PHP4 et MySQL

Hugo Etiévant

Dernière mise à jour : 12 octobre 2002

Sommaire

Liens

PHP*

- http://www.php.net
- http://www.phpinfo.net
- http://www.phpfrance.com
- http://www.developpez.com/php/

MySQL

- http://www.mysql.com/
- http://dev.nexen.net/docs/mysql/

HTML

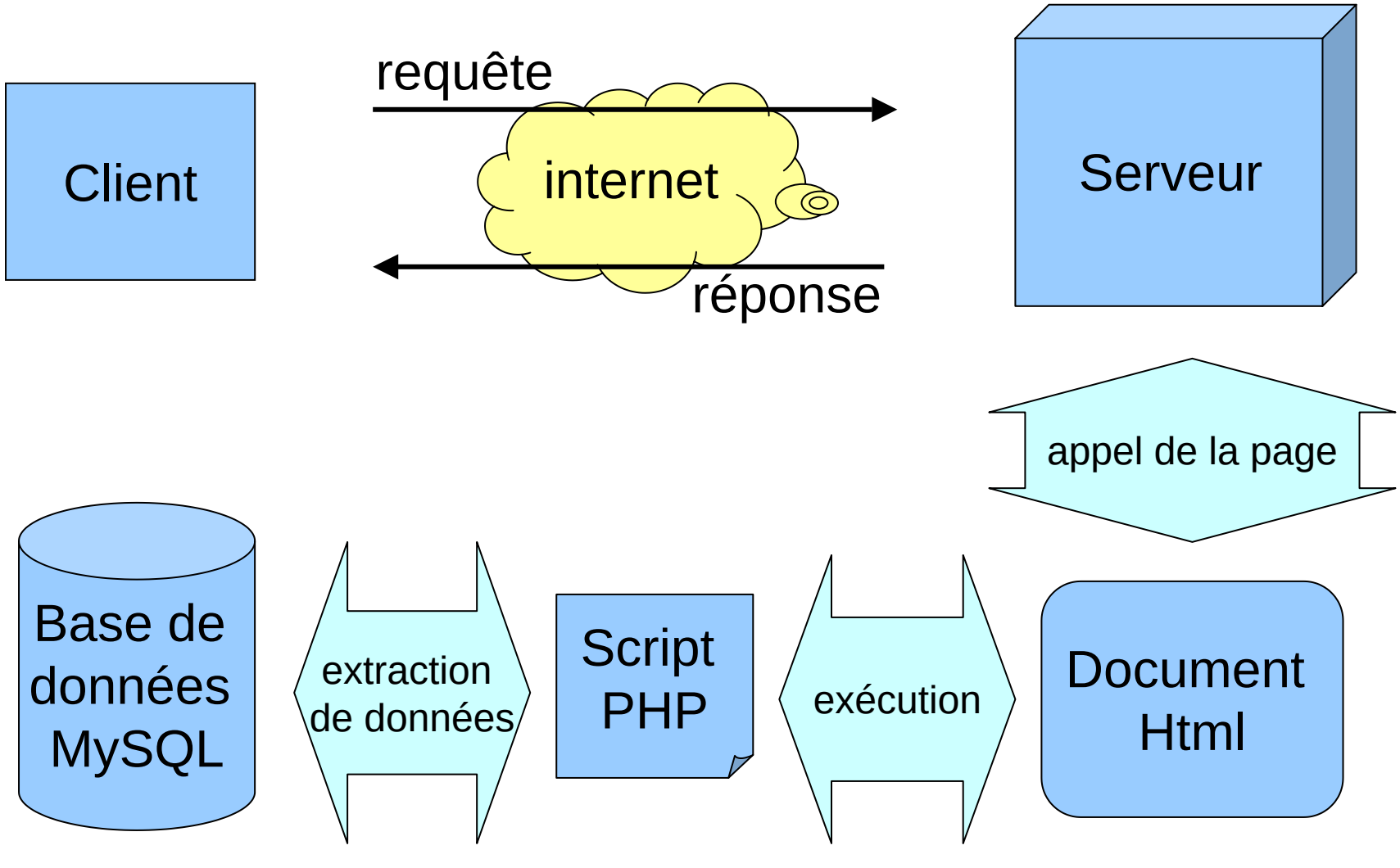
- http://cyberzoide.developpez.com/html/

Exemple concret

- http://www.miag-rezo.net

** PHP Hypertext Preprocessor (encore un acronyme récursif !)*

Modèle


Client
requête
internet
réponse
Serveur
appel de la page
Base de données MySQL
extraction de données
Script PHP
exécution
Document Html

Documentation en ligne

Pour obtenir en ligne toute la documentation officielle (en français) sur une commande, tapez l'URL suivante dans la barre d'adresse de votre navigateur Internet :

http://fr.php.net/

Et rajouter en fin d'URL le nom de la commande.

Exemple :

http://fr.php.net/echo

Partie 1 : PHP

La petite histoire du PHP

Il a été créé en 1994 par Rasmus Lerdorf pour les besoins des pages web personnelles (livre d'or, compteurs, etc.). A l'époque, PHP signifiait *Personnal Home Page*.

C'est un langage incrusté au HTML et interprété (PHP3) ou compilé (PHP4) côté serveur. Il dérive du C et du Perl dont il reprend la syntaxe. Il est extensible grâce à de nombreux modules et son code source est ouvert. Comme il supporte tous les standards du web et qu'il est gratuit, il s'est rapidement répandu sur la toile.

En 1997, PHP devient un projet collectif et son interpréteur est réécrit par Zeev Suraski et Andi Gutmans pour donner la version 3 qui s'appelle désormais *PHP : Hypertext Preprocessor* (acronyme récursif à l'exemple du système Open Source *Linux : Is Not UniX*).

Il existe par ailleurs des applications web prêtes à l'emploi (PHPNuke, PHP SPIP, PHPSlash…) permettant de monter facilement et gratuitement son portail. En juillet 2000 plus de 300.000 sites tournaient déjà sous PHP !

Intégration d'un script dans une page

Les pages web sont au format html. Les pages web dynamiques générées avec PHP4 sont au format php. Le code source php est directement insérer dans le fichier html grâce au conteneur de la norme XML :

<?php … ?>

Exemple:

```
<html>
<body>
<?php
        echo ''Bonjour'';
?>
</body>
</html>
```

Autres syntaxes d'intégration :

- **<? … ?>**
- **<script language=''php''> … </script>**
- **<% … %>**

Exemple de script

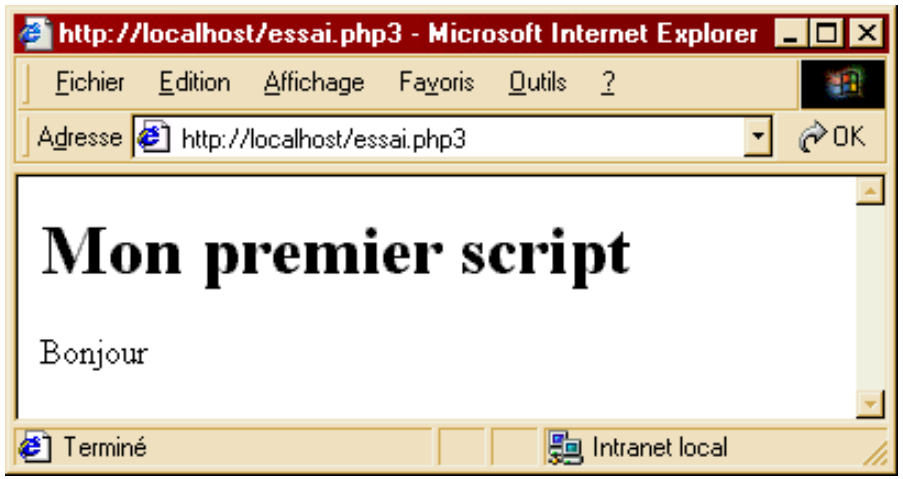
Exemple de script, code source (côté serveur) :

```
<html>
<body>
<h1>Mon premier script</h1>
<?php echo ''Bonjour\n''; ?>
</body>
</html>
```

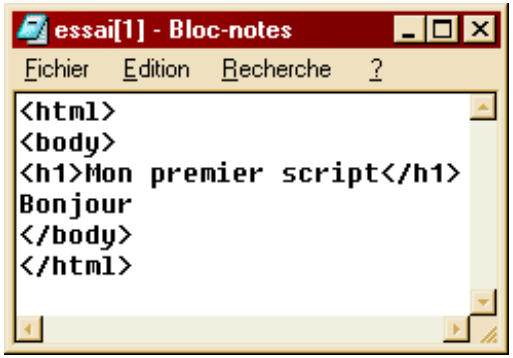
Autre écriture du même script :

```
<?php
echo ''<html>\n<body>\n'';
echo ''<h1>Mon premier script</h1>\n'';
echo ''Bonjour\n'';
echo ''</body>\n</html>\n'';
?>
```

Résultat affiché par le navigateur :



Code source (côté client) de la page essai.ph3 résultant du script



Commentaires

Un script php se commente comme en C :

Exemple :

```
<?php

// commentaire de fin de ligne

/* commentaire
sur plusieurs
lignes */

# commentaire de fin de ligne comme en Shell
?>
```

Tout ce qui se trouve dans un commentaire est ignoré. Il est conseillé de commenter largement ses scripts.

Variables, types et opérateurs (I)

Le typage des variables est implicite en php. Il n'est donc pas nécessaire de déclarer leur type au préalable ni même de les initialiser avant leur utilisation.

Les identificateurs de variable sont précédés du symbole « **$** » (dollars). Exemple : **$toto**.

Les variables peuvent être de type entier (**integer**), réel (**double**), chaîne de caractères (**string**), tableau (**array**), objet (**object**), booléen (**boolean**).

Il est possible de convertir une variable en un type primitif grâce au cast(1) (comme en C).

Exemple :

```
$str = ''12'';          // $str vaut la chaîne ''12''
$nbr = (int)$str;       // $nbr vaut le nombre 12
```

(1) : Le cast est une conversion de type. L'action de caster consiste en convertir une variable d'un type à un autre.

Variables, types et opérateurs (II)

Quelques fonctions :

empty($var) : renvoie vrai si la variable est vide

isset($var) : renvoie vrai si la variable existe

unset($var) : détruit une variable

gettype($var) : retourne le type de la variable

settype($var, ''type'') : convertit la variable en type **type** (cast)

is_long(), **is_double()**, **is_string()**, **is_array()**, **is_object()**, **is_bool()**, **is_float()**, **is_numeric()**, **is_integer()**, **is_int()…**

Une variable peut avoir pour identificateur la valeur d'une autre variable.

Syntaxe : **${$var} = valeur;**

Exemple :

```
$toto = ''foobar'';
${$toto} = 2002;
echo $foobar;                    // la variable $foobar vaut 2002
```

Variables, types et opérateurs (III)

La portée d'une variable est limitée au bloc dans lequel elle a été crée. Une variable locale à une fonction n'est pas connue dans le reste du programme. Tout comme une variable du programme n'est pas connue dans une fonction. Une variable créée dans un bloc n'est pas connue dans les autres blocs, mêmes supérieurs.

Opérateurs arithmétiques :

+ (addition), **-** (soustraction), ***** (multiplié), ***/*** (divisé), **%** (modulo), **++** (incrément), **--**(décrément). Ces deux derniers peuvent être pré ou post fixés

Opérateurs d'assignement :

= (affectation), ***=** (**$x*=$y** équivalent à **$x=$x*$y**), ***/=***, **+=**, **-=**, **%=**

Opérateurs logiques :

and, **&&** (et), **or**, **||** (ou), **xor** (ou exclusif), **!** (non)

Opérateurs de comparaison :

== (égalité), **<** (inférieur strict), **<=** (inférieur large), **>**, **>=**, **!=** (différence)

Variables, types et opérateurs (IV)

Signalons un opérateur très spécial qui équivaut à une structure conditionnelle complexe *if then else* à la différence qu'il renvoie un résultat de valeur pouvant ne pas être un booléen : l'opérateur ternaire.

Syntaxe :

(condition)?(expression1):(expression2);

Si la **condition** est vrai alors évalue et renvoie l'**expression1** sinon évalue et renvoie l'**expression2**.

Exemple :

$nbr = ($toto>10)?($toto):($toto%10);

Dans cet exemple, la variable **$nbr** prend **$toto** pour valeur si **$toto** est strictement supérieur à **10**, sinon vaut le reste de la division entière de **$toto** par **10**.

Constantes

L'utilisateur peut définir des constantes dont la valeur est fixée une fois pour toute. Les constantes ne portent pas le symbole **$** (dollars) en début d'identificateur et ne sont pas modifiables.

define(''var'',valeur) : définit la constante **var** (sans **$**) de valeur **valeur**

Exemple 1 :

```
define(''author'',''Foobar'');
echo author;                    // affiche 'Foobar'
```

Exemple 2 :

```
define(MY_YEAR,1980);
echo MY_YEAR;                   // affiche 1980
```

Contrairement aux variables, les identificateurs de constantes (et aussi ceux de fonction) ne sont pas sensibles à la casse.

Références

On peut à la manière des pointeurs en C faire référence à une variable grâce à l'opérateur **&** (ET commercial).

Exemple 1 :

```
$toto = 100;          // la variable $toto est initialisée à la valeur 100
$foobar = &$toto;     // la variable $foobar fait référence à $toto
$toto++;              // on change la valeur de $toto
echo $foobar;         // qui est répercutée sur $foobar qui vaut alors 101
```

Exemple 2 :

```
function change($var) {
        $var++;       // la fonction incrémente en local l'argument
}
$nbr = 1;             // la variable $nbr est initialisée à 1
change(&$nbr);        // passage de la variable par référence
echo $nbr;            // sa valeur a donc été modifiée
```

Mathématiques (I)

Il existe une miriade de fonctions mathématiques.

abs($x) : valeur absolue
ceil($x) : arrondi supérieur
floor($x) : arrondi inférieur
pow($x,$y) : x exposant y
round($x,$i) : arrondi de x à la ième décimale
max($a, $b, $c …) : retourne l'argument de valeur maximum
pi() : retourne la valeur de Pi

Et aussi : **cos**, **sin**, **tan**, **exp**, **log**, **min**, **pi**, **sqrt**…

Quelques constantes :
M_PI : valeur de pi (3.14159265358979323846)
M_E : valeur de e (2.71828182845904523354)

Mathématiques (II)

Nombres aléatoires :

rand([$x[,$y]) : valeur entière aléatoire entre 0 et RAND_MAX si x et y ne sont pas définis, entre x et RAND_MAX si seul x est défini, entre x et y si ces deux paramètres sont définis.

srand() : initialisation du générateur aléatoire

getrandmax() : retourne la valeur du plus grand entier pouvant être généré

L'algorithme utilisé par la fonction **rand()** - issu de vieilles bibliothèques libcs - est particulièrement lent et possède un comportement pouvant se révéler prévisible. La fonction **mt_rand()** équivalente à **rand()** est plus rapide et plus sûre puisque l'algorithme utilisé se base sur la cryptographie.

En cas d'utilisation de la fonction **mt_rand()**, ne pas oublier d'utiliser les fonctions de la même famille : **mt_rand([$x[,$y]])**, **mt_srand()** et **mt_getrandmax()**.

Mathématiques (III)

Formatage d'un nombre :

number_format ($nbr[,$dec,[$a,$b]]) : retourne une chaîne de caractères représentant le nombre **$nbr** avec **$dec** décimales après formatage. La chaîne **$a** représente le symbole faisant office de virgule et **$b** le séparateur de milliers.

Par défaut, le formatage est anglophone : $a = ''.'' et $b = ''','.

Très utile pour représenter les nombres élevés au format francophone.

Exemples :

```
number_format (1000000.3333);          // affiche 1,000,000
number_format (1000000.3333,2);        // affiche 1,000,000.33
number_format (1000000.3333,2,",","."); // affiche 1.000.000,33
```

Expressions

Presque tout en php est une expression. On les construit de façon très souple. De façon à réduire la taille du code (comme en C). L'inconvénient est l'illisibilité du code ainsi écrit sauf à le commenter suffisamment. Il faut donc les construire avec parcimonie et ne pas hésiter à les fractionner.

Exemple :

echo change($nbr += ($toto>0)?(0):(--$toto));

Attention à utiliser les parenthèses afin de palier aux priorités des opérateurs.

Booléens

Les variables booléennes prennent pour valeurs **TRUE** (vrai) et **FALSE** (faux). Une valeur entière nulle est automatiquement considérée comme **FALSE**. Tout comme une chaîne de caractères vide **‘’’**. Ou encore comme les chaînes **‘’0’’** et **‘0’** castées en l’entier **0** lui même casté en FALSE.

Exemple :

```
if(0) echo 1;       // faux
if("") echo 2;      // faux
if("0") echo 3;     // faux
if("00") echo 4;
if('0') echo 5;     // faux
if('00') echo 6;
if(" ") echo 7;
```

Cet exemple affiche **467**. Donc l’espace ou la chaîne **‘’00’’** ne sont pas considérés castés en **FALSE**.

Chaînes de caractères (I)

Une variable chaîne de caractères n'est pas limitée en nombre de caractères. Elle est toujours délimitée par des simples quotes ou des doubles quotes.

Exemples :

$nom = ‘’Etiévant’’;

$prenom = ‘Hugo’;

Les doubles quotes permettent l'évaluation des variables et caractères spéciaux contenus dans la chaîne (comme en C ou en Shell) alors que les simples ne le permettent pas.

Exemples :

echo ‘’Nom: $nom‘’; // affiche **Nom: Etiévant**

echo ’Nom: $nom‘; // affiche **Nom: $nom**

Quelques caractères spéciaux : **\n** (nouvelle ligne), **\r** (retour à la ligne), **\t** (tabulation horizontale), **** (antislash), **\$** (caractère dollars), **\’’** (double quote).

Exemple : **echo ‘’Hello Word !\n’’;**

Chaînes de caractères (II)

Opérateur de concaténation de chaînes : **.** (point)

Exemple 1 :

```
$foo = ‘’Hello’’;
$bar = ‘’Word’’;
echo $foo.$bar;
```

Exemple 2 :

```
$name = ‘’Henry’’;
$whoiam = $name.‘’IV’’;
```

Exemple 3 :

```
$out = ‘Patati’;
$out .= ‘’ et patata…’’;
```

Chaînes de caractères (III)

Affichage d'une chaîne avec **echo**:

Exemples:

echo 'Hello Word.';

echo ''Hello ${name}\n'';

echo ''Nom : '', $name;

echo(''Bonjour'');

Quelques fonctions:

strlen($str) : retourne le nombre de caractères d'une chaîne

strtolower($str) : conversion en minuscules

strtoupper($str) : conversion en majuscules

trim($str) : suppression des espaces de début et de fin de chaîne

substr($str,$i,$j) : retourne une sous chaîne de **$str** de taille **$j** et débutant à la position **$i**

strnatcmp($str1,$str2) : comparaison de 2 chaînes

addslashes($str) : déspécialise les caractères spéciaux (', '', \)

ord($char) : retourne la valeur ASCII du caractère **$char**

Chaînes de caractères (IV)

On peut délimiter les chaînes de caractères avec la syntaxe *Here-doc.*

Exemple :

```
$essai = <<<EOD
Ma chaîne ''essai''
sur plusieurs lignes.
EOD;
echo $essai;
```

La valeur de la variable **$essai** est délimitée par un identifiant que vous nommez librement. La première apparition de cet identifiant doit être précédée de 3 signes inférieurs **<**. Sa deuxième apparition doit se faire en premier sur une nouvelle ligne. Cette valeur chaîne se comporte comme si elle était délimitée par des doubles quotes **'' ''** dans le sens où les variables seront évaluées. Par contre il est inutile d'échapper les guillemets, c'est-à-dire que la déspécialisation des doubles quotes devient inutile.

Affichage

Les fonctions d'affichage :

echo() : écriture dans le navigateur

print() : écriture dans le navigateur

printf([$format, $arg1, $arg2]) : écriture formatée comme en C, i.e. la chaîne de caractère est constante et contient le format d'affichage des variables passées en argument

Exemples :

```
echo ''Bonjour $name'';
print(''Bonjour $name'');
printf(''Bonjour %s'', $name);
```

Tableaux (I)

Une variable tableau est de type **array**. Un tableau accepte des éléments de tout type. Les éléments d'un tableau peuvent être de types différents et sont séparés d'une virgule.

Un tableau peut être initialisé avec la syntaxe **array**.

Exemple :

```
$tab_colors = array('red', 'yellow', 'blue', 'white');
$tab = array('foobar', 2002, 20.5, $name);
```

Mais il peut aussi être initialisé au fur et à mesure.

Exemples :

```
$prenoms[ ] = ''Clément'';        $villes[0] = ''Paris'';
$prenoms[ ] = ''Justin'';         $villes[1] = ''Londres'';
$prenoms[ ] = ''Tanguy'';         $villes[2] = ''Lisbonne'';
```

L'appel d'un élément du tableau se fait à partir de son indice (dont l'origine est zéro comme en C**)**.

Exemple : **echo $tab[10];** // pour accéder au 11ème élément

Tableaux (II)

Parcours d'un tableau.

$tab = array('Hugo', 'Jean', 'Mario');

Exemple 1 :

```
$i=0;
while($i <= count($tab)) {        // count() retourne le nombre d'éléments
  echo $tab[$i].'\n';
  $i++;
}
```

Exemple 2 :

```
foreach($tab as $elem) {
  echo $elem.''\n'';
}
```

La variable **$elem** prend pour valeurs successives tous les éléments du tableau **$tab**.

Tableaux (III)

Quelques fonctions:

count($tab), **sizeof** : retournent le nombre d'éléments du tableau

in_array($var,$tab) : dit si la valeur de **$var** existe dans le tableau **$tab**

list($var1,$var2…) : transforme une liste de variables en tableau

range($i,$j) : retourne un tableau contenant un intervalle de valeurs

shuffle($tab) : mélange les éléments d'un tableau

sort($tab) : trie alphanumérique les éléments du tableau

rsort($tab) : trie alphanumérique inverse les éléments du tableau

implode($str,$tab), join : retournent une chaîne de caractères contenant les éléments du tableau **$tab** joints par la chaîne de jointure **$str**

explode($delim,$str) : retourne un tableau dont les éléments résultent du hachage de la chaîne **$str** par le délimiteur **$delim**

array_merge($tab1,$tab2,$tab3…) : concatène les tableaux passés en arguments

array_rand($tab) : retourne un élément du tableau au hasard

Tableaux (IV)

Il est possible d'effectuer des opérations complexes sur les tableaux en créant par exemple sa propre fonction de comparaison des éléments et en la passant en paramètre à une fonction de tri de php.

usort($tab, ''fonction'');

Trie les éléments grâce à la fonction **fonction** définie par l'utilisateur qui doit prendre 2 arguments et retourner un entier, qui sera inférieur, égal ou supérieur à zéro suivant que le premier argument est considéré comme plus petit, égal ou plus grand que le second argument. Si les deux arguments sont égaux, leur ordre est indéfini.

Attention, les variables tableaux ne sont pas évaluées lorsqu'elles sont au milieu d'une chaîne ce caractère délimitée par des doubles quottes.

Exemple :

```
echo ''$tab[3]'';          // syntaxe invalide
echo $tab[3];              // syntaxe valide
```

Tableaux associatifs (I)

Un tableau associatif est appelé aussi *dictionnaire* ou *hashtable*. On associe à chacun de ses éléments une clé dont la valeur est de type chaîne de caractères.

L'initialisation d'un tableau associatif est similaire à celle d'un tableau normal.

Exemple 1 :

```
$personne = array(''Nom'' => ''César'', ''Prénom'' => ''Jules'');
```

Exemple 2 :

```
$personne[''Nom''] = ''César'';
$personne[''Prénom''] = ''Jules'';
```

Ici à la clé **''Nom''** est associée la valeur **''César''**.

Tableaux associatifs (II)

Parcours d'un tableau associatif.

```
$personne = array(''Nom'' => ''César'', ''Prénom'' => ''Jules'');
```

Exemple 1 :

```
foreach($personne as $elem) {
  echo $elem;
}
```

Ici on accède directement aux éléments du tableau sans passer par les clés.

Exemple 2 :

```
foreach($personne as $key => $elem) {
  echo ''$key : $elem'';
}
```

Ici on accède simultanément aux clés et aux éléments.

Tableaux associatifs (III)

Quelques fonctions :

array_count_values($tab) : retourne un tableau contenant les valeurs du tableau **$tab** comme clés et leurs fréquence comme valeur (utile pour évaluer les redondances)

array_keys($tab) : retourne un tableau contenant les clés du tableau associatif **$tab**

array_values($tab) : retourne un tableau contenant les valeurs du tableau associatif **$tab**

array_search($val,$tab) : retourne la clé associée à la valeur **$val**

L'élément d'un tableau peut être un autre tableau.

Les tableaux associatifs permettent de préserver une structure de données.

Tableaux associatifs (IV)

Quelques fonctions alternatives pour le parcours de tableaux (normaux ou associatifs) :

reset($tab) : place le pointeur sur le premier élément

current($tab) : retourne la valeur de l'élément courant

next($tab) : place le pointeur sur l'élément suivant

prev($tab) : place le pointeur sur l'élément précédant

each($tab) : retourne la paire clé/valeur courante et avance le pointeur

Exemple :

```
$colors = array(''red'', ''green'', ''blue'');
$nbr = count($colors);
reset($colors);
for($i=1; $i<=$nbr; $i++) {
        echo current($colors).''<br />'';
        next($colors);
}
```

Fonctions (I)

Comme tout langage de programmation, php permet l'écriture de fonctions.

Les fonctions peuvent prendre des arguments dont il n'est pas besoin de spécifier le type. Elles peuvent de façon optionnelle retourner une valeur.

L'appel à une fonction peut ne pas respecter son prototypage (nombre de paramètres). Les identificateurs de fonctions sont insensibles à la casse.

Exemple :

```
function mafonction($toto) {
        $toto += 15;
        echo ''Salut !'';
        return ($toto+10);
}
$nbr = MaFonction(15.1);
/* retourne 15.1+15+10=40.1, les majuscules n'ont pas d'importance */
```

Fonctions (II)

On peut modifier la portée des variables locales à une fonction.

global permet de travailler sur une variable de portée globale au programme. Le tableau associatif **$GLOBALS** permet d'accéder aux variables globales du script (**$GLOBALS[''var'']** accède à la variable **$var**).

Exemple :

```
function change() {
        global $var;       // définit $var comme globale
        $GLOBALS[''toto''] ++; // incrémente la variable globale $toto
        $var++;            // cela sera répercuté dans le reste du programme
}
```

static permet de conserver la valeur d'une variable locale à une fonction.

Exemple :

```
function change() {
        static $var;      // définit $var comme statique
        $var++;           // sa valeur sera conservée jusqu'au prochain appel
}
```

Fonctions (III)

On peut donner une valeur par défaut aux arguments lors de la déclaration de la fonction. Ainsi, si un argument est « oublié » lors de l'appel de la fonction, cette dernière lui donnera automatiquement une valeur par défaut décidée à l'avance par le programmeur.

Exemple :

```
function Set_Color($color=''black'') {
        global $car;
        $car[''color''] = $color;
}
```

Forcer le passage de paramètre par référence (équivalent à user de **global**):

Exemple :

```
function change(&$var) { // force le passage systématique par référence
        $var += 100;       // incrémentation de +100
}
$toto = 12;       // $toto vaut 12
change($toto); // passage par valeur mais la fonction la prend en  référence
echo $toto;      // $toto vaut 112
```

Fonctions (IV)

Les paramètres sont passés par copie et les résultats sont retournés par copie (sauf à forcer la référence). Même sans paramètre, un entête de fonction doit porter des parenthèses **( )**. Les différents arguments sont séparés par une virgule **,** . Et le corps de la fonction est délimité par des accolades **{ }**.

Quelques exemples :

```
function afficher($str1, $str2) {     // passage de deux paramètres
        echo ''$str1, $str2'';
}

function bonjour() {                  // passage d'aucun paramètre
        echo ''Bonjour'';
}

function GetColor() {                 // retour d'une variable de type chaîne
        return ''black'';
}
```

Fonctions (V)

Une fonction peut être définie après son appel (en PHP4 du fait de la compilation avant exécution contrairement au PHP3 qui est interprété).

Exemple :

```
function foo() {                    // définition de la fonction foo
        echo ''Foo…'';
}
foo();              // appel de la fonction foo définie plus haut
bar();              // appel de la fonction bar pas encore définie
function bar() {                    // définition de la fonction bar
        echo ''bar!<br />'';
}
```

Cet exemple affichera : **Foo…bar!**.

Fonctions (VI)

Grâce à une petite astuce, il est possible de faire retourner plusieurs valeurs d'une fonction. En retournant un tableau ayant pour éléments les variables à retourner. Dans l'appel de la fonction, il faudra alors utiliser la procédure **list()** qui prend en paramètre les variables à qui ont doit affecter les valeurs retournées. On affecte à **list()** le retour de la fonction.

Exemple :

```
function trigo($nbr) {
    return array(sin($nbr), cos($nbr), tan($nbr));     // retour d'un tableau
}
$r = 12;
list($a, $b, $c) = trigo($r);          /* affectation aux variables $a,$b et $c des éléments du tableau retourné par la fonction trigo */
echo ''sin($r)=$a, cos($r)=$b, tan($r)=$c'';       // affichage des variables
```

Cet exemple affichera ceci :

sin(12)=-0,5365729180, cos(12)=0,8438539587, tan(12)=-0,6358599286

Fonctions dynamiques (I)

Il est possible de créer dynamiquement des fonctions. Pour les déclarer, on affecte à une variable chaîne de caractères le nom de de la fonction à dupliquer. Puis on passe en argument à cette variable les paramètres normaux de la fonction de départ.

Exemple :

```
function divers($toto) {
        echo $toto;
}
$mafonction = ''divers'';
$mafonction(''bonjour !'');        // affiche 'bonjour !' par appel de divers()
```

Fonctions dynamiques (II)

Quelques fonctions permettant de travailler sur des fonctions utilisateur : **call_user_func_array**, **call_user_func**, **create_function**, **func_get_arg,** **func_get_args**, **func_num_args**, **function_exists**, **get_defined_functions**, **register_shutdown_function**.

call_user_func_array($str [, $tab]) : Appelle une fonction utilisateur **$str** avec les paramètres rassemblés dans un tableau **$tab**.

Exemple :

```
function essai($user, $pass) {          // fonction à appeler
        echo ''USER: $user PASSWORD: $pass'';
}
$params = array(''hugo'',''0478'');     // création du tableau de paramètres
call_user_func_array(''essai'', $params);   // appel de la fonction
```

Le nom de la fonction à appeler doit être dans une chaîne de caractères.

Fonctions dynamiques (III)

call_user_func($str [, $param1, $param2, …]) : Appelle une fonction utilisateur éventuellement avec des paramètres.

Exemple :

```
function essai($user, $pass) {
        echo ‘’USER: $user PASSWORD: $pass’’;
}
call_user_func(‘’essai’’, ‘’hugo’’,’’0478’’);
```

Similaire à **call_user_func_array** à la différence qu’ici les arguments à envoyer à la fonction à appeler ne sont pas dans un tableau mais envoyés directement en paramètre à **call_user_func**.

Fonctions dynamiques (IV)

create_function($params,$code) : Crée une fonction anonyme (style lambda). Prend en argument la liste **$params** des arguments de la fonction à créer ainsi que le code **$code** de la fonction sous la forme de chaînes de caractères. Il est conseillé d'utiliser les simples quotes afin de protéger les noms de variables **'$var'**, ou alors d'échapper ces noms de variables **\$var**. Le nom de la fonction créée sera de la forme : *lambda_x* où *x* est l'ordre de création de la fonction.

Exemple :

```
$newfunc = create_function('$a,$b',''return \$a+\$b;'');
echo ''Nouvelle fonction anonyme : $newfunc <br />'';
echo $newfunc(5,12).''<br />'';
```

On fabrique ici une fonction qui prend en argument 2 paramètres et renvoie leur somme. On l'appelle avec les nombres **5** et **12**. On va donc afficher le nombre **17**.

Cet exemple est équivalent à : **function lambda_1($a,$b) { return $a+$b; }**

Fonctions dynamiques (V)

func_num_args() : Retourne le nombre d'arguments passés à la fonction en cours.

func_get_arg($nbr) : Retourne un élément de la liste des arguments envoyés à une fonction. Elle ne doit être utilisée qu'au sein d'une fonction. L'indice **$nbr** commence à zéro et renvoie le **$nbr**-1 ème paramètre.

Exemple :

```
function foobar() {
        $numargs = func_num_args();
        echo ''Nombre d'arguments: $numargs<br />'';
        if ($numargs >= 2) {
                echo ''Le 2ème param : ''.func_get_arg(1).''<br />'';
        }
}
foobar(''foo'', 'bar', 10.5);
```

Cet exemple affiche la chaîne : '**Le 2ème param : 10.5**'

Fonctions dynamiques (VI)

func_get_args() : Retourne les arguments de la fonction en cours sous la forme d'un tableau.

Exemple :

```
function somme() {
        $params = func_get_args();
        $nbr = 0;
        foreach($params as $elem) {
                $nbr += $elem;
        }
        return $nbr;
}
echo somme(50,20,3,98,50);
```

Cette fonction **somme** retourne la somme de tous ses arguments quel qu'en soit le nombre.

Fonctions dynamiques (VII)

function_exists($str) : Retourne TRUE si la fonction **$str** a déjà été définie.

get_defined_functions() : Retourne un tableau associatif contenant la liste de toutes les fonctions définies. A la clé **''internal''** est associé un tableau ayant pour éléments les noms des fonctions internes à PHP. A la clé **''user''**, ce sont les fonctions utilisateurs.

register_shutdown_function($str) : Enregistre la fonction **$str** pour exécution à l'extinction du script.

Classes (I)

Php gère la programmation orientée objet à l'aide de classes.

Exemple :

```
class Voiture {                          // déclaration de la classe
        var $couleur;                    // déclaration d'un attribut
        var $belle = TRUE;               // initialisation d'un attribut
        function voiture() {             // constructeur
                $this->couleur = ''noire'';
        }   // le mot clé $this faisant référence à l'objet est obligatoire
        function Set_Couleur($couleur) {
                $this->couleur = $couleur;
        }
}
$mavoiture = new Voiture();              // création d'une instance
$mavoiture->Set_Couleur(''blanche'');    // appel d'une méthode
$coul = $mavoiture->couleur;             // appel d'un attribut
```

Classes (II)

Le système de classes de php est très succint, il ne gère que l'héritage simple.

Exemple :

```
class Voituredeluxe extends Voiture {   // déclaration de la sous classe
        var $couleur;
        function voituredeluxe() {      // constructeur
                $this->Voiture();
        }
        function Set_Couleur($couleur) {
                $this->couleur = $couleur;
        }
        function Get_Couleur() {
                return $this->couleur;
        }
}
```

La nouvelle classe **Voituredeluxe** hérite de tous les attributs et méthodes de la classe parente **Voiture** dont elle est une extension (**extends**). Il est possible de surcharger les méthodes, d'en déclarer de nouvelles, etc.

Classes (III)

Quelques fonctions :

get_declared_classes() : retourne un tableau listant toutes les classes définies

class_exists($str) : vérifie qu'une classe dont le nom est passé en argument a été définie

get_class($obj), **get_parent_class** : retournent le nom de la classe de l'objet **$obj**

get_class_methods($str) : retourne les noms des méthodes de la classe **$str** dans un tableau

get_class_vars($str) : retourne les valeurs par défaut des attributs de la classe **$str** dans un tableau associatif

get_object_vars($obj) : retourne un tableau associatif des attributs de l'objet **$obj** les clés sont les noms des attributs et les valeurs, celles des attributs si elles existent

is_subclass_of($obj,$str) : détermine si l'objet **$obj** est une instanciation d'une sous-classe de **$str**, retourne VRAI ou FAUX

method_exists($obj,$str) : vérifie que la méthode **$str** existe pour une classe dont **$obj** est une instance, retourne VRAI ou FAUX

Classes (IV)

Tout objet instancié est une variable et peut à se titre être passé en argument à une fonction ou bien être un retour de fonction ou encore être sauvegardée en donnée de session.

Il n'existe pas de destructeur : comme en Java, les objets qui cessent d'être utilisés sont automatiquement détruits.

Il n'y a pas de notion de visibilité : tous les attributs et méthodes sont publiques et une classe hérite forcément de tous les attributs et méthodes de son père.

Une classe fille hérite de tous les attributs et méthodes de la classe parente dont elle est une extension (d'ou la syntaxe **extends**). Il est possible de surcharger les méthodes, d'en définir de nouvelles…

Structures de contrôle (I)

Structures conditionnelles (même syntaxe qu'en langage C) :

```
if( ... ) {
        ...
} elseif {
        ...
} else {
        ...
}

switch( ... ) {
        case ... : { ... } break
        ...
        default : { ... }
}
```

Structures de contrôle (II)

Structures de boucle (même syntaxe qu'en langage C) :

```
for( ... ; ... ; ... ) {
        ...
}

while( ... ) {
        ...
}

do {
        ...
} while( ... );
```

Structures de contrôle (III)

L'instruction **break** permet de quitter prématurément une boucle.

Exemple :

```
while($nbr = $tab[$i++]) {
        echo $nbr.''<br />'';
        if($nbr == $stop)
                break;
}
```

L'instruction **continue** permet d'éluder les instructions suivantes de l'itération courante de la boucle pour passer à la suivante.

Exemple :

```
for($i=1; $i<=10; $i++) {
        if($tab[$i] == $val)
                continue;
        echo $tab[$i];
}
```

Inclusions

On peut inclure dans un script php le contenu d'un autre fichier.

require insert dans le code le contenu du fichier spécifié même si ce n'est pas du code php. Est équivalent au préprocesseur *#include* du C.

Exemple :

require(''fichier.php'');

include évalue et insert à chaque appel (même dans une boucle) le contenu du fichier passé en argument.

Exemple :

include(''fichier.php'');

Arrêt prématuré

Pour stopper prématurément un script, il existe deux fonctions.

die arrête un script et affiche un message d'erreur dans le navigateur.

Exemple :

```
if(mysql_query($requette) == false)
        die(''Erreur de base de données à la requête : <br />$requet'');
```

exit l'arrête aussi mais sans afficher de message d'erreur.

Exemple :

```
function foobar() {
        exit();
}
```

Ces fonctions stoppent tout le script, pas seulement le bloc en cours.

Variables d'environnement

Le langage php est doté d'une multitude de variables d'environnement que la fonction **phpinfo()** permet d'afficher (ainsi que bien d'autres informations sur la configuration du serveur Apache). Ces variables permettent au script d'accéder à des informations très utiles voire quelques fois indispensables.

Quelques variables :

$PHP_SELF : nom du script en cours

$HTTP_ACCEPT : liste des types MIME supportés par le client

$HTTP_USER_AGENT : signature du navigateur du client

$REMOTE_ADDR : adresse IP du client

$QUERY_STRING : chaîne au format URL contenant les paramètres passés à la page en cours

$HTTP_REFERER : URL de la source ayant renvoyée le client sur la page en cours (en l'analysant, on peut connaître le moteur de recherche utilisé ainsi que les mots clés entrés par l'internaute, s'il vient effectivement d'un moteur de recherche; permet d'évaluer la qualité du référencement d'un site web)

Constantes du PHP (I)

Le langage php met à disposition du programmeur des constantes propres au script qui ne peuvent pas être redéfinies et qui sont bien utiles pour la gestion des erreurs internes au script.

Les constantes prédéfinies du PHP :

__**FILE**__ : nom du fichier en cours

__**LINE**__ : numéro de ligne en cours

PHP_VERSION : version de PHP

PHP_OS : nom du système d'exploitation qui est utilisé par la machine qui fait tourner le PHP

TRUE : la valeur vraie booléenne

FALSE : la valeur faux booléenne

Exemples :

```
$test = true;
echo __file__, __line__;
```

Constantes du PHP (II)

Les constantes suivantes, lorsqu'elles sont déclarées par le programmeur (en général avant toute les autres instructions du script), permettent de spécifier à l'interpréteur php du serveur quel niveau de rigueur appliquer face aux erreurs lors du déroulement du script.

E_ERROR : dénote une erreur autre qu'une erreur d'analyse (''parse error'') qu'il n'est pas possible de corriger.

E_WARNING : dénote un contexte dans lequel le PHP trouve que quelque chose ne va pas. Mais l'exécution se poursuit tout de même. Ces alertes-là peuvent être récupérées par le script lui-même.

E_PARSE : l'analyseur a rencontré une forme syntaxique invalide dans le script, correction de l'erreur impossible.

E_NOTICE : quelque chose s'est produit, qui peut être ou non une erreur. L'exécution continue. Par exemple, la tentative d'accéder à une variable qui n'est pas encore affectée.

E_ALL : toutes les constantes E_* rassemblées en une seule. Si vous l'utilisez avec **error_reporting()**, toutes les erreurs et les problèmes que PHP rencontrera seront notifiés.

Constantes du PHP (III)

La fonction **error_reporting($nbr)** permet de fixer le niveau de rapport d'erreurs PHP. Par défaut php est assez permissif puisque autorise l'utilisation des variables avant leur création, le cast implicite, l'appel de fonction retournant une valeur sans variable pour la récupérer… Cette fonction permet de forcer une plus grande sévérité tout comme l'option *explicite* du Visual Basic ou le paramètre –*Wall* du compilateur gcc.

Exemples :

```
error_reporting( E_ERROR |
                 E_WARNING |
                 E_PARSE );
error_reporting(E_NOTICE);
```

constante	valeur
1	E_ERROR
2	E_WARNING
4	E_PARSE
8	E_NOTICE
16	E_CORE_ERROR
32	E_CORE_WARNING
64	E_COMPILE_ERROR
128	E_COMPILE_WARNING
256	E_USER_ERROR
512	E_USER_WARNING
1024	E_USER_NOTICE

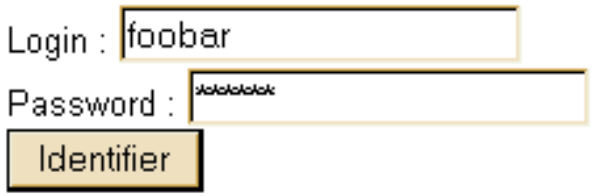
Passage de paramètres à un script (I)

Méthode des formulaires.

La méthode **POST** permet de passer des variables saisies par l'utilisateur d'un script php à l'autre.

Exemple :

```
echo ''<form method=\''post\'' action=\''check.php\''>
Login : <input type=\''text\'' name =\''login\'' value=\''$login\'' /><br />
Password : <input type=\''password\'' name =\''pass\'' value=\''$pass\'' /><br />
<input type=\''submit\'' value=\''Identifier\'' />
</form>'';
```



Cet exemple affiche un formulaire simple dans le navigateur : un champs de saisie de texte et un champ de saisie de mot de passe. Lorsque l'utilisateur valide et envoie les données au serveur, les variables du formulaire seront connues comme variables globales du script php destination (désigné par la valeur de l'attribut **action** de la balise **FORM)**. Les variables porteront le nom des balises (désigné par l'attribut **name** ou **id** des balises de saisie).

Passage de paramètres à un script (II)

Toutes les variables passées en paramètres par cette méthode seront considérées comme des chaînes des caractères. Mais les casts implicites permettront de les récupérer directement dans des variables d'autre type (entier, réel…).

Exemple :

```
if($pass==''xrT12'')
        echo ''Ok, valid user.'';
        /* … + données importantes */
else
        echo ''Acces forbiden.'';
```

Dans cet exemple, on contrôle la validité du mot de passe du formulaire précédent qui a été passé en paramètre au script *check.php* par la méthode **POST**. Par exemple, on affiche des données confidentielles seulement si le mot de passe est le bon.

Les données saisies n'apparaîtront pas dans l'URL et ne seront donc pas stockées dans les fichiers de log du serveur, contrairement à la méthode **GET** (attention, HTTP1.1 implique que les appels de **GET** doivent être idempotents c'est-à-dire doivent toujours retourner la même valeur).

Passage de paramètres à un script (III)

Méthode des ancres.

Les variables peuvent directement être passées par l'intermédiaire des URL.

Exemple :

```
$id = 20;
echo ''<a href=\''fichier.php?action=buy&id=$id\''>Acheter</a>'';
```

Cet exemple va créer dans le script destination les variables globales **$action** et **$id** avec les valeurs respectives **''buy''** et **''20''**.

La barre d'adresse affichera l'URL suivante :

Ainsi une application web écrite en php peut interagir avec l'utilisateur de façon dynamique.

Chargement de fichiers (I)

Les formulaires permettent de transmettre des informations sous forme de chaînes de caractères. Ils peuvent aussi permettre à un internaute de transmettre un fichier vers le serveur.

C'est la balise HTML suivante : **<input type=''file'' />** qui permet le chargement de fichiers. La balise FORM doit nécessairement posséder l'attribut **ENCTYPE** de valeur **''multipart/form-data''**. La méthode utilisée sera **POST**. De plus, il est utile d'imposer au navigateur une taille de fichier limite par le paramètre **MAX_FILE_SIZE** dont la valeur numérique a pour unité l'octet.

Exemple :

```
echo ''<form action=\''$PHP_SELF\'' method=\''POST\''
ENCTYPE=\''multipart/form-data\''>\n
<input type=\''hidden\'' name=\''MAX_FILE_SIZE\'' value=\''1024000\'' />\n
<input type=\''file\'' name=\''mon_fichier\'' /><br />\n
<input type=\''submit\'' value=\''envoyer\'' />\n
</form>\n'';
```

Chargement de fichiers (II)

Pour récupérer le fichier, il faut utiliser la variable d'environnement **$HTTP_POST_FILES** qui est un tableau associatif dont les champs sont les noms des champs HTML **file** du formulaire. Par exemple : **$HTTP_POST_FILES['mon_fichier']** où **mon_fichier** est le nom donné au champs du formulaire HTML de type **file**.

La variable **$HTTP_POST_FILES['mon_fichier']** est elle aussi un tableau associatif possédant les champs suivants :

Champ	Description
name	nom du fichier chez le client
type	type MIME du fichier
size	taille du fichier en octets
tmp_name	nom temporaire du fichier sur le serveur

Si aucun fichier n'a été envoyé par le client, la variable **mon_fichier** vaudra la chaîne de caractères : **''none''** ou bien **''''** (chaîne vide).

La durée de vie du fichier temporaire sur le serveur est limitée au temps d'exécution du script. Pour pouvoir exploiter ultérieurement le fichier sur le serveur, il faut le sauvegarder dans un répertoire et lui donner un vrai nom.

Chargement de fichiers (III)

Exemple du cas du chargement de ce qui doit être une image GIF de moins de 1024.000 octets :

```
// création d'une variable contenant toutes les infos utiles
$file = $HTTP_POST_FILES['mon_fichier'];
// si un fichier a bel et bien été envoyé :
if($file || ($file != ''none'')) {
        // extraction du nom du fichier temporaire sur le serveur :
        $file_tmp = basename($file['tmp_name']);
        // vérification de la taille et du type MIME
        if(($file['size'] <= 1024000) || ereg(''gif$'',$file[type]))
                // nouveau nom, emplacement et extension du fichier :
                $file_def = $dir.'/'.$name.'.'.$ext;
                // copie du fichier temporaire dans son nouvel emplacement :
                copy($file_tmp, $file_def);
        }
}
```

Chargement de fichiers (IV)

Il est important de vérifier avec **is_file()** si un fichier du même nom existe déjà sur le serveur à l'emplacement où on veut copier le nouveau fichier. On pourra supprimer l'ancien fichier avec **unlink()** (qui ne fonctionne pas avec les serveurs fonctionnant sous Windows). **basename()** permet de connaître le nom du fichier à partir de son chemin (+nom) complet.

Même si le paramètre **MAX_FILE_SIZE** est inclus dans le formulaire, il est important de vérifier la taille du fichier réceptionné car rien n'empêche un internaute malveillant de modifier en local sur sa machine le formulaire pour y soustraire le champs caché **MAX_FILE_SIZE** afin de saturer le serveur avec des fichiers trop volumineux.

La vérification du type MIME du fichier est également importante dans le cas où on ne souhaite réceptionner que des types de fichiers bien particuliers (des images GIF, JPEG ou PNG par exemple).

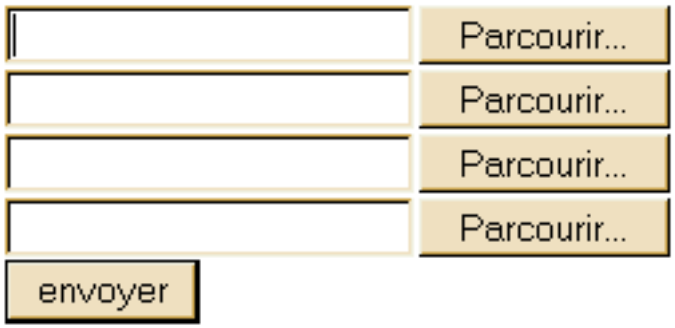
Chargement de fichiers (V)

Pour charger simultanément plusieurs fichiers, il suffit de rajouter des crochets au nom du champ HTML **file**, et de mettre autant de champs **file** que désiré.

Exemple :

```
<input type=''file'' name=''mes_fichiers[]'' />
<input type=''file'' name=''mes_fichiers[]'' />
<input type=''file'' name=''mes_fichiers[]'' />
<input type=''file'' name=''mes_fichiers[]'' />
```

Dans cet exemple, l'internaute pourra charger jusqu'à quatre fichiers.



Chargement de fichiers (VI)

A la réception, la variable **$HTTP_POST_FILES** sera un tableau de tableaux associatifs.

Exemple :

```
$files_tab = $HTTP_POST_FILES['mes_fichiers'];
foreach($files_tab as $file) {
        /* faire le traitement vu précédemment :
        - extraire le nom du fichier temporaire sur le serveur
        - vérifier la taille et le type MIME
        - donner un nouveau nom, emplacement et extension du fichier
        - copier le fichier temporaire dans son nouvel emplacement */
}
```

Les fichiers temporaires sont généralement placés dans le répertoire **/tmp** du serveur. C'est la directive de configuration **upload_tmp_dir** du fichier **php.ini** qui détermine l'emplacement des fichiers chargés par la méthode POST.

Chargement de fichiers (VII)

On aurait pu procéder autrement qu'avec des crochets, en utilisant directement dans le formulaire HTML des champs **file** de noms complètement différents.

Exemple :

```
<input type=''file'' name=''mon_fichier'' />
<input type=''file'' name=''mon_autre_fichier'' />
```

Par contre, à la réception, on ne peut plus utiliser de boucle !

Exemple :

```
$file = $HTTP_POST_FILES['mon_fichier'];
// puis faire le traitement vu précédemment
$file = $HTTP_POST_FILES['mon_autre_fichier'];
// puis refaire encore le même traitement
```

L'approche utilisant des crochets convient au cas où le nombre de fichiers à charger est dynamique (non connu à l'avance, dépend de paramètres divers).

Chargement de fichiers (VIII)

Pour les versions PHP 3 supérieures à la 3.0.16, PHP4 supérieures à la 4.0.2, il existe une autre méthode, plus simple :

Exemple 1 :

```
// vérification que le fichier a bien été envoyé par la méthode POST
if (is_uploaded_file($mon_fichier)) {
        // déplace le fichier dans le répertoire de sauvegarde
        copy($userfile, $dest_dir);
}
```

Exemple 2 :

```
/* déplace directement le fichier dans le répertoire de sauvegarde en faisant les vérifications nécessaires */
move_uploaded_file($mon_fichier, $dest_dir);
```

Sessions (I)

Les sessions sont un moyen de sauvegarder et de modifier des variables tout au cours de la visite d'un internaute sans qu'elles ne soient visibles dans l'URL et quelque soient leurs types (tableau, objet…).

Cette méthode permet de sécuriser un site, d'espionner le visiteur, de sauvegarder son panier (e-commerce), etc.

Les informations de sessions sont conservées en local sur le serveur tandis qu'un identifiant de session est posté sous la forme d'un cookie chez le client (ou via l'URL si le client refuse les cookies).

Quelques fonctions :

session_start() : démarre une session

session_destroy() : détruit les données de session et ferme la session

session_register(''var'') : enregistre la variable **$var** dans la session en cours, attention, ne pas mettre le signe **$** (dollars) devant le nom de variable

session_unregister(''var'') : détruit la variable **$var** de la session en cours

Sessions (II)

Une session doit obligatoirement démarrer avant l'envoi de toute information chez le client : donc pas d'affichage et pas d'envoi de header. On peut par exemple avoir une variable globale contenant le code HTML de la page et l'afficher à la fin du script.

Les variables de session sont accessibles comme des variables globales du script.

Exemple :

```
<?
$out = ''<html><body>'';
session_start();                          // démarrage de la session
if(! isset($client_ip)) {
        $client_ip = $REMOTE_ADDR;
        session_register(''client_ip'');  // enregistrement d'une variable
}
        /* … + code de la page */
$out .= ''</body></html>'';
echo $out;
?>
```

Sessions (III)

Sauvegarder des variables de type objet dans une session est la méthode de sécurisation maximum des données : elles n'apparaîtront pas dans l'URL et ne pourront pas être forcées par un passage manuel d'arguments au script dans la barre d'adresse du navigateur.

Les données de session étant sauvegardées sur le serveur, l'accès aux pages n'est pas ralenti même si des données volumineuses sont stockées.

Une session est automatiquement fermée si aucune requête n'a été envoyée au serveur par le client durant un certain temps (2 heures par défaut dans les fichiers de configuration Apache).

Une session est un moyen simple de suivre un internaute de page en page (sans qu'il s'en rende compte). On peut ainsi sauvegarder son parcours, établir son profil et établir des statistiques précises sur la fréquentation du site, la visibilité de certaines pages, l'efficacité du système de navigation…

Fichiers (I)

La manipulation de fichier se fait grâce à un identifiant de fichier.

Quelques fonctions:

fopen($file [,$mode]) : ouverture du fichier identifié par son nom **$file** et dans un mode **$mode** particulier, retourne un identificateur **$fp** de fichier ou **FALSE** si échec

fopen($fp) : ferme le fichier identifié par le **$fp**

fgets($fp, $length) : lit une ligne de **$length** caractères au maximum

fputs($fp, $str) : écrit la chaîne **$str** dans le fichier identifié par **$fp**

fgetc($fp) : lit un caractère

feof($fp) : teste la fin du fichier

file_exists($file) : indique si le fichier **$file** existe

filesize($file) : retourne la taille du fichier **$file**

filetype($file) : retourne le type du fichier **$file**

unlink($file) : détruit le fichier **$file**

copy($source, $dest) : copie le fichier **$source** vers **$dest**

readfile($file) : affiche le fichier **$file**

rename($old, $new) : renomme le fichier **$old** en **$new**

Fichiers (II)

Exemple typique d'affichage du contenu d'un fichier :

```
<?php
$file = ''fichier.txt'' ;
if( $fd = fopen($file, ''r'')) {                    // ouverture du fichier en lecture
        while ( ! feof($fd) ) {                     // teste la fin de fichier
                $str .= fgets($fd, 1024);
        /* lecture jusqu'à fin de ligne où des 1024 premiers caractères */
        }
        fclose ($fd);                               // fermeture du fichier
        echo $str;                                  // affichage
} else {
        die(''Ouverture du fichier <b>$file</b> impossible.'');
}
?>
```

Fichiers (III)

La fonction **fopen** permet d'ouvrir des fichiers dont le chemin est relatif ou absolu. Elle permet aussi d'ouvrir des ressources avec les protocols HTTP ou FTP. Elle renvoie FALSE si l'ouverture échoue.

Exemples :

$fp = fopen(''../docs/faq.txt'', ''r'');

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

$fp = fopen(''ftp://user:password@cia.gov/'', ''w'');

Les modes d'ouverture :

'**r**' (lecture seule), '**r+**' (lecture et écriture), '**w**' (création et écriture seule), '**w+**' (création et lecture/écriture), '**a**'(création et écriture seule ; place le pointeur de fichier à la fin du fichier), '**a+**' (création et lecture/écriture ; place le pointeur de fichier à la fin du fichier)

Accès aux dossiers (I)

Il est possible de parcourir les répertoires grâce à ces quelques fonctions :

chdir($str) : Change le dossier courant en **$str**. Retourne TRUE si succès, sinon FALSE.

getcwd() : Retourne le nom du dossier courant (en format chaîne de caractères).

opendir($str) : Ouvre le dossier **$str**, et récupère un pointeur **$d** dessus si succès, FALSE sinon et génère alors une erreur PHP qui peut être échappée avec @.

closedir($d) : Ferme le pointeur de dossier **$d**.

readdir($d) : Lit une entrée du dossier identifié par **$d**. C'est-à-dire retourne un nom de fichier de la liste des fichiers du dossier pointé. Les fichiers ne sont pas triés. Ou bien retourne FALSE s'il n'y a plus de fichier.

rewinddir($d) : Retourne à la première entrée du dossier identifié par **$d**.

Accès aux dossiers (II)

Exemple 1:

```
<?php
if ($dir = @opendir('.')) {                    // ouverture du dossier
        while($file = readdir($dir)) {         // lecture d'une entrée
                echo ''$file<br />'';          // affichage du nom de fichier
        }
        closedir($dir);                        // fermeture du dossier
}
?>
```

$dir est un pointeur vers la ressource dossier

$file est une chaîne de caractères qui prend pour valeur chacun des noms de fichiers retournés par **readdir()**

Accès aux dossiers (III)

Il existe un autre moyen d'accéder aux dossiers : l'utilisation de la pseudo-classe **dir**.

En voici les attributs :

handle : valeur du pointeur

path : nom du dossier

En voici les méthodes :

read() : équivalent à **readdir($d)**

close() : équivalent à **closedir($d)**

Constructeur :

dir($str) : retourne un objet **dir** et ouvre le dossier **$str**

Accès aux dossiers (IV)

Exemple 2 :

```
<?php
$d = dir('.');                          // ouverture du dossier courant
echo ''Pointeur: ''.$d->handle.''<br />'';
echo ''Chemin: ''.$d->path.''<br />'';
while($entry = $d->read()) {            // lecture d'une entrée
        echo $entry.''<br />'';
}
$d->close();                            // fermeture du dossier
?>
```

Cet exemple est équivalent au précédent. Ils listent tous les deux les fichiers et sous répertoires du dossier courant.

Dates et heures (I)

Les fonctions de dates et heures sont incontournables sur Internet et sont indispensables pour la conversion en français des dates fournies par la base de données MySQL qui les code au format anglophone (YYYY-DD-MM hh:mm:ss).

Quelques fonctions :

date(''$format'') : retourne une chaîne de caractères contenant la date et/ou l'heure locale au format spécifié

getdate() : retourne un tableau associatif contenant la date et l'heure

checkdate($month, $day, $year) : vérifie la validité d'une date

mktime ($hour, $minute, $second, $month,$day, $year) : retourne le timestamp UNIX correspondant aux arguments fournis c'est-à-dire le nombre de secondes entre le début de l'époque UNIX (1er Janvier 1970) et le temps spécifié

time() : retourne le timestamp UNIX de l'heure locale

Dates et heures (II)

Exemple 1 :

```
echo date("Y-m-d H:i:s");
/* affiche la date au format MySQL : '2002-03-31 22:30:29' */
```

Exemple 2 :

```
if(checkdate(12, 31,2001))
        echo ''La St Sylvestre existe même chez les anglais !!!'';
```

Exemple 3 :

```
$aujourdhui = getdate();
$mois = $aujourdhui['mon'];
$jour = $aujourdhui['mday'];
$annee = $aujourdhui['year'];
echo "$jour/$mois/$annee";        // affiche '31/3/2002'
```

Dates et heures (III)

Les formats pour **date** :

d Jour du mois sur deux chiffres [01..31] **j** Jour du mois sans les zéros initiaux

l Jour de la semaine textuel en version longue et en anglais

D Jour de la semaine textuel en trois lettres et en anglais

w Jour de la semaine numérique [0..6] (0: dimanche)

z Jour de l'année [0..365]

m Mois de l'année sur deux chiffres [01..12] **n** Mois sans les zéros initiaux

F Mois textuel en version longue et en anglais

M Mois textuel en trois lettres

Y Année sur 4 chiffres **y** Année sur 2 chiffres

h Heure au format 12h [01..12] **g** Heure au format 12h sans les zéros initiaux

H Heure au format 24h [00..23] **G** Heure au format 24h sans les zéros initiaux

i Minutes [00..59] **s** Secondes [00.59]

a am ou pm **A** AM ou PM

L Booléen pour savoir si l'année est bisextile (1) ou pas (0)

S Suffixe ordinal anglais d'un nombre (ex: *nd* pour 2)

t Nombre de jour dans le mois donné [28..31]

U Secondes depuis une époque **Z** Décalage horaire en secondes [-43200..43200]

Dates et heures (IV)

Les clés du tableau associatif retourné par **getdate** :

seconds : secondes

minutes : minutes

hours : heures

mday : jour du mois

wday : jour de la semaine, numérique

mon : mois de l'année, numérique

year : année, numérique

yday : jour de l'année, numérique

weekday : jour de la semaine, textuel complet en anglais

month : mois, textuel complet en anglais

Expressions régulières (I)

Les expressions régulières sont un outil puissant pour la recherche de motifs dans une chaîne de caractères.

Fonctions :

ereg($motif, $str) : teste l'existence du motif **$motif** dans la chaîne **$str**

ereg_replace($motif, $newstr, $str) : remplace les occurrences de **$motif** dans **$str** par la chaîne **$newstr**

split($motif, $str) : retourne un tableau des sous-chaînes de **$str** délimitées par les occurrences de **$motif**

Les fonctions **eregi**, **eregi_replace** et **spliti** sont insensibles à la casse (c'est-à-dire ne différencient pas les majuscules et minuscules).

Exemple :

```
if (eregi(''Paris'', $adresse))
        echo ''Vous habitez Paris.'';
```

Expressions régulières (II)

Les motifs peuvent être très complexes et contenir des caractères spéciaux.

Les caractères spéciaux :

[abcdef] : intervalle de caractères, teste si l'un d'eux est présent

[a-f] : plage de caractères : teste la présence de tous les caractères minscules entre **'a'** et **'f'**

[^0-9] : exclusion des caractères de **'0'** à **'9'**

\^ : recherche du caractère **'^'** que l'on déspécialise par l'antislash ****

. : remplace un caractère

? : rend facultatif le caractère qu'il précède

+ : indique que le caractère précédent peut apparaître une ou plusieurs fois

***** : pareil que **+** Mais le caractère précédent peut ne pas apparaître du tout

{i,j} : retrouve une chaîne contenant entre au minimum **i** et au maximum **j** fois le motif qu'il précède

{i,} : idem mais pas de limite maximum

{i} : retrouve une séquence d'exactement **i** fois le motif qu'il précède

^ : le motif suivant doit apparaître en début de chaîne

$: le motif suivant doit apparaître en fin de chaîne

Expressions régulières (III)

Exemples de motifs :

‘’[A-Z]’’ : recherche toutes les majuscules

‘’[a-zA-Z]’’ : recherche toutes les lettres de l’alphabet minuscules ou majuscules

‘’[^aeyuio]’’ : exclu les voyelles

‘’^Le ’’ : toute chaîne commençant par le mot **‘’Le ‘‘** suivi d’un espace

‘’$\.com’’ : toute chaîne se terminant par **’’.com’’** (déspécialise le point)

Exemples :

```
if ( ereg(‘’^.*@wanadoo\.fr’’, $email) ) {
        echo ‘’Vous êtes chez Wanadoo de France Télécom.’’;
}

$email = eregi_replace(‘’@’’, ‘’-nospam@’’, $email);
```

Ce dernier exemple remplace **‘’moi@ici.fr’’** en **‘’moi-nospam@ici.fr’’**.

Expressions régulières (IV)

Il existe des séquences types :

[[:alnum:]] : [A-Za-z0-9] – caractères alphanumériques

[[:alpha:]] : [A-Za-z] – caractères alphabétiques

[[:digit:]] : [0-9] – caractères numériques

[[:blank:]] : espaces ou tabulation

[[:xdigit:]] : [0-9a-fA-F] – caractères hexadécimaux

[[:graph:]] : caractères affichables et imprimables

[[:lower:]] : [a-z] – caractères minuscules

[[:upper:]] : [A-Z] – caractères majuscules

[[:punct:]] : caractères de ponctuation

[[:space:]] : tout type d'espace

[[:cntrl:]] : caractères d'échappement

[[:print:]] : caractères imprimables sauf ceux de contrôle

Entêtes HTTP (I)

Il est possible d'envoyer des entêtes particuliers du protocole HTTP grâce à la commande **header**.

Syntaxe : **header($str);**

Exemples :

```
header(''Content-type: image/gif'');   // spécifie le type d'image gif
header(''Location: ailleurs.php'');    // redirection vers une autre page
header(''Last-Modified: ''.date(''D, d M Y H:i:s'').'' GMT'');
```

Les entêtes doivent obligatoirement être envoyées avant l'affichage de tout caractère dans la page en cours. Car l'affichage force l'envoi des entêtes de base.

headers_sent() : Retourne TRUE si les entêtes ont déjà été envoyées, FALSE sinon.

Entêtes HTTP (II)

Le rôle des entêtes est d'échanger des méta informations entre serveur et client à propos du document, de la connexion, etc.

Voici quelques entêtes HTTP :

```
HTTP/1.0 404 Not Found
HTTP/1.0 301 Moved Permanently
Date: Sun, 07 Apr 2002 14:39:29 GMT
Server: Apache/1.3.9 (Unix) Debian/GNU
Last-Modified: Sun, 07 Apr 2002 14:39:29 GMT
Connection: keep-Alive
Keep-Alive: timeout=15, max=100
Content-type: text/html
Content-length: 1078
Transfert-Encoding: chunked
Pragma: no-cache
WWW-Authenticate: Basic realm="Domaine sécurisé"
Location: home.html
```

Entêtes HTTP (III)

Exemple pratique 1 :

```
<?php
header(''Location: home2.php'');
exit();
?>
```

Ce script effectue une redirection vers une autre page. C'est-à-dire que le navigateur du client en voyant cet entête *Location* va charger directement la page indiquée sans regarder la suite du script. La fonction **exit** est là pour parer au cas impossible où le script continuerait son exécution.

Note: en règle générale, le format d'un entête est le suivant

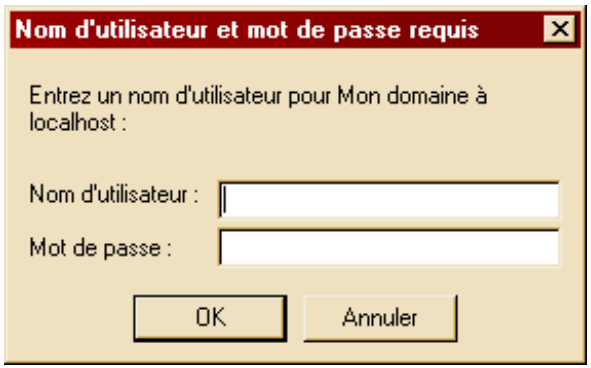
Champ: valeur

Avec un espace entre les deux points '**:**' et la '**valeur**'.

Entêtes HTTP (IV)

Exemple pratique 2 :

```
<?php
if(!isset($PHP_AUTH_USER)) {
    header(''WWW-Authenticate: Basic realm=\''Mon domaine\''');
    header(''HTTP/1.0 401 Unauthorized'');
    echo ''Echec de l'identification.'';
    exit();
} else {
    echo ''Bonjour $PHP_AUTH_USER.<br />'';
    echo ''Votre mot de passe : $PHP_AUTH_PW.'';
}
?>
```



Cet exemple fait intervenir les variables d'environnement **$PHP_AUTH_USER** et **$PHP_AUTH_PW** qui contiennent le login et mot de passe d'un utilisateur préalablement identifié. Si l'utilisateur n'est pas identifié, alors on demande au navigateur d'afficher une boite de saisie de mot de passe et on spécifie que l'accès est interdit. Une fois que l'utilisateur est identifié on peut contrôler la validité des login et mot de passe et s'ils sont corrects afficher des données sensibles.

Attention, cette méthode ne fonctionne pas lorsque PHP est installé en tant que module d'Apache mais seulement lorsqu'il est compilé en même temps que le serveur Apache.

Mail (I)

La fonction **mail** envoie un message électronique.

Syntaxe :

mail($recipient, $subject, $message[, $headers, $params]);

Exemple :

$message = ‘’Allez sur le site du CyberZoïde Qui Frétille, vous y trouverez un tutoriel complet sur le PHP4 et MySQL.’’;

mail(‘’vous@labas.fr’’, ‘’Aide sur PHP’’, $message);

Les arguments obligatoires sont le destinataire, le sujet du message ainsi que le message proprement dit. Les entêtes et paramètres additionnels sont facultatifs.

Note: cette fonction ne marche que si un programme de messagerie électronique (appelé « mailer ») est préalablement installé sur le serveur.

Mail (II)

Exemple plus complet :

```
<?php
$recipient = ''Tony <tony@labas.com>, '';
$recipient .= ''Peter <peter@pwet.net>'';
$subject = ''Notre rendez-vous'';
$message = ''Je vous propose le samedi 15 juin \n'';
$message .= ''--\r\n'';                    // Délimiteur de signature
$message .= ''Le CyberZoïde Qui Frétille'';
$headers = ''From: Hugo Etiévant <cyberzoide@multimania.com>\n'';
$headers .= ''Content-Type: text/html; charset=iso-8859-1\n'' ;
$headers .= ''Cc: bruno@ici.fr\n'';
mail($recipient, $subject, $message, $headers);
?>
```

Mail (III)

Quelques entêtes :

From: Hugo Etiévant <theboss@php-help.com>\n
X-Mailer: PHP\n // maileur
X-Priority: 1\n // Message urgent!
X-Files: Truth is out there\n // entête fantaisiste !
Return-Path: <deamon@php-help.com>\n // @ retour pour erreurs
Content-Type: text/html; charset=iso-8859-1\n // Type MIME
Cc: archives@php-help.com\n // Champs CC
Bcc: bill@php.net, tony@phpinfo.net\n // Champs BCC
Reply-To: <hugo@php-help.com> // @ de retour

Format général des entêtes :

Nom-Entete: valeur\n

Evaluation d'une portion de code PHP

La fonction **eval($str)** évalue la chaîne de caractères **$str** comme du code php. Les variables éventuellement définies dans cette chaîne seront connues dans le reste du programme principal.

Grâce à cette fonction, on peut conserver dans une base de données, des portions de code utilisables ultérieurement.

Le code de la chaîne **$str** doit respecter les mêmes contraintes que le code normal. Notamment :

- point virgule après chaque instruction
- respect des séquences d'échappement
- etc…

Exemple :

```
$foobar = ''Hello Word'';
eval('echo $foobar;');      // affiche 'Hello Word'
```

Colorisation syntaxique (I)

PHP dispose de fonctions qui permettent d'afficher le code source de scripts php et d'en coloriser la syntaxe.

Il n'est pas recommandé d'utiliser les fonctions suivantes afin que vos visiteurs ne connaissent pas votre code source et ne puissent ainsi pas exploiter des vulnérabilité de votre application web.

highlight_file($file), **show_source** : Colorisation de la syntaxe d'un fichier. Affiche la syntaxe colorisée du fichier **$file**, en utilisant les couleurs définies dans le moteur interne de PHP.

highlight_string($str) : Colorisation d'une chaîne de caractères contenant du code php.

Colorisation syntaxique (II)

Exemple :

<?php highlight_file(''sondage.php''); ?>

Résultat affiché :

```php
<?php
$out = "<html><body>";
ConnexionSQL();
echo date("D, d M Y H:i:s");
if($action == "add"){
   AddNew();
}elseif($action == "stats"){
   ShowStats();
}
/* ShowSubmitForm(); */
ShowStats();
$out .= "</body></html>";
echo $out;
?>
```

Colorisation syntaxique (III)

La configuration de la colorisation se trouve dans le fichier *php.ini* :

```
; Colors for Syntax Highlighting mode.
; Anything that's acceptable in <font color=???> would work.
highlight.string        =       #0000FF
highlight.comment       =       #00FF00
highlight.keyword       =       #777700
highlight.bg            =       #FFFFFF
highlight.default       =       #0000BB
highlight.html          =       #000000
```

Et voici comment la commande **phpinfo()** affiche ces informations :

Configuration

PHP Core

Directive	Local Value	Master Value	
highlight.bg	#FFFFFF	#FFFFFF	Couleur de fond
highlight.comment	#00FF00	#00FF00	Couleur des commentaires
highlight.default	#0000BB	#0000BB	Couleur par défaut
highlight.html	#000000	#000000	Couleur des balises HTML
highlight.keyword	#777700	#777700	Couleur des mots réservés
highlight.string	#0000FF	#0000FF	Couleur des chaînes

URL (I)

Les URL (*Uniform Ressource Location*) sont les chemins de ressources sur internet.

Exemples d'URL:

`http://www.google.fr/?q=cours+php`

`http://cyberzoide.developpez.com/php/php4_mysql.ppt`

`ftp://foo:0478@ftp.download.net`

Leur format spécifique leur interdit de comporter n'importe quel caractère (comme l'espace par exemple).

Une URL est une chaîne de caractères composée uniquemment de caractères alphanumériques incluant des lettres, des chiffres et les caractères : **-** (tirêt), _ (souligné), **.** (point).

Tous les autres caractères doivent êtres codés. On utilise le code suivant : **%xx**. Où **%** introduit le code qui le suit et **xx** est le numéro hexadécimal du caractère codé.

URL (II)

Le passage de valeur d'un script à l'autre se fait soit par les sessions, soit par les formulaires ou encore par l'URL.

Exemple par l'URL :

Version imprimable

Dans cet exemple on transmet deux variables au script **index.php** : **$imprim** de valeur **''yes''** et **$user_id** de valeur **''75''**. Les valeurs sont des chaînes de caractères qui pouront être castées implicitement en entier.

Le caractère **?** Indique que la suite de l'URL sont des paramètres et ne font pas partie du nom de fichier. Le caractère **=** sépare un nom de paramètre et sa valeur transmise. Le caractère **&** séparer deux paramètres.

Pour faire face au cas général d'un paramètre dont la valeur contient des caractères interdits, on utilise les fonction de codage.

URL (III)

Quelques fonctions de codage sur l'URL :

Codage de base :

urlencode : Encode une chaîne en URL.

urldecode : Décode une chaîne encodée URL.

Codage complet :

rawurlencode : Encode une chaîne en URL, selon la RFC1738.

rawurldecode : Décode une chaîne URL, selon la RFC1738.

Codage plus évolué :

base64_encode : Encode une chaîne en MIME base64.

base64_decode : Décode une chaîne en MIME base64

URL (IV)

urlencode($str) : code la chaîne **$str**. Les espaces sont remplacés par des signes plus (+). Ce codage est celui qui est utilisé pour poster des informations dans les formulaires HTML. Le type MIME utilisé est *application/x-www-form-urlencoded*.

Exemple 1 :

```
echo <a href=\''$PHP_SELF?foo='.urlencode($foo).''\''>Foo</a>'';
```

rawurlencode($str) : code la chaîne **$str**. Remplace tous les caractères interdits par leur codage équivalent hexadécimal.

Exemple 2 :

```
echo <a href=\''$PHP_SELF?foo='.rawurlencode($foo).''\''>Foo</a>'';
```

Pour être accessible, la valeur du paramètre devra par la suite être décodée dans le script d'arrivé par la fonction réciproque adéquate.

URL (V)

base64_encode($str) : code la chaîne **$str** en base 64. Cet encodage permet à des informations binaires d'être manipulées par les systèmes qui ne gèrent pas correctement les codes 8 bits (code ASCII 7 bit étendu aux accents européens), comme par exemple, les corps de mail qui en général sont américains et ne gère que les 7 bits. Une chaîne encodée en base 64 a une taille d'environ 33% supérieure à celle des données initiales.

Exemple 3 :

```
echo <a href=\''$PHP_SELF?foo='.base64_encode($foo).''\''>Foo</a>'';
```

Comparatif des trois encodages :

Sans codage : **René & Cie : 30%-5*20**

urlencode : **Ren%E9+%26+Cie+%3A+30%25-5%2A20**

rawurlencode : **Ren%E9%20%26%20Cie%20%3A%2030%25-5%2A20**

base64_encode : **UmVu6SAmIENpZSA6IDMwJS01KjIw**

URL (VI)

parse_url($str) : retourne un tableau associatif contenant les différents éléments de l'URL passée en paramètre. Les champs sont les suivants : ''scheme'' (protocol), ''host'' (domaine), ''port'' (n° de port), ''user'' (nom d'utilisateur ftp), ''pass'' (mot de passe ftp), ''path'' (chemin de la ressource), ''query'' (paramètres et valeurs), et ''fragment''.

Exemple :

$tab = parse_url(''http://www.cia.gov:8080/form.php?var=val'');

Cet exemple retourne le tableau suivant :

Champ	Valeur
scheme	http
host	www.cia.gov
port	8080
path	form.php
query	var=val

URL (VII)

parse_str($str) : analyse la chaîne **$str** comme si c'était une URL et en extrait les variables et valeurs respectives qui seront alors connues dans la suite du script.

Cette fonction évite d'avoir à créer ses propres fonctions d'analyse de champs de base de données où l'on aurait sauvegardé une url.

Exemple :

```
$str = ''nom=jean+pierre&email[]=moi@ici.fr&email[]=moi@labas.com'';
parse_str($str);
echo $nom, $email[0], $email[1];
```

Cryptage et autres réjouissances

crypt($str [, $salt]) : Retourne une chaîne de caractères. Crypte la chaîne de caractères **$str**. La chaîne optionnelle **$salt** sert de base au cryptage. Cet argument optionnel est appelé *« grain de sel »* à l'image des germes de nucléation à l'origine des dendrites. L'algorythme de cryptage utilisé par PHP n'est a priori pas défini (il peut varier d'un système à un autre), mais c'est en général le DES standard ou bien encore MD5. On sait aussi que le système utilisé – quel qu'il soit – est injectif, c'est-à-dire qu'il n'existe pas d'algorithme symétrique pour décrypter la chaîne codée résultante (du moins officiellement, il est toujours possible que la NSA ait pu le faire et aurait alors tout intérêt à garder cela secret).

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

crc32($str) : Retourne la somme de vérification de redondance cyclique (32-bit) de la chaîne **$str**. Cette valeur sert généralement à vérifier l'intégrité de données transmises.

uniqid($str [, $lcg]) : Retourne un identifiant en chaîne de caractères préfixé unique, basé sur l'heure courante, en micro-secondes. Si le paramètre optionnel booléen **$lcg** est vrai, **uniqid()** ajoutera une entropie *« combined LCG »* à la fin de la valeur retournée, ce qui renforcera encore l'unicité de l'identifiant.

```
$x = md5 (uniqid (rand()));          // $x est une chaîne difficile à prédire !!!
```

Fonctions de hachage (I)

Les fonctions de hachage s'appliquent à des données afin d'en produire un condensat (dit aussi hashcode). Ce condensat est calculé à partir des données d'origine grâce à un algorithme injectif. C'est-à-dire que l'application de cet algorithme produira toujours le même condensat si on utilise les mêmes données de départ. Par contre, deux condensats égaux ne proviennent pas forcément des mêmes données. Les fonctions de hachage sont généralement utilisées pour créer des sommes de vérification ou signer des messages. La génération de condensats joints aux données est une méthode très utilisée pour assurer l'intégrité des données contre le piratage.

Méthodologie : X veux envoyer un message à Y de façon a s'assurer que personne ne pourra altérer le message (le modifier) sans que Y s'en rende compte. X va calculer un condensat C du message et l'envoier à Y avec le message. A réception du message, Y calcul lui aussi un condensat K du message et le comparé au condensat C envoyé par X. Si K est différent de C, alors le message reçu est différent du message envoyé !

mhash($hash,$data) : retourne le condensat (binaire) de la chaîne **$data**, calculé à partir de l'algorithme d'identifiant numérique **$hash**.

Fonctions de hachage (II)

Il existe différents algorithmes de hachage :

Algorithme	Constante (identifiant)	Valeur	Taille du bloc
CRC32	MHASH_CRC32	0	4
MD5	MHASH_MD5	1	16
SHA1	MHASH_ SHA1	2	20
HAVAL256	MHASH_HAVAL256	3	32
RIPEMD160	MHASH_RIPEMD160	5	20
TIGER	MHASH_TIGER	7	24
GOST	MHASH_GOST	8	32
CRC32B	MHASH_CRC32B	9	4
HAVAL224	MHASH_HAVAL224	10	28
HAVAL192	MHASH_HAVAL192	11	24
HAVAL160	MHASH_HAVAL160	12	20

L'identifiant numérique passé en paramètre à **mhash()** est une constante. Les algorithmes agissent sur les données par bloc de x octets. Cette taille x est spécifiée dans le tableau ci-dessus.

Fonctions de hachage (III)

Exemple :

```
$data = ''TOP SECRET : mise en alerte des missiles nucléaires.'';
$condensat = mhash(MHASH_SHA1, $data);
echo bin2hex($condensat);
/* affiche : ''28424f16ae4a53ae865601372a3462a014614c3b''
(la fonction bin2hex() convertit le binaire en hexadécimal) */
```

mhash_get_hash_name($hash) : retourne le nom de l'algorithme dont l'identifiant numérique est passé en paramètre.

mhash_get_block_size() : retourne la taille des blocs utilisés par l'algorithme dont l'identifiant numérique est passé en paramètre.

mhash_count() : retourne le plus grand identifiant d'algorithme de hachage connu par l'interpréteur PHP.

mhash_keygen_s2k($hash, $pass, $salt, $nbr) : retourne une clef de **$nbr** octets à partir du mot de passe **$pass** et du grain de sel **$salt** (chaîne de 8 octets complétée par des zéros s'il le faut) en utilisant l'algorithme d'identifiant **$hash** associé à l'algorithme *Salted S2K* spécifié dans OpenPGP (RFC 2440).

Fonctions de hachage (IV)

Pour connaître les algorithmes disponibles sur votre système, vous pouvez procéder ainsi :

```
$nbr = mhash_count();
echo ''<table>'';
for($i = 0; $i <= $nbr; $i++) {
  if(mhash_get_hash_name($i))
    echo ''<tr><td>$i</td><td>''.mhash_get_hash_name($i).''</td><td>''.
        mhash_get_block_size($i).''</td></tr>'';
}
echo ''</table>'';
```

Divers

defined($str) : Vérifie qu'une constante existe. Renvoie VRAI si la constante dont le nom est passé en paramètre existe ou FAUX sinon.

sleep($nbr) : Retarde l'exécution du script durant **$nbr** secondes. Attention, Apache peut être configurer pour limiter la durée d'exécution des scripts.

usleep($nbr) : Retarde l'exécution du script durant **$nbr** microsecondes.

Requêtes Apache (I)

getallheaders() : renvoie un tableau associatif contant tous les entêtes de la requête en cours.

Exemple :

```
$headers = getallheaders() ;
foreach($headers as $key => $elem) // affichage des entêtes recueillies
        echo ''$key : $elem<br />\n''  ;
```

Cet exemple affiche :

Accept : */*
Accept-Encoding : gzip, deflate
Accept-Language : fr
Connection : Keep-Alive
Host : 127.0.0.1
User-Agent : Mozilla/4.0 (compatible; MSIE 5.5; Windows 98;)

Les fonctions décrites ici de fonctionnent que si PHP est installé en tant que module du serveur HTTP Apache.

Requêtes Apache (II)

apache_lookup_uri($str) : effectue une requête partielle sur l'URI spécifiée en paramètre et retourne un objet contenant toutes les informations importantes la concernant.

Les propriétés de l'objet retourné sont : status (n° erreur HTTP), the_request (requête HTTP complète), status_line, method (méthode HTTP utilisée), content_type (type MIME de la ressource), handler, uri, filename (nom et chemin de la ressource en local sur le serveur), path_info (chemin du répertoire distant), args, boundary, no_cache (vaut '1' si mise en cache interdite), no_local_copy (vaut '1' si copie locale interdite), allowed, send_bodyct, bytes_sent, byterange, clength, unparsed_uri, mtime, request_time (date de la requête au format timestamp UNIX).

Requêtes Apache (III)

Exemple :

```
$obj = apache_lookup_uri('http://127.0.0.1/cyberzoide/essai.php') ;
$tab = get_object_vars($obj) ;
foreach($tab as $key => $elem)
        echo ''$key : $elem <br />\n'' ;
```

Cet exemple affiche :

status : 403
the_request : GET /cyberzoide/essai.php HTTP/1.1
method : GET
uri : /cyberzoide/http://127.0.0.1/cyberzoide/essai.php
filename : d:/internet/cyberzoide/http:
path_info : //127.0.0.1/cyberzoide/essai.php
no_cache : 0
no_local_copy : 1
allowed : 0
sent_bodyct : 0
bytes_sent : 0
byterange : 0
clength : 0
unparsed_uri : /cyberzoide/http://127.0.0.1/cyberzoide/essai.php
request_time : 1034444645

Réseau

checkdnsrr — Résolution DNS d'une adresse IP.

gethostbyaddr — Retourne le nom d'hôte correspondant à une IP.

gethostbyname — Retourne l'adresse IP correspondant à un hôte.

gethostbynamel — Retourne la liste d'IP correspondants à un hôte.

getprotobyname — Retourne le numéro de protocole associé au nom de protocole

getprotobynumber — Retourne le nom de protocole associé au numéro de protocole

getservbyname — Retourne le numéro de port associé à un service Internet, et un protocole.

getservbyport — Retourne le service Internet qui correspond au port et protocole.

ip2long — Convertit une chaîne contenant une adresse (IPv4) IP numérique en adresse littérale.

long2ip — Convertit une adresse IP (IPv4) en adresse IP numérique

Exercice 1 : conversion de date (I)

Etudions l'exemple complet de la conversion d'une date au format anglophone MySQL en format francophone.

Une date MySQL se présente ainsi : **''YYYY-DD-MM hh:mm:ss''**

YYYY : l'année numérique avec 4 chiffres, **DD** : le jour numérique, **MM** : le mois numérique, **hh** : heures, **mm** : minutes, **ss** : secondes

Tous les nombres sur 2 chiffres prennent un zéro devant si nécessaire.

Par exemple : **''2002-20-04 15:08:20''** correspond au 20 avril 2002 à 15h08 et 20 secondes.

Ce format correspond au type **DATETIME**. Soit **lastmodified** un attribut d'une table MySQL.

La requête suivante permet d'extraire la date de dernière modification d'un enregistrement.

```
SELECT lastmodified
FROM citations
WHERE id=''$id'';
```

Exercice 1 : conversion de date (II)

Etape 1 : extraction de la date d'une base de données MySQL

```
$requet = ''SELECT lastmodified FROM citations WHERE id=\''$id\''';
if($result = mysql_query($requet)) {
        if($ligne = mysql_fetch_row($result)) {
                $lastmodifed = $ligne[0];
        } else die(''Erreur base de données'');
} else die(''Erreur base de données'');
```

Etape 2 : séparation de la date et de l'heure

Le seul espace de la chaîne de caractères qui constitue la variable **$lastmodified** est un séparateur entre la date et l'heure. On va donc la scindée en ses deux morceaux grâce à la fonction **explode** qui renvoie les sous chaînes dans un tableau dont est extrait grâce à **list** les deux variables **$date** et **$time**.

```
list($date, $time) = explode('' '', $lastmodified);
```

Exercice 1 : conversion de date (III)

Etape 3 : extraction des jour, mois, année

On procède selon le même schéma que précédemment sauf qu'ici c'est le tiret qui est séparateur dans la date.

list($year, $day, $month) = explode(''-'', $date);

Etape 4 : extraction facultative des heure, minute, seconde

Ici, le séparateur sont les deux points.

list($hour, $min, $sec) = explode('':'', $time);

Etape 5 : affichage au format francophone

echo $lastmodified = ''$day/$month/$year $time'';

Affiche **''20/04/2002 15:08:20''**.

On a donc transcrit en français notre date anglaise.

On peut aller encore plus loin en affichant les mois en toutes lettres et en français.

Exercice 1 : conversion de date (IV)

Etape 6 : affichage en toutes lettres du mois

On crée d'abord le tableau des mois de l'année.

$months = array("janvier", "février", "mars", "avril", "mai", "juin", "juillet", "août", "septembre", "octobre", "novembre", "décembre");

Ensuite, on affiche l'élément du tableau des mois de l'année dont l'indice est égale au numéro du mois. Comme l'indice du tableau commence à zéro, il faut soustraire 1 à **$month**, cela aura aussi pour effet de caster cette chaîne en entier (et de supprimer le zéro éventuel en première position).

echo $lastmodified = ''le $day ''.$months[$month-1].'' $year à ${hour}h${min}m${sec}s'';

Affiche **''le 20 avril 2002 à 15h08m20s''**.

Et voilà !

Exercice 2 : compteur de visites (I)

On souhaite comptabilisé le nombre de chargement d'une page (la page d'accueil par exemple). On va procéder de la façon suivante : le compteur numérique sera stocké dans un fichier, à la première ligne. Ce fichier contiendra seulement un nombre, celui des visites.

Phase 1 : incrémenter la valeur dans le fichier

Ce fichier va s'appeler **compteur.cpt**.

Principe de l'algorithme : si le fichier n'existe pas encore (**file_exists**), alors on le crée et on l'ouvre en écriture (**fopen w**) et on initialise le compteur à zéro en écrivant la chaîne **'0'** en première ligne (**fputs**) et on le referme (**fclose**). Ensuite, ouverture du fichier en lecture plus écriture (**fopen r+**), lecture du nombre (**fgets**), incrémentation d'une unité du nombre (**++**), positionnement du pointeur courant du fichier en première ligne (**fseek 0**) et réécriture du nombre (**fputs**) puis fermeture (**fclose**).

Cet algorithme (écrit dans la diapo suivante) est écrit directement dans le code source de la page d'accueil.

Exercice 2 : compteur de visites (II)

Algorithme :

```
$fichier = "compteur.cpt";          // affectation du nom de fichier
if( ! file_exists($fichier)) {      // test d'existence
        // initialisation du fichier si n'existe pas encore
        $fp = fopen($fichier,"w");// ouverture en écriture
        fputs($fp,"0");             // écriture
        fclose($fp);                // fermeture
}
$fp = fopen($fichier,"r+");         // ouverture
$hits = fgets($fp,10);              // lecture
$hits++;                            // incrémentation
fseek($fp,0);                       // positionnement
fputs($fp,$hits);                   // écriture
fclose($fp);                        // lecture
```

Exercice 2 : compteur de visites (III)

Phase 2 : généralisation aux autres pages

Comme les internautes peuvent atterrir sur des pages internes à votre site sans passer par l'accueil, il peut être intéressant de pouvoir comptabilité des visites des autres pages. Cela permettra aussi de comparer la popularité relative de vos différentes rubriques.

Créons donc une fonction que l'on placera dans un fichier à part par exemple **compteur.php** et que l'on appellera par inclusion comme ceci :

```
<?php include("compteur.php");
Mon_Compteur("ma_page") ?>
```

Remplacez **"ma_page"** par un identifiant unique pour chaque page.

```
<?
function Mon_Compteur($page) {
  $fichier = $page.".cpt";
  if(!file_exists($fichier)) {
          $fp = fopen($fichier,"w");
          fputs($fp,"0");
          fclose($fp);
  }
  $fp = fopen($fichier,"r+");
  $hits = fgets($fp,10);
  $hits++;
  fseek($fp,0);
  fputs($fp,$hits);
  fclose($fp);
}
?>
```

Exercice 2 : compteur de visites (IV)

Phase 3 : protection contre la redondance

Comme un visiteur peut charger plusieurs fois la même page au cours d'une même visite, ce mode de décompte n'est pas assez précis et va surestimé le nombre réel de visiteurs. Il faut donc garder en mémoire le fait qu'un visiteur est déjà passé par la page et incrémenter le compteur seulement si ce n'est pas le cas. On va donc créer une variable de session : un tableau contenant la liste des pages visitées.

Principe du nouvel algorithme : on teste l'existence de la variable de session **$PAGES_LIST**, si elle n'existe pas on la crée, on y ajoute la page en cours et on appelle la fonction **Mon_Compteur**. Si elle existe, on teste la présence de la page en cours dans ce tableau, si elle n'y est pas alors on l'y met et on appelle **Mon_Compteur**.

L'appel est légèrement différent :

```
<?php
$page = "ma_page";                  // définition du nom de la page
include("compteur.php");            // chargement de l'algorithme
?>
```

Exercice 2 : compteur de visites (V)

Voici le code à rajouter dans le fichier **compteur.php** :

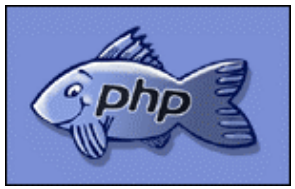
```
session_start();                    // démarrage de la session
if( ! isset($PAGES_LIST)) {  // test de l'existence de la variable de session
        $PAGES_LIST = array($page);      // création de la variable
        session_register($PAGES_LIST); // ajout de la page en cours
        Mon_compteur($page);                // incrémentation du compteur
} else {
        if( ! in_array($page, $PAGES_LIST)) {       // test de redondance
                $PAGES_LIST[ ] = $page;           /* ajout dans la variable
de session pour éviter la redondance */
                Mon_compteur($page);   // incrémentation du compteur
        }
}
```

Le phénomène PHP

Sa gratuité et le libre accès à ses sources en fait un langage très populaire surtout auprès de la communauté GNU Linux.

Sa syntaxe C universellement connue, et sa programmation intuitive en font le langage qui a fait la plus grande percée auprès des webdesigners en 2001.

De nombreux sites lui sont consacrés, on y trouve des scripts, des astuces et même des concours de logos…



Partie 2 : MySQL

Présentation

MySQL est une base de données implémentant le langage de requête SQL un langage relationnel très connu. Cette partie suppose connue les principes des bases de données relationnelles.

Il existe un outil libre et gratuit développé par la communauté des programmeurs libres : phpMyAdmin qui permet l'administration aisée des bases de données MySQL avec php. Il est disponible sur : http://sourceforge.net/projects/phpmyadmin/ et http://www.phpmyadmin.net.

Avec MySQL vous pouvez créer plusieurs bases de données sur un serveur. Une base est composée de tables contenant des enregistrements.

Plus d'informations sont disponibles à http://www.mysql.com/.

La documentation de MySQL est disponibles à http://www.mysql.com/documentation/, ainsi qu'en français chez nexen : http://dev.nexen.net/docs/mysql/.

Connexion (I)

Pour se connecter à une base depuis php, il faut spécifier un nom de serveur, un nom d'utilisateur, un mot de passe et un nom de base.

Les fonctions de connexion :

mysql_connect($server,$user,$password) : permet de se connecter au serveur **$server** en tant qu'utilisateur **$user** avec le mot de passe **$password**, retourne l'identifiant de connexion si succès, FALSE sinon

mysql_select_db($base[,$id]) : permet de choisir la base **$base**, retourne TRUE en cas de succès, sinon FALSE

mysql_close([$id]) : permet de fermer la connexion

mysql_pconnect : idem que **mysql_connect** sauf que la connection est persistante, il n'y a donc pas besoin de rouvrir la connexion à chaque script qui travaille sur la même base.

Les identifiants de connexion ne sont pas nécessaires si on ne se connecte qu'à une seule base à la fois, ils permettent seulement de lever toute ambiguïté en cas de connexions multiples.

Connexion (II)

Exemple 1 :

```
if( $id = mysql_connect(''localhost'',''foobar'',''0478'') ) {
        if( $id_db = mysql_select_db(''gigabase'') ) {
                echo ''Succès de connexion.'';
                /* code du script … */
        } else {
                die(''Echec de connexion à la base.'');
        }
        mysql_close($id);
} else {
        die(''Echec de connexion au serveur de base de données.'');
}
```

Connexion (III)

Exemple 2 :

```
@mysql_connect(''localhost'',''foobar'',''0478'') or die(''Echec de connexion au serveur.'');
@mysql_select_db(''gigabase'') or die(''Echec de sélection de la base.'');
```

Cet exemple est équivalent au précédent mais plus court à écrire. Le symbole @ (arobase) permet d'éviter le renvoie de valeur par la fonction qu'il précède.

On pourra avantageusement intégrer ce code dans un fichier que l'on pourra joindre par **include()**. C'est aussi un moyen de sécuriser le mot de passe de connexion.

Une connexion persistante évite d'avoir à rouvrir une connexion dans chaque script. Les connexions sont automatiquement fermées au bout d'un certain temps en cas d'absence de toute activité…

Interrogation

Pour envoyer une requête à une base de donnée, il existe la fonction : **mysql_query($str)** qui prend pour paramètre une chaîne de caractères qui contient la requête écrite en SQL et retourne un identificateur de résultat ou FALSE si échec.

Les requêtes les plus couramment utilisées sont : **CREATE** (création d'une table), **SELECT** (sélection), **INSERT** (insertion), **UPDATE** (mise à jour des données), **DELETE** (suppression), **ALTER** (modification d'une table), etc.

Exemple :

```
$result = mysql_query(''SELECT address FROM users WHERE name=\''$name\'''');
```

Cet exemple recherche l'adresse de l'utilisateur portant pour nom la valeur de la chaîne **$name**. L'identificateur de résultat **$result** permettra à d'autres fonctions d'extraire ligne par ligne les données retournées par le serveur.

Attention, contrairement à Oracle SQL, les requêtes MySQL ne se terminent pas par un point virgule '**;**' et n'autorisent pas les **SELECT** imbriqués.

Extraction des données (I)

Une fois la requête effectuée et l'identificateur de résultat acquis, il ne reste plus qu'à extraire les données retournées par le serveur.

Sous Oracle, l'affichage des résultats d'une requête se fait ligne par ligne, sous MySQL, c'est pareil. Une boucle permettra de recueillir chacune des lignes à partir de l'identifiant de résultat.

```
SQL > SELECT * FROM users;
 ID   NAME    ADDRESS
----------------------------
  1  Boris    Moscou              ← 1ère ligne
  2  Bill     Washington          ← 2ème ligne
  3  William  London              ← 3è ligne
```

Une ligne contient (sauf cas particulier) plusieurs valeurs correspondants aux différents attributs retournés par la requête. Ainsi, une ligne de résultat pourra être sous la forme d'un tableau, d'un tableau associatif, ou d'un objet.

Extraction des données (II)

mysql_fetch_row($result) : retourne une ligne de résultat sous la forme d'un tableau. Les éléments du tableau étant les valeurs des attributs de la ligne. Retourne FALSE s'il n'y a plus aucune ligne.

Exemple 1 :

```
$requet = ''SELECT * FROM users'';
if($result = mysql_query($requet)) {
        while($ligne = mysql_fetch_row($result)) {
                $id = $ligne[0];
                $name = $ligne[1];
                $address = $ligne[2];
                echo ''$id - $name, $address <br />'';
        }
} else {
        echo ''Erreur de requête de base de données.'';
}
```

Ici, on accède aux valeurs de la ligne par leur indice dans le tableau.

Extraction des données (III)

mysql_fetch_array($result) : retourne un tableau associatif. Les clés étant les noms des attributs et leurs valeurs associées leurs valeurs respectives. Retourne FALSE s'il n'y a plus aucune ligne.

Exemple 2 :

```
$requet = ''SELECT * FROM users'';
if($result = mysql_query($requet)) {
        while($ligne = mysql_fetch_array($result)) {
                $id = $ligne['id'];
                $name = $ligne['name'];
                $address = $ligne['address'];
                echo ''$id - $name, $address <br />'';
        }
} else {
        echo ''Erreur de requête de base de données.'';
}
```

Ici, on accède aux valeurs de la ligne par l'attribut dans le tableau associatif.

Extraction des données (IV)

mysql_fetch_object($result) : retourne un objet. Les attributs de l'objet correspondent à ceux de la ligne de résultat. Et les valeurs des attributs de l'objet correspondent à ceux de la ligne de résultat. Retourne FALSE s'il n'y a plus aucune ligne.

Exemple 3 :

```
$requet = ''SELECT * FROM users'';
if($result = mysql_query($requet)) {
        while($ligne = mysql_fetch_object($result)) {
                $id = $ligne->id;
                $name = $ligne->name;
                $address = $ligne->address;
                echo ''$id - $name, $address <br />'';
        }
} else {
        echo ''Erreur de requête de base de données.'';
}
```

Ici, on accède aux valeurs par leur attribut dans l'objet.

Fonctions additionnelles

Quelques fonctions supplémentaires très utiles :

mysql_free_result($result) : efface de la mémoire du serveur les lignes de résultat de la requête identifiées par **$requet**. Très utile pour améliorer les performances du serveur.

mysql_insert_id([$id]) : retourne l'identifiant d'un attribut clé primaire AUTO_INCREMENT de la dernière insertion.

mysql_num_fields($result) : retourne le nombre d'attributs du résultats.

mysql_num_rows($result) : retourne le nombre de lignes du résultats. Et ainsi permet de remplacer le **while** par un **for**.

Penser à bien tester la valeur de retour des fonctions (**mysql_query** et les autres) afin de détecter toute erreur et d'éviter de polluer votre page avec des *Warnings*.

Partie 3 : Un exemple concret

Présentation

Cet exemple concret d'application dynamique écrite en php et utilisant une base MySQL s'inspire directement de la rubrique *Citations de profs* que j'ai eu à réaliser pour le site web de l'*Association des Miagistes Lyonnais* dont je suis le webmaster (année 2001-2002).

Le site web en question est : http://www.miag-rezo.net

Cette rubrique consiste en un seul script qui propose les services suivants :

1) soumettre une nouvelle citation (laissée à l'approbation préalable d'un administrateur avant affichage publique)

2) lancer une recherche par critères dans la base (parmi les citations validées par l'admin)

3) afficher toutes les citations (validées)

4) afficher des statistiques

Création de la table des citations (I)

Pour stocker les citations, on créé une table citation dans notre base de données MySQL, en voici le schéma :

```
CREATE TABLE ‘citation’ (
        id MEDIUMINT UNSIGNED not null AUTO_INCREMENT,
        body TEXT not null ,
        author VARCHAR (40) not null ,
        sender VARCHAR (40) ,
        accept ENUM ('yes','no') DEFAULT 'no' not null ,
        PRIMARY KEY (id)
)
```

Les simples quotes autour du nom de la table servent à sécuriser ce nom.

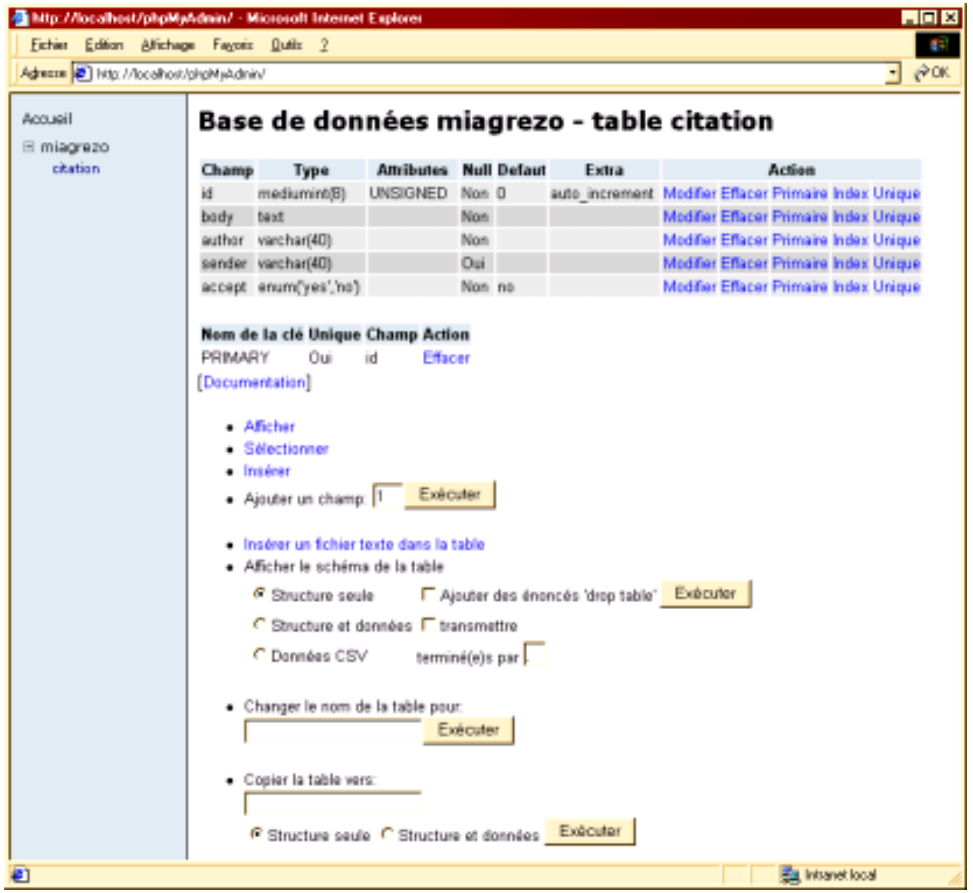
Il est donc créée une table ayant une clé primaire (un entier positif qui s’auto-incrémente lors des insertions), un champ body (le corps de la citation), un champ author (le nom du prof), un champ sender (nom de l’élève qui la propose, facultatif) et un champ accept de validation par un administrateur (on ne va pas laisser afficher n’importe quelle obscénité).

Création de la table des citations (II)

Bien qu'il aurait été tout aussi simple de la créer avec une requête **mysql_query**, cette table a été créée grâce à l'outil phpMyAdmin (qui a été sécurisé avec les fichiers .htaccess et .htpasswd).

Cet utilitaire rend la gestion d'une base de données (surtout la création de tables) très intuitive et sans effort.

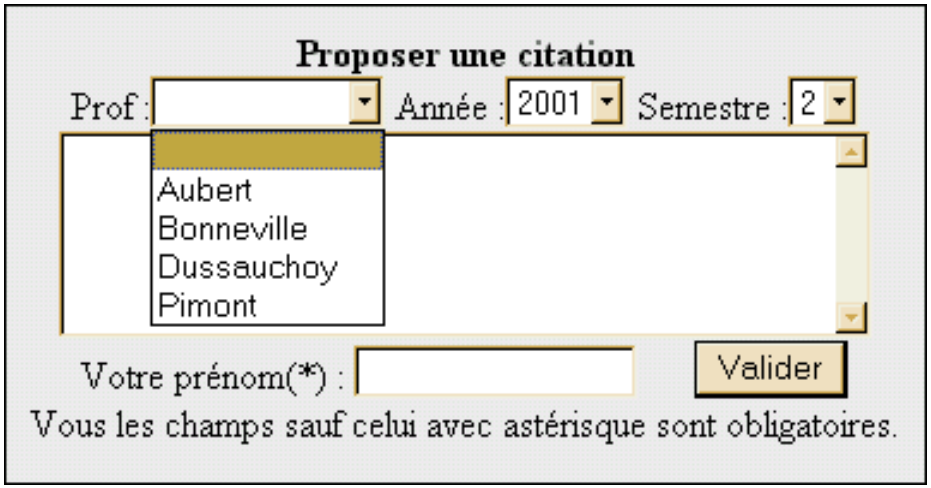
Il permet aussi de créer un fichier de sauvegarde, de modifier les valeurs des enregistrements, etc.



Formulaire de soumission (I)

Il est intéressant de permettre aux élèves de poster aux mêmes des paroles de profs entendues en cours. Pour cela, ils remplissent un formulaire qui contient les champs suivants :

1) Une balise SELECT qui contient la liste de tous les profs de la base
2) Une autre SELECT avec la liste des années
3) Une autre SELECT avec la liste des semestres
4) Un TEXTAREA pour écrire le corps de la citation
5) Un INPUT TEXT pour que l'internaute laisse son pseudonyme
6) Un champs HIDDEN qui spécifie l'action à réaliser (ici : *proposer*)



Formulaire de soumission (II)

En voici la source HTML :

```
<form action=‘’$PHP_SELF" method="post">
<b>Proposer une citation</b><br />
 Prof : <select name="author">
<option value=""> </option>
<option value="Aubert ">Aubert</option>
<option value="Bonneville">Bonneville</option>
</select>                    // même chose pour year et semestre
<textarea name="body" cols="40" rows="5"></textarea><br />
Votre pseudo(*) : <input type="text" name=‘sender" value=""  />
<input type="hidden" name="action" value="proposer’’ />
<input type="submit" value="Valider’’  /><br />
Tous les champs sauf celui avec astérisque sont obligatoires.
</form>
```

Formulaire de soumission (III)

En voici la source php :

```php
function PrintSubForm()
{
  global $out, $author, $body, $year, $semestre, $proposer;
  $out .= "<form action=\"$PHP_SELF\" method=\"post\">";
  $out .= "<b>Proposer une citation</b><br />";
  $out .= " Prof : <select name=\"author\">";
  $requet = "SELECT DISTINCT author FROM citation ORDER BY author ASC";
  $result = mysql_query($requet);
  $out .= "<option value=\"\"> ";
  while($ligne = mysql_fetch_object($result)) {
    if($author == $ligne->author) { $selected = "selected"; } else { $selected=""; }
    $out .= "<option value=\"".$ligne->author."\" $selected>".ligne->author. "</option>";
  }
  $out .= "</select><br />";   // même chose pour year et semestre
  $out .= "<textarea name=\"body\" cols=\"40\" rows=\"5\">$body</textarea><br />";
  $out .= "Votre pseudo(*) : <input type=\"text\" name=\"sender\" value=\"$proposer\" />";
  $out .= "<input type=\"hidden\" name=\"action\" value=\"proposer\" />";
  $out .= "<input type=\"submit\" value=\"Valider\" /><br />";
  $out .= "Tous les champs sauf celui avec astérisque sont obligatoires.";
  $out .= "</form>";
}
```

Contrôle des données soumisses (I)

Lors du démarrage du script, la variable **$action** (qui provient du champ HIDDEN **action**) est testée :

```
switch($action)
{
        case "chercher" : PrintSearchResults(); break;
        case "proposer" : InsertCitation(); break;
        case "stats" : PrintStats(); break;
        default : PrintCitations();
}
```

Si l'action résulte de la validation du formulaire de soumission d'une nouvelle citation (**$action == ''proposer''**), alors on va inserer la nouvelle citation (après vérification des données) par l'appel de la fonction **InsertCitation()**.

Contrôle des données soumisses (II)

Le contrôle des données fournies par le formulaire se fait ainsi :

1) On vérifie que les attributs obligatoires ont bien été saisis.

2) On vérifie que le nom du prof et les valeurs année et semestre sont bien présents dans la base, des fois qu'un petit malin essaierait de pirater notre base en y rajoutant des valeurs incohérentes ou en voulant faire planter notre script !

3) Et si tout se passe bien jusque là, on insert les données par la commande suivante :

```
$requet = "INSERT INTO citation(author,body,year,semestre,sender)
VALUES(\"$author\",\"$body\",\"$year\",\"$semestre\" \"$sender\")";
if(!($result = mysql_query($requet))) {
        $out .= "Erreur de base de données.";
}
```

Affichage

La requête suivante permet d'afficher les citations dans l'ordre des plus récentes, et dans l'ordre alphabétique des nom de prof.

```
SELECT *
FROM citation
WHERE accept="yes"
ORDER BY year DESC, semestre DESC, author ASC, id DESC;
```

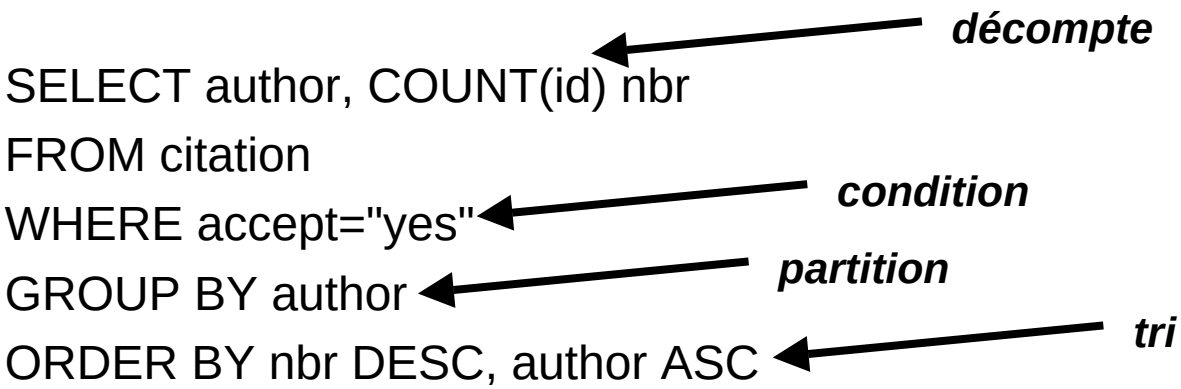
2002 semestre 2 - Aubert
C'est pas XMLment correct.
(proposée par X-Ray)

2002 semestre 2 - Hassas
La dame parle français !
(proposée par Nadine)

2002 semestre 1 - Perret
Vous enculez les mouches.
(proposée par Pweter)

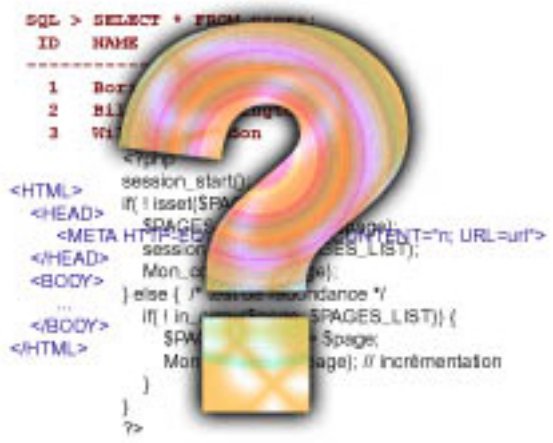
Statistiques

La requête suivante permet de calculer des statistiques sommaires :



Cette requête partitionne la table des citations par prof et compte le nombre de citations par profs en les triant dans l'ordre décroissant des profs les plus productifs. Si deux profs arrivent ex æquo, alors on les affichent dans l'ordre alphabétique. On sélectionne une citation à condition qu'elle ait été validée par un administrateur.

PARTIE 4 : Méthodologie



Méthodologie générale

Ce manière générale, il faut :

1) Programmer orienté objet en créant des objets, comme par exemple un objet *citation* dont les méthodes sont : *ajouter*, *afficher*, *proposer*, *statistiques*… dont peut hériter un autre objet : *citationvalidée*.

2) Toujours contrôler les données saisies par un utilisateur afin d'éviter tout piratage ; ne jamais présupposer l'honnêteté de l'internaute dans un univers aussi impersonnel que l'est Internet.

3) Ne pas répercuter automatiquement les données saisies par un utilisateur auprès des autres internautes sans contrôle ; faire valider les données par un administrateur.

4) Utiliser des variables de session, puisque celles globales peuvent êtres forcées par des passages en URL.

Le respect de ces quelques conseils devrait vous éviter des mauvaises surprises.

Récupération et sauvegarde de données

Pour récupérer les données d'un visiteur, utilisez un formulaire HTML.

Ce formulaire pourra envoyer les données saisies par mail à votre adresse à condition que le client soit équipé d'un logiciel de messagerie et d'un compte mail valide (<FORM action=''mailto:webmaster@monsite.com'' method=''post'' enctype=''text/plain''>). Ou bien envoyez les données saisies à un script PHP qui saura quoi en faire (<FORM action=''monscript.php'' method=''post''>).

Le script de sauvegarde aura plusieurs méthodes au choix :

Méthode 1 : Fichier de sauvegarde (texte ou binaire), pratique pour les compteurs de visites.

Méthode 2 : Base de données (généralement MySQL), souvent plus simple, plus rapide et plus adapté pour les données volumineuses.

Méthode 3 : Variables de session, inconvénient : durée de vie limitée au seul temps de visite de l'internaute.

Méthode 4 : Envoyer des cookies aux clients, qui peut les refuser…

Rafraîchissement de la page

Pour rafraîchir la page en cours (la recharger ou en changer automatiquement) sans intervention de l'internaute, vous disposez d'au moins trois solutions.

Solution 1 :

Utiliser une méta balise dans l'entête de votre page HTML :

*<HEAD><META HTTP-EQUIV="Refrech" CONTENT="**n**; URL=**url**"></HEAD>*

Où **n** est la durée en secondes après laquelle faire la redirection et **url** est l'adresse de la page à charger.

Solution 2 :

Utiliser le JavaScript dans le corps de votre page HTML :

*<SCRIPT language=''JavaScript''> document.location = ''**url**''; </SCRIPT>*

Solution 3 :

Utiliser la fonction **header** avec l'entête **Location** :

*<?php header(''Location: **url**''); ?>*

Historique

- **6 octobre 2002** : nombres aléatoires améliorés, constantes mathématiques, formatage d'un nombre, url parsing, chargement de fichiers, requêtes Apache, algos de hachage (154 diapos)
- **30 août 2002** : partie 4 « Méthodologie », documentation en ligne, quelques corrections (135 diapos)
- **22 avril 2002** : cryptage, exercices : conversion de date et compteur de visites (130 diapos)
- **11 avril 2002** : fonctions dynamiques, accès aux dossiers, entêtes HTTP, URL, Mail, coloration syntaxique, exemple de script, la petite histoire du PHP, évaluation d'une portion de code, quelques développements (chaînes, tableaux, fonctions, classes) (123 diapos)
- **2 avril 2002** : quelques corrections
- **1er avril 2002** : création du document par Hugo Etiévant (83 diapos)

Et remerciements spéciaux à tout ceux qui par leurs commentaires ont contribué à l'amélioration de ce document.

Hugo Etiévant
cyberzoide@yahoo.fr
http://cyberzoide.developpez.com/