo "<div align='center'>";

    if(checkdate($mois,$jour,$annee))
    {
        $horodatage = mktime(0,0,0,$mois,$jour,$annee);
        $nom_jour = $tab_jours[date("w",$horodatage)];
        echo "Le $jour/$mois/$annee est un <font color='blue'
            size='5'>$nom_jour</font><br><br>";
    }
    else
    {
        echo "<font color='red' size ='4'>";
        echo "< Date erron&eacute;e >";
        echo "</font><br><br>";
    }


    echo "<input type='button' name='ressaisir' value='Ressaisir'
    OnClick=\"document.location='selection_date.php3'\">>";
    echo "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~";
    echo "<input type='button' name='quitter' value='Quitter' OnClick='parent.close();'>";
    echo "</div>";
    echo "</form>";

?>
</body>
</html>
```

Autre possibilité, sans le tableau des jours en français, mais en utilisant la fonction *setlocale()* qui change les informations locales.

Syntaxe :

*setlocale(string categorie, string langue\_locale);*

*categorie* est une chaîne qui spécifie la catégorie de fonction qui va être affectée par les informations locales :

- LC\_ALL                    pour toutes les fonctions ci-dessous
- LC\_COLLATE            pour les comparaisons de chaînes (en cours d'implémentation)
- LC\_CTYPE               pour la classification de caractères et les conversions, par exemple strtoupper()
- LC\_MONETARY          pour localeconv() (en cours d'implémentation)
- LC\_NUMERIC            pour les séparateurs décimaux
- LC\_TIME                pour le format des dates et heures avec strftime()

```
setlocale(LC_ALL,"FR");
if(checkdate($mois,$jour,$annee))
{
    $horodatage = mktime(0,0,0,$mois,$jour,$annee);
    $nom_jour = strftime("%A",$horodatage);
    echo "Le $jour/$mois/$annee est un <font color='blue' size='5'>$nom_jour</font><br><br>";
}
```

## 10. Les objets

Un objet est un ensemble de variables et de fonctions entre accolades conçues depuis un modèle spécial nommé classe.

Les objets cachent la majorité de leurs rouages internes au code qui les utilise, fournissant des interfaces simples par le biais desquelles vous pouvez leur envoyer des ordres et recevoir des informations de leur part.

Ces interfaces sont des fonctions particulières nommées méthodes. Toutes les méthodes d'un objet ont accès à des variables spéciales nommées propriétés.

En définissant une classe, vous fixez un ensemble de caractéristiques. En créant des objets de ce type, vous créez des entités qui partagent ces caractéristiques, mais qui sont susceptibles de les initialiser sous différentes valeurs.

Vous pouvez créer une classe **automobile**, par exemple. Cette classe pourrait posséder une caractéristique **couleur**. Tous les objets **automobile** partageraient la caractéristique **couleur**, mais certains l'initialiseraient en tant que 'bleu', d'autres en tant que 'vert' et ainsi de suite.

### Définitions :

- Une **classe** est un ensemble de fonctions spéciales nommées méthodes et de variables particulières nommées propriétés (ou attributs). Vous pouvez déclarer une classe à l'aide du mot clé *class*. Les classes sont les modèles à partir desquels sont créés les objets.
- Un **objet** est une instance de classe. Cela signifie qu'un objet est l'incarnation de la fonctionnalité fixée dans une classe. Un objet est instancié en utilisant l'instruction *new* conjointement au nom de la classe de laquelle il est membre. Lorsqu'un objet est instancié, vous pouvez accéder à toutes ses propriétés ainsi qu'à toutes ses méthodes.

Le plus grand atout du code orienté objet est peut-être sa possibilité de réutilisation. Les classes employées pour créer des objets étant autonomes, elles peuvent facilement être retirées d'un projet pour être employées dans un autre. De plus, il est possible de créer des classes 'enfant', qui héritent et prennent la priorité sur les caractéristiques de leurs 'parents'. Cette technique nécessite que les classes doivent donc être définies selon le principe des couches successives, par empilage des classes de haut niveau sur les classes de bas niveau, comme pour les fonctions.

La programmation objet consiste donc à écrire des classes et à les utiliser. Une bibliothèque de composants est un ensemble de fichiers qui contiennent des définitions de classes, que l'on peut inclure en tête des fichiers qui réutilisent ces classes.

## 76. Création d'un objet

Pour créer un objet, vous devez auparavant concevoir le modèle à partir duquel il pourra être instancié. Ce modèle est connu sous le nom de classe, et doit être déclaré avec le mot clé *class*.

Créer un objet (ou déclarer l'instance d'une classe, ou instancier un objet) se réalise à l'aide du mot clé *new*.

Dans l'exemple qui suit, nous allons construire une classe nommée *premiere\_classe* et créer deux objets *\$obj1* et *\$obj2* :

```
class premiere_classe
{
    // emplacement des propriétés et méthodes
}

$obj1 = new premiere_classe(); // instanciation de l'objet $obj1
$obj2 = new premiere_classe(); // instanciation de l'objet $obj2
```

L'instanciation d'un objet signifie la récupération d'un pointeur sur une zone mémoire disponible, qui est de la taille de l'objet et peut donc être utilisée pour en gérer le contenu dynamique. Elle gère donc l'allocation de la mémoire ainsi que la mise à disposition d'une variable PHP qui permet de manipuler une donnée du format de la classe.

## 77. Les propriétés d'un objet

Les objets peuvent accéder à des variables spéciales nommées propriétés ou attributs et créées à l'aide du mot clé *var*. Elles peuvent être déclarées à tout endroit du corps de la classe, et leur visibilité s'étend sur l'ensemble du code qui compose la classe, mais dans un souci de clarté, il est préférable de les définir au début.

Une propriété peut être une valeur, un tableau ou même un autre objet.

Exemple :

```
class premiere_classe
```

```
{  
    var $nom ;  
}
```

```
$obj1 = new premiere_classe() ;
```

```
$obj2 = new premiere_classe() ;
```

La propriété *\$nom* a été déclarée à l'aide du mot clé *var*. Cela est essentiel dans le contexte d'une classe. Vous obtiendrez une erreur d'analyse si vous l'oubliez.

Vous pouvez accéder à cette propriété au travers de l'objet.

L'opérateur '->' (les deux caractères '-' et '>' forment une flèche appelée aussi accesseur) permet d'accéder aux propriétés d'un objet ou de les modifier.

Exemple :

```
$obj1->nom = "DUPONT" ;           // initialisation de la propriété $nom
```

En PHP, le programmeur ne peut définir aucun niveau de confidentialité pour les propriétés et méthodes (il n'y a donc pas de concept d'encapsulation).

Vous pouvez accéder à tous les champs d'un objet, ce qui cause des problèmes si une propriété n'est pas destinée à être modifiée.



## 78. Les méthodes d'un objet

Une méthode est une fonction définie dans une classe. Chaque objet instancié depuis la classe possédera la fonctionnalité de la méthode.

Vous pouvez accéder à la méthode au travers de l'objet grâce à l'opérateur '->'

L'exemple suivant ajoute une méthode à la classe *premiere\_classe* et crée un objet *\$obj1* instancié depuis cette classe :

```
class premiere_classe
{
    var $nom;
    function aff_hello()
    {
        print("Hello !");
    }
}
$obj1 = new premiere_classe() ;
$obj1->aff_hello();           // affichera 'Hello !'
```

Les méthodes possèdent un accès aux variables (propriétés) membres de classe.

Une classe utilise une variable spéciale nommée *\$this* pour faire référence à l'objet en cours d'instanciation. Par le biais de cette variable, un objet peut donc faire référence à lui-même.

En combinant la variable *\$this* et l'opérateur ->, vous pouvez accéder à toute propriété ou méthode d'une classe depuis la classe elle-même.

Exemple :

```
class premiere_classe
{
    var $prenom ;
    function aff_hello()
    {
        print("Hello $this->prenom !") ;
    }
}

$obj1 = new premiere_classe() ;
$obj1->prenom = "Florian";
$obj1->aff_hello();           // affichera 'Hello Florian !'
```

Vous pouvez également transmettre des arguments à la méthode de la même façon que vous le feriez avec une fonction.

Exemple :

```
class premiere_classe
{
    var $prenom ;
    var $nom ;
    function init_nom($name)      // l'argument aurait pu également être nommé $nom (variable locale)
    {
        $this->nom = $name ;
    }
    function aff_hello()
    {
        print("Hello $this->prenom $this->nom !") ;
    }
}

$obj1 = new premiere_classe() ;
$obj1->prenom = "Florian";
$obj1->init_nom("DUPONT") ;
$obj1->aff_hello();           // affichera 'Hello Florian DUPONT !'
```

Si vous créez une méthode dont le nom est identique à la classe *premiere\_classe*, elle sera automatiquement appelée lors de l'instanciation d'un nouvel objet. De cette façon, vous pouvez donner à vos objets des arguments à traiter au moment où vous les instanciez.

Les objets peuvent exécuter du code pour s'initialiser eux-mêmes en fonction d'arguments ou d'autres facteurs. Ces méthodes spéciales sont nommées constructeurs. Il s'agit d'une norme de programmation qui s'est imposée au fil du temps. Le constructeur doit être considéré non seulement en tant qu'étape d'initialisation des variables, mais également en tant que première phase de contrôle et d'harmonisation des données disponibles aux traitements de la classe.

L'exemple suivant ajoute un constructeur à la classe *premiere\_classe* :

```
class premiere_classe
{
    var $prenom ;
    var $nom ;
    function premiere_classe ($name)
    {
        $this->nom = $name ;
    }
    function aff_hello()
    {
        print("Hello $this->prenom $this->nom !") ;
    }
}

$obj1 = new premiere_classe("DUPONT") ; // initialisation de la propriété $nom
$obj1->prenom = "Florian";
$obj1->aff_hello(); // affichera 'Hello Florian DUPONT !'
```

Dans l'exemple suivant, nous établissons un argument par défaut au constructeur, de sorte que la chaîne "DUPONT" soit attribuée au paramètre si nous n'incluons pas d'argument lors de la création de l'objet :

```
class premiere_classe
{
    var $prenom ;
    var $nom ;
    function premiere_classe ($name = "DUPONT")
    {
        $this->nom = $name ;
    }
    function aff_hello()
    {
        print("Hello $this->prenom $this->nom !<br>") ;
    }
}

$obj1 = new premiere_classe() ;
$obj1->prenom = "Florian";
$obj1->aff_hello(); // affichera 'Hello Florian DUPONT !'
$obj2 = new premiere_classe("DURAND") ;
$obj2->prenom = "François";
$obj2->aff_hello(); // affichera 'Hello François DURAND !'
```

## 79. L'héritage

L'héritage est un principe propre à la programmation orientée objet, permettant de créer une nouvelle classe à partir d'une classe existante. Le nom d'"*héritage*" (pouvant parfois être appelé *dérivation de classe*) provient du fait que la classe dérivée (la classe nouvellement créée) contient les propriétés et les méthodes de sa super classe (la classe dont elle dérive).

L'intérêt majeur de l'héritage est de pouvoir définir de nouvelles propriétés et de nouvelles méthodes pour la classe dérivée, qui viennent s'ajouter à celles héritées.

Par ce moyen on crée une hiérarchie de classes de plus en plus spécialisées. Cela a comme avantage majeur de ne pas avoir à repartir de zéro lorsque l'on veut spécialiser une classe existante. De cette manière il est possible d'acheter dans le commerce des librairies de classes, qui constituent une base, pouvant être spécialisées à loisir (on comprend encore un peu mieux l'intérêt pour l'entreprise qui vend les classes de protéger les données membres grâce à l'encapsulation...).

L'héritage ne fonctionne qu'à sens unique, la sous-classe hérite de la super classe (classe mère), mais pas l'inverse. Contrairement à certains langages orientés objet, PHP ne supporte pas l'héritage multiple. Chaque classe ne peut hériter que d'une seule super classe.

**Syntaxe de l'héritage :** `class nom_de_la_nouvelle_classe extends nom_de_la_super-classe`  

```
{
    // nouvelles propriétés et méthodes
}
```

**Exemple :**

*// Création de la classe de base 'vehicule' :*

```
class vehicule
{
    var $marque ;
    var $modele ;
    var $puissance ;
    var $couleur ;
    var $moteur = "arrêté";

    // implémentation des méthodes agissant sur tous les types de véhicules
    function identifier($marque,$modele)
    {
        $this->marque = $marque ;
        $this->modele = $modele ;
    }
    function demarrer()
    {
        $this->moteur = "en marche" ;
    }
    function arreter()
    {
        $this->moteur = "arrêté" ;
    }
    function assigner_puissance($puiss)
    {
        $this->puissance = $puiss ;
    }
    function peindre($coul)
    {
        $this->couleur = $coul ;
    }
    function etat()
    {
        echo "couleur : $this->couleur<br>" ;
        echo "moteur : $this->puissance chevaux, $this->moteur<br>" ;
    }
}
```

```
// Création de la classe 'voiture' héritée de la classe 'vehicule' :
class voiture extends vehicule
{
// Ajout d'une propriété, d'une méthode permettant de gérer cette propriété et d'une méthode d'affichage
// propre aux voitures portant le même nom que la méthode d'affichage de la classe mère.
    var $nb_portes = 5 ;

    function assigner_portes($portes)
    {
        $this->nb_portes = $portes ;
    }
    function etat()
    {
        echo "voiture : $this->marque $this->modele $this->nb_portes portes<br>" ;
        echo "couleur : $this->couleur<br>" ;
        echo "moteur de $this->puissance chevaux $this->moteur<br>" ;
    }
}

// Création de la classe 'camion' héritée de la classe 'vehicule' :
class camion extends vehicule
{
// Ajout d'une propriété, d'une méthode permettant de gérer cette propriété et d'une méthode d'affichage
// propre aux camions portant le même nom que la méthode d'affichage de la classe mère.
    var $poids;

    function assigner_poids($poids)
    {
        $this->poids = $poids;
    }
    function etat()
    {
        echo "camion : $this->marque $this->modele de $this->poids tonnes<br>" ;
        echo "couleur : $this->couleur<br>" ;
        echo "moteur de $this->puissance chevaux $this->moteur<br>" ;
    }
}
}
```

Le polymorphisme est la possibilité d'utiliser une fonction en lui passant en paramètre n'importe quel objet (dans l'exemple ci-dessus de classe 'voiture' ou de classe 'camion') implémentant une méthode de même nom (dans l'exemple ci-dessus la méthode 'etat()').

Exemple :

```
function affichage($objet)
{
    $objet->etat() ;
}
```

## 80. Exercice pratique

Nous allons illustrer l'instanciation et l'utilisation des objets de classes 'voiture' et 'camion' définies précédemment :

< ?

*// instanciation de la classe 'voiture' et affectation des propriétés de cette voiture*

*\$v1 = new voiture();*

*\$v1->identifier("Renault","Safrane");*

*\$v1->assigner\_puissance(7);*

*\$v1->peindre("rouge");*

*\$v1->assignerportes(3);*

*// instanciation de la classe 'camion' et affectation des propriétés de ce camion*

*\$c1 = new camion();*

*\$c1->identifier("Volvo","V530");*

*\$c1->assigner\_puissance(100);*

*\$c1->peindre("vert");*

*\$c1->demarrer();*

*\$c1->assigner\_poids(25);*

*// affichage de l'état de chacun des objets 'voiture' et 'camion' par l'intermédiaire de la même fonction affichage()*

*affichage(\$v1);*

*echo "<br><br>";*

*affichage(\$c1);*

*?>*

Résultat de l'affichage :

*voiture : Renault Safrane 3 portes*

*couleur : rouge*

*moteur de 7 chevaux arrêté*

*camion : Volvo V530 de 25 tonnes*

*couleur : vert*

*moteur de 100 chevaux en marche*

- Vous pouvez remarquer que la méthode *etat()* appelée via les objets *\$v1* et *\$c1* est celle de la classe dérivée et non celle de la classe mère. En effet, les méthodes et propriétés redéfinies dans les sous-classes sont prépondérantes et remplacent les définitions des classes mères.
- Contrairement à d'autres langages orientés objet, PHP ne permet pas de redéfinir une méthode tout en gardant la possibilité d'invoquer la version définie dans la classe mère.
- Il est recommandé d'accéder aux propriétés d'un objet à l'aide d'une méthode et non directement (ex : *\$c1->marque = "Volvo"* ;). A première vue, une méthode telle que *initialiser()* peut sembler de peu d'intérêt, cependant les méthodes ont une excellente raison d'être : elles permettent d'effectuer tous les accès aux propriétés via une seule section de code, et d'opérer une vérification avant d'autoriser toute modification.

# 11. Les formulaires

Les formulaires HTML représentent pour l'utilisateur le moyen principal de transmettre des informations au serveur. PHP est conçu pour recevoir et exploiter les informations soumises par le biais des formulaires HTML.

## 81. Les variables PHP

PHP est en mesure de localiser les variables globales définies par le programmeur, mais également les variables globales décrivant à la fois les environnements client et serveur. Celles-ci sont plus connues sous le nom de variables d'environnement. La disponibilité de ces variables variera en fonction de votre système, de votre serveur et de votre configuration. Elles peuvent néanmoins s'avérer précieuses.

La fonction *phpinfo()* vous affiche le tableau des variables d'environnement PHP dont les plus communes sont décrites ci-dessous. Il est possible d'y accéder directement ou en tant que partie du tableau \$GLOBALS.

Par exemple la variable \$GLOBALS['PHP\_SELF'] vous donne le chemin d'accès vers le script en cours d'exécution.

Variable	Ce qu'elle contient
\$HTTP_USER_AGENT	Le nom et la version du client
\$REMOTE_ADDR	L'adresse IP du client
\$REQUEST_METHOD	Indique si la méthode était GET ou POST
\$QUERY_STRING	Le nom et la valeur des variables envoyées et accolées à l'URL (chaîne de requête)
\$REQUEST_URI	L'adresse complète du script y compris la chaîne de requête
\$HTTP_REFERER	L'adresse de la page à partir de laquelle la requête a été effectuée

Tableau des variables d'environnement PHP affiché lors de l'appel à la fonction *phpinfo()* :

### PHP Variables

Variable	Value
_POST["articles"]	Array ( [0] => tournevis )
_POST["envoyer"]	Envoyer
_SERVER["HTTP_ACCEPT"]	application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, image/gif, image/x- xbitmap, image/jpeg, image/pjpeg, */*
_SERVER["HTTP_REFERER"]	http://160.xxx.xxx.xxx/formul2.php3
_SERVER["HTTP_ACCEPT_LANGUAGE"]	fr
_SERVER["CONTENT_TYPE"]	application/x-www-form-urlencoded
_SERVER["HTTP_ACCEPT_ENCODING"]	gzip, deflate
_SERVER["HTTP_USER_AGENT"]	Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0; H010818)
_SERVER["HTTP_HOST"]	160.xxx.xxx.xxx
_SERVER["CONTENT_LENGTH"]	40
_SERVER["HTTP_CONNECTION"]	Keep-Alive
_SERVER["HTTP_CACHE_CONTROL"]	no-cache
_SERVER["PATH"]	C:\WINNT\SYSTEM32;C:\WINNT;C:\PROGRAM FILES\BORLAND\CBUILD3\BIN
_SERVER["SystemRoot"]	C:\WINNT
_SERVER["COMSPEC"]	C:\WINNT\system32\cmd.exe
_SERVER["PATHEXT"]	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
_SERVER["WINDIR"]	C:\WINNT
_SERVER["SERVER_SIGNATURE"]	<address>Apache/2.0.40 Server at 160.xxx.xxx.xxx Port 80</address>
_SERVER["SERVER_SOFTWARE"]	Apache/2.0.40 (Win32) PHP/4.3.0-dev
_SERVER["SERVER_NAME"]	160.xxx.xxx.xxx
_SERVER["SERVER_ADDR"]	160.xxx.xxx.xxx
_SERVER["SERVER_PORT"]	80

<code>_SERVER["REMOTE_ADDR"]</code>	160. xxx.xxx.xxx
<code>_SERVER["DOCUMENT_ROOT"]</code>	F:/http/developpement
<code>_SERVER["SERVER_ADMIN"]</code>	philippe.wagner@esat.terre.def
<code>_SERVER["SCRIPT_FILENAME"]</code>	F:/http/developpement/articles.php3
<code>_SERVER["REMOTE_PORT"]</code>	1221
<code>_SERVER["GATEWAY_INTERFACE"]</code>	CGI/1.1
<code>_SERVER["SERVER_PROTOCOL"]</code>	HTTP/1.1
<code>_SERVER["REQUEST_METHOD"]</code>	POST
<code>_SERVER["QUERY_STRING"]</code>	
<code>_SERVER["REQUEST_URI"]</code>	/articles.php3
<code>_SERVER["SCRIPT_NAME"]</code>	/articles.php3
<code>_SERVER["PHP_SELF"]</code>	/articles.php3

## 82. Les éléments SELECT multiples

Jusqu'à présent, les exemples nous permettent de réunir des informations depuis des éléments HTML qui soumettent une valeur unique par nom d'élément.

Les éléments *select* permettent à l'utilisateur de choisir des options multiples.

Le nom de ces éléments doit se terminer par une paire de crochets vide (notation tableau).

Les éléments doivent contenir l'option *multiple*.

Exemple :

```
<html>
<head>
<title>Exercice sur les formulaires</title>
</head>
<body>
<?
echo "<form method='post' action='articles.php3'>";
echo "<div align='center'>";
echo "Sélectionner un ou plusieurs articles<br>";
echo "<select name='articles[]' multiple>";
echo "<option value='tournevis'>tournevis</option>";
echo "<option value='marteau'>marteau</option>";
echo "<option value='pied de biche'>pied de biche</option>";
echo "</select>";
echo "<br><br>";
echo "<input type='submit' name='envoyer' value='Envoyer'>";
echo "</div>";
echo "</form>";
?>
</body>
</html>
```

Sélectionner un ou plusieurs articles

Envoyer

Fichier 'articles.php3' :

```
<?
$articles = $HTTP_POST_VARS["articles"];
$nb_articles = sizeof($articles);
echo "<form method='post' action='#'>";
echo "<div align='center'>";
echo "Vous avez sélectionné $nb_articles articles :<br>";
for( $art = 0 ; $art < $nb_articles ; $art++ )
    echo "$articles[$art]<br>";
echo "</div>";
echo "</form>";
?>
</body>
</html>
```

Vous avez sélectionné 2 articles :

tournevis  
pied de biche

L'élément cases à cocher `<input type='checkbox' name='xx[]'>` autorise également les valeurs multiples dans un même nom de champ. Le nom doit également se terminer par une paire de crochets vide.



### 83. Les champs d'un formulaire dans un tableau associatif

Les techniques étudiées jusqu'à présent fonctionnent bien aussi longtemps que votre script connaît le nombre de champs qu'il va traiter. Vous pouvez cependant avoir à écrire du code qui s'adapte aux modifications réalisées dans un formulaire, ou même qui desserve plusieurs formulaires, sans se soucier des noms des champs spécifiques du formulaire.

Les variables globales fournies par PHP4 permettent de résoudre le problème. En fonction de l'utilisation de la méthode POST ou GET dans le formulaire soumis, vous accéderez à `$HTTP_POST_VARS` ou à `$HTTP_GET_VARS`. Il s'agit de tableaux associatifs qui contiennent les paires nom/valeur des données soumises.

Dans l'exemple suivant, nous définissons un formulaire qui contient un champ texte nommé 'utilisateur', une zone de texte nommée 'adresse' et un bouton de soumission nommé 'envoyer' qui fera appel au formulaire 'trait.php3' :

Votre nom	<input type="text" value="MARTIN"/>
Votre adresse	<input type="text" value="1, rue de Paris"/>
	<input type="text" value="35000 RENNES"/>
<input type="button" value="Envoyer"/>	

```
<html>  
<head>  
<title>Exercice simple sur les formulaires</title>  
</head>  
<body>  
<?  
echo "<form method='post' action='trait.php3'>" ;  
echo "<div align='center'>" ;  
echo "Votre nom&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&";  
echo "<input type='text' name='utilisateur' /><br><br>" ;  
echo "Votre adresse<br>" ;  
echo "<textarea name='adresse' rows='5' cols='40'></textarea><br><br>" ;  
echo "<input type='submit' name='envoyer' value='Envoyer'>" ;      // bouton de soumission  
echo "</div>" ;  
echo "</form>" ;  
?>  
</body>  
</html>
```

L'argument *action* de l'élément *form* pointe sur un fichier nommé 'trait.php3' qui traitera les données du formulaire. Comme nous n'avons spécifié qu'un nom de fichier à l'argument *action*, le fichier doit se trouver dans le même répertoire du serveur que le document contenant notre code.

Le fichier 'trait.php3' décrit ci-dessous est appelé lors de la soumission :

```
<html>
<head>
<title>Exercice sur les formulaires</title>
</head>
<body>
<?
$utilisateur = $_HTTP_POST_VARS["utilisateur"];
$adresse = $_HTTP_POST_VARS["adresse"];
$bouton_soumission = $_HTTP_POST_VARS["envoyer"] ;
$page = $_HTTP_SERVER_VARS["HTTP_REFERER"] ;

echo "<form method='post' action='#>";
echo "<div align='center'>";
echo "<h2>Bienvenue</h2>votre nom est : $utilisateur<br>";
echo "votre adresse est :$adresse";
echo "<br><br>Ce formulaire provient de la page '$page' et a été appelé à l'aide du bouton '$bouton_soumission'";
echo "</div>";
echo "</form>";
?>
</body>
</html>
```

## Bienvenue

votre nom est : MARTIN  
votre adresse est : 1, rue de Paris 35000 RENNES

Ce formulaire provient de la page 'http://160.xxx.xxx.xxx/formul1.php3' et a été appelé à l'aide du bouton 'Envoyer'

Le code suivant dresse la liste des noms et valeurs de tous les paramètres (les tableaux font partie d'un traitement particulier : la sérialisation, qui fera l'objet d'un autre chapitre) qui lui sont transmis via une transaction POST :

```
foreach( $_HTTP_POST_VARS as $nom=>$valeur )
    echo "$nom ➔ $valeur<br>";
```

Résultat à l'affichage :

utilisateur ➔ MARTIN  
adresse ➔ 1, rue de Paris 35000 RENNES  
envoyer ➔ Envoyer

La balise <INPUT> permet également de définir un champ de type 'HIDDEN' (caché). Ce champ n'est pas visible, il est principalement destiné à définir une donnée dont la valeur est fixée, et à passer cette donnée en même temps que celles saisies par l'utilisateur.

Exemple :

```
echo "<form method='post' action='trait.php3'>";
echo "<input type=HIDDEN name='var' value=' ".$var. ">";
```

Il est important de noter que « caché » ne veut pas dire « secret ». Rien n'empêche un utilisateur de consulter la valeur d'un champ caché en regardant le code source du document HTML.

Il faut pouvoir aider l'utilisateur lors de la saisie d'un formulaire, et surtout contrôler que les valeurs entrées respectent certaines règles (ex : champ saisi obligatoirement, valeurs numériques et positives etc.). Ce genre de contrôle peut être fait par le serveur après que l'utilisateur a validé sa saisie, mais ce mode de fonctionnement a l'inconvénient de multiplier les échanges sur le réseau. Une solution plus séduisante est d'effectuer les contrôles par le navigateur, à l'aide d'un langage comme JavaScript qui permet de faire de la programmation au niveau du client (la présentation du langage JavaScript mérite un ouvrage complet).

## 84. Echanges client/serveur

Toutes les données saisies dans un formulaire doivent être transmises au serveur web, et inversement ce programme doit produire un document (généralement un document HTML) et le transmettre au navigateur via le serveur web.

Au moment où l'utilisateur déclenche la soumission, le navigateur concatène dans une chaîne de caractères une suite de descriptions de la forme *nomchamp=valeurchamp* où *nomchamp* est le nom du champ dans le formulaire et *valeurchamp* la valeur saisie. Les différents champs sont séparés par le caractère '&'.

Il existe alors deux manières de transmettre cette chaîne au serveur, selon que la méthode utilisée est GET ou POST.

**Méthode GET :** la chaîne est placée à la fin de l'URL appelée, après un caractère '?'.  
**Méthode POST :** la chaîne est transmise séparément de l'URL. Cette méthode est généralement préférée car elle évite d'avoir à gérer des URL très longues.

## 85. Combinaison de codes HTML et PHP

Dans certaines circonstances, vous pourriez désirer contrôler la validité des champs saisis par l'utilisateur. Il peut se révéler alors utile de présenter le même formulaire à l'utilisateur tant que les champs ne sont pas valides.

Quel que soit le nom que nous attribuons à la page contenant le formulaire, le bouton de soumission 'Envoyer' l'appellera toujours, car nous avons affecté la valeur de \$PHP\_SELF (nom de la page courante) à l'argument ACTION de l'élément FORM.

L'exemple suivant simule l'envoi de plusieurs messages, le numéro du message est affiché en haut de la page :

```
<html>
<head>
<title>Un formulaire qui s'appelle lui-même</title>
</head>
<body>
<form method='GET' action="<?print($PHP_SELF)?>">
<?
echo "<div align='center'>";
// si l'élément nb_messages du tableau $HTTP_GET_VARS existe (vrai à partir du 2ième appel)
// sa valeur sera incrémentée et stockée dans la variable $nb_messages
// sinon celle-ci sera initialisée à 1
$nb_messages = ($HTTP_GET_VARS["nb_messages"]) ? $HTTP_GET_VARS["nb_messages"]+1 : 1;

echo "<h2>$nb_messages ".(($nb_messages == 1) ? "ier" : "ième")." message</h2>";
echo "Entrez votre message : <input type='text' name='mess'>";
echo "<br>Entrez le nom du destinataire : <input type='text' name='dest'>";

echo "<input type='HIDDEN' name='nb_messages' value='". $nb_messages. "'>";
echo "<br><br><input type='submit' name='envoyer' value='Envoyer'>";
echo "</div>";
echo "</form>";
?>
</body>
</html>
```

Vous remarquerez la présence du champ caché nommé 'nb\_messages' et contenant la valeur de la variable \$nb\_messages. La valeur de ce champ est, comme la valeur des autres champs, passée automatiquement à la page appelée à l'aide du bouton 'submit'.

Le champ caché doit être placé après l'obtention de la valeur avec laquelle il sera initialisé.

Résultat à l'exécution :

---

'formul1.php3

---

## 1 ier message

Entrez votre message :

Entrez le nom du destinataire :

---

'formul1.php3?mess=message1&dest=destinataire1&nb\_messages=1&envoyer=Envoyer

---

## 2 ième message

Entrez votre message :

Entrez le nom du destinataire :

## 12. Les fichiers

PHP est un langage de script interprété sur un serveur. Il offre de larges possibilités de gestion et d'utilisation de son système de fichiers semblables au langage C. Toutefois, ces possibilités ne sont en aucun cas applicables au système de fichiers du poste client.

Si un accès aux fichiers du poste client se révèle nécessaire, il faut recourir à des mécanismes de upload et de download, implémentés par PHP, qui permettent de transférer ces fichiers sur le serveur. Ce dernier pourra ainsi y effectuer les traitements requis.

D'autres mécanismes d'accès aux fichiers clients reposant sur des applets Java ou des composants ActiveX qui s'exécutent sur le client sont envisageables, dans certaines conditions, mais sortent du cadre de la programmation PHP, qui se localise exclusivement sur le serveur.

### 86. Structure d'un fichier pour PHP

En effet, PHP reprend toute la base des fonctions de gestion de fichiers du langage C, et y ajoute une série de fonctions adaptées aux applications fondées sur une architecture Intranet.

Pour PHP, tous les fichiers sont ouverts avec la même structure. Seul le mode d'ouverture change (lecture, écriture et modification). Cela implique qu'aucune différence ne sera faite entre l'ouverture d'un fichier binaire et l'ouverture d'un fichier texte, par exemple. Les mêmes méthodes sont applicables, quel que soit le type du fichier ouvert.

### 87. Lire et écrire dans un fichier

Qu'il s'agisse de lire ou d'écrire dans un fichier, l'opération se déroule généralement en trois phases :

- Ouverture du fichier dans le mode approprié,
- Lecture / écriture dans le fichier,
- Fermeture du fichier.

Toutefois PHP propose certaines fonctions qui permettent de réaliser des opérations simples de lecture de fichier qui évitent les phases d'ouverture et de fermeture.

La manière la plus simple de visualiser le contenu d'un fichier en PHP est d'utiliser la fonction ***readfile()***.

Elle permet uniquement le renvoi vers la sortie standard du contenu du fichier passé en paramètre, mais se passe de toute opération d'ouverture et de fermeture de fichier :

```
Readfile("mon_fichier.txt") ; // affiche à l'écran le contenu du fichier 'mon_fichier.txt'
```

La fonction ***file()*** s'utilise de la même manière, mais elle retourne un tableau qui contient les lignes du fichier passé en paramètre, au lieu de l'afficher à l'écran.

### 88. Ouverture d'un fichier

En PHP, l'ouverture d'un fichier s'effectue au moyen de la fonction ***fopen()***. Lors de l'ouverture d'un fichier, vous devez spécifier le nom du fichier et le mode d'ouverture, c'est-à-dire la manière dont vous voulez utiliser le fichier. ***fopen()*** retourne un pointeur sur le fichier, qui est utilisé en tant qu'identifiant dans toutes les fonctions relatives à la manipulation du fichier. Si l'ouverture échoue (ex : fichier en lecture non trouvé), cette fonction retourne faux.

Syntaxe : *Int ptr\_file fopen(string nom\_de\_fichier, string mode\_d\_ouverture) ;*

- Si '*nom\_de\_fichier*' commence par "http://" (insensible à la casse), une connexion HTTP est ouverte avec le serveur spécifié, et un pointeur sur la réponse fournie est retourné.
- Si '*nom\_de\_fichier*' commence par "ftp://" (insensible à la casse), une connexion est ouverte avec le serveur spécifié, et un pointeur sur la réponse fournie est retourné. Si le serveur ne supporte pas le mode ftp passif, cette fonction échouera. Vous pouvez ouvrir des fichiers en lecture seule, ou en écriture seule (le full duplex n'est pas supporté).

Il existe 6 modes d'ouverture de fichiers possibles :

- "r" ouvre le fichier en lecture seule. Le pointeur est positionné en tête du fichier.
- "r+" ouvre le fichier en lecture / écriture et place le pointeur en tête du fichier.
- "w" ouvre le fichier en écriture seule et place le pointeur en tête du fichier. Si le fichier existe déjà, son contenu est effacé. Si le fichier n'existe pas, il est créé.
- "w+" ouvre le fichier en lecture / écriture et place le pointeur en tête du fichier. Si le fichier existe déjà, son contenu est effacé. Si le fichier n'existe pas, il est créé.
- "a" ouvre le fichier en écriture seule et place le pointeur à la fin du fichier (ajout en fin de fichier). Si le fichier n'existe pas, il est créé.
- "a+" ouvre le fichier en lecture / écriture et place le pointeur à la fin du fichier (ajout en fin de fichier). Si le fichier n'existe pas, il est créé.

## 89. Fermeture d'un fichier

La fermeture du fichier est effectuée par la fonction ***fclose()***, qui prend en paramètre l'identifiant du fichier.

Syntaxe :

*int fclose(ptr\_file) ;*

## 90. Accès aux fichiers

**int copy(string source, string dest);** Fait une copie du fichier *source* vers le fichier *dest*. Renvoie vrai (TRUE) en cas de succès, faux (FALSE) sinon.

**float diskfreespace(string directory);** La fonction retourne le nombre d'octets disponibles sur le disque Windows ou le système de fichiers dans lequel se trouve le dossier UNIX

**int feof(int fp);** Retourne vrai (TRUE) si le pointeur fp est à la fin du fichier, ou si une erreur survient, sinon, retourne faux (FALSE). Le pointeur de fichier doit être valide, et avoir été correctement ouvert par *fopen()*.

**char fgetc(int fp);** Retourne le caractère lu depuis le fichier pointé par fp qui est avancé d'une position. Retourne FALSE à la fin du fichier (tout comme *feof()*). Le pointeur de fichier doit être valide, et avoir été correctement ouvert par *fopen()*.

**string fgets(int fp, int length);** Retourne la chaîne lue jusqu'à la longueur (*length* - 1) octets, ou bien la fin du fichier, ou encore un retour chariot (le premier des trois que l'on rencontre). Si une erreur survient, retourne faux. Erreur courante : Les programmeurs habitués à la programmation 'C' noteront que *fgets()* ne se comporte pas comme son équivalent 'C' lors de la rencontre de la fin du fichier. Le pointeur de fichier doit être valide, et avoir été correctement ouvert par *fopen()*.

Exemple :

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

**int file\_exists(string filename);** Retourne vrai si le fichier *filename* existe, et faux sinon. Le résultat de cette fonction est mis en cache.

**int filesize(string filename);** Retourne la taille du fichier *filename*, ou faux en cas d'erreur. Le résultat de cette fonction est mis en cache.

**bool flock(int fp, int operation);** PHP dispose d'un système complet de verrouillage de fichier. Tous les programmes qui accèdent au fichier doivent utiliser la même méthode de verrouillage pour qu'il soit efficace.

*flock()* agit sur le fichier *fp* qui doit avoir été ouvert. *operation* est une des valeurs suivantes :

- 1 · Verrouillage en lecture. Le fichier peut être partagé avec d'autres lecteurs.
- 2 · Verrouillage en écriture. Ce type de verrouillage est exclusif : le fichier ne peut être partagé.
- 3 · Libération d'un verrou existant (partagé ou exclusif).
- +4 · Si vous voulez que *flock()* ne se bloque pas durant le verrouillage, ajoutez 4 à *operation*.

*flock()* permet de réaliser un système de verrous écriture / lecture simple, qui peut être utilisé sur n'importe quelle plate-forme (Unix et Windows compris).

*flock()* retourne vrai en cas de succès, et faux sinon. (le verrou n'a pas pu être obtenu).

**int fpassthru(int fp);** Lit le contenu d'un fichier depuis la position du pointeur jusqu'à la fin, et dirige le résultat vers la sortie standard (l'écran). Si une erreur survient, *fpassthru()* retourne faux. Le pointeur de fichier doit être valide, et doit avoir été correctement ouvert par *fopen()*. Après la lecture, *fpassthru()* va fermer le fichier (le pointeur sera alors invalide). Si vous voulez simplement afficher le contenu d'un fichier, il suffit d'utiliser *readfile()*, ce qui épargnera l'appel à *fopen()*.

**int fputs(int fp, string str, [int length]);** La fonction écrit la chaîne *str* dans le fichier décrit par *fp*. *fputs()* est un alias de *fwrite()*, et lui est identique en tout point. Notez que *length* est un paramètre optionnel, et s'il n'est pas spécifié, toute la chaîne est écrite. Elle retourne vrai si l'opération réussit.

**string fread(int fp, int length);** Lecture d'un fichier en mode binaire. *fread()* lit jusqu'à *length* octets dans le fichier référencé par *fp*. La lecture s'arrête lorsque *length* octets ont été lus, ou que l'on a atteint la fin du fichier, ou qu'une erreur survient (le premier des trois).

**int fseek(int fp, int offset);** Positionne le pointeur interne de fichier *fp* à *offset* octets à partir du début du fichier. Equivalent à la fonction 'C' `fseek( fp, offset, SEEK_SET )`. Retourne vrai en cas de succès, et sinon -1. Notez que positionner le pointeur au-delà de la fin du fichier n'est pas une erreur. Ne peut pas être utilisé sur les pointeurs retournés par *fopen()* s'ils sont au format HTTP ou FTP.

**int ftell(int fp);** Retourne la position courante du pointeur *fp* (nombre d'octets par rapport au début du fichier). Si une erreur survient, retourne faux. Le pointeur de fichier doit être valide, et avoir été correctement ouvert par *fopen()*.

**int fwrite(int fp, string str, [int length]);** La fonction écrit le contenu de la chaîne *str* dans le fichier pointé par *fp*. Si la longueur *length* est fournie, l'écriture s'arrêtera après *length* octets, ou à la fin de la chaîne (le premier des deux).

**int rewind(int fp);** Remplace le pointeur du fichier *fp* au début. Si une erreur survient, la fonction retourne 0. Le pointeur de fichier doit être valide, et avoir été correctement ouvert par *fopen()*.

## 91. Accès aux répertoires

**new dir(string directory);** *dir* est un mécanisme orienté pseudo-objet qui permet la lecture d'un dossier. L'argument *directory* doit être ouvert.

```
class dir {
    dir(string directory);
    string path;
    resource handle;

    string read();
    void rewind();
    void close();
}
```

Deux propriétés sont disponibles une fois le dossier ouvert : le pointeur *handle* peut être utilisé avec d'autres fonctions telles que *readdir()*, *rewinddir()* et *closedir()*. Le chemin du dossier *path* est le chemin fourni lors de la construction de l'objet. Trois méthodes permettent de lire *read()*, remettre à zéro *rewind()* et fermer le dossier *close()*.

**Exemple** : affichage du nom de toutes les entrées d'un répertoire en programmation objet :

```
<?
$rep = dir("C:\Program Files");
echo "chemin : $rep->path<br>";

while($entry = $rep->read())
    echo "$entry<br>";

$rep->close();
?>
```

**string getcwd (void);** Retourne le nom du dossier courant.

**int chdir(string directory);** Permet de se positionner dans le dossier *directory*. Retourne faux (FALSE ) si l'opération échoue, et vrai (TRUE ) sinon.

**void closedir(int ptr\_dir);** Ferme le pointeur de dossier *ptr\_dir*. Le dossier doit avoir été ouvert avec *opendir()*.

**int mkdir(string pathname, int permissions);** Tente de créer un dossier dans le chemin *pathname*. Notez que vous aurez à préciser les permissions en base octale (codées comme sous UNIX), ce qui signifie que vous aurez probablement un 0 comme premier chiffre, **exemple** :

```
mkdir("/path/to/my/dir", 0700);
```

La fonction retourne vrai en cas de succès, et faux en cas d'échec.

**int opendir(string path);** Retourne un pointeur sur le dossier spécifié par le chemin *path*, pour être utilisé avec les fonctions *closedir()*, *readdir()*, et *rewinddir()*.

**string readdir(int ptr\_dir);** Retourne le nom du fichier suivant dans le dossier identifié par *ptr\_dir*. Les noms sont retournés dans n'importe quel ordre.

**int rename(string oldname, string newname);** Tente de renommer le fichier *oldname* en *newname*. Retourne vrai en cas de succès et faux sinon.

**void rewinddir(int ptr\_dir);** Retourne à la première entrée du dossier pointé par *ptr\_dir* : le prochain fichier lu sera le premier.

**int rmdir(string dirname);** Tente d'effacer le dossier dont le chemin est *dirname*. Le dossier doit être vide, et le script doit avoir les autorisations adéquates. Si une erreur survient, la fonction retourne 0.

**int unlink(string filename);** Tente d'effacer le fichier *filename*. Identique à la fonction Unix C *unlink()*. La fonction retourne 0 ou FALSE en cas d'échec.

**Exemple** : affichage du nom de tous les fichiers d'un répertoire :

```
<?
echo "<form method='POST' action='#'>";
echo "<div align='center'>";
$path = "F:\\http\\developpement";
$dp = opendir($path);
echo "Liste des noms de fichiers du répertoire : $path<br><br>";
while($fic = readdir($dp))
    echo "$fic<br>";
closedir($dp);
echo "</div>";
echo "</form>"; ?>
```



## 92. Lire un fichier défini par une URL

PHP a été créé spécialement pour développer des applications Internet/Intranet, et cette fonctionnalité en est une illustration typique. De la même manière qu'on a lu un fichier local, on va pouvoir lire un fichier distant, en passant par HTTP ou FTP. Pour cela, il suffit de passer une URL en paramètre à *fopen()* au lieu d'un nom de fichier.

Exemple :

```
$fp = fopen("http://www.php.net", "r") ;
```

Cette facilité permet d'analyser le code source d'une URL au fur et à mesure de son arrivée, avant son interprétation par le navigateur client. On peut ainsi filtrer le contenu des pages, neutraliser des liens ou encore aspirer un site complet avec un minimum de code. Il est à noter que seul le mode lecture fonctionne avec HTTP.

La même opération est possible en utilisant le protocole FTP. Dans ce cas les lectures / écritures sont possibles mais dépendent étroitement des libertés accordées par le serveur référencé.

## 93. Exemples de traitements sur les fichiers

Exemple1 : Le script suivant va dupliquer un fichier texte en le mettant au format HTML, avec mise en gras de la première lettre de chaque ligne et augmentation de la taille de sa police. Un retour à la ligne HTML sera inséré entre chaque ligne :

```
<html>
<head>
<title>Un formulaire qui duplique un fichier texte en fichier HTML</title>
</head>
<body>
<?
echo "<form method='POST' action='#'>";
echo "<div align='center'>";
$nom_fic_w = "fic_w.html";
$nom_fic_r = "fic_r.txt";

// ouverture des fichiers
$fpr = fopen($nom_fic_r,"r");
$fpw = fopen($nom_fic_w,"w");

// parcours du fichier à dupliquer
while($ligne_r = fgets($fpr,4096))
{
    $ligne_w = "<font size=+1><b>"; // augmentation de la taille et mise en gras
    $ligne_w .= substr($ligne_r,0,1); // retourne le premier caractère
    $ligne_w .= "</b></font>"; // retour à la taille standard et fin de mise en gras
    $ligne_w .= substr($ligne_r,1); // retourne la ligne complète à partir du 2ième caractère
    $ligne_w .= "<br>"; // ajout de la balise 'nouvelle ligne'

    fwrite($fpw,$ligne_w); // ecriture dans le fichier html
}

// fermeture des deux fichiers
fclose($fpr);
fclose($fpw);

// affichage d'un message de fin de traitement
echo "L'écriture du fichier $nom_fic_w est terminée !";
echo "</div>";
echo "</form>";
?>
</body>
</html>
```

Exemple2 : Parcours d'un fichier "/etc/passwd" et affichage des utilisateurs d'un serveur :

```
$data = file("/etc/passwd") ;
foreach ( $data as $ligne )
{
    $user = strtok($ligne,":") ;
    $passwd = strtok(":") ;
    $uid = strtok(":") ;
    $gid = strtok(":") ;
    $gecos = strtok(":") ;
    $homedir = strtok(":") ;
    $shell = strtok(":") ;
    echo "$user (uid=$uid) => $gecos<br>";
}
```

## 94. La fonction popen

L'accès au système de fichiers de la machine peut se faire par l'intermédiaire de l'exécution de commandes système, dont on récupère le résultat, comme s'il s'agissait d'un fichier ordinaire.

La fonction `popen()` permet d'exécuter ces commandes. Le code suivant permet d'afficher le contenu du répertoire qui contient la page PHP interprétée :

```
<html>
<head>
<title>Obtention d'une liste de fichiers à l'aide de popen</title>
</head>
<body>
<?
echo "<form method='POST' action='#>";
echo "<div align='center'>";
$commande = "dir /B";           // sous UNIX la commande sera 'ls'

// exécution de la commande et stockage du résultat dans un fichier pointé par $fpr
$fpr = popen($commande,"r");

// parcours du fichier contenant le résultat de la commande
// lecture de chaque ligne et suppression des espaces blancs de début et fin de ligne
while($ligne_r = trim(fgets($fpr,4096)))
    echo $ligne_r."<br>";

echo "</div>";
echo "</form>";
?>
</body>
</html>
```

## 95. Problèmes posés par l'usage de fichiers plats

L'utilisation de fichiers pose divers problèmes :

- Dès lors qu'un fichier devient volumineux, sa manipulation peut se révéler très lente.
- La recherche d'un enregistrement ou d'un ensemble d'enregistrements dans un fichier plat est difficile.
- La gestion des accès concurrents est problématique : si le trafic sur le site prend de l'ampleur, il peut arriver que de nombreux utilisateurs aient à attendre le déverrouillage du fichier pour valider leur commande. Les longues attentes sont pénalisantes pour l'utilisateur.
- Dans toutes les manipulations de fichiers décrites jusqu'ici, les traitements étaient mis en œuvre de façon séquentielle, c'est-à-dire en partant du début du fichier, et en parcourant le contenu du fichier jusqu'à la fin de celui-ci. S'il apparaît nécessaire d'insérer ou de supprimer des enregistrements à partir du milieu du fichier (accès aléatoire), le mode de traitement séquentiel peut poser un problème : il implique de lire et de placer en mémoire l'intégralité du fichier, d'apporter les modifications, et de réécrire à nouveau le fichier. La charge de traitement peut alors devenir très lourde avec des fichiers de données volumineux.
- Au-delà de la limite offerte par le dispositif de permissions sur les fichiers, il n'existe pas de moyen simple d'implémenter des niveaux différents d'accès aux données.

Vous pouvez consulter la section du manuel en ligne PHP (<http://no.php.net/manual/fr/>) consacrée au système de fichiers.

## 13. Les bases de données

Les systèmes de gestion de base de données relationnelle (SGBD) apportent des solutions à tous les problèmes évoqués au chapitre précédent. Les SGBD :

- Permettent des accès plus rapides aux données. MySQL est apparu comme le SGBD le plus rapide du marché. Vous pouvez consulter les statistiques de ses performances sur le site <http://web.mysql.com/benchmark.html>
- Peuvent être facilement consultés et explorés, pour extraire les ensembles de données qui répondent à des critères spécifiques.
- Comprennent des mécanismes prédéfinis pour le traitement des accès concurrents.
- Fournissent un accès aléatoire aux données.
- Comprennent des systèmes de privilèges intégrés. MySQL est particulièrement performant sur ce point.

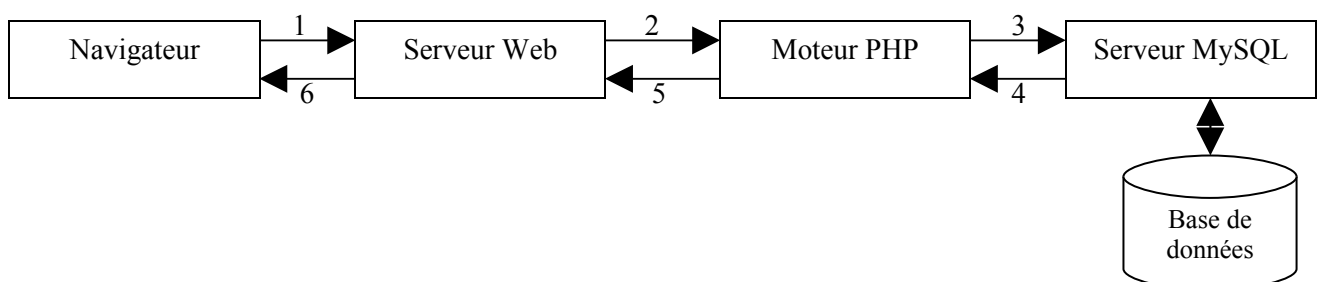
Par rapport à ses concurrents, MySQL possède plusieurs avantages majeurs, comme ses performances élevées, son coût réduit (disponible gratuitement sous licence Open Source, ou pour un prix très raisonnable pour les licences commerciales), sa simplicité de configuration et d'apprentissage, sa portabilité et son code source disponible gratuitement, au même titre que celui de PHP.

MySQL peut être utilisé sur un grand nombre de systèmes UNIX, ainsi qu'avec Windows.

## 96. Architecture d'une base de données Web

Le fonctionnement fondamental d'un serveur Web est articulé autour d'un système composé de deux objets : un navigateur Web et un serveur Web. Le navigateur Web effectue des requêtes auprès du serveur, et le serveur renvoie des réponses. Cette architecture est parfaitement adaptée à un serveur qui fournit des pages statiques.

L'architecture qui permet de servir des sites Web faisant intervenir des bases de données est un peu plus complexe. L'architecture fondamentale nécessaire pour les accès aux bases de données Web est composée d'un navigateur Web, d'un serveur Web, d'un moteur de scripts (PHP) et d'un serveur de bases de données (MySQL).



Une transaction de base de données Web typique est composée des étapes suivantes :

1. Le navigateur Web d'un utilisateur envoie une requête HTTP pour une page Web particulière. La requête peut se présenter sous la forme d'un formulaire HTML. Le serveur Web reçoit la requête, récupère le formulaire et le passe au moteur PHP afin qu'il soit traité.
2. Le moteur PHP commence à analyser le script. A l'intérieur de ce script se trouve une commande permettant de se connecter à la base de données et d'exécuter une requête. PHP ouvre une connexion vers le serveur MySQL, et transmet la requête appropriée.
3. Le serveur MySQL reçoit la requête et la traite, puis envoie les résultats au moteur PHP.
4. Le moteur PHP termine l'exécution du script, ce qui consiste généralement en un formatage des résultats de la requête en HTML.
5. Le moteur PHP envoie ensuite le fichier HTML obtenu au serveur Web.
6. Le serveur Web transmet la page HTML au navigateur, pour que l'utilisateur puisse visualiser les résultats.

Le plus souvent, le serveur Web, le moteur PHP et le serveur de bases de données sont tous exécutés sur le même ordinateur. Cependant, il arrive que le serveur de bases de données soit exécuté sur un autre ordinateur. Cette dernière approche répond à des problèmes de sécurité, d'augmentation des capacités, et de répartition de la charge.

PHP côté serveur : quand un fichier avec une extension .php ou .php3 est demandé au serveur Web, ce dernier le charge en mémoire et y cherche toutes les instructions PHP qu'il transmet à l'interpréteur. L'interpréteur exécute les instructions, ce qui a pour effet de produire du code HTML qui vient remplacer les instructions PHP dans le document finalement fourni au navigateur. Ce dernier reçoit donc du HTML « pur » et ne voit jamais la moindre instruction PHP.

## 97. Structure de l'installation de MySQL

Répertoires de 'mysql'	Contenu des répertoires
'bin'	Programmes client et serveur 'mysqld' ( <b>Compiled with full debugging and automatic memory allocation checking</b> )
'data'	Fichiers log et bases de données
'include'	Fichiers 'header'
'lib'	Librairies
'scripts'	Fichiers liés à la configuration du serveur MySQL dont 'mysql_install_db' ( <b>Vous pouvez utiliser ce script comme base, pour ajouter de nouveaux utilisateurs, modifier les privilèges</b> )
'share'	Fichiers contenant les tables de caractères des pays
'docs'	<b>Manuels de documentation</b>
'bench'	<b>Benchmarks</b>

## 98. Les principales instructions SQL

### 98.1. SELECT

Cette instruction est utilisée pour lire des données dans une base de données, en sélectionnant les lignes qui correspondent à certains critères.

Sa forme principale est la suivante :

```
SELECT colonnes FROM tables [WHERE condition] [GROUP BY group_type] [HAVING définition]
[ORDER BY ordre] [LIMIT limite] ;
```

Vous pouvez spécifier autant de colonnes que vous le souhaitez, en les mentionnant après le mot clé SELECT. Le caractère '\*' symbolise toutes les colonnes de la table spécifiée.

La clause **WHERE** spécifie les critères utilisés pour sélectionner les lignes. En plus du test d'égalité, MySQL supporte plusieurs opérateurs et expressions régulières :

Opérateur	Nom	Exemple	Description
=	Egalité	Index = 5	Teste si les deux opérandes sont égaux
>	Supérieur	Total > 100	Teste si un opérande est supérieur à un autre
<	Inférieur	Total < 100	Teste si un opérande est inférieur à un autre
>=	Supérieur ou égal	Total >= 100	Teste si un opérande est supérieur ou égal à un autre
<=	Inférieur ou égal	Total <= 100	Teste si un opérande est inférieur ou égal à un autre
!= ou <>	Différent de	Total != 100	Teste si les deux opérandes sont différents
IS NOT NULL	Présence	Total IS NOT NULL	Teste si le champ contient une valeur
IS NULL	Présence	Total IS NULL	Teste si le champ ne contient aucune valeur
BETWEEN	Intervalle	Total BETWEEN 100 and 200	Teste si la valeur est comprise dans l'intervalle spécifié
IN	Existence	Ville IN ("Paris","Lyon")	Teste si la valeur se trouve dans un ensemble spécifié
NOT IN	Existence	Ville NOT IN ("Paris","Lyon")	Teste si la valeur ne se trouve pas dans un ensemble spécifié
LIKE	Comparaison de motif	Nom LIKE ("Fred %")	Teste si la valeur correspond à un motif spécifié
NOT LIKE	Comparaison de motif	Nom NOT LIKE ("Fred %")	Teste si la valeur ne correspond pas à un motif spécifié
REGEXP	Comparaison de motif	Nom REGEXP	Teste si la valeur correspond à une expression régulière (séquence de caractères)

- Les motifs peuvent contenir du texte classique, plus les caractères % et \_ . % remplace n'importe quelle chaîne de caractères, et \_ remplace n'importe quel caractère. Dans MySQL les motifs ne respectent pas la casse.
- Il est possible de sélectionner des lignes en associant plusieurs critères avec AND et OR ou && et ||.

- Pour rassembler des informations qui se trouvent dans différentes tables, vous devez effectuer une opération appelée ‘fusion’. Cela revient simplement à réunir plusieurs tables en fonction des relations qui existent entre leurs données.

Exemple :

```
SELECT clients.nom FROM clients, emprunts, livres WHERE clients.num_client = emprunts.num_client AND emprunts.nombre_emprunts >= 4 AND livres.titre LIKE '%PHP%';
```

- Il est souvent pratique de pouvoir faire référence aux tables avec d’autres noms que l’on appelle des ‘alias’. Vous pouvez les créer au début d’une requête à l’aide du mot clé AS, et les utiliser tout au long de la requête.

Exemple :

```
SELECT c.nom FROM clients as c, emprunts as e, livres as l WHERE c.num_client = e.num_client AND e.nombre_emprunts >= 4 AND l.titre LIKE '%PHP%';
```

Il faut passer par les alias pour fusionner une table avec elle-même. Cette approche peut être utile si nous voulons trouver dans une table les lignes qui possèdent des valeurs en commun.

Exemple, trouver les clients qui habitent dans la même ville :

```
SELECT c1.nom, c2.nom, c1.ville FROM clients as c1, clients as c2 WHERE c1.ville = c2.ville AND c1.nom != c2.nom ;
```

La clause **ORDER BY** est utilisée pour trier les lignes en fonction de plusieurs colonnes, listées dans la clause **SELECT**. Par défaut, l’ordre de tri est croissant. Vous pouvez le spécifier explicitement à l’aide du mot clé **ASC**. Il est possible de trier par ordre décroissant, en spécifiant le mot clé **DESC**.

Exemple :

```
SELECT nom, ville FROM clients ORDER BY nom, ville ASC ;
```

MySQL possède plusieurs fonctions d’agrégation pour répondre à certains types de requêtes. Ces fonctions peuvent être appliquées à une table prise comme un ensemble, ou à un groupe de données dans une table :

Nom	Description
AVG(colonne)	Moyenne des valeurs dans la colonne spécifiée.
COUNT(colonne)	Nombre de valeurs non NULL dans la colonne. Si vous ajoutez le mot <b>DISTINCT</b> devant le nom de la colonne, vous obtiendrez le nombre de valeurs distinctes dans cette colonne. Si vous spécifiez <b>COUNT (*)</b> , vous obtiendrez un compte global, indépendamment des valeurs NULL.
MIN(colonne)	Minimum des valeurs dans la colonne spécifiée.
MAX(colonne)	Maximum des valeurs dans la colonne spécifiée.
STD(colonne)	Ecart type des valeurs dans la colonne spécifiée.
STDDEV(colonne)	Idem STD(colonne)
SUM(colonne)	Somme des valeurs dans la colonne spécifiée.

La clause **GROUP BY** permet d’obtenir des informations plus détaillées. Par exemple cela nous permet d’afficher la moyenne des commandes pour différents groupes (ex : en fonction du numéro de société). Grâce à ce mécanisme, nous pouvons connaître la société qui a dépensé le plus d’argent :

```
SELECT num_societe, avg(montant) FROM factures GROUP BY num_societe ;
```

La clause **HAVING** permet de tester le résultat d’une agrégation. Elle doit être spécifiée aussitôt après la clause **GROUP BY**, et elle fonctionne comme une clause **WHERE** qui ne s’appliquerait qu’aux groupes et aux agrégats.

Exemple :

```
SELECT num_societe, avg(montant) FROM factures GROUP BY num_societe HAVING avg(montant) > 1000;
```

La clause **LIMIT** est utilisée pour spécifier les lignes du résultat qui doivent être renvoyées. Elle accepte deux paramètres : le numéro de ligne de départ et le nombre de lignes à renvoyer (les lignes sont numérotées à partir de 0). Cela est très utile pour les applications Web, par exemple pour permettre aux clients qui parcourent le catalogue des produits, d’afficher uniquement 10 articles par page.

Exemple :

```
SELECT * FROM produits LIMIT 0, 9 ;
```

Au lieu d’afficher le résultat d’un ordre **SELECT** à l’écran, on peut le placer dans un fichier.

Exemple :

```
SELECT * INTO OUTFILE 'svfilm.txt' FROM films ;
```

Il faut avoir le privilège *file* (voir auprès de l'administrateur du serveur MySQL) pour utiliser INTO OUTFILE. On obtient alors tous les droits d'accès du serveur *mysql*.

Par défaut, le fichier est créé dans le répertoire contenant les bases de données. On peut indiquer explicitement le chemin d'accès, à condition que *mysql* ait le droit d'écriture.

Les lignes de la table sont écrites en séparant chaque valeur par une tabulation, ce qui permet de recharger le fichier par la suite avec la commande LOAD DATA. On peut indiquer, après le nom du fichier, les options de séparation des lignes et des attributs, avec une syntaxe identique à celle utilisée pour LOAD DATA.

## 98.2. INSERT

Cette instruction permet d'ajouter des lignes dans la base de données.

Sa forme principale est la suivante :

```
INSERT [INTO] nom_de_la_table [(colonne1, colonne2, ...)] VALUES (valeur1, valeur2, ...);
```

Les valeurs spécifiées sont utilisées pour remplir la table, dans l'ordre où elles sont fournies. Cependant, si vous souhaitez remplir uniquement certaines colonnes, ou si vous voulez spécifier les valeurs dans un ordre différent, vous pouvez préciser le nom des colonnes dans l'instruction.

Exemples :

```
INSERT INTO clients VALUES (NULL, 'DUPOND', 'Jules', '10, rue de Brest', 'RENNES');
```

```
INSERT INTO clients (nom, prenom, adresse, ville) VALUES ('DURANT', 'Pierre', '4, rue de Paris', 'RENNES');
```

Voici une autre syntaxe similaire :

```
INSERT INTO clients SET nom='DURANT', prenom='Pierre', adresse='4, rue de Paris', ville='RENNES';
```

- Les chaînes de caractères doivent toujours être mises entre simples ou doubles guillemets. Les nombres et les dates n'ont pas besoin de guillemets.
- Nous avons spécifié la valeur NULL lorsque nous avons ajouté *Jules DUPOND*, car la première colonne (*num\_client*) a été déclarée comme étant la clé primaire de la table *clients*, avec l'attribut AUTO\_INCREMENT. Cela signifie que si nous insérons une ligne avec la valeur NULL (ou aucune valeur) dans ce champ, MySQL génère le prochain numéro dans la séquence d'auto-incrémentation, et insère automatiquement la prochaine valeur.
- Il est également possible d'insérer plusieurs lignes dans une table en une seule instruction. Chaque ligne doit être mise entre parenthèses, et les ensembles de parenthèses doivent être séparés par des virgules.

Exemple :

```
INSERT INTO clients VALUES (NULL, 'DUPOND', 'Jules', '10, rue de Brest', 'RENNES'),  
(NULL, 'DURANT', 'Pierre', '4, rue de Paris', 'RENNES'),  
(NULL, 'MARTIN', 'Alain', '32, rue de Nantes', 'RENNES');
```

## 98.3. UPDATE

Cette instruction met à jour une table, en modifiant toutes les colonnes spécifiées en fonction des expressions fournies. Vous pouvez restreindre une instruction UPDATE à certaines lignes avec la clause WHERE, et limiter le nombre total de lignes affectées avec la clause LIMIT.

Sa forme principale est la suivante :

```
UPDATE nom_de_la_table SET colonne1=expression1, colonne2=expression2, ... [WHERE condition]  
[LIMIT nombre];
```

Exemple :

```
UPDATE produits SET prixHT = prixHT*1.15 WHERE categorie='tabac'; // un peu d'humour ! ;-)
```

## 98.4. DELETE

Cette instruction permet de supprimer des enregistrements dans la base de données.

Sa syntaxe est la suivante :

```
DELETE FROM nom_de_la_table [WHERE condition] [LIMIT nombre];
```

Si vous ne spécifiez aucune clause, toutes les lignes de la table seront supprimées.

La clause WHERE permet de spécifier les lignes à supprimer.

La clause LIMIT peut être utilisée pour limiter le nombre maximal de lignes à supprimer.

Exemple :

```
DELETE FROM clients WHERE num_client = 6;
```

## 98.5. ALTER TABLE

Cette instruction permet de modifier la structure d'une table.

Sa forme principale est la suivante :

ALTER TABLE *nom\_de\_la\_table* *modification1* [, *modification2* ...] ;

En SQL ANSI, il n'est possible d'apporter qu'une seule modification par instruction ALTER TABLE, avec MySQL vous pouvez en faire autant que vous le souhaitez. Chaque clause de modification peut être utilisée pour modifier différents aspects de la table.

Les modifications possibles sont les suivantes :

Syntaxe	Description
ADD [COLUMN] <i>description</i> [FIRST   AFTER <i>colonne</i> ]	Ajoute une nouvelle colonne à l'emplacement spécifié (s'il n'est pas spécifié, la colonne est ajoutée à la fin de la table). Notez que <i>description</i> nécessite un nom et un type, tout comme une instruction CREATE.
ADD [COLUMN] ( <i>description1</i> , <i>description2</i> , ...)	Ajoute une ou plusieurs colonnes à la fin de la table.
ADD INDEX [index] ( <i>colonne1</i> , ...)	Ajoute un index dans la table, sur les colonnes spécifiées.
ADD PRIMARY KEY ( <i>colonne1</i> , ...)	Transforme les colonnes spécifiées en clé primaire de la table.
ADD UNIQUE [index] ( <i>colonne1</i> , ...)	Ajoute un index unique dans la table, sur les colonnes spécifiées.
ALTER [COLUMN] <i>colonne</i> {SET DEFAULT <i>valeur</i>   DROP DEFAULT}	Ajoute ou supprime une valeur par défaut pour une colonne particulière.
CHANGE [COLUMN] <i>colonne</i> <i>nouvelle_description</i>	Modifie la colonne spécifiée en lui affectant la nouvelle description. Vous pouvez vous en servir pour modifier le nom d'une colonne, puisque la description d'une colonne contient son nom.
MODIFY [COLUMN] <i>description</i>	Modifie le type d'une colonne mais pas son nom.
DROP [COLUMN] <i>colonne</i>	Supprime la colonne spécifiée.
DROP PRIMARY KEY	Supprime l'index principal, mais pas la colonne.
DROP INDEX <i>index</i>	Supprime l'index spécifié.
RENAME [AS] <i>nouvelle_table</i>	Renomme une table.

Exemples :

Modification de la taille d'une colonne : *ALTER TABLE clients MODIFY nom VARCHAR(60) NOT NULL ;*

Ajout d'une colonne : *ALTER TABLE clients ADD code\_postal CHAR(5) AFTER adresse ;*

Suppression d'une colonne : *ALTER TABLE clients DROP age ;*

## 98.6. DROP TABLE

Cette instruction permet de supprimer une table de la base de données.

Sa syntaxe est la suivante :

DROP TABLE *nom\_de\_la\_table* ;

L'instruction DROP TABLE supprime toutes les lignes de la table et la table elle-même.

## 98.7. DROP DATABASE

Cette instruction permet de supprimer l'intégralité d'une base de données.

Sa syntaxe est la suivante :

DROP DATABASE *nom\_de\_la\_base\_de\_donnees* ;

L'instruction DROP DATABASE supprime toutes les lignes, toutes les tables, tous les index et la base de données elle-même.



## 99. Compatibilité avec les standards

### 99.1. Extensions MySQL à la norme ANSI SQL 92

Vous ne trouverez probablement ces extensions dans aucune autre base de données. Soyez conscient que si vous utilisez ces extensions, votre code ne sera pas portable sur une autre base SQL. Dans certains cas, vous pouvez écrire du code qui utilise ces extensions MySQL, mais qui est portable, en utilisant les commentaires de la forme `/*! ... */`. Dans ce cas, MySQL va analyser puis exécuter le code de ce commentaire comme n'importe quelle commande MySQL, mais les autres serveur SQL les ignoreront. Par exemple :

```
SELECT /*! STRAIGHT_JOIN */ nom_colonne FROM table1,table2 WHERE ...
```

Voici la liste des extensions MySQL:

- Les types de colonnes *MEDIUMINT*, *SET*, *ENUM* et tous les types *BLOB* et *TEXT*.
- Les attributs de champs *AUTO\_INCREMENT*, *BINARY*, *UNSIGNED* et *ZEROFILL*.
- Toutes les comparaisons de chaînes sont insensibles à la casse par défaut, et le classement des chaînes est basé sur la table de caractères courante (par défaut, ISO-8859-1 Latin1). Si vous souhaitez que ces opérations soient sensibles à la casse, ajoutez l'attribut *BINARY* aux colonnes concernées, ou bien utilisez l'opérateur de transpypage *BINARY* lors des comparaisons pour qu'elles prennent en compte l'ordre ASCII en cours sur l'hôte MySQL.
- MySQL fait correspondre à chaque base de données un dossier, et les tables de cette base seront représentées par des fichiers. Cela a deux conséquences :
  1. Les noms de bases de données et de tables sont dépendants du système d'exploitation sur lequel MySQL tourne, notamment concernant la casse des noms. Si vous avez des difficultés à vous souvenir des noms de tables, utilisez une convention cohérente, comme par exemple celle de mettre tous les noms en minuscules.
  2. Les noms de bases, de tables, d'index, de colonnes ou d'alias peuvent commencer par un chiffre (mais ne doivent pas être constitués seulement de chiffres).
 Remarque : Vous pouvez utiliser les commandes du système pour sauvegarder, renommer, déplacer, effacer et copier des tables. Par exemple, vous pouvez copier le `'.ISD'`, `'.ISM'` et `'.frm'` pour créer une nouvelle table.
- En SQL pur, vous pouvez accéder à une table d'une autre base de données avec la syntaxe : `nom_base_de_donnees.nom_table`. Certains serveurs SQL proposent la même fonctionnalité, mais l'appelle *User space*. MySQL n'accepte pas les espaces dans les noms de tables.
- *LIKE* est utilisable avec les colonnes de type numérique
- Possibilité d'utilisation de *INTO OUTFILE* et *STRAIGHT\_JOIN* dans les commandes *SELECT*.
- L'option *SQL\_SMALL\_RESULT* dans une commande *SELECT*.
- *EXPLAIN SELECT* permet de voir de quelle manière les tables ont été regroupées.
- L'utilisation des index de noms, ou des index sur le préfixe d'un champ et l'utilisation de *INDEX* ou *KEY* dans une commande *CREATE TABLE*.
- Attribut *TEMPORARY* et *IF NOT EXISTS* dans la commande *CREATE TABLE*.
- Utilisation de *COUNT(DISTINCT liste)* lorsque 'liste' contient plus d'un élément.
- Utilisation de *CHANGE nom\_colonne*, *DROP nom\_colonne* ou *DROP INDEX* dans les commandes *ALTER TABLE*.
- Utilisation de *IGNORE* dans les commandes *ALTER TABLE*.
- Utilisation de multiples *ADD*, *ALTER*, *DROP* ou *CHANGE* dans les commandes *ALTER TABLE*.
- Utilisation de *DROP TABLE* avec les mots clés *IF EXISTS*.
- Vous pouvez supprimer plusieurs tables avec une seule commande *DROP TABLE*.
- Clause *LIMIT* dans les commandes *DELETE*.
- L'option *DELAYED* dans les commandes *INSERT* et *REPLACE*.
- L'option *LOW\_PRIORITY* dans les commandes *INSERT*, *REPLACE*, *DELETE* et *UPDATE*.
- Utilisation de *LOAD DATA INFILE*. Dans certains cas, cette syntaxe est compatible avec la fonction *LOAD DATA INFILE* d'Oracle. Elle évite dans beaucoup de cas d'avoir à écrire un programme spécifique pour charger des fichiers dans une base. Cette commande est capable de lire de nombreux formats différents.

Voici comment on utilise la commande *LOAD DATA INFILE* pour insérer en une seule fois le contenu du fichier `'films.txt'` dans la table `'films'` :

```
LOAD DATA LOCAL INFILE 'films.txt' INTO TABLE films FIELDS TERMINATED BY ';' ;
```

- L'option *LOCAL* indique au serveur que le fichier se trouve sur la machine du client *mysql*. Par défaut, le serveur cherche le fichier sur sa propre machine, dans le répertoire contenant la base de données.
- Si le client *mysql* n'a pas été lancé dans le répertoire où se trouve le fichier '*films.txt*', il faut indiquer le chemin complet du fichier.
- Enfin il existe de nombreuses options pour indiquer le format du fichier. Ici on indique qu'une ligne dans le fichier correspond à une ligne dans la table, et que les valeurs des attributs dans le fichier sont séparées par des points-virgules.
- La commande *OPTIMIZE TABLE..*
- La commande *SHOW*.
- Les chaînes de caractères peuvent être définies avec " ou ', et pas seulement avec '.
- Utilisation du caractère d'échappement '\ '.
- La commande *SET OPTION*.
- Vous n'avez pas à nommer expressément toutes les colonnes dans une clause *GROUP BY*. Mais cela peut permettre d'accélérer quelques requêtes très spécifiques
- Pour aider les utilisateurs d'autres environnements SQL, MySQL dispose d'aliases pour de nombreuses fonctions. Par exemple, toutes les fonctions sur les chaînes de caractères supportent la syntaxe ANSI SQL et ODBC.
- MySQL accepte les opérateurs logiques || et && (*OR* et *AND*, comme en langage C). Sous MySQL, || et OR sont synonymes, tout comme le sont && et AND. A cause de ce double emploi, MySQL n'accepte pas les opérateurs ANSI SQL || comme additionneur de chaînes; (utilisez à la place *CONCAT()* ). Etant donné que *CONCAT()* prend un nombre arbitraire d'arguments, il est facile de remplacer ||.
- *CREATE DATABASE* ou *DROP DATABASE*.
- L'opérateur % est synonyme de *MOD()*. C'est à dire que,  $N \% M$  est équivalent à *MOD(N,M)*. % est hérité du langage C, et permet la compatibilité avec PostgreSQL.
- Les opérateurs =, <>, <=, <, >=, >, <<, >>, <=>, *AND*, *OR* ou *LIKE* peuvent être utilisés dans les comparaisons de colonnes avec la clause *FROM* dans les commandes *SELECT*. Par exemple :  

```
SELECT coll=1 AND col2=2 FROM nom_table
```
- La fonction *MYSQL\_INSERT\_ID()*.
- Les opérateurs d'expressions régulières *REGEXP* et *NOT REGEXP*.
- *CONCAT()* ou *CHAR()* avec plus d'un argument. (Avec MySQL, ces fonctions peuvent prendre un nombre arbitraire d'arguments).
- Les fonctions supplémentaires *BIT\_COUNT()*, *ELT()*, *FROM\_DAYS()*, *FORMAT()*, *IF()*, *PASSWORD()*, *ENCRYPT()*, *md5()*, *ENCODE()*, *DECODE()*, *PERIOD\_ADD()*, *PERIOD\_DIFF()*, *TO\_DAYS()*, ou *WEEKDAY()*.
- Utilisation de *TRIM()* pour supprimer les espaces de début et fin de chaîne. ANSI SQL ne supporte que la suppression de caractères uniques.
- Les fonctions de *GROUP BY*: *STD()*, *BIT\_OR()* et *BIT\_AND()*.
- Utilisation de *REPLACE* à la place de *DELETE + INSERT..*
- La commande *FLUSH flush\_option*.

## 99.2. Fonctionnalités manquantes

### Sous sélection :

La requête suivante ne fonctionne pas encore sous MySQL:

```
SELECT * FROM table1 WHERE id IN (SELECT id FROM table2)
SELECT * FROM table1 WHERE id NOT IN (SELECT id FROM table2)
```

Cependant, il est souvent possible de se passer d'une sous sélection en utilisant les jointures :

```
SELECT table1.* FROM table1,table2 WHERE table1.id=table2.id
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id where
table2.id IS NULL
```

Pour les sous requêtes compliquées, vous pouvez toujours créer une table temporaire, et y appliquer votre requête.

### SELECT INTO TABLE :

MySQL ne supporte pas encore cette extension du SQL Oracle.: `SELECT ... INTO TABLE ...`. A la place, MySQL supporte la syntaxe de la norme ANSI SQL `INSERT INTO ... SELECT ...`, ce qui est pratiquement la même chose.

Alternativement, vous pouvez utiliser `SELECT INTO OUTFILE...` ou `CREATE TABLE ... SELECT` pour résoudre votre problème.

### Transactions :

Les transactions ne sont pas encore supportées par MySQL. Le serveur va bientôt accepter des opérations atomiques, ce qui permettra des transactions, mais sans le rollback. Avec les opérations atomiques, vous pourrez exécuter des groupes de commandes `INSERT/SELECT/` avec n'importe quelle commande, et être sûr qu'il n'y aura aucune interférence avec un autre thread. Dans ce contexte, vous n'aurez alors pas besoin de rollback. Actuellement, vous pouvez empêcher les autres threads d'interférer en utilisant les fonctions `LOCK TABLES` et `UNLOCK TABLES`.

Pour éviter d'utiliser le `ROLLBACK`, vous pouvez suivre la stratégie suivante :

1. Utilisez `LOCK TABLES <nom des tables>` pour verrouiller les tables auxquelles vous accédez
2. Testez les conditions d'utilisation.
3. Modifiez si tout est OK.
4. Utilisez `UNLOCK TABLES` pour libérer la table.

Généralement, cette méthode est beaucoup plus rapide que les transactions, et le `ROLLBACK` est souvent possible, mais pas toujours. Le seul point critique est que si le thread est tué au milieu de la modification, les verrous seront libérés, mais une partie des modifications ne sera pas faite.

Vous pouvez aussi utiliser les fonctions qui modifient un seul enregistrement à la fois. Vous pouvez créer des applications très efficaces avec la technique suivante :

- Modifier un champ par rapport à sa valeur actuelle
- Modifier seulement les champs qui ont changés

Par exemple, lorsque vous modifiez les informations concernant un client, ne modifiez que les informations qui ont changées, et non pas celles qui sont restées constantes. La recherche des valeurs est faite avec la clause `WHERE` de la commande `UPDATE`. Si l'enregistrement a changé, on peut retourner au client un message du type : "Les informations n'ont pas été modifiées, car un autre utilisateur est en train de modifier les valeurs ". Alors, on affiche l'ancienne valeur et la nouvelle, ce qui permet à l'utilisateur de décider quelle version utiliser.

Cela fournit un mécanisme du genre 'verrouillage de colonne' mais c'est en fait un peu mieux, car seules les colonnes qui en ont besoin sont utilisées. Une commande `UPDATE` ressemblera alors à ceci :

```
UPDATE tablename SET pay_back=pay_back+'différence de valeur' WHERE ...
```

Comme vous pouvez le voir, c'est une méthode très efficace, et qui fonctionne même si un autre client a changé la valeur entre temps.

Dans certains cas, l'utilisateur a demandé le `ROLLBACK` et/ou `LOCK TABLES` dans le but de gérer des identifiants uniques dans des tables. Il vaut mieux utiliser le type de colonne `AUTO_INCREMENT` et la fonction SQL `LAST_INSERT_ID()`.

**Fonctions enregistrées et triggers :**

Une fonction enregistrée est un ensemble de commandes SQL qui peut être compilé et enregistré sur le serveur. Une fois fait, les clients peuvent se référer à cette fonction pour exécuter l'ensemble des commandes. Cela accélère le traitement des requêtes, car elles n'ont pas à être analysées, et moins d'informations circulent entre le client et le serveur. Il est aussi possible d'élever le niveau de conception, en bâtissant des bibliothèques.

Un trigger est une fonction enregistrée qui est invoquée à chaque fois qu'un événement particulier survient. Par exemple, vous pourriez installer une fonction qui sera lancée à chaque fois qu'un enregistrement sera effacé dans une table de transaction, pour effacer automatiquement les informations correspondantes dans les tables de clients. Lors de modifications ultérieures, MySQL sera capable de gérer les fonctions enregistrées, mais pas les triggers. En général, les triggers ralentissent le serveur, même pour des requêtes pour lesquelles ils ne sont pas appelés.

**Clés étrangères :**

Remarque : les clés externes en SQL ne sont pas utilisées pour effectuer des regroupements de table, mais pour assurer l'intégrité référentielle. Si vous voulez lire des informations depuis plusieurs tables avec une commande SELECT, c'est un regroupement !. Exemple :

```
SELECT * from table1,table2 where table1.id = table2.id
```

**Il n'y a pas besoin de clé étrangère pour joindre deux tables :**

La seule chose que MySQL ne fait pas est de vérifier (CHECK) que les clés que vous utilisez existent vraiment dans la table que vous référencez, et qu'il n'efface pas de lignes dans une table avec une définition de clé étrangère. Si vous utilisez vos clés de manière habituelle, cela fonctionnera parfaitement.

**Exemples :**

```
CREATE TABLE persons (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id) )

CREATE TABLE shirts (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES persons,
    PRIMARY KEY (id) )
```

```
INSERT INTO persons VALUES (NULL, 'Antonio Paz')
```

```
INSERT INTO shirts VALUES
(NULL, 'polo', 'blue', LAST_INSERT_ID()),
(NULL, 'dress', 'white', LAST_INSERT_ID()),
(NULL, 't-shirt', 'blue', LAST_INSERT_ID())
```

```
INSERT INTO persons VALUES (NULL, 'Lilliana Angelovska')
```

```
INSERT INTO shirts VALUES
(NULL, 'dress', 'orange', LAST_INSERT_ID()),
(NULL, 'polo', 'red', LAST_INSERT_ID()),
(NULL, 'dress', 'blue', LAST_INSERT_ID()),
(NULL, 't-shirt', 'white', LAST_INSERT_ID())
```

```
SELECT * FROM persons
```

```
+-----+-----+
| id | name                |
+-----+-----+
|  1 | Antonio Paz         |
|  2 | Lilliana Angelovska |
+-----+-----+
```

```
SELECT * FROM shirts
```

id	style	color	owner
1	polo	blue	1
2	dress	white	1
3	t-shirt	blue	1
4	dress	orange	2
5	polo	red	2
6	dress	blue	2
7	t-shirt	white	2

```
SELECT s.* FROM persons p, shirts s
WHERE p.name LIKE 'Lilliana%'
AND s.owner = p.id
AND s.color <> 'white'
```

id	style	color	owner
4	dress	orange	2
5	polo	red	2
6	dress	blue	2

La syntaxe *FOREIGN KEY* de MySQL n'existe que pour la compatibilité avec les commandes *CREATE TABLE* des autres bases SQL : elle n'a aucun effet. La syntaxe *FOREIGN KEY* sans la partie *ON DELETE . . .* n'est utilisée que pour des raisons de documentation. Certaines applications ODBC l'utilise pour générer des clauses *WHERE* automatiques, mais il est généralement simple à remplacer. *FOREIGN KEY* est parfois utilisé comme contrainte, mais ce genre de vérification n'est pas nécessaire si les lignes ont été insérées dans l'ordre. MySQL ne supporte que ces clauses, car certaines applications en ont besoin (qu'elles fonctionnent ou pas).

Avec MySQL, vous pouvez contourner le problème sans la clause *ON DELETE . . .* en ajoutant une commande *DELETE* adéquate lors de l'effacement d'un enregistrement d'une table qui a une clé externe. En pratique, c'est aussi rapide (voire plus), et beaucoup plus portable que les clés externes.

Dans un futur proche, nous allons implémenter *FOREIGN KEY* de manière à sauver les informations dans la table de spécifications, pour qu'elles soient accessibles par *mysqldump* et ODBC.

### **Vues :**

MySQL ne supporte pas les vues, mais c'est sur la liste des fonctionnalités futures.

### **Opérations ensemblistes :**

La norme SQL ANSI comprend des opérations qui considèrent les tables comme des ensembles, et effectuent des intersections, des unions ou des différences avec les mots clés *INTERSECT*, *UNION* ou *EXCEPT*. Chaque opérateur s'applique à deux tables de schéma identique (même nombre d'attributs, mêmes noms, mêmes types).

#### **Exemples :**

Sélectionner toutes les années des films et des artistes :

```
SELECT annee FROM films UNION SELECT annee_naissance AS annee FROM artistes ;
```

Sélectionner les noms de rôles qui sont aussi des titres de films :

```
SELECT nom_role AS nom FROM roles INTERSECT SELECT titre AS nom FROM films ;
```

Sélectionner les noms de rôles qui ne sont pas des titres de films :

```
SELECT nom_role AS nom FROM roles EXCEPT SELECT titre AS nom FROM films ;
```

MySQL ignore ces trois opérateurs. Ce n'est pas grave pour *INTERSECT* qui peut être exprimé avec une jointure.

Il n'y a pas d'équivalent à *UNION* mais cette opération est rarement utile. En revanche *EXCEPT* permet d'exprimer des négations, à savoir toutes les requêtes où on effectue une recherche en prenant des lignes qui n'ont pas telle ou telle propriété. Il s'agit de la véritable limite de MySQL car il n'y a pas d'équivalent.

**Début de commentaire :**

Sur d'autres bases SQL les commentaires commencent par `'--'` (2 signes moins). MySQL utilise `'#'` pour débiter un commentaire, même si MySQL supprime aussi les lignes qui commencent par `'--'`. Vous pouvez aussi utiliser le style de commentaires du langage C : `/* Ceci est un commentaire */` avec MySQL..

MySQL n'accepte pas les commentaires commençant par `'--'` car ce style de commentaire obsolète a déjà causé de nombreux problèmes avec les requêtes générées automatiquement, lorsque la base utilise un code comme celui ci :

```
UPDATE nom_table SET credit=credit-!paiement!
```

Mais que se passe-t-il si la valeur de paiement est négative?

## 100. Programmation MySQL

### 100.1. Syntaxe des chaînes et nombres

Une chaîne est une séquence de caractères, entourée par des guillemets simples (') ou doubles("").

Exemple :

'une chaîne'

"une autre chaîne"

A l'intérieur d'une chaîne, on trouve des séquences spéciales. Celles-ci commencent avec le caractère backslash ('\\'), dit aussi caractère d'échappement.

**MySQL reconnaît les séquences suivantes :**

\\0	ASCII 0 (NUL) le caractère nul.
\\n	Une nouvelle ligne.
\\t	Une tabulation.
\\r	Un retour chariot.
\\b	Un effacement.
\\'	Un guillemet simple (').
\\"	Un guillemet double (").
\\	Un backslash (\\).
\\%	Un pourcentage '%'. Cela permet de rechercher le caractère '%' dans un contexte où il pourrait être considéré comme un caractère spécial.
\\_	Un souligné '_'. Cela permet de rechercher le caractère '_' dans un contexte où il pourrait être considéré comme un caractère spécial.'

**Nombres :**

Les entiers sont représentés comme une séquence de chiffres. Les nombres réels utilisent le point ('.') comme séparateur décimal. Les entiers et les réels peuvent être précédés par le signe moins ('-'), pour indiquer un nombre négatif.

- Lorsqu'un nombre entier est utilisé avec un nombre réel, il est considéré lui aussi, comme un nombre réel.
- Valeurs hexadécimales : MySQL supporte les valeurs hexadécimales. Dans un contexte numérique, elles se comportent comme des entiers (précision 64bits). Dans un contexte de chaînes, elles se comportent comme des chaînes binaires dont chaque paire de digits sera convertie en caractère.

*SELECT 0xa+0* #donnera '10'

*SELECT 0x5061756c* #donnera 'Paul'

Les chaînes hexadécimales sont souvent utilisées avec ODBC pour donner des valeurs aux colonnes *BLOB*.

- La valeur *NULL* signifie : 'aucune information'. Cette valeur est différente de 0 ou de la chaîne vide. *NULL* est parfois représenté par \N quand on utilise un fichier d'import ou d'export (*LOAD DATA INFILE*, *SELECT ... INTO OUTFILE*).

### 100.2. Types des colonnes et espaces mémoire requis

**Types numériques :**

type	espace mémoire requis
TINYINT	1 octet
SMALLINT	2 octets
MEDIUMINT	3 octets
INT	4 octets
INTEGER	4 octets
BIGINT	8 octets
FLOAT(4)	4 octets
FLOAT(8)	8 octets
FLOAT	4 octets
DOUBLE	8 octets
DOUBLE PRECISION	8 octets
REAL	8 octets
DECIMAL(M,D)	M octets (D+2, si M < D)



**Types date et heure :**

type	espace mémoire requis
DATETIME	8 octets
DATE	3 octets
TIMESTAMP	4 octets
TIME	3 octets
YEAR	1 octet

**Types chaîne :**

type	espace mémoire requis
CHAR(M)	M octets, $1 \leq M \leq 255$
VARCHAR(M)	L+1 bytes, avec $L \leq M$ et $1 \leq M \leq 255$
TINYBLOB, TINYTEXT	L+1 octets, avec $L < 2^8$
BLOB, TEXT	L+2 octets, avec $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	L+3 octets, avec $L < 2^{24}$
LONGBLOB, LONGTEXT	L+4 octets, avec $L < 2^{32}$
ENUM('value1','value2',...)	1 ou 2 octets, suivant le nombre de valeurs dans l'énumération (65535 au maximum)
SET('value1','value2',...)	1, 2, 3, 4 ou 8 octets, suivant le nombre de membres de l'ensemble (64 membres au maximum)

- Les types *VARCHAR*, *BLOB* et *TEXT* sont des types à longueur variable, dont la taille de stockage dépend plus de la valeur qui leur est assignée que de leur taille maximale. Par exemple, une colonne de type *VARCHAR*(10) peut contenir une chaîne de 10 caractères au maximum. La taille réelle nécessaire est la longueur de la chaîne, plus 1 octet, qui stockera la taille réelle de la chaîne. Par exemple, la chaîne 'abcd' occupe 5 octets.
- Les types *BLOB* et *TEXT* ont besoin de 1, 2, 3 ou 4 octets pour stocker la taille de la colonne, en fonction du type.
- Si une table possède au moins une colonne de longueur variable, l'enregistrement sera aussi de longueur variable. Il faut noter que lorsqu'une table est créée, MySQL peut, sous certaines conditions, changer le type d'une colonne de longueur variable en un type de colonne de longueur fixe, et vice-versa..
- La taille d'un objet *ENUM* est déterminé par le nombre d'énumérations différentes. 1 octet est suffisant pour décrire une énumération qui a jusqu'à 255 valeurs différentes; 2 octets sont nécessaires pour décrire une énumération qui aurait jusqu'à 65535 valeurs différentes.
- La taille d'un objet *SET* est déterminée par le nombre d'éléments distincts qu'il contient. Si la taille d'un SET est N, le SET occupera (N+7)/8 octets, arrondie aux entiers 1,2,3,4, ou 8 octets. Un ensemble peut contenir jusqu'à 64 éléments.
- MySQL tente d'interpréter un grand nombre de format de date, l'année devra toujours être placée à gauche. Les dates doivent être données dans l'ordre année-mois-jour (ex : '98-09-04'), plutôt que dans l'ordre mois-jour-année ou l'ordre jour-mois-année utilisés habituellement. (ex : '09-04-98', '04-09-98'). MySQL convertit automatiquement une date ou une heure en un nombre, si cette valeur est utilisée dans un contexte numérique, et vice-versa. Quand MySQL rencontre une valeur pour une date ou une heure qui n'est pas valide, il la convertit en valeur 'zéro'. Les problèmes de dépassement de capacités sont réglés comme pour les types numériques, en ramenant la valeur au maximum ou au minimum de l'intervalle autorisé.



## 101. Connexion et accès à la base de données MySQL

Pour que vous puissiez créer votre propre base de données, votre administrateur doit configurer un utilisateur et une base de données, et vous fournir le nom d'utilisateur, le mot de passe, et le nom de la base de données qu'il a choisis.

Quand vous avez recours aux services d'un fournisseur d'accès à distance, ce dernier propose généralement d'entrer des commandes par l'intermédiaire d'une interface Web d'administration de bases de données MySQL : *phpMyAdmin*. Cet outil est très populaire, et fournit un environnement de travail graphique très convivial.

Dans ce support, nous nous consacrerons uniquement à l'interpréteur de commandes *mysql*.

On peut utiliser indifféremment les majuscules et les minuscules pour les mots clés de SQL. De même les sauts de ligne, les tabulations et les espaces successifs dans un ordre SQL équivalent à un seul espace pour l'interpréteur et peuvent donc être utilisés librement pour clarifier la commande.

## 102. Créer et sélectionner une base de données MySQL

Si l'administrateur vous a créé une base de données pour vous, alors vous pouvez directement commencer à l'utiliser. Sinon, il vous faut la créer vous même.

Exemple :

```
CREATE DATABASE menagerie
```

Sous Unix, les noms de base de données sont sensibles à la casse (contrairement aux mots clés SQL), donc il faudra faire référence à votre base de données sous le nom **menagerie**, et non pas **Menagerie**, **MENAGERIE** ou tout autre variante. Sous Windows, cette restriction ne s'applique pas, même si vous devez faire référence à vos bases et tables de la même manière tout au long d'une même commande.

Créer une base de données ne la sélectionne pas automatiquement. Il faut le faire explicitement. Pour faire de **menagerie** votre base courante, il faut utiliser la commande:

```
USE menagerie
```

La base n'a besoin d'être créée qu'une seule fois, mais il faudra la sélectionner à chaque fois que vous commencerez une session MySQL.

## 103. Fonctions d'accès à MySQL

Ces fonctions vont vous permettre d'accéder aux bases de données MySQL :

**int mysql\_affected\_rows([int link\_identifiant]);** Retourne le nombre de lignes modifiées lors de la dernière requête *INSERT*, *UPDATE* ou *DELETE* sur le serveur associé à l'identifiant de connexion (de lien) *link\_identifiant*. Si cet identifiant n'est pas précisé, cette fonction utilise la dernière connexion ouverte.

**int mysql\_close([int link\_identifiant]);** Ferme la connexion au serveur MySQL associée à l'identifiant de lien *link\_identifiant*. Si cet identifiant n'est pas spécifié, cette commande s'applique à la dernière connexion ouverte.

**int mysql\_connect([string hostname [:port] [:path/to/socket] ] , [string username] , [string password]);** Etablit une connexion à un serveur MySQL. Tous les arguments sont optionnels, et si ils manquent les valeurs par défaut sont utilisées ('localhost', nom du propriétaire du process, mot de passe vide).

**int mysql\_create\_db(string database\_name, [int link\_identifiant]);** Tente de créer une nouvelle base de données *database\_name* sur le serveur associé à l'identifiant de lien *link\_identifiant*, ou la dernière connexion ouverte.

**int mysql\_data\_seek(int result\_identifiant, int row\_number);** Déplace le pointeur interne de résultat, dans le résultat associé à l'identifiant de résultat *result\_identifiant*. Il le fait pointer à la ligne *row\_number*. Le prochain appel à *mysql\_fetch\_row()* retournera cette ligne.

**int mysql\_db\_query(string database\_name, string query, [int link\_identifiant]);** Sélectionne la base de données *database\_name* et envoie la requête *query* à un serveur MySQL. Si l'identifiant de lien *link\_identifiant* n'est pas précisé, la fonction prendra par défaut la dernière base de données ouverte sur le serveur, et, si elle n'en trouve pas, elle tentera de se connecter, en utilisant la fonction *mysql\_connect()* sans argument. Elle retourne un identifiant de résultat si la requête aboutit, et faux (FALSE) sinon.

**int mysql\_drop\_db(string database\_name, [int link\_identifiant]);** Essaie d'effacer la base de données MySQL *database\_name* entière sur le serveur associé à l'identifiant de lien *link\_identifiant*. Elle retourne vrai (TRUE) en cas de succès, et faux (FALSE) sinon.

**int mysql\_errno([int link\_identifiant]);** Retourne le numéro d'erreur de la dernière requête.

**int mysql\_error([int link\_identifiant]);** Retourne le texte associée avec l'erreur générée lors de la dernière requête.

**array mysql\_fetch\_array(int result, [int result\_type]);** Retourne une ligne de résultat sous la forme d'un tableau associatif en utilisant les noms des champs comme indices. L'option *result\_type* est une constante qui peut prendre les valeurs suivantes : *MYSQL\_ASSOC*, *MYSQL\_NUM*, et *MYSQL\_BOTH*.

**object mysql\_fetch\_field(int result, [int field\_offset]);** Retourne les informations concernant les colonnes d'un résultat sous la forme d'un objet. Elle sert à obtenir des informations à propos des champs, dans certaines requêtes. Si l'offset du champ *field\_offset* n'est pas spécifié, le champ suivant est retourné.

**array mysql\_fetch\_lengths(int result);** Retourne la taille de chaque colonne d'un résultat. Elle stocke la taille de chaque colonne de la dernière ligne retournée par *mysql\_fetch\_row()* dans un tableau, en commençant à la position 0.

**object mysql\_fetch\_object(int result, [int result\_type]);** Retourne une ligne de résultat sous la forme d'un objet. Elle est identique à *mysql\_fetch\_array()*, à la différence qu'elle retourne un objet à la place d'un tableau. Vous pourrez ainsi accéder aux valeurs des champs par leur nom, et non plus par leur offset (les nombres ne sont pas des noms MySQL). L'argument optionnel *result\_type* est une constante qui peut prendre les valeurs suivantes : *MYSQL\_ASSOC*, *MYSQL\_NUM*, et *MYSQL\_BOTH*. Concernant la vitesse, cette fonction est aussi rapide que *mysql\_fetch\_array()*, et presque aussi rapide que *mysql\_fetch\_row()* (la différence est insignifiante).

**array mysql\_fetch\_row(int result);** Retourne une ligne de résultat sous la forme d'un tableau. Elle va rechercher une ligne dans le résultat associé à l'identifiant de résultat *result*. La ligne est retournée sous la forme d'un tableau.. Chaque colonne est enregistrée sous la forme d'un tableau commençant à la position 0. L'appel suivant à *mysql\_fetch\_row()* retournera la ligne suivante dans le résultat, ou faux (FALSE) s'il n'y a plus de ligne disponible.

**string mysql\_field\_name(int result, int field\_index);** Retourne le nom d'un champ à partir de son index. Les arguments de la fonction sont l'identifiant de résultat *result*, et l'index de champ *field\_index* numéroté à partir de 0. Ex : *mysql\_field\_name(\$result,2);* retournera le nom du troisième champ dans le résultat associé à *\$result*. Pour des raisons de compatibilité ascendante, *mysql\_fieldname()* peut encore être utilisée.

**int mysql\_field\_seek(int result, int field\_index);** Place le pointeur de résultat à l'index de champ *field\_index*.

**string mysql\_field\_table(int result, int field\_index);** Retourne le nom de la table où se trouve une colonne. Pour des raisons de compatibilité *mysql\_fieldtable()* peut encore être utilisée.

**string mysql\_field\_type(int result, int field\_index);** Retourne le type de la colonne spécifiée dans le résultat courant *result*. Il vaudra "int", "real", "string", "blob", ou d'autres, comme détaillé dans la documentation MySQL. Pour des raisons de compatibilité ascendante, *mysql\_fieldtype()* peut encore être utilisée.

**string mysql\_field\_flags(int result, int field\_index);** Retourne le sémaphore associé à la colonne spécifiée dans le résultat courant. Les sémaphores sont retournés comme des mots, séparés par des espaces, ce qui les rend facile à séparer, avec la commande *explode()*. Les valeurs suivantes (pour une version suffisamment récente de MySQL) sont disponibles : "not\_null", "primary\_key", "unique\_key", "multiple\_key", "blob", "unsigned", "zerofill", "binary", "enum", "auto\_increment", "timestamp". Pour des raisons de compatibilité ascendante, *mysql\_fieldflags()* peut encore être utilisée.

**int mysql\_field\_len(int result, int field\_index);** Retourne la taille du champ spécifié par *field\_index*. Pour des raisons de compatibilité ascendante, *mysql\_fieldlen()* peut encore être utilisée.

**int mysql\_free\_result(int result);** Libère toute la mémoire associée à l'identifiant de résultat *result*. *mysql\_free\_result()* ne doit être appelée que si vous avez utilisé trop de mémoire durant l'exécution de votre script. Pour des raisons de compatibilité ascendante, *mysql\_freeresult()* peut encore être utilisée.

**int mysql\_insert\_id([int link\_identifier]);** Retourne l'identifiant généré par un champ de type *AUTO\_INCREMENTED* lors de la dernière requête *INSERT*.

**int mysql\_list\_fields(string database\_name, string table\_name, [int link\_identifier]);** Liste les champs du résultat MySQL. Elle recherche les informations à propos de la table *table\_name*. Les arguments sont la base de données *database\_name*, et le nom de la table *table\_name*. Un pointeur de résultat est retourné, et pourra être passé à *mysql\_field\_flags()*, *mysql\_field\_len()*, *mysql\_field\_name()*, ou *mysql\_field\_type()*. Un identifiant de résultat est un entier positif. La fonction retourne -1 si une erreur survient. Une chaîne décrivant l'erreur rencontrée sera placée dans la variable *\$phperrormsg*, et, à moins que la fonction n'ait été appelée sous la forme *@mysql()*, cette erreur sera aussi affichée. Pour des raisons de compatibilité ascendante, *mysql\_listfields()* est encore disponible.

**int mysql\_list\_dbs([int link\_identifier]);** Retourne un identifiant de résultat, qui contiendra le nom des bases de données disponibles sur le serveur MySQL.. Pour des raisons de compatibilité ascendante, *mysql\_listdbs()* est encore disponible.

**int mysql\_list\_tables(string database\_name, [int link\_identifier]);** Prend le nom d'une base de données comme argument, et retourne un identifiant de résultat, qui contiendra la liste des tables. La fonction *mysql\_tablename()* est le meilleur moyen d'extraire le nom des tables depuis l'identifiant de résultat. Pour des raisons de compatibilité ascendante, *mysql\_listtables()* est encore disponible.

**int mysql\_num\_fields(int result);** Retourne le nombre de champs d'un résultat. Pour des raisons de compatibilité ascendante *mysql\_numfields()* est encore disponible.

**int mysql\_num\_rows(int result);** Retourne le nombre de lignes d'un résultat. Pour des raisons de compatibilité ascendante *mysql\_numrows()* est encore disponible.

**int mysql\_pconnect([string hostname [:port] [:path/to/socket] ] , [string username] , [string password]);** Ouvre une connexion persistante à un serveur MySQL. Elle retourne un lien persistant positif en cas de succès, et sinon faux (FALSE) en cas d'erreur. Tous les arguments sont optionnels, et des valeurs par défaut sont utilisées en cas d'omission ('localhost', nom d'utilisateur propriétaire du processus, mot de passe vide).

Le nom de l'hôte peut aussi inclure le numéro de port, c'est à dire "hostname:port" ou un chemin jusqu'à la socket ":path/to/socket" pour l'hôte local.

*mysql\_pconnect()* se comporte exactement comme *mysql\_connect()*, mais avec deux différences majeures :

Premièrement, lors de la connexion, la fonction essaie de trouver une connexion permanente déjà ouverte sur cet hôte, avec les mêmes noms d'utilisateur et de mot de passe. Si une telle connexion est trouvée, son identifiant est retourné, sans ouvrir de nouvelle connexion.

Deuxièmement, la connexion au serveur MySQL ne sera pas terminée avec la fin du script. Au lieu de cela, le lien sera conservé pour un prochain accès (*mysql\_close()* ne terminera pas une connexion persistante établie par *mysql\_pconnect()*). C'est pourquoi ce type de connexion est dite 'persistant'.

**int mysql\_query(string query, [int link\_identifier]);** Envoie une requête SQL à la base de données actuellement active sur le serveur MySQL. Si *link\_identifier* n'est pas précisé, la dernière connexion est utilisée. Si aucune connexion n'a été ouverte, la fonction tentera d'en ouvrir une, avec la fonction *mysql\_connect()* mais sans aucun paramètre (c'est à dire avec les valeurs par défaut). La chaîne de requête ne devrait pas se terminer par un point virgule.

*mysql\_query()* retourne un identifiant de résultat que vous pouvez passer à *mysql\_result()*.

*mysql\_query()* retourne vrai (TRUE, ou non-zéro) ou faux (FALSE), pour indiquer le succès ou l'échec de la requête. En cas de retour TRUE, la requête était valide et a pu être exécutée sur le serveur. Cela n'indique pas le nombre de lignes affectées, ou retournées. Il est parfaitement possible qu'une requête valide n'affecte aucune ligne ou ne retourne aucune ligne.

*mysql\_query()* échouera aussi et retournera aussi FALSE si les droits d'accès ne sont pas suffisants.

**int mysql\_result(int result, int row, [mixed field]);** Retourne le contenu d'un champ dans un résultat MySQL. L'argument de champ peut être un index de champ, ou le nom du champ, ou le nom de la table + point + le nom du champ (table.champ). Si la colonne a été aliassée, utilisez de préférence l'alias.

Lorsque vous travaillez sur des résultats de grande taille, vous devriez utiliser une des fonctions qui vont rechercher une ligne entière dans un tableau. Ces fonctions sont NETTEMENT plus rapide. De plus, l'utilisation d'un index numérique est aussi beaucoup plus rapide que de spécifier un nom littéral.

Les appels *mysql\_result()* ne devraient pas être mélangés avec d'autres fonctions qui travaillent aussi sur le résultat. Les fonctions suivantes à haut rendement sont RECOMMANDÉES : *mysql\_fetch\_row()*, *mysql\_fetch\_array()*, et *mysql\_fetch\_object()*.

**int mysql\_select\_db(string database\_name, [int link\_identifier]);** Selectionne une base de données MySQL. Elle retourne vrai (TRUE) en cas de succès, faux (FALSE) sinon.

*mysql\_select\_db()* change la base de données active sur la connexion représentée par l'identifiant de connexion *link\_identifier*. Si aucun identifiant n'est spécifié, la dernière connexion est utilisée. S'il n'y a pas de dernière connexion, la fonction tentera de se connecter seule, avec *mysql\_connect()* et les paramètres par défaut. Toutes les requêtes suivantes avec *mysql\_query()* seront faites avec la base de données active. Pour des raisons de compatibilité ascendante *mysql\_selectdb()* est encore disponible.

**string mysql\_tablename(int result, int link\_identifier);** Retourne le nom de la table que contient le champ spécifié par *link\_identifier*. Elle prend le pointeur de résultat *result* obtenu avec *mysql\_list\_tables()* ou bien un index entier, et retourne le nom de la table. La fonction *mysql\_num\_rows()* peut être utilisée pour déterminer le nombre de tables dans le pointeur de résultat *result*.

Exemple :

```
mysql_pconnect("localhost:3306");
$result = mysql_list_tables("france");
for ($i=0 ; $i < mysql_num_rows($result) ; $i++)
{
    $tb_names[$i] = mysql_tablename($result, $i);
    echo $tb_names[$i] . "<BR>";
}
```

### 103.1. mysql\_connect

Avant de penser à travailler avec votre base de données, vous devez dans un premier temps vous connecter au serveur. Pour ce faire, PHP fournit la fonction `mysql_connect()`.

Cette fonction ne nécessite aucun argument, mais accepte jusqu'à trois chaînes : le nom d'hôte (*hostname*), un nom d'utilisateur (*username*), et un mot de passe (*password*). Si vous omettez l'un de ces arguments, la fonction suppose que *localhost* est le nom d'hôte et qu'aucun mot de passe ou nom d'utilisateur n'a été défini dans la table *mysql user*, à moins que des valeurs n'aient été définies dans le fichier *php.ini*. Les valeurs par défaut utilisées sont : *'localhost'*, nom du propriétaire du process, mot de passe vide.

`mysql_connect()` renvoie un identifieur de lien si la connexion est réussie, sinon faux (FALSE). Vous pouvez stocker cette valeur de retour dans une variable de façon à être en mesure de continuer à travailler avec le serveur de base de données.

Si un second appel à `mysql_connect()` est fait avec les mêmes arguments, PHP ne va pas ouvrir une nouvelle connexion, mais va retourner l'identifieur de la connexion déjà ouverte.

Le lien sera fermé automatiquement dès que l'exécution du script sera terminée, à moins d'être fermé explicitement avec `mysql_close()`.

Syntaxe :

```
Int identifieur = mysql_connect([string hostname [:port] [:/path/jusqu_a/socket]], [string username], [string password]);
```

Exemple :

```
$lien = mysql_connect("localhost","utilisateur1","mdp1") ;  
if( !$lien )  
    die("Connexion au serveur MySQL impossible ! ") ;
```

la fonction `mysql_pconnect()` est similaire à `mysql_connect()` et peut être utilisée lorsque vous utilisez PHP conjointement à Apache. La différence est que la connexion ne prend pas fin lors de l'achèvement de l'exécution de votre script ou lors de l'appel de `mysql_close()` qui met fin à une connexion standard au serveur MySQL. La connexion reste active, attendant un nouvel appel de `mysql_pconnect()`.

### 103.2. mysql\_close

`mysql_close()` ferme la connexion au serveur MySQL associée à l'identifieur. Si cet identifieur n'est pas spécifié, cette commande s'applique à la dernière connexion ouverte.

Notez que cette commande n'est pas nécessaire, car toutes les connexions non persistantes seront automatiquement fermées à la fin du script.

`mysql_close()` ne ferme pas les connexions persistantes générées par `mysql_pconnect()`.

La fonction retourne vrai (TRUE) en cas de succès, et faux (FALSE) sinon.

Syntaxe :

```
int mysql_close([int identifieur] );
```

### 103.3. mysql\_select\_db

Après avoir établi une connexion avec MySQL, vous devez choisir la base de données avec laquelle vous désirez travailler. La fonction `mysql_select_db()` permet de réaliser cette opération.

Cette fonction nécessite un nom de base de données et accepte en option un identifieur de lien. Si vous omettez celui-ci, l'identifieur supposé sera celui renvoyé lors de la dernière connexion au serveur.

S'il n'y a pas de dernière connexion, la fonction tentera de se connecter seule, avec les paramètres par défaut.

`mysql_select_db()` renvoie vrai (TRUE) si la base de données existe et si vous êtes en mesure d'y accéder, sinon faux (FALSE).

Pour des raisons de compatibilité ascendante `mysql_selectdb()` est encore disponible.

Syntaxe :

```
int mysql_select_db(string database_name, [int identifieur] );
```

Exemple :

```
$database_name = "films" ;
```

```
mysql_select_db($database_name) or die("Ouverture de la base de données $database_name impossible !") ;
```

### 103.4. mysql\_error

Jusqu'à présent, nous avons testé les valeurs de retour des fonctions MySQL et appelé `die()` pour mettre fin à l'exécution du script si un problème se produit. Vous pouvez cependant dériver afficher des messages d'information plus nombreux dans le navigateur pour aider à corriger les erreurs. MySQL définit une chaîne et un numéro d'erreur lorsqu'une opération échoue. Vous pouvez accéder au numéro d'erreur avec la fonction `mysql_errno()`, et à la chaîne avec `mysql_error()`.

`mysql_error()` retourne le texte associé à l'erreur générée lors de la dernière requête

Syntaxe :

```
string mysql_error([int identifieur] );
```

Exemple :

```
<html>
```

```
<head>
```

```
<title>Ouverture d'une connexion et sélection d'une base de données</title>
```

```
</head>
```

```
<body>
```

```
<?
```

```
echo "<form method='POST' action='#>";
```

```
$utilisateur = "utilisateur1" ;
```

```
$mdp = "mdp1" ;
```

```
$bdd = "films" ;
```

```
$lien = mysql_pconnect("localhost",$utilisateur,$mdp) ;
```

```
if( !lien )
```

```
    die("Connexion au serveur MySQL impossible ! : ". mysql_error() ) ;
```

```
echo "<h2>Connexion au serveur réussie !</h2>";
```

```
mysql_select_db($bdd) or die("Ouverture de la base de données $bdd impossible ! : ". mysql_error() ) ;
```

```
echo "<h2>Sélection de la base de données '$bdd' réussie !</h2>";
```

```
echo "</form>";
```

```
?>
```

```
</body>
```

```
</html>
```

## 103.5. mysql\_query

Pour effectuer réellement une requête SQL (select, create, delete, update , insert, ...) vous pouvez utiliser la fonction `mysql_query()`.

`mysql_query()` envoie la requête SQL à la base de données actuellement active sur le serveur MySQL. Si l'identifiant n'est pas précisé, la dernière connexion est utilisée. Si aucune connexion n'a été ouverte, la fonction tentera d'en ouvrir une, avec la fonction `mysql_connect()` mais sans aucun paramètre (c'est à dire avec les valeurs par défaut).

La chaîne de requête ne doit pas se terminer par un point virgule.

`mysql_query()` retourne un identificateur de résultat (qui permet de récupérer les résultats des requêtes) vrai (TRUE) ou faux (FALSE), pour indiquer le succès ou l'échec de la requête. En cas de retour TRUE, la requête était valide et a pu être exécutée sur le serveur. Cela n'indique pas le nombre de lignes affectées, ou retournées. Il est parfaitement possible qu'une requête valide n'affecte aucune ligne ou ne retourne aucune ligne.

`mysql_query()` échouera aussi et retournera aussi FALSE si les droits d'accès ne sont pas suffisants.

Pour connaître le nombre de lignes affectées par les commandes DELETE, INSERT, REPLACE, ou UPDATE vous pouvez appeler la fonction `mysql_affected_rows()`. Cette commande n'est pas possible après un SELECT, car elle ne fonctionne qu'après des commandes qui modifient les enregistrements. Pour connaître le nombre de lignes retournées par un SELECT, utilisez `mysql_num_rows()`.

Pour les commandes SELECT , `mysql_query()` retourne un identificateur de résultat que vous pouvez passer à `mysql_result()`. Lorsque vous avez terminé avec le résultat, libérez la mémoire avec `mysql_free_result()`.

Syntaxe :

```
int mysql_query(string requete, [int identifieur] );
```

Exemple :

```
$requete = "select distinct num_titre from films.titres" ; //sélection de tous les titres de la table 'titres' de la base de données 'films'
```

```
$result = mysql_query($requete) ;
```

```
echo "la base de données $bdd contient ".mysql_num_rows($result)." titres de films différents<br>" ;
```

L'exemple suivant décrit la création d'une table 'grades' :

```
echo "Début de la création de la table grades..." ;
```

```
echo "<br>" ;
```

```
$requete = "CREATE TABLE grades " ;
```

```
$requete.= "(" ;
```

```
$requete.= "CodeGrade INTEGER(11) UNSIGNED NOT NULL DEFAULT 0 PRIMARY KEY  
AUTO_INCREMENT," ;
```

```
$requete.= "TriGramme VARCHAR(7) BINARY NULL DEFAULT NULL," ;
```

```
$requete.= "ClairGrade VARCHAR(50) BINARY NULL DEFAULT NULL" ;
```

```
$requete.= ")" ;
```

```
$result = mysql_query($requete) ; // cette requête peut être valide et retourner zéro
```

```
if(mysql_errno())
```

```
    echo "Une erreur s'est produite lors de la création de la table grades : ".mysql_error() ;
```

```
else
```

```
    echo "Fin de la création de la table grades.<br>" ;
```

## 103.6. mysql\_fetch\_array

Retourne un tableau qui contient la ligne demandée, ou faux si il ne reste plus de ligne.

*mysql\_fetch\_array()* est une version étendue de *mysql\_fetch\_row()*. En plus d'enregistrer les données sous forme d'un tableau d'indices numériques, il peut aussi les enregistrer dans un tableau associatif, en utilisant les noms des champs comme indices.

Syntaxe :

```
array mysql_fetch_array(int result, [int result_type] );
```

Si plusieurs colonnes ont le même nom, la dernière colonne aura la priorité. Pour accéder aux autres colonnes du même nom, vous devez utiliser l'index numérique, ou faire un alias (as) pour chaque colonne.

```
select t1.f1 as foo t2.f1 as bar from t1, t2
```

Il est important de souligner que cette fonction N'est PAS plus lente que *mysql\_fetch\_row()*, tandis qu'elle ajoute un confort d'utilisation.

L'option *result\_type* de *mysql\_fetch\_array()* est une constante qui peut prendre les valeurs suivantes : *MYSQL\_ASSOC*, *MYSQL\_NUM*, et *MYSQL\_BOTH*.

Exemple :

```
<?
```

```
mysql_pconnect($host,$user,$password);
```

```
$id_result = mysql_db_query("base_RH","select nom, prenom from table_personnes");
```

```
while($row = mysql_fetch_array($id_result))
```

```
{
```

```
    echo "Nom : $row['nom'], prénom : $row['prenom']<br>";
```

```
}
```

```
mysql_free_result($result);
```

```
mysql_close() ;
```

```
?>
```



## 104. Exercice pratique

Nous allons nous connecter au serveur MySQL, créer une base de données 'resa\_salles', afficher le nombre et le nom des bases de données existantes sur le serveur, puis tenter de sélectionner la base de données 'resa\_salles' :

```
<html>
<head>
<title>Création et connexion à une base de donnée MySQL</title>
</head>
<body>
<?
echo "<form method='POST' action='#'>";
echo "<div align='center'>";
$lien = mysql_pconnect("localhost", "", "");
if( !$lien )
{
    echo "Connexion au serveur MySQL impossible ! ".mysql_stat();
}
else
{
    echo "Connexion au serveur MySQL réussie ! ".mysql_get_server_info() ;
    $bdd = "resa_salles";
    echo "<br>Création de la base de données : $bdd";
    mysql_create_db($bdd);
    $id_result = mysql_list_dbs();
    echo "<br>Nombre de bases de données : ".mysql_num_rows($id_result)."<br>";
    $result = mysql_fetch_array($id_result);

    while ($result)
    {
        echo "Base de données : ".$result[0]."<br>";
        $result = mysql_fetch_array($id_result);
    }

    mysql_select_db($bdd) or die("Sélection de la base de données $bdd impossible ! : ".mysql_error() );
    echo "<h2>Sélection de la base de données '$bdd' réussie !</h2>";
}

mysql_close() ;
echo "</div>";
echo "</form>";
?>
</body>
</html>
```

Résultat à l'affichage :

```
Connexion au serveur MySQL réussie ! 3.23.47-nt
Création de la base de données : resa_salles
Nombre de bases de données : 3
Base de données : mysql
Base de données : resa_salles
Base de données : test
```

**Sélection de la base de données 'resa\_salles' réussie !**

Vous pouvez compléter cet exercice avec :

1. La création de la table 'grades'
2. L'affichage des tables de la base de données "resa\_salles"
3. L'affichage des champs de la table 'grades'

```

<?
echo "<form method='POST' action='#'>";
echo "<div align='center'>";
$lien = mysql_pconnect("localhost","","");
if( !$lien )
{
    echo "Connexion au serveur MySQL impossible ! ".mysql_stat();
}
else
{
    echo "Connexion au serveur MySQL réussie ! ".mysql_get_server_info() ;
    $bdd = "resa_salles";
    echo "<br>Création de la base de données : $bdd";
    mysql_create_db($bdd);
    $id_result = mysql_list_dbs();
    echo "<br>Nombre de bases de données : ".mysql_num_rows($id_result)."<br>";
    $result = mysql_fetch_array($id_result);

    while ($result)
    {
        echo "Base de données : ".$result[0]."<br>";
        $result = mysql_fetch_array($id_result);
    }

    mysql_select_db($bdd) or die("Sélection de la base de données $bdd impossible ! : ".mysql_error() );
    echo "<h2>Sélection de la base de données '$bdd' réussie !</h2>";
    echo "Début de la création de la table grades...";
    echo "<br>";
    $requete = "CREATE TABLE grades ";
    $requete.= "(";
    $requete.= "CodeGrade INTEGER(11) UNSIGNED NOT NULL DEFAULT 0 PRIMARY KEY";
    $requete.= "AUTO_INCREMENT,";
    $requete.= "TriGramme VARCHAR(7) BINARY NULL DEFAULT NULL,";
    $requete.= "ClairGrade VARCHAR(50) BINARY NULL DEFAULT NULL";
    $requete.= ")";
    $result = mysql_query($requete) ;      // cette requête peut être valide et retourner zéro

    if(mysql_errno())
    echo "Une erreur s'est produite lors de la création de la table grades : ".mysql_error() ;
    else
    echo "Fin de la création de la table grades.<br><br>";

    $id_tables = mysql_list_tables($bdd);
    $result_tables = mysql_fetch_array($id_tables);

    while ($result_tables)
    {
        echo "Table : ".$result_tables[0]."<br>";
        $req = "select * from ".$result_tables[0];
        $result_champs = mysql_query($req);
        $nb_champs = mysql_num_fields($result_champs);
        for($index_champ = 0 ; $index_champ < $nb_champs ; $index_champ++)
        {
            echo "Champ : ".mysql_field_name($result_champs, $index_champ)."<br>";
        }
    }
}

```

```
        echo "type : ". mysql_field_type($result_champs, $index_champ) . "<br>";
        echo "longueur : ". mysql_field_len($result_champs, $index_champ) . "<br><br>";
    }

    $result_tables = mysql_fetch_array($id_tables);
}

mysql_close();
echo "</div>";
echo "</form>";
?>
```

Résultat à l'affichage :

```
Connexion au serveur MySQL réussie ! 3.23.47-nt
Création de la base de données : resa_salles
Nombre de bases de données : 3
Base de données : mysql
Base de données : resa_salles
Base de données : test
```

## **Sélection de la base de données 'resa\_salles' réussie !**

```
Début de la création de la table grades...
Fin de la création de la table grades.
```

```
Table : grades
Champ : CodeGrade
type : int
longueur : 11
```

```
Champ : TriGramme
type : string
longueur : 7
```

```
Champ : ClairGrade
type : string
longueur : 50
```

## 105. Méthodes d'accès à MySQL par programmation objet

**La classe Database** décrite ci-dessous contient les propriétés et les méthodes qui permettront de se connecter à une base de données MySQL et d'accéder aux données de la base.

Le constructeur initialise les propriétés de l'objet à l'aide des quatres paramètres suivants : nom de l'hôte qui héberge le serveur, nom de la base de données, nom et mot de passe de l'utilisateur permettant de se logger.

Cette classe sera incluse dans les exemples suivants à l'aide de la fonction : *include "base.php3"* ;

Script 'base.php3' :

```
<html>
<head>
<title>
classe Database
</title>
</head>
<body>
<?
class Database
{
    var $_host;
    var $_database_name;
    var $_login;
    var $_password;
    var $_result_mysql;
    var $_result_database;
    var $_result_query;

    Function Database($host, $database_name, $login, $password)
    {
        $this->_host = $host;
        $this->_database_name = $database_name;
        $this->_login = $login;
        $this->_password = $password;
    }

    Function connect_database()
    {
        $this->_result_mysql = mysql_pconnect($this->_host, $this->_login, $this->_password);
        if (!$this->_result_mysql)
        {
            mysql_error();
        }
        $this->_result_database = mysql_select_db($this->_database_name);
        if (!$this->_result_database)
        {
            mysql_error();
        }
    }

    Function query_database($query)
    {
        $this->_result_query = mysql_query($query, $this->_result_mysql);
        if (!$this->_result_query)
        {
            mysql_error();
        }
    }
}
```

```
Function fetch_row()
{
    return (mysql_fetch_row($this->_result_query));
}

Function free()
{
    return (mysql_free_result($this->_result_query));
}

Function fetch_array()
{
    return (mysql_fetch_array($this->_result_query));
}

Function fetch_field()
{
    return (mysql_fetch_field($this->_result_query));
}

Function num_rows()
{
    return (mysql_num_rows($this->_result_query));
}

Function num_fields()
{
    return (mysql_num_fields($this->_result_query));
}
}
?>
</body>
</html>
```

**Le script 'config.php3'** définit les variables globales qui permettront la connexion à la base de données :

```
<?
/*ce fichier de configuration permet de définir la base de données
*(variables $host_resa, $base_resa, $user_resa, $password_resa).
*/
$host_resa = "localhost";
$base_resa = "resa_salles";
$user_resa = "";
$password_resa = "";
?>
```

**La classe *grade*** décrite ci-dessous contient les propriétés et les méthodes qui permettront d'instancier un objet à partir duquel nous soumettrons les requêtes au serveur MySQL.

Le constructeur crée un pointeur (\$this->\_new\_database) sur un objet de la classe *Database*. C'est à l'aide de ce pointeur que nous ferons appel aux fonctions élémentaires d'accès à MySQL.

Cette classe sera incluse dans les exemples suivants à l'aide de la fonction : *include "grade.php3"* ;

Script 'grade.php3' :

```
<html>
<head>
<title>
classe grade
</title>
</head>
<body>
<?
require "config.php3";

class grade
{
    var $_search_result;
    var $_new_database;

    Function grade()
    {
        global $host_resa, $base_resa, $user_resa, $password_resa;
        $this->_new_database = new Database($host_resa, $base_resa, $user_resa, $password_resa);
        $this->_new_database->connect_database();
    }

    Function ask_query($query)
    {
        $this->_new_database->query_database($query);
    }

    Function access_result()
    {
        return ($this->_search_result);
    }

    Function read_results()
    {
        $this->_search_result = $this->_new_database->fetch_array();
    }

    Function num_results()
    {
        return ($this->_new_database->num_rows());
    }
}
?>
</body>
</html>
```

Ce script remplit la table *grades* à partir des données (séparées par un ‘;’) stockées dans le fichier "grades.csv" dont le contenu est affiché à la page suivante :

```
<html>
<head>
<title>
Remplissage de la table grades
</title>
</head>
<body>
<b>
Début du remplissage de la table grades...
</b>
<br>
<br>
<?
include "base.php3";
include "grade.php3";

$file = "grades.csv";
$new_grade = new grade();

echo "Début du vidage de l'ancienne table grades...";
echo "<br>";
$query = "DELETE FROM resa_salles.grades";
$new_grade->ask_query($query);
echo "Fin du vidage de l'ancienne table grades.";
echo "<br>";
echo "<br>";

echo "Début du remplissage de la nouvelle table grades...";
echo "<br>";
$result = file($file);
for ($compteur = 1; $compteur <= 51; $compteur++)
{
    $array = explode(";", $result[$compteur]);
    $query = "INSERT INTO resa_salles.grades (TriGramme,ClairGrade) VALUES ";
    $query .= "("";
    $query .= $array[1] . ",";
    $query .= addslashes($array[2]) . ")";
    $new_grade->ask_query($query);
}
echo "Fin du remplissage de la nouvelle table grades.";

?>
<br>
<br>
<b>
Fin du remplissage de la table grades.
</b>
</body>
</html>
```

Contenu du fichier 'grades.csv' :

CodeGrade;TriGramme;ClairGrade;  
1;TRS;Transmetteur;  
2;1TRS;1er Transmetteur;  
3;SCI;Scientifique du contingent;  
4;CAL;Caporal;  
5;CCH;Caporal-chef;  
6;BRI;Brigadier;  
7;BCH;Brigadier-chef;  
8;SGT;Sergent;  
9;SCH;Sergent-chef;  
10;MDL;Maréchal des logis;  
11;MCH;Maréchal des logis-chef;  
12;ADJ;Adjudant;  
13;ADC;Adjudant-chef;  
14;MAJ;Major;  
15;EOR;Elève officier de réserve;  
16;ASP;Aspirant;  
17;SLT;Sous-lieutenant;  
18;LTN;Lieutenant;  
19;CNE;Capitaine;  
20;CDT;Commandant;  
21;CBA;Chef de bataillon;  
22;CEN;Chef d'escadron;  
23;CES;Chef d'escadron;  
24;LCL;Lieutenant-colonel;  
25;COL;Colonel;  
26;GDI;Général de division;  
27;GBA;Général de brigade;  
28;GCA;Général de corps d'armée;  
29;GAR;Général d'armée;  
30;AGT/C;Agent sur contrat;  
31;OGR;Ouvrier de groupe;  
32;ATE;Agent technique de l'électronique;  
33;ATPE;Agent technique principal de l'électronique;  
34;AA;Adjoint administratif;  
35;AAP;Adjoint administratif principal;  
36;TSEF;Technicien supérieur d'études et fabrication;  
37;CCN;Contrôleur de classe normale;  
38;CCS;Contrôleur de classe supérieure;  
39;CCE;Contrôleur de classe exceptionnelle;  
40;SACN;Secrétaire administratif de classe normale;  
41;SACS;Secrétaire administratif de classe supérieure;  
42;SACE;Secrétaire administratif de classe exceptionnelle;  
43;IEF;Ingénieur d'études et fabrication;  
44;INSP;Inspecteur des services;  
45;IP;Inspecteur principal des transmissions;  
46;ASA;Attaché de service administratif;  
47;CSA;Chef de service administratif;  
48;AAC;Attaché d'administration centrale;  
49;M.;Monsieur;  
50;MME;Madame;  
51;MLLE;Mademoiselle;



Ce script affiche le contenu de la table *grades* :

```
<html>
<head>
<title>
Affichage de la table grades
</title>
</head>
<body>
<?
include "base.php3";
include "grade.php3";

    $new_grade = new grade();

    echo "Affichage du contenu de la table grades...<br><br>";

    $query = "SELECT * FROM resa_salles.grades";
    $new_grade->ask_query($query);

    $new_grade->read_results();
    $result = $new_grade->access_result();
    $cpt=0;
    echo "<table border='1'>";

    while($result)
    {
        $cpt++;

        if( ($cpt%2) == 1 )    // $cpt impair
            echo "<tr>";

            echo "<td><font size='2'>";
            print $result['TriGramme'];
            echo "</font></td> <td><font size='2'>";
            print $result['ClairGrade'];
            echo "</font></td>";

            if( ($cpt%2) == 0)    // $cpt pair
                echo "</tr>";

            $new_grade->read_results();
            $result = $new_grade->access_result();
        }

        echo "</table>";
    ?>
</body>
</html>
```

Résultat à l'affichage :

Affichage du contenu de la table grades...

TRS	Transmetteur	1TRS	1er Transmetteur
SCI	Scientifique du contingent	CAL	Caporal
CCH	Caporal-chef	BRI	Brigadier
BCH	Brigadier-chef	SGT	Sergent
SCH	Sergent-chef	MDL	Maréchal des logis
MCH	Maréchal des logis-chef	ADJ	Adjudant
ADC	Adjudant-chef	MAJ	Major
EOR	Elève officier de réserve	ASP	Aspirant
SLT	Sous-lieutenant	LTN	Lieutenant
CNE	Capitaine	CDT	Commandant
CBA	Chef de bataillon	CEN	Chef d'escadron
CES	Chef d'escadron	LCL	Lieutenant-colonel
COL	Colonel	GDI	Général de division
GBA	Général de brigade	GCA	Général de corps d'armée
GAR	Général d'armée	AGT/C	Agent sur contrat
OGR	Ouvrier de groupe	ATE	Agent technique de l'électronique
ATPE	Agent technique principal de l'électronique	AA	Adjoint administratif
AAP	Adjoint administratif principal	TSEF	Technicien supérieur d'études et fabrication
CCN	Contrôleur de classe normale	CCS	Contrôleur de classe supérieure
CCE	Contrôleur de classe exceptionnelle	SACN	Secrétaire administratif de classe normale
SACS	Secrétaire administratif de classe supérieure	SACE	Secrétaire administratif de classe exceptionnelle
IEF	Ingénieur d'études et fabrication	INSP	Inspecteur des services
IP	Inspecteur principal des transmissions	ASA	Attaché de service administratif
CSA	Chef de service administratif	AAC	Attaché d'administration centrale
M.	Monsieur	MME	Madame
MLLE	Mademoiselle		

## 14. Fonctionnalités Internet

### 106. Le courrier électronique

#### 106.1. mail

**Mail()** est la principale fonction d'envoi de courrier électronique utilisée en PHP.

Elle nécessite :

- Une configuration préalable de la machine sur laquelle tourne l'application PHP, afin de servir elle-même de serveur e-mail (par exemple avec Sendmail sous Linux, ou le serveur SMTP de IIS sous Windows), ou pour communiquer avec un serveur e-mail donné ;
- Une configuration du fichier « php.ini », dans la section [mail function]. Les paramètres SMTP et Sendmail\_from (Windows) ou Sendmail\_path (Unix) doivent être renseignés.

La fonction *mail()* contient quatre paramètres de type chaîne de caractères, et retourne une valeur booléenne.

Sa syntaxe est la suivante :

*boolean mail(string adresse, string sujet, string message, [string en\_tetes]) ;*

*adresse :* adresse des destinataires du message, séparées par des virgules.

*sujet :* objet du message.

*message :* corps du message.

*en\_tetes :* lignes supplémentaires facultatives à faire figurer dans l'en-tête du message, séparées par des retours.

Les attributs particuliers du message (pièce jointe, copie, etc.) doivent être construits respectivement dans le corps et dans l'en-tête du message.

Exemple simple :

```
$adresse = "destinataire@fai1.com";
$sujet = "Envoi de message" ;
$message = "Bonjour, voici mon premier message envoyé avec PHP" ;
$en_tetes = "From : expéditeur@fai2.com\r\nReply-To : expéditeur@fai2.com " ;
// envoi du message
if( mail($adresse, $sujet, $message, $en_tetes) )
    echo "Votre message a été envoyé avec succès à : $adresse<br>" ;
else
    echo "L'envoi de votre message à : $adresse a échoué<br>" ;
```

le script suivant permet d'envoyer un message avec une pièce jointe :

```
$pièce_jointe = $path_data."piece.txt";
$nom_fichier_pj = "piece.txt";
$content_mail = "Vous trouverez la réponse à votre demande dans la pièce jointe \"$ nom_fichier_pj \";
$from = "Mon application PHP";
$objet_mail = "Envoi d'un message avec pièce jointe";
$limite = "_parties_.md5(uniqid(rand()));
$mail_mime = "Date: ".date("l j F Y, G:i")."\n";
$mail_mime .= "MIME-Version: 1.0\n";
$mail_mime .= "Content-Type: multipart/alternative;\n";
$mail_mime .= " boundary=\"-----$limite\n\n";

// Le message en texte simple pour les navigateurs qui n'acceptent pas le HTML
$texte = "Ceci est un message au format MIME.\n";
$texte .= "-----$limite\n";
$texte .= "Content-Type: text/html; charset=\"US-ASCII\"\n";
$texte .= "Content-Transfer-Encoding: 7bit\n\n";
$texte .= $content_mail;
$texte .= "\n\n";
// Attachement de la pièce jointe
$sattachement = "-----$limite\n";
```

```

$attachelement .= "Content-Type: application/octet-stream; name=\"$nom_fichier_pj\"\\n\";
$attachelement .= "Content-Transfer-Encoding:base64\\n\\n\";
$attachelement .= chunk_split(base64_encode(implode("", file($piece_jointe))));

mail($adresse, $objet_mail, $texte.$attachelement, "From: $from\\n\".$mail_mime);

```

## 107. Le « file upload »

Le « file upload » est l'action de transférer un fichier depuis le poste client vers le serveur (par opposition au « download », qui transfère un fichier depuis le serveur vers le client).

Avec PHP, cette fonctionnalité clé des applications Internet est nativement incluse dans le moteur, et il n'y a rien à programmer ! En effet, aucune fonction particulière ni aucune bibliothèque ne sont nécessaires. Il suffit de référencer une page PHP dans le formulaire qui contient le champ de transfert de fichier. Le fichier est transféré automatiquement dans le répertoire d'upload spécifié dans le fichier « php.ini » (paramètre *upload\_tmp\_dir*), et ses caractéristiques sont intégrées à des variables globales de la page référencée.

Exemple :

```

<FORM ENCTYPE = "multipart/form-data" ACTION = "upload.php4" METHOD = POST>
<INPUT TYPE = "hidden" NAME = "MAX_FILE_SIZE" VALUE = "1000">
Fichier à transférer : <INPUT NAME = "nom_de_fichier" TYPE = "file">
<INPUT TYPE = "submit" VALUE = "Transférer">
</FORM>

```

Dans cette page, il est impératif que le champ caché « MAX\_FILE\_SIZE » précède le champ de type « file ».

Quelle que soit la taille spécifiée dans cette variable cachée, la taille du fichier transféré ne pourra pas dépasser la taille maximale spécifiée dans le fichier « php.ini » (paramètre *upload\_max\_filesize*).

Si l'upload s'est déroulé correctement, à ce stade, le fichier est déjà présent sur le serveur, et des variables globales ont intégré toutes les valeurs utiles à la gestion de ce fichier. Elles sont au nombre de quatre :

<i>\$nom_de_fichier :</i>	nom temporaire sous lequel le fichier a été enregistré sur le serveur.
<i>\$nom_de_fichier_name :</i>	nom original du fichier sur la machine de l'expéditeur.
<i>\$nom_de_fichier_size :</i>	taille du fichier transféré, en octets.
<i>\$nom_de_fichier_type :</i>	type MIME du fichier transféré, si le navigateur du client envoie cette information.

Le préfixe de ces quatre variables, à savoir "nom\_de\_fichier", est le nom donné au champ de type « file ». Il peut bien entendu prendre n'importe quel nom.

On peut donc afficher ces renseignements, en guise de confirmation de transfert :

```

< ?
echo "Le fichier $nom_de_fichier_name ($nom_de_fichier_size octets) de type $nom_de_fichier_type";
echo " a été sauvegardé sous le nom $nom_de_fichier";
?>

```

## 108. Le traitement des URL

### 108.1. Codage des URL

Les URL peuvent contenir, outre l'adresse de la page, des variables et leurs valeurs. Cependant un certain nombre de caractères spéciaux (tous les caractères non alphanumériques, à l'exception de '-' et de '\_') ne peuvent pas être insérés tels quels dans l'URL, et doivent être codés afin de ne pas être interprétés comme des délimiteurs. Le format de codage est constitué du symbole '%', suivi d'un code hexadécimal, à l'exception de l'espace, qui est codé par un '+'.

Voici le tableau de codage des principaux caractères :

Caractère	Code	Caractère	Code	Caractère	Code	Caractère	Code
Espace	+		%7C	°	%B0	%	%25
&	%26	è	%E8	+	%2B	*	%2A
é	%E9	`	%60	=	%3D	/	%2F
"	%22	\	%5C	}	%7D	:	%3A
~	%7E	Ç	%E7	¨	%A8	\$	%A7
#	%23	^	%5E	!	%21	'	%27
à	%E0	\$	%24	µ	%B5	{	%7B
@	%40	£	%A3	?	%3F	(	%28
)	%29	□	%A4	,	%2C	[	%5B
]	%5D	U	%F9	;	%3B		

PHP propose deux fonctions pour gérer ce codage :

*urlencode()* permet de coder une chaîne qui contient des caractères spéciaux.

*urldecode()* est sa réciproque, c'est-à-dire qu'elle remplace les codes hexadécimaux par leur caractère équivalent.

### 108.2. urlencode

Syntaxe :

*string urlencode(string str);*

Cette fonction retourne une chaîne dont les caractères non alphanumérique (hormis '-' et '\_') sont remplacés par des séquences commençant par un caractère pourcentage (%) suivi de deux chiffres hexadécimaux. Les espaces sont remplacés par des signes plus (+). Cette fonction est pratique pour transmettre des informations via une URL. C'est aussi un moyen de passer des informations d'une page à l'autre.

Exemple :

```
< ?
echo urlencode("http://www.url.org/page.php3? var1=valeur1& var2=valeur2") ;
?>
```

Résultat à l'affichage :

```
http%3A%2F%2Fwww.url.org%2Fpage.php3%3F var1%3Dvaleur1%26 var2%3Dvaleur2
```

### 108.3. urldecode

Syntaxe :

```
string urldecode(string str);
```

Cette fonction décode toutes les séquences commençant par un ‘%’ et les remplace par leur valeur. La chaîne ainsi décodée est retournée.

Exemple :

```
< ?  
echo urldecode("http%3A%2F%2Fwww.url.org%2Fpage.php3%3F var1%3Dvaleur1%26 var2%3Dvaleur2") ;  
>
```

Résultat à l’affichage :

```
http://www.url.org/page.php3? var1=valeur1 & var2=valeur2
```

### 108.4. parse\_url

Syntaxe :

```
array parse_url(string url);
```

Cette fonction retourne un tableau associatif contenant les différents éléments de l’URL.

Les éléments recherchés sont : "scheme", "host", "port", "user", "pass", "path", "query", et "fragment".

Ils constituent chacun une des clés du tableau retourné. Seuls les éléments qui possèdent une valeur affectée sont retournés dans le tableau.

Exemple :

```
< ?  
$tab = parse_url("http://www.url.org/page.php3?var1=valeur1&var2=valeur2") ;  
  
foreach($tab as $cle => $val)  
    echo "$cle : $val<br>" ;  
>
```

Résultat à l’affichage :

```
scheme : http  
host : www.url.org  
path : /page.php3  
query : var1=valeur1&var2=valeur2
```

## 109. Les méthodes de gestion du contexte applicatif

### 109.1. Le contexte applicatif

Une des principales particularités d'une application fondée sur une architecture Internet est qu'elle ne procède qu'à des connexions ponctuelles avec la base de données. En effet, les connexions sont ouvertes, puis fermées, pour chaque page PHP.

La conséquence immédiate de cette particularité est que les informations, relatives notamment à l'utilisateur courant, doivent être renvoyées à la base lors de chaque connexion, afin que celui-ci puisse être identifié et retrouver ses droits sur les données.

La plupart du temps, ces informations ne sont pas suffisantes. Il est nécessaire par exemple, de connaître les actions et les choix effectués antérieurement (sur d'autres pages) par l'utilisateur, afin de générer un contenu de page cohérent. Toutes ces informations doivent donc pouvoir être prises en compte à tout moment, sur n'importe quelle page.

Cet ensemble de données est appelé le contexte applicatif. Il va permettre de replacer l'utilisateur dans son contexte, lors de chaque connexion à la base.

La gestion du contexte est donc la capacité de passer des paramètres d'une page à une autre. Pour cela, il existe cinq grandes méthodes.

### 109.2. Les différentes méthodes de gestion

Méthode	Avantages	Inconvénients
URL longue	<ul style="list-style-type: none"> <li>- Simplicité d'utilisation</li> <li>- Lisibilité</li> </ul>	Quantité d'informations limitée à 2 ou 4 Ko <ul style="list-style-type: none"> <li>- Informations visibles</li> <li>- Encodage des données</li> </ul>
Variables cachées	<ul style="list-style-type: none"> <li>- Données cachées (méthode POST)</li> <li>- Pas de limitation de longueur des données transmises</li> </ul>	<ul style="list-style-type: none"> <li>- Nécessité de passer par des formulaires HTML</li> <li>- Importance du code HTML généré si la quantité de données est volumineuse</li> </ul>
Cookies	<ul style="list-style-type: none"> <li>- Disponibilité de l'information à tout moment, sans nécessité de passage de page en page</li> </ul>	<ul style="list-style-type: none"> <li>- Dépendants de la configuration du navigateur du poste client</li> </ul>
Session	<ul style="list-style-type: none"> <li>- Idem cookies</li> </ul>	<ul style="list-style-type: none"> <li>- Disponibles uniquement à partir de PHP4</li> </ul>
Base de données	<ul style="list-style-type: none"> <li>- Idem cookies</li> <li>- Adaptée aux gros volumes de données</li> </ul>	<ul style="list-style-type: none"> <li>- Traitements importants supplémentaires dus aux échanges entre PHP et le SGBD</li> </ul>

#### 109.2.1. URL longue

Cette méthode se base sur l'URL de la page. En effet, il est possible de passer des variables au sein même de l'URL. pour cela, il suffit de compléter l'URL de base de la page avec le nom des variables à passer, suivi de leur valeur. L'URL se forme sur le modèle suivant :

*http://nom\_du\_serveur/nom\_de\_la\_page.php3 ?variable1=valeur1&variable2=valeur2*

- Le '?' marque la séparation entre l'URL de base, qui désigne la page, et l'extension de l'URL, qui contient les variables.
- Le '&' marque la séparation entre chaque couple « variable-valeur ». Dans la page PHP référencée, ces variables seront accessibles simplement par leur nom. Cette méthode permet donc très facilement de transmettre des informations de page en page. Cependant, plusieurs facteurs sont à prendre en compte :
- La taille d'une URL est le plus souvent limitée à 2 ou 4 Ko (en fonction de l'OS utilisé), ce qui limite la quantité de variables et la longueur de leur valeur.
- Les URL s'accommodent très mal des caractères spéciaux. Il faut donc penser à encoder l'URL au moment de sa construction, à l'aide de la fonction `urlencode()`, afin de ne pas provoquer d'erreurs.
- L'appel à la page s'effectue nécessairement à l'aide d'un hyperlien ou d'une fonction JavaScript.
- Le contenu des variables est visible dans la zone d'affichage d'URL du navigateur, ou dans la fenêtre d'information qu'il propose, ce qui peut être gênant pour des raisons de sécurité ou de robustesse.

### 109.2.2. Variables cachées

Cette méthode consiste à utiliser un formulaire, afin de faire passer les données sous forme de variables cachées, selon la syntaxe suivante :

```
<INPUT TYPE = HIDDEN NAME = "nom_de_la_variable" VALUE = "valeur">
```

	Si le formulaire est envoyé au moyen de la méthode GET, on se retrouve dans un contexte très proche des URL longues, ce qui implique la même limitation de longueur et la même visibilité des valeurs transmises.
--	---

On considérera donc que le formulaire est envoyé au moyen de la méthode **POST**.

Pour transmettre les mêmes valeurs que celles utilisées dans l'exemple précédent, on aura le code HTML suivant :

```
<FORM ACTION = "nom_de_la_page.php3" METHOD = POST NAME = "form1">
<INPUT TYPE = HIDDEN NAME = "variable1" VALUE = "valeur1">
<INPUT TYPE = HIDDEN NAME = "variable2" VALUE = "valeur2">
</FORM>
```

Pour valider ce formulaire, et ainsi transmettre les variables à la page "nom\_de\_la\_page.php3", plusieurs solutions sont possibles :

1. Passer par un bouton classique de type « submit », inclus dans le formulaire `<INPUT TYPE = SUBMIT ...>`
2. Passer par une image cliquable, incluse dans le formulaire `<INPUT TYPE = IMAGE ...>`
3. Passer par une fonction JavaScript, qui valide le formulaire ( `document.form1.submit();` ) déclenché par un clic sur un hyperlien ou sur un simple bouton `<INPUT TYPE = BUTTON ...>` (cette fonction fait appel à la méthode POST du formulaire).

Exemple :

```
<INPUT TYPE = BUTTON VALUE = "Envoyer" NAME = "envoyer" OnClick="document.form1.submit();">
```

L'exemple suivant est identique mais il ne fait pas appel à la balise `<FORM ....>`, ce qui oblige à écrire les couples 'variable-valeur', qui doivent être passés, après le nom de la page appelée :

```
<INPUT TYPE = BUTTON VALUE = "Envoyer" NAME = "envoyer" OnClick="document.location =
"nom_de_la_page.php3?variable1=\"valeur1\"&variable2=\"valeur2\"">
```

Seule la troisième méthode possède une restriction. En effet, toute action fondée sur du langage JavaScript est soumise au paramétrage du navigateur de l'utilisateur, qui peut refuser son exécution. Aucune autre contrainte ne vient trancher entre ces trois méthodes. Le choix doit donc être effectué en fonction de l'ergonomie et du design des pages concernées.

### 109.2.3. Cookies

Un cookie est une information stockée sur le poste client, c'est-à-dire sur celui où s'exécute le navigateur. Comme pour les fonctions JavaScript, l'utilisation des cookies est dépendante de la configuration du navigateur du client, ce qui peut constituer un handicap.

En pratique, chaque cookie peut stocker une ou plusieurs informations. Le principal avantage de cette technique est qu'une information stockée dans un cookie peut être consultée depuis n'importe quelle page ; il n'est pas nécessaire de la faire passer de page en page tout au long de l'application.

En PHP, l'utilisation des cookies est extrêmement simple, et ne nécessite qu'une seule fonction : `setcookie()`

La syntaxe de la fonction `setcookie()` est la suivante :

```
int setcookie(string nom [,string valeur] [,int expiration] [,string chemin] [, string domaine] [, int secure]) ;
```

- Le champ *expiration* spécifie la date au-delà de laquelle le cookie ne sera plus valide (lorsque ce paramètre n'est pas spécifié, le cookie devient permanent, à moins d'être supprimé manuellement).
- Les paramètres *chemin* et *domaine* peuvent être utilisés en conjonction pour spécifier les URL auxquelles doit être associé le cookie.
- Le mot clé *secure* vient indiquer que le cookie ne peut être envoyé que via une connexion HTTPS sécurisée.
- Si vous souhaitez affecter plusieurs valeurs à un seul cookie, ajoutez [] au nom du cookie.

	Si le formulaire est envoyé au moyen de la méthode GET, on se retrouve dans un contexte très proche des URL longues, ce qui implique la même limitation de longueur et la même visibilité des valeurs transmises.
--	---



Les Cookies font parties de l'entête HTTP, ce qui impose que la fonction *setcookie()* soit appelée avant tout affichage sur le client. Elle doit donc être appelée avant la première balise HTML, et avant n'importe quel envoi de commande PHP. C'est une erreur très courante que de lire du code avec la fonction *include()* ou avec *auto\_prepend* et d'avoir des espaces ou des lignes vides dans ce code qui produisent un début de sortie avant que *setcookie()* n'ait été appelée.

Exemple1 :

```
< ?
    // Ecriture des cookies ayant pour nom "variable1" et "variable2" sur le poste client
    setcookie("variable1", "valeur1") ;
    setcookie("variable2", "valeur2") ;
    // Pour accéder à ces cookies, il suffit d'utiliser la variable globale du même nom créée
    // automatiquement par PHP
    echo $variable1 ;      // affiche la valeur du cookie "variable1"
?>
```

Vous pouvez également accéder à la valeur de la variable \$variable1 via \$HTTP\_COOKIE\_VARS["variable1"]

Exemple2 :

```
// $login et $password ont été préalablement saisis par l'utilisateur
// la classe 'utilisateur' a été préalablement définie
$new_utilisateur = new utilisateur();
$req = "select index_utilisateur, nom, password from utilisateurs"; // le contenu du champ password est crypté
$new_utilisateur->ask_query($req);
$new_utilisateur->read_results();
$result = $new_utilisateur->access_result();
$password_code = crypt($password, "xx");
$login = strtolower($login);
$nom = strtolower($result['nom']);

if( ($login == $nom) && ($password_code == $result['password']) )
{
    $index_util = $result['index_utilisateur'];
    setcookie("cook1", $index_util);
    setcookie("cook2", $login);
    setcookie("cook3", $password_code);
    header("Location: menu.php3"); // redirige le navigateur sur la page 'menu.php3'
}
else
{
    echo "<script language=\"JavaScript\">";
    echo "alert('Erreur ! Vous n'avez pas été enregistré')";
    echo "</script>";
}
```

A la fin de l'application il ne faut pas oublier de réinitialiser les cookies, car leur existence sur le poste client est persistante.

Exemple :

```
// contenu de la page 'quitter.php3'
<?
    setcookie("cook1","");
    setcookie("cook2","");
    setcookie("cook3","");
?>
<html>
<head>
<title></title>
</head>
<body onload="parent.close();" bgproperties="fixed">    // fermeture de la fenêtre active
</body>
</html>
```

### 109.2.4. Variables de session

Ce type de variables n'est implémenté qu'à partir de PHP4. Dans PHP3, plusieurs bibliothèques de composants les simulent, à l'aide d'une base de données, mais elles n'en offrent pas les avantages.

Une variable de session est une variable stockée sur le serveur. Elle est propre à un utilisateur donné, et est accessible à tout moment, depuis n'importe quelle page appelée par cet utilisateur.

Elle offre donc les mêmes avantages qu'un cookie. Toutefois, l'identifiant de l'utilisateur doit être passé dans toutes les pages. Pour cela, il est nécessaire d'employer une des méthodes vues précédemment.

#### Gestion des sessions :

**session\_start()** : Initialise les données de session.

**session\_destroy()** : Supprime l'identifiant de la session courante.

**session\_name([string nom])** : Affecte et/ou retourne le nom de la session courante. Si *nom* est fourni, le nom de la session changera, et prendra la valeur fournie.

**session\_module\_name([string module])** : Affecte et/ou retourne le module de la session courante. Si *module* est fourni, ce module sera utilisé à la place du courant.

**session\_save\_path([string path])** : Affecte et/ou retourne le chemin du dossier utilisé pour enregistrer les données de sessions. Si *path* est fourni, le chemin prendra alors la valeur fournie.

**session\_id([string id])** : Affecte et/ou retourne l'identifiant de la session courante. Si *id* est fourni, il remplacera l'identifiant courant de la session.

**session\_register(string name1 [, string name2] {, ...})** : Enregistre une ou des variables avec les noms *name1*, *name2*, ... dans la session courante. Cette fonction retourne TRUE lorsque la ou les variables ont été correctement enregistrées. Ex : *session\_register("name1","name2") ;*

**session\_unregister(string name)** : Supprime la variable nommée *name* dans la session courante. Cette fonction retourne TRUE lorsque la variable a été correctement supprimée de la session. Ex : *session\_unregister("name") ;*

**session\_unset()** : Supprime toutes les variables de la session courante.

**session\_is\_registered(string name)** : Retourne TRUE si il y a une variable du nom de *name* enregistrée dans la session courante. L'existence d'une variable en tant que variable de session peut également être vérifiée à l'aide du tableau \$HTTP\_SESSION\_VARS. Ex : *\$HTTP\_SESSION\_VARS["name"];*

**session\_decode(string data) :** Décode les données de session à partir de la chaîne *data*, et affecte les valeurs des variables de session.

**session\_encode() :** Encode les données de session dans la chaîne retournée.

### **Mise en œuvre :**

La première opération consiste, lors de l'accès au site, à créer une session, puis à transmettre son identifiant au navigateur.

On fait alors appel à la fonction **session\_start()**, sans lui passer aucun paramètre. Cette fonction doit être placée en tout début de script. En effet, étant donné qu'elle génère un cookie, elle doit être placée avant l'envoi des headers HTTP, sinon une erreur se produit.

A ce moment, la session est créée. Elle charge les variables de session enregistrées, afin que vous puissiez les utiliser. L'identifiant est alors stocké dans la constante SID, et dans la suite du script, l'appel à la fonction **session\_id()** retourne la valeur de cet identifiant unique.

Pour transmettre la valeur d'une variable d'une page à une autre, il faut spécifier que l'on souhaite faire enregistrer cette variable en tant que variable de session. Pour cela on utilise la fonction **session\_register()**, à laquelle on passe en paramètre le nom de la variable, débarrassé de son préfixe \$.

Exemple1 :

```

< ?
    // test d'existence de session, sinon création d'une nouvelle session
    if ( $PHPSESSID )
        session_start($PHPSESSID) ;
    else
        session_start() ;
?>
<!-- Début du HTML, envoi des entêtes par le serveur HTTP →
<html>
<head>
<title> PHP4 Session Start</title>
</head>
<body>
<h1>PAGE 1 : PHP4 Session Start </h1>
<hr><br>
<!-- Affichage de l'identifiant de session, enregistrement d'une variable au contexte de la session →
< ?
    $id = session_id() ;
    $sname = session_name() ;
    echo "Votre identifiant de session est : $id<br>" ;
    echo "Le nom de cette session est : $sname<br>" ;

    // ajout de la variable $venant_depuis au contexte de la session
    $venant_depuis = $HTTP_REFERER ;          // adresse de la page de référence
    session_register("venant_depuis" ) ;
?>
<!-- Pour passer la clé de session à la page suivante, il est conseillé de la placer dans l'URL de la page
de destination. PHP4 propose un raccourci, afin de rendre cette écriture plus concise et plus facile à
interpréter. Le lien vers la page de destination s'écrira : <?=SID?> →
<br>
<hr>
<center>
<!--Ecriture courte →
<a href = "page_de_destination.php?<?=SID?>">Cliquer ici (écriture courte)</a>
<br><br>
<!--Ecriture longue →
<a href = "page_de_destination.php?session_name()=session_id()">Cliquer ici (écriture longue)</a>
</center>
</body>
</html>

```

Par la suite, toute autre page utilisée dans la session devra répercuter la clé de session sur les pages suivantes. Il faudra donc commencer chaque page par l'écriture du test d'existence de session ( les 7 premières lignes de l'exemple1).

Exemple2 : Compter le nombre de visites d'un utilisateur.

```

<? session_start() ;
session_register("compteur");
$compteur++;
?>
Bonjour visiteur, vous avez vu cette page <? echo $compteur; ?> fois.<p>
<!-- le <?=SID?> est nécessaire pour transmettre l'identifiant de session →
Pour continuer, <A HREF="nextpage.php?<?=SID?>">cliquez ici </A>

```

Notez que la valeur de la variable *\$compteur* est ajustée après que la variable a été enregistrée. La procédure inverse aurait également été correcte : ajuster la valeur puis enregistrer la variable. La valeur finale de la variable dans la page est celle qui sera disponible dans les pages subséquentes.

A la fin du script, la variable de session est gelée jusqu'à son rechargement (automatique) par un autre appel de la fonction `session_start()`.

Le prochain script doit par conséquent lui aussi débiter par un appel de la fonction `session_start()`.

### **Configuration du contrôle de session :**

Diverses options peuvent être ajustées dans le fichier 'php.ini', pour configurer le contrôle de session PHP.

Le tableau ci-dessous décrit les options de configuration les plus utiles :

Nom de l'option	Valeur par défaut	Effet
<code>session.auto_start</code>	0 (désactivée)	Ouverture automatique des sessions.
<code>session.cache_expire</code>	180	Spécifie la durée de vie, en minutes, des pages de session mises en mémoire cache.
<code>session.cookie_domain</code>	Aucune	Spécifie le domaine à utiliser avec le cookie de session.
<code>session.cookie_lifetime</code>	0	Fixe la durée de vie maximale du cookie de session sur la machine de l'utilisateur. La valeur 0, qui est la valeur par défaut, spécifie que le cookie doit expirer à la fermeture du navigateur.
<code>session.cookie_path</code>	/	Spécifie le chemin d'accès à utiliser avec le cookie de session.
<code>session.name</code>	PHPSESSID	Spécifie le nom de la session qui est utilisé comme nom de cookie dans le système de l'utilisateur.
<code>session.save_handler</code>	Files	Définit l'emplacement de stockage des données de session. Par défaut, les sessions sont enregistrées dans des fichiers. Elles peuvent être enregistrées dans une base de données, mais il faut alors écrire ses propres fonctions de sauvegarde.
<code>session.save_path</code>	/tmp	Spécifie le chemin d'accès de l'emplacement de stockage des données de session. Plus généralement, c'est l'argument qui est passé à la fonction de sauvegarde, et défini par <code>session.save_handler</code> .
<code>session.use_cookies</code>	1 (activé)	Configure les sessions de sorte que des cookies soient utilisés côté client.

## **109.2.5. Base de données**

Le principe d'utilisation d'une base de données dans le but de gérer les contextes applicatifs est le même que pour les sessions. En effet, ici aussi, il faut transmettre un identifiant utilisateur, mais la méthode de récupération des données est différente.

Avec les sessions, les variables peuvent être stockées dans un fichier local au serveur, et également dans une base de données.

Dans ce dernier cas, la programmation des pages est strictement identique.

Il existe une bibliothèque de composants très utilisée, qui simule les sessions par l'utilisation d'une base de données. Il s'agit de la phplib, disponible à l'adresse : <http://phplib.shonline.de>

## 15. Implémenter une authentification avec PHP et MySQL

Le Web est un support relativement anonyme, mais il est souvent intéressant de savoir qui visite votre site. Heureusement pour la confidentialité des utilisateurs, il n'est possible d'obtenir que peu d'informations personnelles sans leur accord explicite.

Avec peu de travail, les serveurs peuvent déterminer plusieurs informations sur les ordinateurs et les réseaux qui tentent de s'y connecter. Les navigateurs Web ont l'habitude de s'identifier en fournissant aux serveurs leur nom, leur numéro de version et le nom de votre système d'exploitation. Vous pouvez également déterminer la résolution et le nombre de couleurs de l'écran de vos visiteurs, ainsi que la largeur de leur écran de navigateur.

Chaque ordinateur connecté à Internet possède une adresse IP qui lui est propre. A partir de l'adresse IP d'un visiteur, vous pouvez découvrir certaines informations le concernant. Certaines adresses sont plus utiles que d'autres. Généralement, les personnes disposant d'une connexion permanente à Internet possèdent une adresse IP permanente. En revanche, les personnes qui doivent se connecter par modem à leur fournisseur d'accès à Internet pour ouvrir chaque session se voient affecter temporairement l'une des adresses IP de leur fournisseur.

Heureusement pour les utilisateurs du Web, aucune des informations fournies par leur navigateur ne permet de les identifier personnellement.

Après avoir demandé et reçu les informations concernant l'un de vos visiteurs, vous avez besoin d'un moyen d'associer ces informations au même utilisateur la prochaine fois qu'il viendra sur votre site.

Si vous partez de l'hypothèse qu'une seule personne visite votre site avec un compte donné, et que chaque visiteur ne se sert que d'un seul ordinateur, vous pouvez enregistrer un cookie sur l'ordinateur de cette personne pour l'identifier.

Ces suppositions ne sont certainement pas vraies pour tout le monde, puisqu'il arrive souvent que plusieurs personnes partagent un même ordinateur et qu'une même personne se serve de plusieurs ordinateurs. Il arrivera donc que vous ayez besoin de demander à vos utilisateurs de s'identifier à nouveau. Vous devrez en plus leur demander de prouver leur identité. La méthode d'authentification la plus courante sur le Web consiste à demander aux utilisateurs de fournir un nom d'utilisateur et un mot de passe valides. L'authentification est également utilisée pour autoriser ou interdire l'accès à certaines pages ou à certaines ressources.

## 110. Implémenter un contrôle d'accès

Le script suivant (secret.php3) fournit un mécanisme d'authentification simple, permettant aux utilisateurs autorisés de voir le contenu d'une page :

```
< ?
echo "<div align = 'center'>";
if( !isset($HTTP_POST_VARS["nom"]) && !isset($HTTP_POST_VARS["mdp"]) )
{
    // les variables ne sont pas définies, le visiteur doit donc saisir un nom et un mot de passe
?>

    <h1>Veuillez vous identifier</h1>
    <form method = POST action = 'secret.php3'>
    Nom&nbsp;:&nbsp;
    <input type = text name = 'nom' />
    <br><br>
    Mot de passe&nbsp;:&nbsp;
    <input type = password name = 'mdp' />
    <br><br>
    <input type = submit value = 'Envoyer' />
    </form>

< ?
}
else if( ($HTTP_POST_VARS["nom"] == "util1") && ($HTTP_POST_VARS["mdp"] == "passwd1") )
{
    // le nom et le mot de passe du visiteur sont corrects
    echo "<h1>Bienvenue sur notre site !</h1>";
}
}
```

```
else
{
    // le nom et le mot de passe du visiteur ne sont pas corrects
    echo "<h1>Vous n'êtes pas autorisé à visiter notre site !</h1>";
}
echo "</div>";
?>
```

Ce script pose néanmoins plusieurs problèmes :

- Il possède un nom d'utilisateur et un mot de passe inscrits directement dans son code source.
- Il enregistre et transmet le mot de passe en clair.
- Il ne protège qu'une seule page.

Ces problèmes peuvent être résolus de différentes manières.

### 110.1. Enregistrement des mots de passe

Il vaut mieux éviter d'enregistrer les noms d'utilisateur et les mots de passe dans un script. A l'intérieur d'un script, ces données sont difficiles à modifier. L'enregistrement de ces informations dans un autre fichier de votre serveur vous permet d'écrire plus facilement un programme pour ajouter ou pour supprimer des utilisateurs, et de modifier leur mot de passe.

Si vous avez l'intention d'enregistrer un grand nombre d'éléments (plus de 100) dans un fichier pour pouvoir les parcourir ultérieurement, il vaut mieux utiliser une base de données.

L'utilisation d'une base de données pour enregistrer les noms d'utilisateur et les mots de passe ne rend pas le script beaucoup plus complexe, et cela vous permet d'authentifier très rapidement plusieurs utilisateurs différents. Cela vous permet aussi d'écrire facilement un script pour ajouter de nouveaux utilisateurs, pour supprimer des utilisateurs, et pour permettre aux utilisateurs de modifier leur mot de passe.

Le script suivant (secret.php3) permet d'authentifier les visiteurs d'une page grâce à une base de données :

```
<html>
<head>
<title>Authentification des visiteurs avec base de données</title>
</head>
<body>
< ?
echo "<div align = 'center'>";
if( !isset($HTTP_POST_VARS["nom"]) && !isset($HTTP_POST_VARS["mdp"]) )
{
    // les variables ne sont pas définies, le visiteur doit donc saisir un nom et un mot de passe
    ?>
    <h1>Veuillez vous identifier</h1>
    <form method = POST action = 'secretdb.php3'>
    Nom&nbsp;  :&nbsp;  
    <input type = text name = 'nom' />
    <br><br>
    Mot de passe&nbsp;  :&nbsp;  
    <input type = password name = 'mdp' />
    <br><br>
    <input type = submit value = 'Envoyer' />
    </form>
< ?
}
else
{
    $nom = $HTTP_POST_VARS["nom"];
    $mdp_code = crypt($HTTP_POST_VARS["mdp"],"xx");

    // connexion à MySQL
    mysql_pconnect("localhost","","") or die("Connexion au serveur MySQL impossible ! : ".mysql_stat());
```

```
// sélection de la base de données 'resa_salles'
mysql_select_db('resa_salles') or die("Sélection de la base de données 'resa_salles' impossible ! : ".
mysql_error());

// interrogation de la table 'util' pour chercher un enregistrement qui correspond
$req = "select * from util where nom_util = '". $nom. "' and mdp_util = '". $mdp_code. "'";
$id_result = mysql_query($req);

if(!$id_result)
    echo "Requête non valide !";
else
{
    $nb = mysql_num_rows($id_result);

    if( $nb > 0 )
    {
        // le nom et le mot de passe du visiteur sont corrects
        $row = mysql_fetch_array($id_result);
        echo "<h1>". $row['nom_util']. ", Bienvenue sur notre site !</h1>";
    }
    else
    {
        // le nom et le mot de passe du visiteur ne sont pas corrects
        echo "<h1>Vous n'êtes pas autorisé à visiter notre site !</h1>";
    }
}
}
echo "</div>";
?>
</body>
</html>
```

## Veillez vous identifier

Nom :

Mot de passe :

**util1, Bienvenue sur notre site !**



## 110.2. Exercice pratique

Ecrire le script (create\_table\_util.php3) qui réalisera les opérations suivantes :

Connexion au serveur MySQL...  
Connexion au serveur MySQL réussie ! 3.23.47-nt  
Sélection de la base de données 'resa\_salles'...

### **Sélection de la base de données 'resa\_salles' réussie !**

Début de la destruction de l'ancienne table util...  
Fin de la destruction de l'ancienne table util.  
Début de la création de la table util (clé primaire index\_util, nom\_util, mdp\_util)...  
Fin de la création de la table util.

Insertion de l'utilisateur 'util1' mot de passe 'passwd1' crypté, dans la table util...  
affichage du nom et du mot de passe codé des utilisateurs :

nom : util1, mdp : xxRnesCEoZKio

### 110.3. Protéger plusieurs pages

Le moyen le plus simple de protéger plusieurs pages consiste à utiliser les mécanismes de contrôle d'accès fournis par votre serveur Web.

Vous pouvez ajouter les informations que les utilisateurs ont saisies dans chaque lien hypertexte de la page. Comme les noms d'utilisateurs peuvent contenir des espaces, ou d'autres caractères interdits dans les URL, vous devez vous servir de la fonction `urlencode()` pour coder correctement ces caractères.

Cependant, cette approche possède quelques problèmes. En effet, comme les données d'authentification sont incluses dans les pages Web envoyées à l'utilisateur, elles sont visibles dans l'historique du navigateur. De plus, comme ces données sont échangées entre le navigateur et le serveur pour chaque page demandée, elles sont transmises beaucoup plus souvent qu'il n'est nécessaire.

Il existe deux méthodes intéressantes pour résoudre ces problèmes : les sessions et l'authentification de base du protocole HTTP.

L'authentification de base permet de résoudre le problème de l'historique, mais le navigateur envoie toujours le mot de passe au serveur lors de chaque requête.

Le mécanisme de contrôle de session, vu précédemment, permet de résoudre ces deux problèmes, mais il est implémenté qu'à partir de PHP4.

### 110.4. Utiliser l'authentification de base dans PHP

Heureusement, l'authentification des utilisateurs est une tâche très courante, c'est pourquoi le protocole HTTP intègre cette fonction. Les scripts et les serveurs Web peuvent demander une authentification à un navigateur Web. Ce dernier s'occupe alors d'afficher une boîte de dialogue ou une fenêtre comparable pour demander les informations nécessaires à l'utilisateur.

Bien que le serveur Web redemande les détails d'authentification pour chaque requête d'utilisateur, le navigateur Web n'a pas besoin de demander ces informations à l'utilisateur pour chaque page. Généralement, le navigateur conserve ces informations tant qu'une fenêtre de navigation reste ouverte, et les renvoie automatiquement au serveur Web, sans les demander à l'utilisateur.

Cette caractéristique du protocole HTTP est appelée l'authentification de base, et vous pouvez l'activer avec PHP.

L'authentification de base transmet le nom de l'utilisateur et son mot de passe en texte brut, c'est pourquoi elle n'est pas très sécurisée. Le protocole HTTP 1.1 définit une méthode plus sécurisée que l'on appelle une authentification de résumé, qui se sert d'un algorithme de hachage (généralement MD5) pour cacher les détails de cette transaction. L'authentification de résumé est supportée par plusieurs serveurs Web, mais n'est pas supportée par la majorité des navigateurs Web. L'authentification de résumé est désormais supportée depuis la version 5.0 d'Internet Explorer, et elle pourrait être incluse dans la version 6.0 de Netscape Navigator.

L'authentification de base fournit donc un niveau de sécurité assez faible, comparable à celui qui est utilisé traditionnellement pour se connecter à des ordinateurs via Telnet ou FTP, en transmettant les mots de passe en texte brut. L'authentification de résumé est un peu plus sécurisée, puisqu'elle crypte les mots de passe avant de les transmettre, mais aucune de ces deux méthodes ne fournit à l'utilisateur l'assurance qu'il communique avec l'ordinateur auquel il avait l'intention d'accéder.

Grâce à l'utilisation de SSL et des certificats numériques, toutes les parties d'une transaction Web peuvent être protégées d'une manière fiable. Cependant, dans la plupart des cas, une méthode relativement rapide et peu sécurisée, comme l'authentification de base, est suffisante.

L'authentification de base permet de protéger un domaine de sécurité en demandant aux utilisateurs de fournir un nom d'utilisateur et un mot de passe valides. Ces domaines de sécurité possèdent chacun un nom, pour qu'il puisse en exister plusieurs sur un même serveur. Différents fichiers ou dossiers d'un même serveur peuvent appartenir à différents domaines de sécurité, chacun étant protégé par un nom d'utilisateur et un mot de passe différents. Les domaines de sécurité permettent également de regrouper plusieurs dossiers sur un même hôte, pour les protéger avec un seul mot de passe.

Les scripts PHP sont en général compatibles entre différentes plates-formes, mais l'utilisation de l'authentification de base se fonde sur des variables d'environnement définies par le serveur. Pour qu'un script d'authentification HTTP puisse être exécuté sur Apache avec PHP, sous la forme d'un module Apache, ou sur IIS à l'aide de PHP, sous la forme d'un module ISAPI, le script doit détecter le type du serveur et se comporter en conséquence.

Dans le script suivant (auth\_base.php3), nous allons voir que si l'utilisateur n'a pas encore fourni d'informations d'authentification, elles lui seront demandées. Si des informations erronées ont été fournies, l'utilisateur obtient un message d'erreur. Si l'utilisateur saisit un nom d'utilisateur et un mot de passe valides, il peut accéder au contenu de la page.

```
<?
//si nous utilisons IIS, nous devons définir $PHP_AUTH_USER et $PHP_AUTH_PW
if( (substr($_SERVER["SERVER_SOFTWARE"],0,9) == "Microsoft") && !
isset($_SERVER["PHP_AUTH_USER"]) &&
!isset($_SERVER["PHP_AUTH_PW"]) &&
(substr($_SERVER["HTTP_AUTHORIZATION"],0,6) == "Basic ") )
{
    list($_SERVER["PHP_AUTH_USER"],$_SERVER["PHP_AUTH_PW"]) =
    explode(":",base64_decode(substr($_SERVER["HTTP_AUTHORIZATION"],6)));
}

if (!isset($_SERVER["PHP_AUTH_USER"]))
{
    // utilisation de l'authentification de base, realm est le nom de domaine de sécurité
    header("WWW-Authenticate: Basic realm=\"Mon application\"");
}

// l'instruction suivante peut être remplacée par une requête de base de données
if (($_SERVER["PHP_AUTH_USER"] != "util1") || ($_SERVER["PHP_AUTH_PW"] != "passwd1"))
{
    if(substr($_SERVER["SERVER_SOFTWARE"],0,9) == "Microsoft")
        header("Status: 401 Unauthorized"); // Microsoft IIS
    else
        header("HTTP/1.0 401 Unauthorized"); // Apache

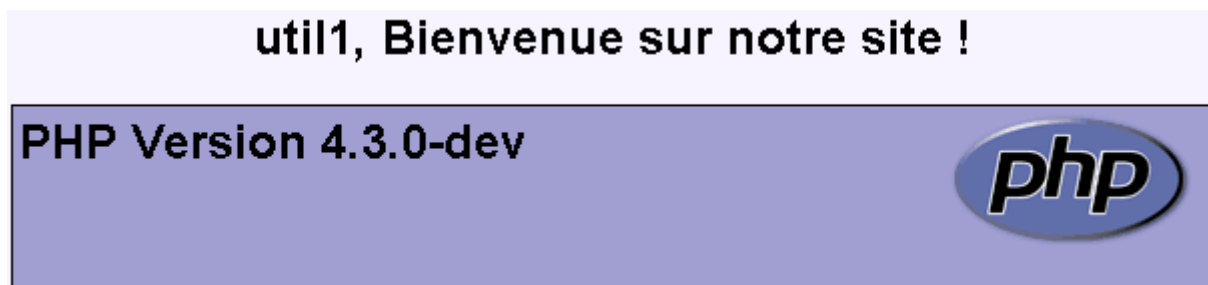
    echo "<div align='center'>";
    echo "<h1> Impossible ! Vous n'êtes pas autorisé !</h1>";
    echo "</div>";
}
else
{
    // l'utilisateur est identifié
    echo "<div align='center'>";
    echo "<h1>".$_SERVER["PHP_AUTH_USER"].", Bienvenue sur notre site !</h1>";
    echo "</div>";
    ?>
    <html>
    <head>
    <title>Utilisation d'une authentification de base dans PHP</title>
    </head>
    <body>
    <?
    phpinfo();
    ?>
    </body>
    </html>
    <?
}
?>
```

	L'authentification de base doit être placée au début du script, car elle doit être la première requête d'en-tête envoyée par le navigateur.
--	---

Exécution du script auth\_base.php3 :

La boîte de dialogue est affichée par le navigateur de l'utilisateur :

Les informations saisies par l'utilisateur sont valides :



Comme le mécanisme d'authentification est assisté par des caractéristiques intégrées dans le navigateur, celui-ci peut gérer à sa façon les échecs d'authentification. Par exemple, Internet Explorer permet aux utilisateurs d'effectuer trois tentatives d'authentification avant d'afficher une page d'erreur. Netscape Navigator permet aux utilisateurs d'effectuer autant de tentatives qu'ils le veulent, en affichant une boîte de dialogue de confirmation après un échec. Netscape n'affiche la page d'erreur que si l'utilisateur clique sur le bouton 'Annuler' de cette boîte de dialogue.

### 110.5. Utiliser l'authentification de base avec les fichiers .htaccess d'Apache

Vous pouvez obtenir des résultats analogues à ceux du script précédent sans écrire de code PHP.

Le serveur Web Apache contient plusieurs modules d'authentification différents qui peuvent être utilisés pour vérifier la validité des informations saisies par l'utilisateur.

Le plus simple est *mod\_auth*, qui compare une paire *nom\_d'utilisateur/mot\_de\_passe* aux lignes d'un fichier texte qui se trouve sur le serveur.

Pour obtenir le même résultat que celui du script précédent, vous devez créer deux fichiers HTML différents, un pour le contenu à afficher, et un autre pour la page d'erreur.

Le fichier 'content.html' contient la page protégée que les utilisateurs autorisés ont le droit d'afficher.

Le fichier 'rejet.html' contient la page d'erreur. Il est facultatif de proposer une page d'erreur, mais il s'agit d'une petite amélioration que vous pouvez utiliser pour fournir certaines informations à vos utilisateurs. Etant donné que cette page sera affichée lorsqu'un utilisateur ne s'est pas authentifié correctement, il peut être intéressant de lui expliquer comment s'enregistrer pour obtenir un mot de passe, ou comment se faire envoyer son mot de passe s'il l'a oublié.

Script content.html (exemple de page protégée) :

```
<html>
<body>
<div align='center'><h1>Bienvenue sur notre site !</h1></div>
</body>
</html>
```

Script rejet.html (exemple de page d'erreur) :

```
<html>
<body>
<div align='center'><h1>Vous n'êtes pas autorisé à visiter notre site !</h1></div>
</body>
</html>
```

Le fichier '.htaccess' permet d'activer l'authentification de base dans un dossier et aux sous-dossiers de son propre dossier. Plusieurs paramètres peuvent être définis dans un fichier '.htaccess', mais les six lignes de l'exemple ci-dessous sont toutes en rapport avec l'authentification.

Exemple de fichier '.htaccess' :

```
ErrorDocument 401 /rejet.html
AuthUserFile /home/book/.htpass
AuthGroupFile /dev/null
AuthName "nom_du_domaine_sécurisé"
AuthType Basic
Require valid-user
```

La première ligne :

```
ErrorDocument 401 /rejet.html
```

indique à Apache le document à afficher pour les visiteurs qui n'ont pas réussi à s'authentifier. Vous pouvez utiliser d'autres directives ErrorDocument pour proposer vos propres pages d'erreur, à la place des erreurs HTTP standard. En voici la syntaxe : *ErrorDocument numero\_d\_erreur URL*

- Pour une page qui gère l'erreur 401, il est important que son URL soit disponible pour tout le monde. En effet, il n'est pas très intéressant de fournir une page d'erreur personnalisée indiquant aux utilisateurs que leur authentification a échoué, si cette page se trouve dans un dossier nécessitant une authentification.

La ligne :

```
AuthUserFile /home/book/.htpass
```

indique à Apache l'endroit où il peut trouver le fichier contenant les mots de passe des utilisateurs autorisés. Ce fichier s'appelle souvent '.htpass', mais vous pouvez lui donner n'importe quel nom. Le nom de ce fichier n'a aucune importance, contrairement à son emplacement. Il ne doit pas se trouver dans l'arborescence Web (/Apache Group/...), ou dans n'importe quel dossier accessible depuis le serveur Web. Un exemple de fichier 'htpass' est présenté un peu plus loin.

La ligne :

```
AuthGroupFile /dev/null
```

permet d'indiquer que seuls les utilisateurs autorisés et qui appartiennent à certains groupes peuvent accéder aux ressources. Ici *AuthGroupFile* pointe vers le fichier '/dev/null', cela signifie que ce mécanisme ne nous intéresse pas. En effet, le fichier '/dev/null' est un fichier spécial des systèmes Unix qui ne contient rien du tout.

La ligne :

```
AuthName "nom_du_domaine_sécurisé"
```

fournit un nom de domaine de sécurité pour se servir de l'authentification HTTP. Ce nom apparaîtra à vos visiteurs.

La ligne :

```
AuthType Basic
```

spécifie la méthode d'authentification utilisée.

La dernière ligne :

```
Require valid-user
```

spécifie que n'importe quel utilisateur authentifié a le droit d'accéder aux pages. Vous pouvez également désigner des utilisateurs ou des groupes particuliers.

Exemple de fichier '.htpass' :

```
util1 : 0bL5eKkd7beL4  
util2 : hd4Kkj2gFR22s  
util3 : zK5kjGF25fhLK  
util4 : 2JgiK4Fk1jffD
```

Chaque ligne du fichier '.htpass' contient un nom d'utilisateur, le caractère deux-points ':', et le mot de passe crypté de cet utilisateur. Pour créer ce fichier, vous pouvez vous servir d'un petit programme appelé 'htpasswd', fourni avec la distribution d'Apache.

Le programme 'htpasswd' peut être utilisé de l'une des manières suivantes :

```
htpasswd -[cmdps] nom_du_fichier_des_mots_de_passe nom_d_utilisateur
```

Ou :

```
htpasswd -b[cmdps] nom_du_fichier_des_mots_de_passe nom_d_utilisateur mot_de_passe_utilisateur
```

La seule option que vous devrez utiliser est -c. cette option demande à htpasswd de créer le fichier. Vous devez utiliser cette syntaxe pour le premier utilisateur que vous ajoutez. Faites attention de ne pas l'utiliser pour les autres utilisateurs, puisque si le fichier existe déjà, htpasswd le supprime et en crée un nouveau.

Les options m, d, p et s peuvent être utilisées si vous souhaitez spécifier l'algorithme de cryptage à utiliser (ou bien aucun d'entre eux).

L'option b demande au programme de prendre le mot de passe en paramètre, au lieu de le demander. Cette approche est intéressante si vous souhaitez appeler htpasswd d'une manière non interactive, dans un processus de traitement par lots, mais elle ne doit pas être utilisée si vous invoquez htpasswd à partir de la ligne de commande.

Les commandes suivantes ont été utilisées pour créer le fichier présenté ci-dessus :

```
htpasswd -bc /home/book/.htpass util1 passwd1  
htpasswd -b /home/book/.htpass util2 passwd2  
htpasswd -b /home/book/.htpass util3 passwd3  
htpasswd -b /home/book/.htpass util4 passwd4
```

Utilisation :

Les noms d'utilisateurs et les mots de passe sont enregistrés dans un fichier texte. Chaque fois qu'un navigateur demande un fichier qui est protégé par le fichier '.htaccess', le serveur doit analyser le fichier '.htaccess', puis analyser le fichier des mots de passe, et tenter de trouver un nom d'utilisateur et un mot de passe qui correspondent. Au lieu d'utiliser un fichier '.htaccess', vous pouvez spécifier les mêmes éléments dans le fichier 'httpd.conf', fichier de configuration principal du serveur Web (/Apache Group/...).

Un fichier '.htaccess' est analysé chaque fois qu'un fichier est demandé. En revanche, le fichier 'httpd.conf' n'est analysé qu'une seule fois, au démarrage du serveur. Cette approche est donc plus rapide, mais elle signifie que si vous souhaitez apporter des modifications, il faut interrompre et redémarrer le serveur.

Le fichier des mots de passe quant à lui, devra être analysé pour chaque requête. Cela signifie que, comme pour les autres techniques, cette approche n'est pas appropriée pour authentifier des centaines voire des milliers d'utilisateurs.

## 110.6. Utiliser l'authentification mod\_auth\_mysql

Comme vous l'avez déjà vu, l'utilisation de *mod\_auth* avec Apache est à la fois simple à configurer et relativement efficace. Comme cette méthode enregistre les utilisateurs dans un fichier texte, elle n'est pas tout à fait adaptée aux sites importants possédant un grand nombre d'utilisateurs.

Heureusement, vous pouvez bénéficier de la simplicité de *mod\_auth* et de la vitesse des bases de données en utilisant *mod\_auth\_mysql*. Ce module fonctionne de la même manière que *mod\_auth*, mais comme il se sert d'une base de données MySQL à la place d'un fichier texte, il peut parcourir beaucoup plus rapidement de grandes listes d'utilisateurs.

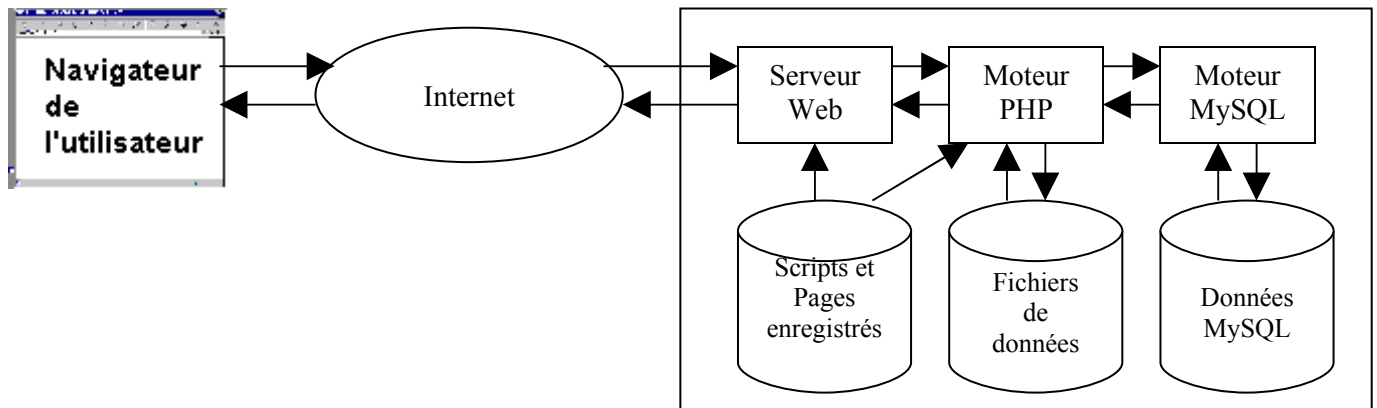
Pour pouvoir l'utiliser, vous devez compiler et installer le module correspondant sur votre système, ou demander à votre administrateur système de l'installer. Vous devrez donc configurer Apache et MySQL en suivant les instructions que vous trouverez dans les fichiers README et USAGE qui font partie de la distribution. Vous pourrez obtenir la dernière version de la distribution sur le site <http://www.zend.com> ou sur <http://www.mysql.com/download/contrib.html>.

## 111. Notions de transactions sécurisées SSL

Pour fournir des transactions sécurisées sur Internet, il faut examiner le flux d'informations de votre système, et vous assurer qu'en chaque point vos informations sont sécurisées. Dans le contexte de la sécurité réseau, il n'existe aucune méthode absolue. Aucun système n'est jamais parfaitement impénétrable.

Une transaction sécurisée est donc une transaction qui demande beaucoup d'efforts pour être piratée, par rapport aux informations qu'elle transporte.

Si vous devez réellement étudier la sécurité de votre système, il faut donc examiner le flux des informations qui passent au travers de chacune des parties de ce système. Le flux des informations d'un utilisateur dans une application typique, écrite à l'aide de PHP et de MySQL est le suivant :



Les détails de chaque transaction sur votre système peuvent varier légèrement, en fonction de l'architecture de votre système, et des données ou des actions des utilisateurs qui ont entraîné la transaction, mais le principe reste le même. Chaque transaction entre une application Web et un utilisateur commence par un envoi au niveau du navigateur de l'utilisateur d'une requête vers le serveur Web, passant par Internet. Si la page est un script PHP, le serveur Web délègue le traitement de la page au moteur PHP.

Le script PHP peut lire ou écrire des informations sur le disque. Il peut également faire appel à d'autres fichiers PHP ou HTML, avec *include()* ou *require()*. Il peut également envoyer des requêtes SQL au démon MySQL, et recevoir les réponses correspondantes. Le moteur MySQL s'occupe de lire et d'écrire ses propres données sur le disque.

Ce système est composé de trois parties :

1. **L'ordinateur de l'utilisateur** : celui-ci fait tourner un navigateur Web. Vous ne pouvez pas contrôler les autres facteurs de ce système, par exemple sa sécurité. Il faut se rappeler que cet ordinateur peut être très peu sécurisé, ou qu'il peut même s'agir d'un terminal partagé. Il faut également savoir que certaines personnes désactivent plusieurs caractéristiques qu'ils considèrent comme peu sécurisées, par exemple Java, les cookies ou JavaScript.

Les fonctionnalités fournies par PHP peuvent être compatibles avec n'importe quel navigateur, puisque le résultat final est essentiellement une page HTML. A partir du moment où l'on traite avec des scripts JavaScript (à part peut-être les plus simples), il faut prendre en compte les capacités et la configuration de chaque navigateur.

Du point de vue de la sécurité, il vaut mieux avoir recours à des scripts PHP au niveau du serveur pour la validation des données, puisque de cette manière votre code source n'est pas accessible aux utilisateurs. Si vous validez des données avec JavaScript, les utilisateurs pourront récupérer la source du script, et éventuellement la modifier.

Les données qui doivent être conservées peuvent être enregistrées sur le serveur Web, dans votre base de données, ou sur l'ordinateur de l'utilisateur sous la forme de cookies. Néanmoins, si les informations sont stockées en dehors de votre système, vous ne pouvez pas contrôler leur niveau de sécurité, vous ne pouvez pas vous assurer que l'utilisateur ne les effacera pas, et vous ne pouvez pas empêcher l'utilisateur de les modifier.

2. Internet : celui-ci possède plusieurs caractéristiques très intéressantes, mais il s'agit par essence d'un réseau peu sécurisé. Lorsque vous envoyez des informations d'un point à un autre, il ne faut pas oublier que d'autres personnes peuvent intercepter ou modifier les informations transmises. Vous pouvez donc choisir de crypter les informations avant de les transmettre, pour garantir leur confidentialité ou pour les protéger contre d'éventuelles tentatives de modification.

Il existe au moins deux moyens de sécuriser les informations que vous transmettez sur Internet/Intranet :

- SSL (*Secure Sockets Layer*)
- S-HTTP (*Secure Hyper Text Transfer Protocol*)

Ces deux technologies permettent de transmettre des messages à la fois confidentiels et robustes, mais SSL est déjà largement utilisé et disponible, alors que S-HTTP n'en est qu'à ses débuts.

3. Votre système : Il convient d'envisager le cas d'une transmission de données entre plusieurs composants de votre système, en passant par un réseau. A titre d'exemple, on peut citer une base de données MySQL qui se trouverait sur un autre ordinateur que votre serveur Web. Dans ce cas, PHP se connecte à votre serveur MySQL via TCP/IP, et cette connexion n'est pas cryptée. Si les deux ordinateurs participants à cette communication appartiennent à un réseau local privé, vous devez vous assurer que ce réseau est sécurisé. Si les ordinateurs communiquent via Internet, votre système sera probablement ralenti, et vous devrez traiter cette connexion de la même manière que toute autre connexion passant par Internet.

PHP ne possède aucun moyen intégré d'établir ses connexions via SSL. L'instruction *fopen()* supporte HTTP, mais pas HTTPS. Cependant, vous pouvez utiliser SSL grâce à la bibliothèque cURL.

Il est important que vos visiteurs, à partir du moment où ils pensent traiter avec vous, soient réellement en connexion avec votre site. Vous pouvez vous procurer un certificat numérique pour les protéger d'un autre site qui se ferait passer pour le votre, mais aussi pour vous permettre d'utiliser SSL sans afficher de message d'avertissement chez vos utilisateurs.

### 111.1. Utilisation de SSL

Le SSL a été créé par Netscape ; il est intégré dans tous les navigateurs du marché. Il assure l'authentification, la confidentialité et l'intégrité des données échangées, au moyen d'un système de cryptographie, qui utilise le résultat d'opérations effectuées sur des nombres premiers : l'algorithme à clé publique RSA (dont le nom est formé des initiales de ses concepteurs : Rivest, Shamir et Adleman).

Les versions 2 et 3 de SSL sont très largement supportées. La plupart des serveurs Web ont été conçus pour intégrer ce protocole, ou pour accepter des modules correspondants. Internet Explorer et Netscape Navigator supportent ce protocole depuis la version 3. Dans un futur proche, SSL 3.0 sera remplacé par TLS 1.0 (*Transport Layer Security*). TLS devrait être un standard réellement ouvert et utilisable par le reste du monde, et non un standard défini par une organisation particulière. Il est fondé directement sur SSL 3.0, mais il contient des améliorations destinées à combler certaines faiblesses de SSL.



Les protocoles réseau et les logiciels qui les implémentent sont généralement organisés en couches, que l'on appelle aussi des niveaux. Chaque niveau peut transmettre des données (ou demander des services) au niveau supérieur ou au niveau inférieur.

Lorsque vous vous servez du protocole HTTP pour transférer des informations, celui-ci fait appel au protocole TCP (*Transmission Control Protocol*), qui à son tour s'appuie sur le protocole d'Internet (IP). Ce protocole a lui-même besoin d'un protocole approprié pour le périphérique réseau qui s'occupe de transmettre les paquets de données vers leur destination, en passant par un support électrique.

Lorsque des données sont envoyées, elles traversent ces couches de l'application jusqu'au réseau physique. Lorsque des données sont reçues, elles traversent ces couches dans l'autre sens, du réseau physique jusqu'à l'application.

L'utilisation de SSL ajoute un niveau supplémentaire, ainsi que des protocoles dans le niveau d'application. Le niveau SSL se situe entre le niveau de transport et le niveau d'application.

Le niveau SSL modifie les données de votre application HTTP avant de les envoyer au niveau de transport, qui les enverra vers leur destination.

HTTP	Protocole d'accord SSL	Changement de Cryptage SSL	Protocole d'alerte SSL	...	Niveau d'application
Protocole d'enregistrement SSL					Niveau SSL
TCP					Niveau de transport
IP					Niveau réseau
Hôte vers réseau					Niveau hôte vers réseau

## 111.2. Processus

Le navigateur de l'utilisateur entre en communication avec le serveur. Ce dernier possède déjà sa paire de clés publique/privée. C'est cette paire de clés qu'il utilise dans ses communications avec tous les logiciels clients. Le processus est le suivant :

1. Une fois reconnu par le logiciel serveur, le logiciel client génère une paire de clés publique/privée.
2. Le logiciel client demande au logiciel serveur de lui fournir sa clé publique (celle du serveur).
3. La clé publique du client est aussitôt cryptée avec la clé publique du serveur, puis transmise au serveur.
4. Le serveur décode le message à l'aide de sa clé privée, puis authentifie la clé publique de l'utilisateur.
5. Le serveur envoie ensuite au logiciel client une confirmation cryptée, du bon déroulement de l'opération.

Toutes les informations futures, échangées entre l'utilisateur et le serveur, seront désormais cryptées. De plus, seul ce serveur est en mesure de communiquer avec cet utilisateur, car il est le seul à connaître la clé publique de cet utilisateur. L'utilisateur et le serveur peuvent maintenant échanger toutes les données voulues, de façon sûre. L'ensemble de ce processus est complètement transparent pour l'utilisateur.

Avec ce protocole, une nouvelle paire de clés est générée à chaque établissement de la communication entre le logiciel client de l'utilisateur et le logiciel serveur. La communication est donc entièrement sûre, mais en aucun cas le serveur ne peut s'assurer de l'identité de l'utilisateur à l'autre extrémité.

Une façon de résoudre ce problème est de joindre à ce processus un système de validation, par exemple un numéro d'identification personnel, qui s'obtient par une inscription préalable.

## 111.3. Fonctionnement

Le système repose sur l'algorithme RSA, un standard utilisé pour le cryptage des données et la signature de messages électroniques. Cet algorithme est très utilisé pour l'authentification et le cryptage des données dans le domaine informatique.

Deux paires de clés (une pour le verrouillage, l'autre pour le déverrouillage) à 40 bits sont utilisées. Chaque paire est composée d'une clé publique et d'une clé privée. La clé publique est distribuée, alors que la clé privée ne l'est jamais ; cette dernière est toujours gardée secrète. Les données cryptées avec la clé publique peuvent uniquement être décryptées avec la clé privée, et inversement.

## 111.4. Certificats

Un certificat est un document électronique qui atteste qu'une clé publique est bien liée à une organisation ou à un individu. Il permet la vérification de la propriété d'une clé publique, afin de prévenir la contrefaçon de clés publiques. Un certificat contient généralement une clé publique, un nom, ainsi que d'autres champs qui identifient le propriétaire, une date d'expiration, un numéro de série, le nom de l'organisation qui contresigne le certificat, et la signature elle-même. Le format des certificats est défini par la norme X509.

Le certificat donne donc l'assurance que les informations qu'il contient sont exactes. Pour cela, le certificat doit être généré par un tiers de confiance, c'est-à-dire un organisme indépendant, qui contrôle la véracité de ces informations. Le CA (*Certifying Authority*, l'organisme certificateur) donne la crédibilité au certificat.

Il existe deux types de certificats utilisés avec SSL : pour le serveur et pour le client. Techniquement, ils utilisent le même format, mais diffèrent par l'information qu'ils contiennent.

Ainsi, un certificat côté client sert à identifier un utilisateur ; il contiendra donc des informations sur cet utilisateur. Côté serveur, le certificat a pour mission d'authentifier le serveur, ainsi que l'organisme qui l'exploite. c'est ce type de certificat dont vous avez besoin dans la mise en place d'un serveur « sécurisé » HTTPS.

## 111.5. Installation de SSL

Pour pouvoir utiliser le protocole SSL avec le serveur HTTP Apache, il faut installer des modules spécifiques. Les modules nécessaires sont disponibles gratuitement sur Internet :

- OpenSSL 0.9.2b, disponible à l'adresse <http://www.openssl.org/>
- Mod\_ssl 2.2.8-1.3.6, disponible à l'adresse <http://www.modssl.org/>

Une fois ces modules installés, vous pouvez créer des certificats de test, que vous signerez en tant qu'autorité de certification. Ces certificats permettent de tester le bon fonctionnement du protocole SSL. Les navigateurs préviennent l'utilisateur que le serveur utilise des certificats qui n'ont pas été signés par une CA reconnue, mais cela permet malgré tout de tester le bon fonctionnement de l'installation.

Cependant, avant cela, il faut configurer le serveur Apache, afin de lui indiquer qu'il doit se mettre à l'écoute du port 443 (port classiquement utilisé par HTTPS), et également activer la protection par SSL sur les répertoires qui doivent être protégés.

# 16. Autres fonctionnalités offertes par PHP

## 112. La sérialisation

La sérialisation est le processus qui consiste à convertir un élément qui peut être stocké dans une variable, un tableau ou un objet PHP en un flux d'octets dédié à être stocké dans une base de données ou passé de pages en pages via une URL.

Sans sérialisation, il se révèle difficile de stocker ou de passer l'intégralité du contenu d'un tableau ou d'un objet.

La sérialisation a toutefois perdu de son utilité avec l'introduction du contrôle de session. La sérialisation des données était généralement utilisée pour des opérations pour lesquelles le contrôle de session est aujourd'hui mis en œuvre. En effet, les fonctions de contrôle sérialisent les variables de session.

Le programmeur PHP doit néanmoins encore régulièrement stocker des tableaux ou des objets PHP dans des fichiers et des bases de données. Deux fonctions se révèlent alors précieuses : *serialize()* et *unserialize()*.

Leur syntaxe est la suivante :

*String serialize(mixed objet) ;*

Exemple : *\$serial\_tab = serialize(\$tab) ;*

*Mixed unserialize(string serial\_objet) ;*

Exemple : *\$tab = unserialize(\$serial\_tab);*

Nous allons examiner le résultat renvoyé par la sérialisation d'un tableau d'entiers :

```
$tab = array();
for($i=0 ; $i < 11 ; $tab[$i] = $i*2, $i++);
$serial_tab = serialize($tab);
echo $serial_tab;
```

Résultat à l'affichage :

*a:11:{i:0;i:0;i:1;i:2;i:2;i:4;i:3;i:6;i:4;i:8;i:5;i:10;i:6;i:12;i:7;i:14;i:8;i:16;i:9;i:18;i:10;i:20;}*

Le premier caractère indique le type de l'objet sérialisé :

a : array  
o : objet  
s : string

Le nombre suivant correspond à la taille de l'objet (nombre d'éléments).

Entre les accolades nous trouvons le contenu de l'objet :

- pour un tableau nous avons le type et la valeur de chaque indice suivi du type et de la valeur de chaque élément,
- pour un objet nous avons le type, la longueur (pour les *string*), le nom suivi du type de la valeur, de la longueur de la valeur et de la valeur de chaque propriété.

Voici, en exemple, la sérialisation de l'objet *\$v1* défini et initialisé précédemment :

```
$serial_v1 = serialize($v1);
echo $serial_v1;
```

Résultat à l'affichage :

*O:7:"voiture":6:
{s:6:"marque";s:7:"Renault";s:6:"modele";s:7:"Safrane";s:9:"puissance";i:7;s:7:"couleur";s:5:"rouge"
;s:6:"moteur";s:6:"arrêté";s:9:"nb\_portes";i:3;}*

- Ne pas oublier d'utiliser la fonction *addslashes()* avant d'écrire des données de type chaîne de caractères dans une base de données.
- Pour replacer les données dans leur format d'origine, il suffit d'invoquer la fonction *unserialize()*.
- Bien entendu, si la fonction *addslashes()* a été invoquée avant le stockage de l'objet dans la base de données, la fonction *stripslashes()* doit être appelée avant de dé-sérialiser la chaîne.

## 113. Chargement dynamique d'extensions sous Windows

Vous pouvez choisir quelles extensions vous voulez charger en décommentant la ligne 'extension=php\_\*.dll' dans le fichier *php.ini*.

Des extensions (bibliothèques) peuvent être également chargées en cours d'exécution, si elles ne sont pas déjà compilées avec PHP, au moyen de la fonction *dl()*.

Syntaxe :

*dl(string nom\_du\_fichier\_contenant\_l\_extension) ;*

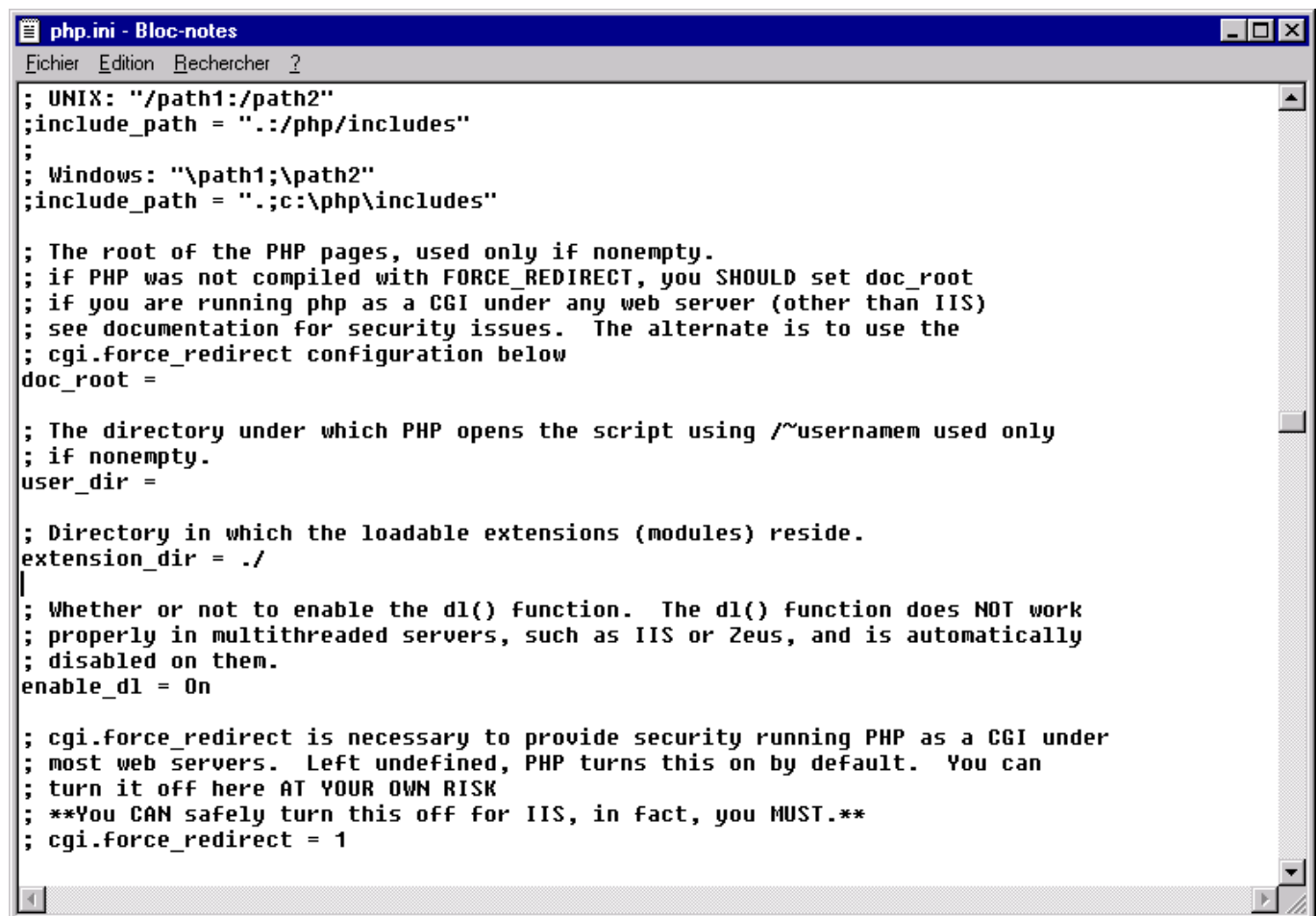
La fonction *dl()* prend comme argument le nom du fichier contenant la bibliothèque.

Sous Unix, les bibliothèques logicielles sont stockées dans des fichiers portant l'extension *.so*, tandis que sous Windows, elles sont stockées dans des fichiers dont le nom se termine par *.dll*.

La ligne de code, dans l'exemple suivant, provoque le chargement dynamique de l'extension FTP sur une machine Windows :

*dl("php\_ftp.dll") ;*

Ne spécifiez pas le dossier de stockage des extensions, mais spécifiez-le par contre dans le fichier de configuration *php.ini*. Le paramètre appelé *extension\_dir* spécifie en effet le dossier de stockage dans lequel PHP doit rechercher les bibliothèques à charger dynamiquement.



```
; UNIX: "/path1:/path2"
;include_path = "./:php/includes"
;
; Windows: "\\path1;\\path2"
;include_path = ".;c:\\php\\includes"

; The root of the PHP pages, used only if nonempty.
; if PHP was not compiled with FORCE_REDIRECT, you SHOULD set doc_root
; if you are running php as a CGI under any web server (other than IIS)
; see documentation for security issues.  The alternate is to use the
; cgi.force_redirect configuration below
doc_root =

; The directory under which PHP opens the script using /~username used only
; if nonempty.
user_dir =

; Directory in which the loadable extensions (modules) reside.
extension_dir = ./
|
; Whether or not to enable the dl() function.  The dl() function does NOT work
; properly in multithreaded servers, such as IIS or Zeus, and is automatically
; disabled on them.
enable_dl = On

; cgi.force_redirect is necessary to provide security running PHP as a CGI under
; most web servers.  Left undefined, PHP turns this on by default.  You can
; turn it off here AT YOUR OWN RISK
; **You CAN safely turn this off for IIS, in fact, you MUST.**
; cgi.force_redirect = 1
```

Si vous rencontrez des difficultés lors du chargement dynamique d'une extension, vérifiez l'ajustement du paramètre de configuration *enable\_dl* dans le fichier *php.ini* (voir la copie d'écran ci-dessus). Ce paramètre doit être activé (On) pour que l'interpréteur PHP puisse charger dynamiquement des extensions. Il peut toutefois être désactivé pour des raisons de sécurité, notamment si vous travaillez sur un ordinateur qui ne vous appartient pas. Par ailleurs, la fonction *dl()* n'est pas utilisable si PHP est exécuté en mode sécurisé.

Les fichiers DLLs des extensions PHP sont préfixés par 'php\_' en PHP 4, et 'php3\_' en PHP 3. Cela évite la confusion des extensions PHP et de leurs librairies.

En PHP 4.0.4pl1, les extensions BCMath, Calendar, COM, FTP, MySQL, ODBC, PCRE, Sessions, WDDX et XML sont activées **par défaut**. Vous n'avez rien à faire pour qu'elles soient incluses. Lisez le fichier *README.txt* ou *install.txt* dans votre distribution pour connaître la liste des modules par défaut.

Certaines de ces extensions requièrent des bibliothèques DLL supplémentaires pour fonctionner correctement. Certaines d'entre elles sont disponibles dans la distribution, dans le dossier *dlls* mais certaines (comme Oracle (php\_oci8.dll)), demandent des dlls qui ne sont pas dans la distribution.

Copiez les dlls fournies depuis le dossier *dlls* dans votre PATH Windows. Les bons emplacements sont typiquement

- c:\windows\system pour Windows 9x/Me
- c:\winnt\system32 pour Windows NT/2000

Si elles sont déjà installées sur votre système, ne les écrasez que si PHP ne fonctionne pas correctement (et de toutes manières, faites une sauvegarde de ces DLL, en cas de problème).

### Extensions PHP :

Extension	Description	Notes
php_bz2.dll	Fonctions de compression Bzip2	Aucune
php_calendar.dll	Fonctions de conversions calendaires (Depuis PHP 4.03, elles sont activées par défaut)	Activée automatiquement depuis PHP 4.0.3
php_cpdf.dll	Fonctions ClibPDF	
php_crypt.dll	Fonctions de cryptage	Aucune
php_ctype.dll	Fonctions ctype	Aucune
php_curl.dll	Fonctions CURL	Requiert :@: libeay32.dll, ssleay32.dll (fournies)
php_cybercash.dll	Fonctions de paiement Cybercash	Aucune
php_db.dll	Fonctions DBM	Obsolète. Utilisez DBA à la place (php_dba.dll)
php_dba.dll	Fonctions dbm-style	Aucune
php_dbase.dll	Fonctions DBase	Aucune
php3_dbm.dll	Librairie d'émulation GDBM via Berkely DB2	Aucune
php_domxml.dll	Fonctions DOM XML	Requiert :@: libxml2.dll (fournie)
php_dotnet.dll	Fonctions .NET	Aucune
php_exif.dll	Entêtes EXIF des images JPEG	Aucune
php_fbsql.dll	Fonctions FrontBase	Aucune
php_fdf.dll	Fonction FDF (Forms Data Format)	Requiert:@: fdfk.dll (fournie)
php_filepro.dll	Lecture des bases filepro	Accès en lecture seule
php_ftp.dll	Fonctions FTP	Activée par défaut depuis PHP 4.0.3
php_gd.dll	Bibliothèque GD (pour les manipulations d'images)	Aucune
php_gettext.dll	Fonctions GNU Gettext	Requiert :@: gnu_gettext.dll (fournie)
php_hyperwave.dll	Fonctions HyperWave	Aucune
php_iconv.dll	Fonctions de conversions ICONV	Requiert :@: iconv-1.3.dll (fournie)
php_ifx.dll	Fonctions Informix	Requiert:@: bibliothèques Informix
php_iisfunc.dll	Fonctions IIS	Aucune
php_imap.dll	Fonctions IMAP 4(en PHP 3:@: php3_imap4r1.dll)	PHP 3:@: php3_imap4r1.dll
php_ingres.dll	Fonctions Ingres II	Requiert:@: bibliothèques Ingres II
php_interbase.dll	Fonctions InterBase	Requiert:@: gds32.dll (fournie)
php_java.dll	Extension Java	Requiert:@: jvm.dll (fournie)
php_ldap.dll	Fonctions LDAP	Requiert:@: libsasl.dll (fournie)
php_mhash.dll	Fonctions Mhash	Aucune
php_ming.dll	Fonctions Ming pour Flash	Aucune
php_msql.dll	Fonctions mSQL	Requiert:@: msql.dll (fournie)
php3_msql1.dll	Fonctions mSQL 1	Aucune
php3_msql2.dll	Fonctions mSQL 2	Aucune
php_mssql.dll	Fonctions MSSQL (anciennement php_mssql70.dll, requiert MSSQL DB-Libraries)	Requiert:@: ntwdbib.dll (fournie)
php3_mysql.dll	Fonctions MySQL (Activées par défaut en PHP 4)	Aucune
php_nsmail.dll	Fonctions Netscape mail	Aucune
php3_oci73.dll	Fonctions Oracle	Aucune

php_oci8.dll	Fonctions Oracle 8	Requiert:@: librairies clientes Oracle 8
php_openssl.dll	Fonctions OpenSSL	Requiert:@: libeay32.dll (fournie)
php_oracle.dll	Fonctions Oracle	Requiert:@: librairies clientes Oracle 7
php_pdf.dll	Fonctions PDF	Aucune
php_pgsql.dll	Fonctions PostgreSQL	Aucune
php_printer.dll	Fonctions d'impression	Aucune
php_sablot.dll	Fonctions XSLT	Requiert:@: sablot.dll (fournie)
php_snmp.dll	Fonctions SNMP get et walk	NT uniquement!
php_sybase_ct.dll	Fonctions Sybase	Requiert:@: librairies clientes Sybase
php_yaz.dll	Fonctions YAZ	Aucune
php_zlib.dll	Fonctions ZLib	Aucune

### 113.1. Liste des extensions chargées

Vous pouvez facilement connaître les jeux de fonctions qui sont disponibles, et les fonctions disponibles au sein de chacun de ces jeux, via les fonctions `get_loaded_extensions()` et `get_extension_funcs()`.

Syntaxes :

`Array get_loaded_extensions(void) ;`

`Array get_extension_funcs(string extension) ;`

La fonction `get_loaded_extensions()` renvoie un tableau de tous les jeux de fonctions actuellement disponibles dans l'installation PHP. Cette fonction ne prend aucun paramètre.

La fonction `get_extension_funcs()` renvoie un tableau contenant toutes les fonctions disponibles dans le jeu de fonctions ou l'extension qui lui est passé comme argument.

L'information ainsi obtenue peut se révéler précieuse pour déterminer si l'installation d'une extension s'est déroulée avec succès.

Le script '`affiche_extensions.php3`' suivant liste toutes les extensions disponibles dans l'installation PHP courante et donne, pour chaque extension, la liste des fonctions disponibles :

```
<html>
<head>
<title>affichage des extensions disponibles</title>
</head>
<body>
<?
    echo "<div align='center'>";
    echo "<h2>Liste des fonctions supportées dans votre installation PHP</h2>";
    echo "</div>";
    $extensions = get_loaded_extensions();

    foreach($extensions as $each_extension)
    {
        echo "$each_extension<br>";
        echo "<ul>";
        $ext_fonctions = get_extension_funcs($each_extension);

        foreach($ext_fonctions as $fonction)
            echo "<li>$fonction";

        echo "</ul>";
    }
?>
</body>
</html>
```

# 17. Installation d'Apache, PHP et MySQL

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

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

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

## 114. Télécharger les sources

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

## 115. Installer Apache et PHP

1. Décompresser les archives:  

```
tar zxvf apache_1.3.x.tar.gz
tar zxvf php-3.0.x.tar (ou php-4.0.x.tar pour php4)
```
2. Configurer Apache  

```
cd apache_1.3.x
./configure --prefix=/www
```
3. Configurer PHP  

```
cd ../php-3.0.x (ou php-4.0.x pour php4)
./configure --with-mysql --with-apache=../apache_1.3.x --enable-track-vars
```

Si vous préférez installer PHP dans un autre répertoire, il faut utiliser l'option de configuration `--with-config-file-path=/path`
4. Compiler PHP  

```
make
make install
```
5. Installer Apache  

```
cd ../apache_1.3.x
./configure --prefix=/www --activate-module=src/modules/php3/libphp3.a
( OU pour php4 :
./configure --prefix=/www --activate-module=src/modules/php4/libphp4.a)
make
make install
```
6. Modifier le fichier de configuration de PHP  

```
cd ../php-3.0.x (ou php-4.0.x pour php4)
cp php3.ini-dist /usr/local/lib/php3.ini (ou php.ini pour php4)
```

(Vous pouvez désormais éditer le fichier de configuration `/usr/local/lib/php3.ini`.)
7. Editez le fichier de configuration du serveur apache (généralement `httpd.conf` ou `srm.conf`) et ajoutez la ligne suivante:  

```
AddType application/x-httpd-php3 .php3
```

(Il s'agit de choisir l'extension associée aux scripts PHP. Par souci d'homogénéité, il est courant de choisir l'extension **.php3** pour php3.0 ou **.php** pour php4)
8. Démarrez le serveur Apache. (Il est essentiel d'arrêter et redémarrer le serveur, et non uniquement de le relancer). Il suffit généralement de taper `apachectl stop`, puis `apachectl start`.

## 116. Premier lancement

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

```
<html>
<head><title>Exemple</title></head>
<body>
<?php
echo "PHP fonctionne!";
?>
</body>
</html>
```

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

`http://localhost/toto.php3`

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

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

---



## 18. Bibliographie

### Les ouvrages qui ont servi à l'élaboration de ce support :

- PHP & MySQL de Luke Welling et Laura Thomson aux éditions Campus Press
- Programmation Web avec PHP de L. Lacroix, N. Leprince, C. Boggero, et C. Lauer aux éditions Eyrolles
- Pratique de MySQL et PHP de Philippe Rigaux aux éditions O'REILLY
- L'intro PHP4 de Matt Zandstra aux éditions Campus Press

### Les sites Internet qui ont servi à l'élaboration de ce support :

- [www.commentcamarche.fr](http://www.commentcamarche.fr)
- [fr2.php.net/manual/fr](http://fr2.php.net/manual/fr)
- [www.yellis.net/docs](http://www.yellis.net/docs)

Avec les remerciements d'Olivier SAINT MARTIN et de Yannick GAILLABAUD.