

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>

Intégration d'un script dans une page

`<?php ... ?>`

Exemple:

`<html>`

`<body>`

`<?php`

`echo "Bonjour";`

`?>`

`</body>`

`</html>`

Autres syntaxes d'intégration :

▶ `<? ... ?>`

▶ `<script language="php"> ... </script>`

▶ `<% ... %>`

Exemple de script

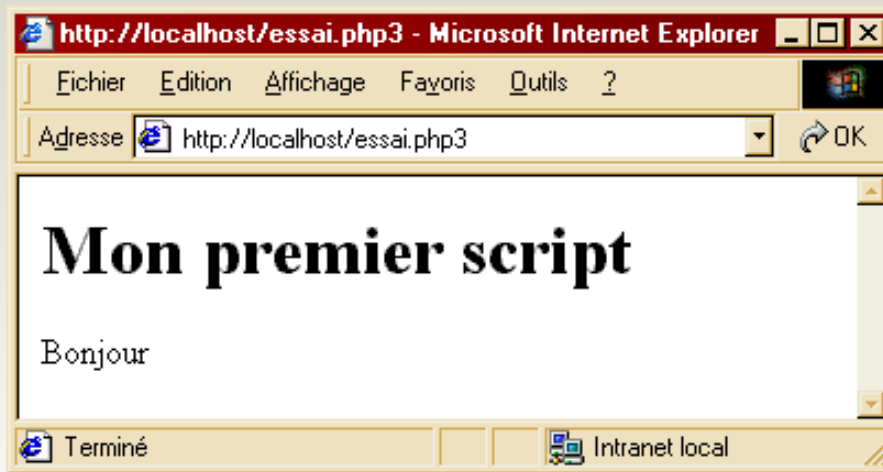
Exemple de script, code source (côté serveur) :

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

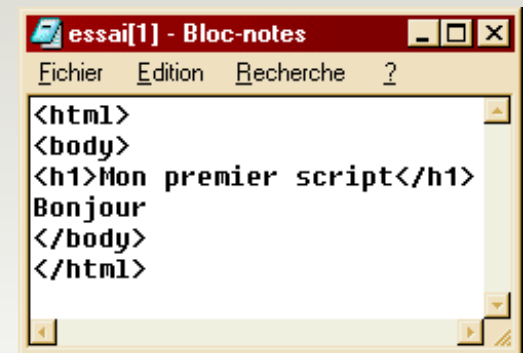
Autre écriture du même script :

```
<?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.php3
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

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 créée. 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 (2.7182818284590452354)

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) : désécialise 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 déspecialisation 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 de caractère délimitée par des doubles quotes.

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+15+10=40, 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+)

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 dynamiques (I)

Il est possible de créer dynamiquement des fonctions. Pour les déclarer, on affecte à une variable chaîne de caractères le nom 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.

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($requete) == 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.

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>";
```



Login : foobar

Password : xoxoxoxox

Identifier

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 forbidden.";
```

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 par les ancres d'url (GET)

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"**.



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

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 uniquement de caractères alphanumériques incluant des lettres, des chiffres et les caractères : - (tirêt), _ (souligné), . (point).

Tous les autres caractères doivent être 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 :

`<a href="index.php?imprim=yes&user_id=75">Version imprimable</a>`

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 pourront 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'arrivée 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];
```

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 toutes 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 :

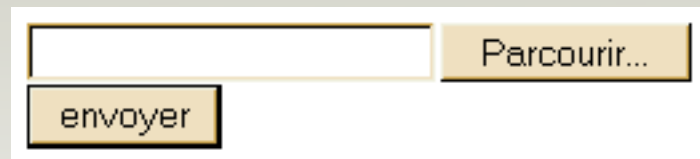
```
error_reporting( E_ERROR |  
                E_WARNING |  
                E_PARSE );  
error_reporting(E_NOTICE);
```

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

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 **\$_FILES** qui est un tableau associatif dont les champs sont les noms des champs HTML **file** du formulaire. Par exemple : **\$_FILES['mon_fichier']** où **mon_fichier** est le nom donné au champs du formulaire HTML de type **file**.

La variable **\$_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 = \$_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 :

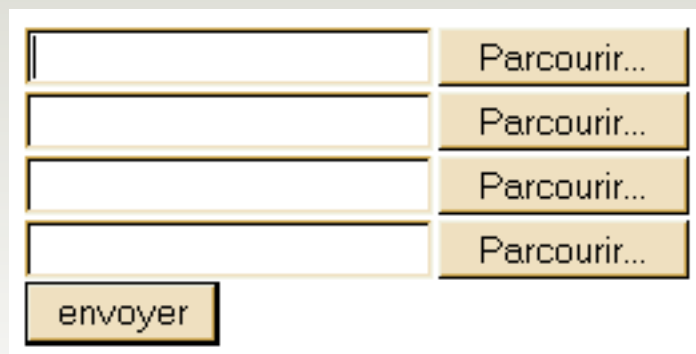
```
<input type="file" name="mes_fichiers[]" />
```

```
<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.



The image shows a web form with four rows, each containing a file input field and a 'Parcourir...' button. At the bottom of the form is an 'envoyer' button.

	Parcourir...
	Parcourir...
	Parcourir...
	Parcourir...
envoyer	

Chargement de fichiers (VI)

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

Exemple :

```
$files_tab = $_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 = $_FILES['mon_fichier'];
```

```
// puis faire le traitement vu précédemment
```

```
$file = $_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 protocoles 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 ($dir = @opendir('.') {
    while($file = readdir($dir)) {
        echo "$file<br />";
    }
    closedir($dir);
}
?>
```

// ouverture du dossier
// lecture d'une entrée
// affichage du nom de fichier
// 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 (ereg("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 minuscules entre 'a' et 'f'

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

\^ : recherche du caractère '^' que l'on déspecialise 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'exactly **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

“**[^aeuyio]**” : exclu les voyelles

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

“**\$\com**” : toute chaîne se terminant par “.com” (déspecialise 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