

Inclusions

Il arrive souvent qu'un ensemble de pages web en PHP, que nous appellerons une application, fasse souvent appel au même code.

Au lieu de copier ce code et de le coller sur toutes les pages, on peut inclure "virtuellement" ce code dans une page avec les commandes **require** et **include**.

require : inclut le contenu d'un autre fichier appelé, et provoque une erreur bloquante s'il est indisponible.

require_once : même chose que **require**, mais ne le fait qu'une seule fois en tout et pour tout dans le même document, si **require** a déjà été appelé auparavant avec le même nom de fichier.

include : inclut le contenu d'un autre fichier appelé, mais ne provoque pas d'erreur bloquante s'il est indisponible

include_once : même chose que **include**, mais ne le fait qu'une seule fois en tout et pour tout dans le même document, si **include** a déjà été appelé auparavant avec le même nom de fichier

Exemples :

```
require("fichier.php");  
include("fichier.html");
```

Inclusions

Dans quel cas de figure peut-on avoir besoin de mutualiser le code entre différents fichiers ?

Pour toutes les parties qui se répètent sur un site :

- En-tête de page
- Pied de page
- Menu de navigation
- Barre contextuelle
- Paramètres communs PHP (par exemple connexion à MySQL, variables de configuration, chaînes de texte...)

Exemple :

```
<?php require 'header.php'; ?>
```

```
<p>Ceci est le formulaire de contact</p>
```

```
<form>
```

```
...
```

```
</form>
```

```
<?php require 'footer.php'; ?>
```

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_USER_AGENT : signature du navigateur du client.

\$DOCUMENT_ROOT : localiser le fichier actuellement consulté.

\$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).

Exemples:

```
echo $_SERVER["PHP_SELF"];  
echo getenv("DOCUMENT_ROOT");
```

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.

__DIR__ : nom du dossier du fichier 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 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 produite, 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.

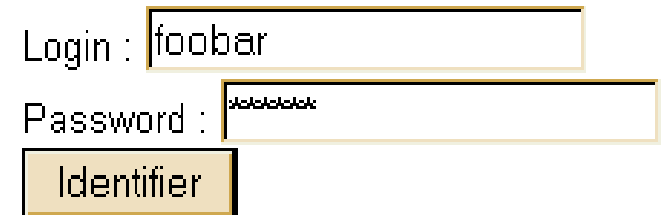
Passage de paramètres à un script (I)

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 :

Password :

Cet exemple affiche un formulaire simple dans le navigateur : un champ 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=="etude2005")
    echo "Mot de passe valide.";
    /* ... + données importantes */
else
    echo "Mot de passe invalide.";
```

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**

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.

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 quels que soient leurs types (tableau, objet...).

Un visiteur se connecte. On demande à créer une session pour lui :
PHP génère un numéro. Pour cela, on utilise la fonction **`session_start()`**.

Ce numéro sert d'identifiant et est appelé "**ID de session**" (ou **PHPSESSID**).
PHP transmet automatiquement cet ID de page en page en utilisant un cookie ou via l'url.

(**ex** : `mapage.php?PHPSESSID=a02bbfffc6198e6e0cc2715047bc3766f`).

A partir de là, on peut créer une infinité de variables de session.

Par exemple:

`$_SESSION['login']` : contient le login du visiteur,

`$_SESSION['password']` : contient le mot de passe, etc...

L'avantage, c'est que vous pourrez récupérer par exemple le login et le mot de passe du visiteur, quelle que soit la page de votre site.

Sessions (II)

La seule chose qu'il ne faut pas oublier de faire, c'est d'appeler **`session_start()`** sur chacune de vos pages AVANT d'écrire le moindre code HTML.

Les variables sessions peuvent être déclarées n'importe où dans le code.

Quelques fonctions :

`session_start()` : démarre une session.

`session_destroy()` : met fin à la session et efface les données.

`session_name()` : retourne le nom de la session en cours.

`session_is_registered("var")` : vérifie si la variable **`$var`** est enregistrée dans la session.

`session_register("var")` : enregistre la variable **`$var`** dans la session courante.

`session_unregister("var")` : supprime la variable **`$var`** de la session courante.

N.B. : Ne pas mettre le signe **`$`** (dollars) devant le nom de variable.

Sessions (III)

```
<?php
session_start(); // On démarre la session AVANT toute chose

$_SESSION['prenom'] = "Jamal";
$_SESSION['age'] = 24;

?>

<html > <body>

  <p>

    Salut <?php echo $_SESSION['prenom']; ?> ! <br>
    Tu es à l'accueil de mon site (index.php). <br>
    Tu veux aller sur une autre page ?

  </p>

  <p>

    <a href="mapage.php"> Lien vers mapage.php </a> <br>
    <a href="monscript.php"> Lien vers monscript.php </a> <br>
    <a href="informations.php"> Lien vers informations.php </a>

  </p>

</body> </html>
```

Sessions (IV)

Notez quelque chose de très important :

Les liens sont tous simples. On ne s'occupe de rien : ni de transmettre le prénom, âge du visiteur, ni de transmettre l'ID de session.

PHP s'occupe automatiquement du transfert des variables entre les pages.

En effet, sur chacune des pages "mapage.php", "monscript.php", "informations.php", vous retrouverez les variables `$_SESSION['prenom']` et `$_SESSION['age']`, vous pourrez les récupérer et manipuler leur contenu.

```
<?
session_start();    // On démarre la session AVANT toute chose
?>
<html >
  <p> Re-bonjour ! </p>
  <p>
    Tu t'appelles <? echo $_SESSION['prenom'] ; ?> ! <br>
    Et ... Tu as <? echo $_SESSION['age']; ?> ans, c'est ça ?
  </p>
</html>
```

Expressions régulières (I)

A quoi ça sert ?

En fait, c'est un système **très puissant** et très rapide pour faire des recherches dans des chaînes de caractères (des phrases par exemple). C'est une sorte de fonctionnalité **Rechercher/Remplacer** très poussée.

Des exemples ?

- Vérifier automatiquement si l'adresse e-mail entrée par le visiteur a une forme valide (comme "**dupont@free.fr**")
 - Modifier une date que vous avez au format américain (**08-05-1985**) pour la mettre dans le bon ordre en français (**05/08/1985**)
 - Remplacer automatiquement toutes les adresses "**http://**" par des liens cliquables, comme ça se fait sur certains forums.
-

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 :

^ : le motif suivant doit apparaître en début de chaîne

\$: le motif précédant doit apparaître en fin de chaîne

[abcdef] : teste si l'un des caractères "a", "b", "c", "d", "e", "f" est présent

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

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

? : rend facultatif le caractère qu'il précède (0 ou une fois)

+ : indique que le caractère précédent est obligatoire et peut apparaître **une ou plusieurs fois**

***** : pareil que **+** Mais le caractère précédent peut apparaître **0 ou plusieurs fois**

{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

() : les opérateurs de quantification concernent tous les éléments entre ().

"un|le": chaîne qui contient "un" ou "le"

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

"[^aeuio]" : 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)

"^abk+" : commence par **ab** et suivi de *un ou plusieurs k*

"^yahoo+\$" : commence et se termine par **yaho** (avec le **o** qui peut se répéter)

"abd?" : contient **ab** suivi de *un ou aucun d*

"abc{2,4}" : contient **ab** suivi de 2, 3 ou 4 **c**

"abc{2,}" : contient **ab** suivi de 2 ou plus **c**

Exemples :

```
if ( ereg("^.@usms.ac.ma", $email) ) {  
    echo "C'est un email institutionnel de l'USMS."  
}
```

```
$email = eregi_replace("@", "21@", $email);
```

Ce dernier exemple remplace **"moi@usms.ac.ma"** en **"moi21@usms.ac.ma"**.

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

Chargement des fichiers joints (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";
```



Parcourir...

envoyer

Chargement des fichiers joints (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 des fichiers joints (III)

Exemple :

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 des fichiers joints (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 des fichiers joints (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.

<input type="text"/>	Parcourir...
<input type="text"/>	Parcourir...
<input type="text"/>	Parcourir...
<input type="text"/>	Parcourir...
<input type="button" value="envoyer"/>	

Chargement des fichiers joints (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 des fichiers joints (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 des fichiers joints (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);
```

Manipulation de Fichiers en PHP (I)

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

Quelques fonctions:

\$fp=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

fclose(\$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**

Manipulation de Fichiers en PHP (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.");
}
?>
```

Manipulation de Fichiers en PHP (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 : écriture seule (création du fichier s'il n'existe pas)

w+ : lecture/écriture (création du fichier s'il n'existe pas)

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 en PHP (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 en PHP (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 en PHP (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 en PHP (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.

Envoi de Mail en PHP (I)

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

Syntaxe :

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

Exemple :

```
$message = "Allez sur estbm.ac.ma , vous y trouverez toutes les actualités.";  
mail("m.outanoute@usms.ma", "Actualités ESTBM", $message);
```

Les arguments obligatoires sont :

- **le destinataire (\$recipient)**,
- **le sujet** du message (**\$subject**)
- **le message** proprement dit (**\$message**).

Les entêtes (\$headers) et **paramètres** additionnels (**\$params**) sont facultatifs.

Envoi de Mail en PHP (II)

Exemple plus complet :

```
<?php
$recipient = "GI2-G1 <gi2.groupe1@estbm.ma>, ";
$recipient .= "GI2-G2 <gi2.groupe2@estbm.ma>, ";
$recipient .= "GI2-G3 <gi2.groupe3@estbm.ma>";
$subject = "Examen PHP";
$message = "Je vous propose le début de mois de janvier pour l'examen\n";
$message .= "--\r\n"; // Délimiteur de signature
$message .= "Professeur de Programmation Web";
$headers = "From: M'hamed OUTANOUTE <m.outanoute@usms.ma>\n";
$headers .= "Cc: responsable.gi@usms.ma\n";
mail($recipient, $subject, $message, $headers);
?>
```

Envoi de Mail en PHP (III)

Quelques entêtes :

From: Webmaster <webmaster@php-help.com>\n	// Expéditeur
X-Mailer: PHP\n	// mailleur
X-Priority: 1\n	// Message urgent!
X-Files: Truth is out there\n	// entête fantaisiste !
Return-Path: <daemon@php-help.com>\n	// adresse réponse si erreurs
Content-Type: text/html; charset=iso-8859-1\n	// Type MIME
Cc: archives@php-help.com\n	// Copie Conforme
Bcc: toto@php.net, tata@phpinfo.net\n	// copie conforme invisible
Reply-To: <papou@php-help.com>	// adresse de réponse

Format général des entêtes :

Nom-Entete: valeur\n

(espace entre : et valeur)

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 :

```
$testvar = "Hello Word";
```

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



Coloration syntaxique en PHP (I)

PHP dispose de fonctions qui permettent d'afficher le code source de scripts PHP et d'en colorer 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 coloriée du fichier **\$file**, en utilisant les couleurs définies dans le moteur interne de PHP.

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

Coloration syntaxique en PHP (II)

Exemple :

```
<?php highlight_file("sondage.php"); ?>
```

Résultat affiché :

```
<?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;
?>
```

Coloration syntaxique en PHP (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 Resource Location*) sont les chemins de ressources sur internet.

Exemples d'URL:

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

http://cyberzoide.multimania.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 :

`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 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 fonctions 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 Foo;

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**" (le fragment dans la page).

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];
```
