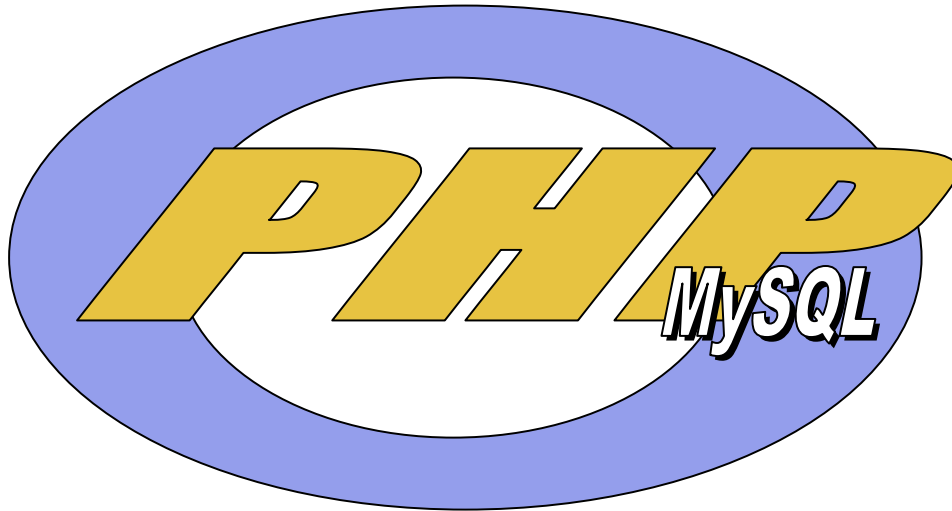


Module 151



- Objectifs :**
- Mettre en œuvre les structures de contrôle PHP.
 - Développer des formulaires dynamiques.
 - Permettre l'administration d'une base de données depuis une page web.
 - Manipuler les fichiers textes et chaînes de caractère.
 - Manipuler les objets Session

Table des matières.

FONCTIONS AVEC PHP 4

1.1 DEFINITIONS	4
1.2 LA DECLARATION D'UNE FONCTION	4
1.3 APPEL DE FONCTION	4
1.4 RENVOI D'UNE VALEUR PAR UNE FONCTION	4
1.5 LES ARGUMENTS D'UNE FONCTION	4
1.6 TRAVAILLER SUR DES VARIABLES DANS LES FONCTIONS	5
1.7 PASSAGE DE PARAMETRE PAR REFERENCE	6
1.8 RETOURNER PLUSIEURS VARIABLES	6
1.9 LA RECURSIVITE	6

PAGES DYNAMIQUES AVEC PHP 7

2.1 REDIRECTION	7
2.2 FORMULAIRES	7
2.3 PASSAGE DE VARIABLES PAR L'URL (QUERYSTRINGS)	8
2.4 ENVOI DE EMAILS	8
2.5 FONCTIONS INCLUDE	8
2.6 LES COOKIES	9

FICHIERS ET STRINGS 11

3.1 LES FONCTIONS DE BASE DES FICHIERS	11
3.1.1 LA FONCTION FOPEN()	11
3.1.2 LA FONCTION FCLOSE() :	11
3.1.3 LA FONCTION FPUTS() :	11
3.1.4 LA FONCTION FGETS() :	11
3.1.5 LA FONCTION FEOF() :	12
3.2 LES TESTS DE FICHIERS	13
3.3 AUTRES FAÇONS DE LIRE ET ECRIRE	13
3.4 TRAITEMENT DE STRINGS	14
3.4.1 CHR()	14
3.4.2 COUNT_CHARS()	14
3.4.3 EXPLODE()	15
3.4.4 IMplode()	15
3.4.5 FONCTION SUBSTR	16
3.4.6 HTMLSPECIALCHARS()	16

BASES DE DONNÉES 17

4.1 PHPMYADMIN	17
4.2 LES FONCTIONS DE BASE	18
5.3 EXPLOITATION D'UNE BASE DE DONNEES	18
5.4 GESTION DES ERREURS DE CONNEXION : 2 METHODES	19
4.4 TRAITEMENT DES RESULTATS	19
4.5 DETECTER UN RESULTAT NUL	20

LES SESSIONS **21**

5.1 INTRODUCTION	21
5.2 SESSION ET LA SECURITE	21
5.3 UTILISATION DE SESSION	22

IMPLEMENTATION D'UN CADDIE **24**

6.1 SCHEMA DU PIPELINE SUIVI PAR L'UTILISATEUR.	24
6.2.- PRESENTATION D'UN EXEMPLE D'ARCHITECTURE	24
6.3.- TECHNIQUES D'IMPLEMENTATION D'UN CADDIE	25
6.3.1.- TECHNIQUE DES COOKIES	25
6.3.2.- TECHNIQUE DES SESSIONS	25
6.4.- EXEMPLE DE CADDIE AVEC DES SESSIONS	26
6.5.- CLASSE POUR UN CADDIE (PANIER VIRTUEL)	28

BIBLIOGRAPHIE **30**

LIVRES	30
SITES	30
AUTRES	

ERREUR ! SIGNET NON DEFINI.

FONTIONS AVEC PHP

1.1 Définitions

Une fonction est un bloc de code autonome et indépendant qui accomplit une tâche spécifique. En général cette tâche peut être répétée plusieurs fois au cours de l'exécution du programme. Lorsqu'une fonction est écrite au sein d'un programme, le code inclus est ignoré jusqu'à ce que celle-ci soit invoquée.

Il y a deux types de fonctions : les fonctions intégrées de PHP et les fonctions définies par l'utilisateur.

Attention	Concernant les fonctions en PHP : <ul style="list-style-type: none">- Pour les fonctions intégrées, il n'est pas nécessaire de la déclarer avant de l'invoquer.- Une fois déclarée, une fonction ne peut être redéfinie, pas de surcharge possible (mécanisme qui permet de créer plusieurs fonctions en leur attribuant le même nom).
------------------	---

1.2 La déclaration d'une fonction

La déclaration de fonction se fait comme suit :

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

1.3 Appel de fonction

Pour appeler, il suffit d'écrire le nom de la fonction suivi de parenthèses avec les arguments s'il y en a.

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

1.4 Renvoi d'une valeur par une fonction

Une fonction peut renvoyer une valeur au code qui l'appelle (une autre fonction ou programme principal). En PHP, comme dans la plupart des langages de programmation (exemple le C++), l'instruction `return` est employée pour renvoyer la valeur d'une variable comme par exemple :

```
return valeur_ou_variable;
```

1.5 Les arguments d'une fonction

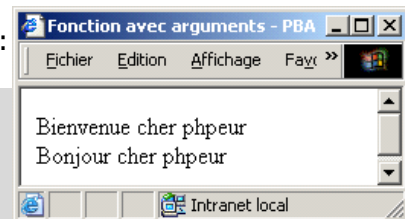
Les arguments peuvent être de simples variables, mais aussi des tableaux ou des objets. À noter qu'il est possible de donner une valeur par défaut à ces arguments en faisant suivre le nom de la variable par le signe "=" puis la valeur que l'on affecte par défaut à la variable.

```
<?
function dire_texte($qui, $texte = 'Bonjour')
{
    if(empty($qui))
    {
        // $qui est vide, on retourne faux
        return false;
    }
    else
    {
        echo "$texte $qui"; // on affiche le texte
        return true; // fonction exécutée avec succès
    }
}
?>
```

Cette fonction peut être appelée de deux façons différentes:

```
<?
// Passage des deux paramètres
dire_texte("cher phpeur<BR>", "Bienvenue");
// affiche "Bienvenue cher phpeur"

// Utilisation de la valeur par défaut du deuxième paramètre
dire_texte("cher phpeur<BR>"); // affiche "Bonjour cher phpeur"
?>
```



1.6 Travailler sur des variables dans les fonctions

Une variable précédée du mot clé **global** sera visible dans l'ensemble du code, c'est-à-dire que sa portée ne sera pas limitée à la fonction seulement. Ainsi, toutes les fonctions pourront utiliser et modifier cette même variable

Le niveau **static** permet de définir une variable locale à la fonction, qui persiste durant tout le temps d'exécution du script

Par défaut, la variable possède le niveau **local**, c'est-à-dire que la variable ne sera modifiée qu'à l'intérieur de la fonction et retrouvera la valeur qu'elle avait juste avant l'appel de fonction à la sortie de celle-ci

```
<?

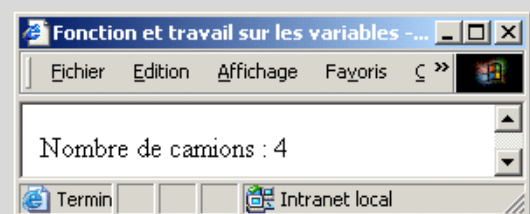
$chaine = "Nombre de camions : ";

function ajoute_camion($mode='')
{
    global $chaine;
    static $nb=0;

    $nb++; // on incrémente le nombre de camions

    if($mode == "affiche")
        echo $chaine.$nb; // on affiche le nombre de camions
}

ajoute_camion(); // nb == 1
ajoute_camion(); // nb == 2
ajoute_camion(); // nb == 3
ajoute_camion("affiche"); // affiche Nombre de camions : 4
```



?>

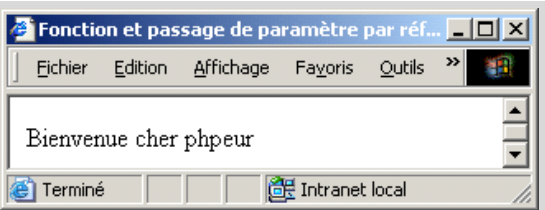
1.7 Passage de paramètre par référence

Une autre méthode pour modifier une variable consiste à la faire précéder du caractère &, précisant qu'il s'agit alors d'un alias: la valeur de la variable est modifiée à la sortie de la fonction. On parle alors de passage par référence. Dans ce cas on passe la référence (adresse mémoire) de la variable à la fonction, ce qui permet de modifier sa valeur.

```

<?
function dire_texte($qui, &$texte)
{
    $texte = "Bienvenue $qui";
}

$chaine = "Bonjour ";
dire_texte("cher phpeur", $chaine);
echo $chaine; // affiche "Bienvenue cher phpeur"
?>
```




1.8 Retourner plusieurs variables

Lorsque vous souhaitez qu'une fonction retourne plusieurs valeurs, le plus simple est d'utiliser un tableau.

```

<?
function nom_fonction()
{
    $variable1 = 10; $variable2 = 5; $variable3 = 2;
    return array( $variable1, $variable2, $variable3 );
    //Retourne les valeurs voulues dans un tableau
}
$retour = nom_fonction();
echo "$retour[0] - $retour[1] - $retour[2]";
?>
```

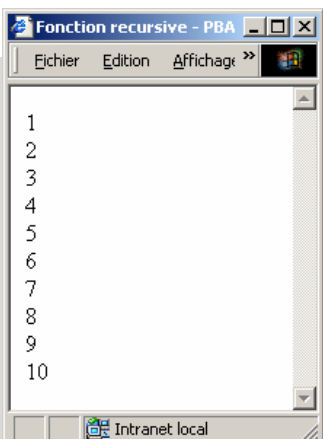


1.9 La récursivité

Les fonctions récursives sont des fonctions qui s'appellent elles-mêmes. Voici un exemple simple.

```

<?
function fonction_recursive($n=0)
{
    $n++;
    echo "$n <br>";
    if($n < 10)
    {
        // si n est inférieur à 10 on continue
        fonction_recursive($n);
    }
}
fonction_recursive(); // affiche les nb de 1 à 10
?>
```



PAGES DYNAMIQUES AVEC PHP

2.1 Redirection

Les fonctions **Header** et **Location** de PHP permettent la redirection vers une autre page. La fonction Header génère l'entête d'une page HTML. Il est important que Header soit la première fonction à afficher quelque chose à l'écran.

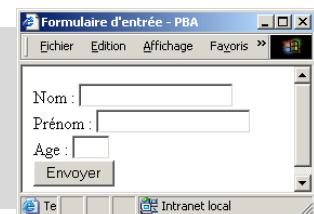
Exemple : `Header("Location: http://www.raclette.com/fromage.html ")`

2.2 Formulaires

Lorsque l'on soumet un formulaire à un fichier Php, toutes les données du formulaire lui sont passées en tant que variables, c'est-à-dire chacun des noms associés aux champs (ou boutons) du formulaire précédés du caractère \$.

Exemple de formulaire HTML:

```
<FORM Method="GET" Action="test.php">
  Nom : <INPUT type="text" size=20 name="nom"><BR>
  Prénom : <INPUT type="text" size=20 name="prenom"><BR>
  Age : <INPUT type="text" size=2 name="age"><BR>
  <INPUT type="submit" value="Envoyer">
</FORM>
```



NB : Afin d'avoir un maximum de compatibilité avec les différents navigateurs, il est fortement conseillé de mettre, dans le code HTML, tous les **attributs des balises** entre **guillemets** (ex. : problèmes sous Firefox pour la méthode POST).

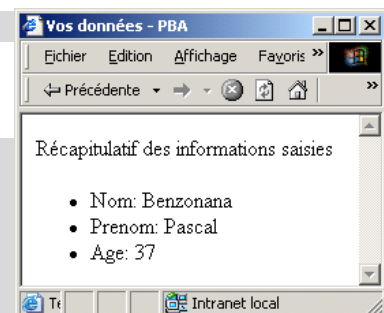
Ainsi, dans cet exemple, le fichier test.php reçoit les variables:

nom, prenom, age

Exemple de test.php :

```
<?php
$Nom=@$_POST['nom'];
$Prenom=@$_POST['prenom'];
$Age=@$_POST['age'];

if (($Nom=="") || ($Prenom=="") || ($Age==""))
{
    if($Nom=="") print("Nom de l'utilisateur :<BR>\n");
    if($Prenom=="") print(" Prénom de l'utilisateur :<BR>\n");
    if($Age=="") print("Veuillez saisir l'âge de l'utilisateur<BR>\n");
}
else
{
    echo "Récapitulatif des informations saisies<BR>\n";
    <UL>
    <LI>Nom: $Nom</LI>
    <LI>Prenom: $Prenom</LI>
    <LI>Age: $Age</LI>
    </UL>;
}
?>
```



2.3 Passage de variables par l'URL (Querystrings)

La variable passée en querystring est automatiquement définie comme variable dans la nouvelle page.

Exemple : `http : // bdo_mag/input.php ?ref=174&nom=eula`

Dans input.php, on va récupérer ces variables ref et eula. On peut afficher les variables de la manière suivante.

```
$Vref=@$_GET['ref'];
$Veula=@$_GET['eula'];
echo $Vref ;
echo $Veula ;
```

2.4 Envoi de mails

L'envoi de messages en php est simple. Il faut définir l'adresse du destinataire, le sujet, le message et (pas obligatoire) l'expéditeur.

```
Mail("x@y.com","sujet ","message", "From : z@epsic.ch ")
```

2.5 Fonctions include

Les deux fonctions include et require permettent d'insérer dans un script le code contenu dans un fichier.

Exemple : `<?php include "phrase.txt";?>`

En mettant cet include dans toutes les pages du site et en mettant par exemple la phrase du jour dans le fichier "phrase.txt", il suffit de modifier le contenu du fichier texte pour que la modification s'effectue sur toutes les pages.

Mais l'inclusion ne se limite pas à du texte, on peut tout aussi bien insérer du code PHP.

```
Exemple : <?php include "phrase.php";?>
          <? FctGenerale(); //appel de la fonction de l'include ?>
```

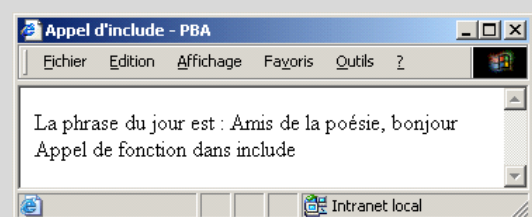
Où le contenu du fichier phrase.php serait :

```
<?php

$phrase="Amis de la poésie, bonjour";
print("La phrase du jour est : ");
print($phrase);

function FctGenerale()
{
    echo "<BR>Appel de fonction dans include";
}

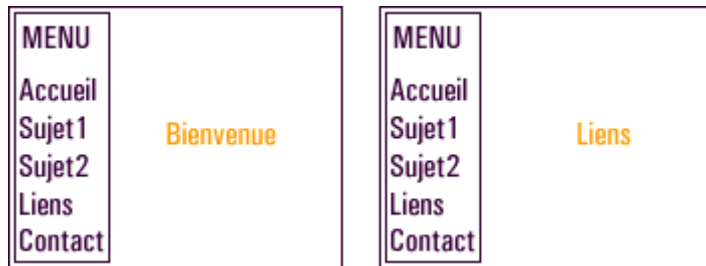
?>
```



L'utilisation la plus courante de ces fonctions est de loin l'inclusion des entêtes, des menus et des bas de page qui sont présents sur toutes les pages du site. Par exemple, si on ne veut pas utiliser de

"frames", qui sont source de conflit avec certains navigateurs, on est obligé de remettre le menu sur chaque page.

Alors qu'avec une inclusion PHP, il suffit de définir le menu dans un fichier et de l'appeler sur chaque page.



Ensuite lors d'une modification du menu, un seul fichier est à modifier pour mettre à jour le site entier. Cette manière de procéder prend toute son importance avec la mise à jour d'un site comportant des dizaines de pages.

Note : Attention cependant à limiter à 2 voir 3 appels d'include, ceci pour des raisons de ressources serveurs.

2.6 Les Cookies

Les cookies permettent de stocker des informations sur le PC du client. Ces informations pourront être récupérées ultérieurement pour identifier la personne et ses habitudes. Les cookies ne contiennent pas de virus ni de données personnelles non entrées par l'utilisateur lui-même par le biais d'un formulaire

Écriture du cookie :

```
setcookie("nom_var","ceci est une donnée",time()+10000);
```

Ceci stocke une variable \$nom_var dans le cookie. Le contenu de la variable est 'ceci est une donnée'. La durée d'existence du cookie dans le cas présent est 10000 sec. Utilisez :

```
$nom_var = @$_COOKIE['nom_var'] ;
```

Attention : Il ne doit **pas** y avoir **de code HTML ou d'instruction d'affichage avant l'écriture du cookie** sinon le parser vous générera une erreur comme montré ci-dessous. En effet, les cookies étant passé en en-tête HTTP vers le serveur, ils ne peuvent être traités qu'avant l'affichage du moindre caractère.

Lecture du cookie :

La variable est automatiquement transmise du client au serveur.

Pour afficher la variable 'nom_var' de l'exemple ci-dessus, tapez simplement :

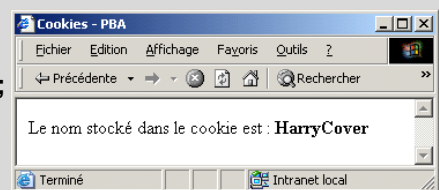
```
$nom_var = @$_COOKIE['nom_var'] ; echo $nom_var;
```

Limitations :

1. Les utilisateurs peuvent désactiver les cookies sur leur navigateur, dans ce cas on ne pourra plus les utiliser.
2. Le **nombre de cookies** autorisé **par site** pour un même client est de **20** maximum.
3. Le **nombre total de cookies** autorisé pour une même machine est de **300** maximum.
4. La **taille maximum d'un cookie** est de **4 kB**.
5. Un cookie n'est accessible que depuis un site, avec le navigateur depuis lequel il a été créé.
6. Les informations contenues dans un cookie peuvent être facilement décodées → évitez de les utiliser pour mémoriser un mot de passe.

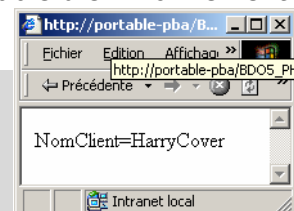
Exemple :

```
<?
if (isset ($Nom))
{
    setcookie("Nom","Harry Cover",time()+10000);
    echo "Vous n'avez pas de cookie stocké";
}
else
{
    echo "Le nom stocké dans le cookie est : <B>";
    echo @$_COOKIE['Nom']."</B>";
}
?>
```



Pour pouvoir voir qu'est-ce qu'il y a dans le cookie, on utilise la variable d'environnement HTTP_COOKIE

```
<?
echo getenv("HTTP_COOKIE");
?>
```



Remarque : Certaines compagnies de marketing utilisent " intensément " les cookies. Elles sont, grâce à des **bannières installées sur plusieurs sites**, capables de déterminer les **habitudes de navigation** des internautes.

Un exemple est la compagnie **DoubleClick**. Celle-ci, grâce à des bannières sur les sites Web, permet de suivre le cheminement d'un internaute. Doubleclick n'est pas capable d'identifier spécifiquement un individu (heureusement !). Mais elle peut déterminer que parmi les visiteurs du site Web de la compagnie XYZ, 80% ont visité le site de CZZ, 40 % le site érotique de XXX et 30 % le site ludique de VVV. Il donc possible de dresser un " profil " du visiteur qui est un consommateur potentiel.

FICHIERS ET STRINGS

3.1 Les fonctions de base des fichiers

3.1.1 La fonction fopen()

Permet d'ouvrir un fichier, que ce soit pour le lire, le créer, ou y écrire :

```
$fp = fopen("../fichier.txt","r"); //lecture  
$fp = fopen("http://epsic.ch/file.txt","a");//écriture depuis fin du  
fichier
```

Le mode indique le type d'opération qu'il sera possible d'effectuer sur le fichier après ouverture :

r ouverture en lecture seulement

w ouverture en écriture seulement (la fonction crée le fichier s'il n'existe pas)

a ouverture en écriture seulement avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas)

r+ ouverture en lecture et écriture

w+ ouverture en lecture et écriture (la fonction crée le fichier s'il n'existe pas)

a+ ouverture en lecture et écriture avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas)

Il est généralement utile de tester si l'ouverture de fichier s'est bien déroulée ainsi que d'éventuellement stopper le script PHP si cela n'est pas le cas:

```
<?  
if (!$fp = fopen("http://www.epsic.ch","r"))  
{  
    echo "Échec de l'ouverture du fichier";  
    exit;  
}  
else  
{  
    // votre code;  
}  
?>
```

3.1.2 La fonction fclose() :

Un fichier ouvert avec la fonction fopen() doit être fermé, à la fin de son utilisation, par la fonction fclose() en lui passant en paramètre l'entier retourné par la fonction fopen()

3.1.3 La fonction fputs() :

Permet d'écrire une chaîne de caractères dans le fichier

Exemple:

```
$fch = fopen("fichier.txt","w");  
fputs ($fch, "texte a mettre dans fch ") ;
```

4.1.4 La fonction fgets() :

Permet de récupérer une ligne du fichier

Exemple:

```
$fch = fopen("fichier.txt","w");  
$f_recup = fgets($fch, 1024) ;
```

la deuxième expression '1024' correspond au nombre maximal de caractères à lire, toutefois la lecture s'arrêtera si un caractère retour à la ligne est rencontré. Pour lire toutes les lignes, on utilisera une boucle.

3.1.5 La fonction feof() :

Fonction testant si la fin du fichier n'a pas été atteinte.

Exemple 1:

```
<?  
if (!$fp = fopen("fichier.txt","r"))  
{  
    echo "Échec de l'ouverture du fichier";  
    exit;  
}  
else  
{  
    while(!feof($fp))  
    {  
        $Ligne = fgets($fp,255); // On récupère une ligne  
        echo $Ligne;           // On affiche la ligne  
        $Fichier .= $Ligne; // Stocke l'ensemble des lignes dans 1 variable  
    }  
    fclose($fp); // On ferme le fichier  
}  
?>
```

Exemple 2 :

```
<?  
$fp = fopen("php_8_fichier.txt","a"); // ouverture du fichier en écriture  
fputs($fp, "\n"); // on va a la ligne  
fputs($fp, "$nom|$email"); // on écrit le nom et email dans le fichier  
fclose($fp);  
?>
```

Exemple 3 :

```
<?  
$fp = fopen("http://www.epsic.ch","r"); //lecture du fichier home  
while (!feof($fp))  
{ //on parcourt toutes les lignes  
    $page .= fgets($fp, 4096); // lecture du contenu de la ligne  
}  
$titre = eregi("<title>(.)</title>", $page, $regs); //on isole le titre  
echo $regs[1];  
fclose($fp);  
?>
```

3.2 Les tests de fichiers

is_dir() : permet de savoir si le fichier dont le nom est passé en paramètre correspond à un répertoire

```
is_dir(chaine Nom_du_fichier);
```

La fonction is_dir() renvoie 1 s'il s'agit d'un répertoire, 0 dans le cas contraire

```
<?
    if (!is_dir("install"))
    {
        echo "Il ne s'agit pas d'un répertoire";
    }
    else
    {
        echo "Il s'agit bien d'un répertoire";
    }
?>
```

is_executable() : permet de savoir si le fichier dont le nom est passé en paramètre est exécutable

```
is_executable(chaine Nom_du_fichier);
```

La fonction is_executable() renvoie 1 si le fichier est exécutable, 0 dans le cas contraire

is_file() : permet de savoir si le fichier dont le nom est passé en paramètre ne correspond ni à un répertoire, ni à un lien symbolique

```
is_file(chaine Nom_du_fichier);
```

La fonction is_file() renvoie 1 s'il s'agit d'un fichier, 0 dans le cas contraire

is_link() : permet de savoir si le fichier dont le nom est passé en paramètre correspond à un lien symbolique

```
is_link(chaine Nom_du_fichier);
```

La fonction is_link() renvoie 1 s'il s'agit d'un lien symbolique, 0 dans le cas contraire

3.3 Autres façons de lire et écrire

La fonction file() :

Permet de retourner dans un tableau l'intégralité d'un fichier en mettant chacune de ces lignes dans un élément du tableau (rappel: le premier élément d'un tableau est repéré par l'indice 0).

```
file(chaine nomdufichier);
```

L'exemple suivant montre comment parcourir l'ensemble du tableau afin d'afficher le fichier.

```
<?
$Fichier = "fichier.txt";
if (is_file($Fichier))
{
    if
    ($TabFich = file($Fichier))
        for($i = 0; $i < count($TabFich); $i++)
        {
            echo $TabFich[$i];
        }
    else
    {
        echo "Le fichier ne peut être lu...<br>";
    }
}
else
{
    echo "Désolé le fichier n'est pas valide<br>";
}
?>
```

3.4 TRAITEMENT DE STRINGS

3.4.1 chr()

Retourne un caractère spécifique

```
string chr ( int  ascii )
```

chr retourne une chaîne d'un seul caractère, dont le code ASCII est donné par le paramètre `ascii`.

Exemple avec [chr](#)

```
<?php
/* Ajoute un caractère d'échappement à la fin de la chaîne $str */
$str .= chr(27);

/* Ceci est souvent plus pratique, et réalise la même chose */
$str = sprintf("The string ends in escape: %c", 27);
?>
```

3.4.2 count_chars()

Retourne des statistiques sur les caractères utilisés dans une chaîne

```
count_chars ( string , int  mode )
```

`count_chars` compte le nombre d'occurrences de tous les octets présents dans la chaîne `string` et retourne différentes statistiques. Le paramètre optionnel `mode` vaut par défaut 0.

Suivant la valeur de `mode`, [count_chars](#) retourne les informations suivantes :

- 0 - un tableau avec l'octet en index, et la fréquence correspondante pour chaque octet.
- 1 - identique à 0 mais seules les fréquences supérieures à zéro sont listées.
- 2 - identique à 0 mais seules les fréquences nulles sont listées.
- 3 - une chaîne contenant tous les octets utilisés est retournée.
- 4 - une chaîne contenant tous les octets non utilisés est retournée.

3.4.3 explode()

Coupe une chaîne en segments

```
array explode ( string separator , string string , int limit )
```

`explode` retourne un tableau de chaînes. Ce sont les sous-chaînes, extraites de `string`, en utilisant le séparateur `separator`. Si `limit` est fourni, le tableau retourné aura un maximum de `limit` éléments, et le dernier élément du tableau contiendra le reste de la chaîne `string`.

Si `separator` est une chaîne vide (""), `explode` retournera `FALSE`. Si `separator` contient une valeur qui n'est pas contenue dans `string`, alors `explode` retournera un tableau, contenant la chaîne `string` entière.

Exemple avec `explode`

```
<?php
$pizza = "garniture1 garniture2 garniture3 garniture4";
$pieces = explode(" ", $pizza);

$data = "foo*:1023:1000::/home/foo:/bin/sh";
list($user,$pass,$uid,$gid,$gecos,$home,$shell) = explode(":",$data);
?>
```

3.4.4 implode()

Rassemble les éléments d'un tableau en une chaîne

```
string implode ( string glue , array pieces )
string implode ( array pieces , string glue )
```

`implode` retourne une chaîne contenant la représentation en chaîne de caractères de tous les éléments du tableau `pieces`, dans le même ordre, avec la chaîne `glue`, placée entre deux éléments.

Exemple avec `implode`

```
<?php
$array = array('nom', 'email', 'telephone');
$comma_separated = implode(",", $array);

print $comma_separated;
// nom,email,telephone
?>
```

3.4.5 Fonction substr

La fonction substr permet d'extraire une portion de chaîne de caractères.

Syntaxe : chaîne substr(chaine, int position, int longueur)

- a) chaine : chaîne à traiter
- b) position : position du caractère de départ ('0' étant la position du 1er caractère)
- c) longueur : longueur de la portion à extraire

La position peut être positive ou négative :

- positive : position à partir du début de la chaîne
- négative : position à partir de la fin de la chaîne

Exemple :

```
<?php
    $machaine="Lundi 1 Janvier 2001";
    $nomdujour=substr($machaine,0,5);
    $jour=substr($machaine,6,1);
    $mois=substr($machaine,8,7);
    $annee=substr($machaine,-4);
?>
```

3.4.6 htmlspecialchars()

Convertis les caractères spéciaux en entités HTML

htmlspecialchars est pratique pour éviter que des données fournies par les utilisateurs contiennent des balises HTML, comme pour un forum ou un chat. Cette fonction prend un deuxième argument optionnel, qui indique comment doivent être traités les guillemets doubles et simples.

Les remplacements effectués sont :

- '&' (ampersand) devient '&'
- '"' (guillemets doubles) devient '"'
- '"' (single quote) devient '''
- '<' (supérieur à) devient '<'
- '>' (inférieur à) devient '>'

Exemple avec [htmlspecialchars](#)

```
<?php
$new = htmlspecialchars("<a href='test'>Test</a>");
echo $new;
// <a href=&#039;test&#039;>Test</a>
?>
```

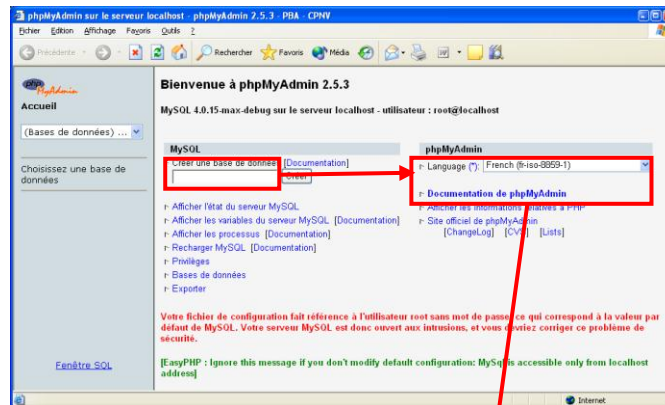


BASES DE DONNÉES

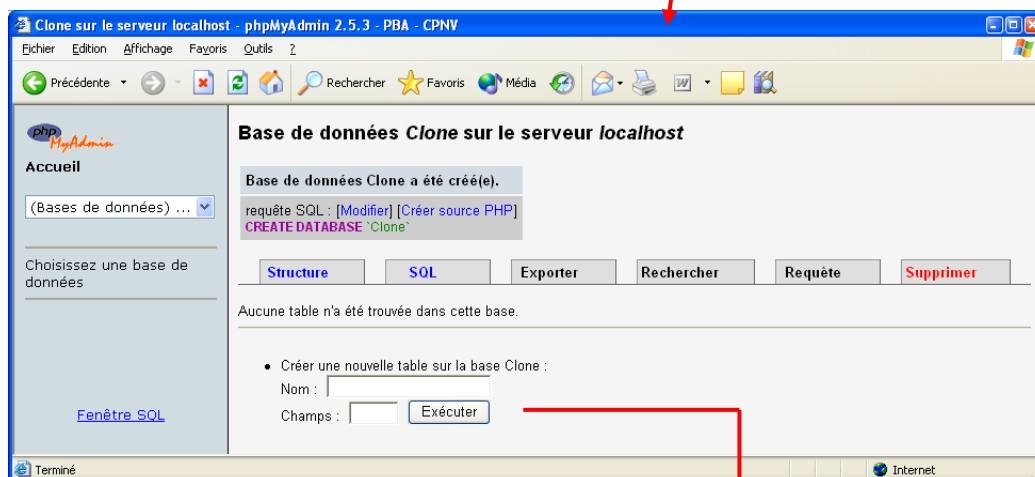
4.1 PhpMyadmin

PhpMyadmin est une interface web de gestion de la base de données MySQL. Son utilisation est relativement simple.

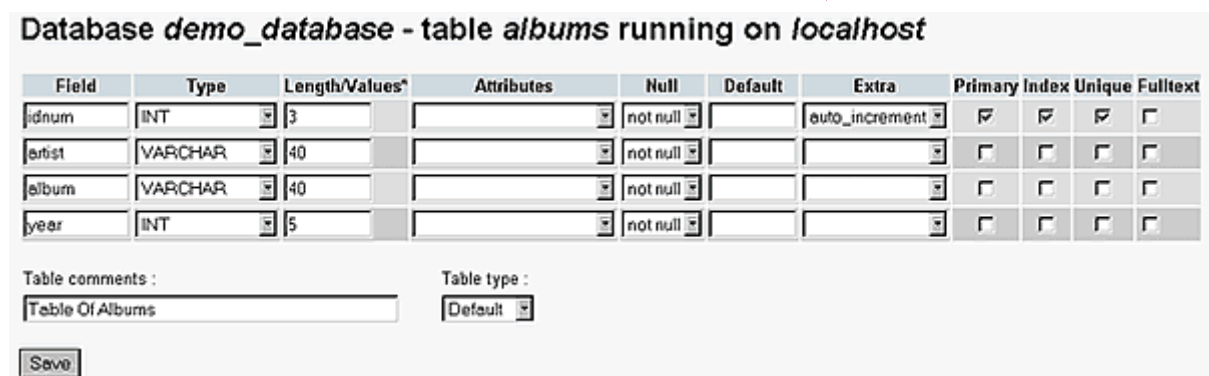
Pour y accéder, il est possible de passer par la page d'administration d'EasyPHP. En cliquant sur le bouton PHP MyAdmin, vous avez accès à l'interface Web.



Pour créer une table : créez une nouvelle base ...



Puis une nouvelle table avec un certain nombre de champs.



Les champs peuvent être des entiers (INT ou LONGINT), des Réels (FLOAT) des caractères (CHR) ou des chaînes (TEXT ou LONGTEXT). La clé de table est définie comme ci-dessus. Cependant, notez que MySql ne gère pas automatiquement les relations entre tables.

4.2 Les fonctions de base

Dans les exemples ci-dessous, le système de gestion de bases de données utilisé est MySQL.

La déclaration des variables :

```
$user : Le nom d'utilisateur  
$passwd : Le mot de passe  
$host : L'hôte  
$bdd : Le nom de la base de données
```

Ces variables sont en fait des paramètres de la fonction permettant la connexion à la base de données.

Php fournit un grand choix de fonctions permettant de manipuler les bases de données. Toutefois, parmi celles-ci quatre fonctions sont essentielles:

La fonction de connexion au serveur

Avec MySQL, ces fonctions sont les suivantes:

```
mysql_connect  
mysql_select_db  
mysql_db_query  
mysql_close
```

5.3 Exploitation d'une base de données

Exemple 1 : connexion.php3

```
<?  
$host = "Localhost" ;  
$user = "utilisateur" ;  
$password = "passe" ;  
  
function connect ()  
{  
    if ( ! $linkid=mysql_connect($host, $user, $password))  
    {  
        echo" impossible d'établir la connexion." ;  
        exit ;  
    }  
    return $linkid;  
}  
  
$nom = "Dupont" ;  
$tel = "021 987 654 " ;  
$ok = mysql_db_query($database, "insert into clients (nom, tel) values  
('$nom', '$tel') ");  
?>
```

5.4 Gestion des erreurs de connexion : 2 méthodes

Variable de stockage du résultat de l'exécution :

```
$connect = mysql_connect($host,$user,$password);
```

la fonction `die()` : en cas d'erreur d'exécution. Si la fonction retourne la valeur 0 (c'est-à-dire s'il y a une erreur) la fonction `die()` renvoie un message d'erreur :

```
mysql_connect($host,$user,$password) or die("erreur de connexion au serveur $host");
```

4.4 Traitement des résultats

Lorsque l'on effectue une requête de sélection à l'aide de la fonction `mysql_query`, il est essentiel de stocker le résultat de la requête dans une variable, que l'on nomme généralement `$result`.

Toutefois, cette variable contient l'ensemble des enregistrements et n'est donc pas exploitable telle quelle. Ainsi, on utilise la fonction `mysql_fetch_row()`, qui découpe les lignes de résultat en colonnes (par exemple Nom,adresse,...) et les affecte à une variable tableau dans l'ordre où elles arrivent.

Ainsi, imaginons une table appelée liens contenant le nom et l'URL de sites Internet. Il est possible de récupérer l'ensemble des enregistrements et de les afficher dans un tableau:

```
<?php
    $host = la_machine;
    $user = votre_login;
    $bdd = Nom_de_la_base_de_donnees;
    $password = Mot_de_passe;

    // Connexion au serveur
    mysql_connect($host, $user,$password) or die("erreur de connexion au
    serveur");
    mysql_select_db($bdd) or die("erreur de connexion a la base de données");

    // Création et envoi de la requête
    $query = "SELECT nom,url FROM sites ORDER BY nom";
    $result = mysql_query($query);

    // Récupération des résultats
    while($row = mysql_fetch_row($result))
    {
        $Nom = $row[0];
        $Url = $row[1];

        echo "<tr>\n
        <td><a href=\""$Url\""$>$Nom</a></td>\n
        <td>$Url</td>\n
        </tr>\n";
    }
    // Déconnexion de la base de données
    mysql_close();
?>
</tr></table>
```

Dans l'exemple ci-dessus, les requêtes retournent les champs nom et url. La fonction `mysql_fetch_row()` analyse donc chaque ligne de résultat de la requête et stocke les colonnes dans le tableau `row[]`. Ainsi, le champ nom sera stocké dans `row[0]` et url dans `row[1]`.

4.5 Détecter un résultat nul

Il peut être utile, avant d'insérer des données dans une table, de détecter la présence d'un enregistrement dans une table, afin d'éviter de stocker des doublons. Cela peut se faire en effectuant une requête SQL avec un ordre `SELECT` et une clause `WHERE` permettant de vérifier la présence ou non d'enregistrements correspondant à la requête. La non-présence de résultat se traduit par un retour nul de la part de la fonction `mysql_fetch_row()`.

Voici un exemple affichant le résultat d'une requête le cas échéant, et dans le cas contraire une phrase expliquant qu'aucun enregistrement correspondant n'a été trouvé

```
<?php
    $host = la_machine;
    $user= votre_login;
    $bdd = Nom_de_la_base_de_donnees;
    $password = Mot_de_passe;

    // Connexion au serveur
    mysql_connect($host, $user,$password) or die("erreur de connexion au
serveur");
    mysql_select_db($bdd) or die("erreur de connexion a la base de données");

    // Création et envoi de la requête
    $query = "SELECT nom,url FROM sites ORDER BY nom";
    $result = mysql_query($query);

    // Recuperation des resultats
    if (!mysql_fetch_row($result)) {
        echo "Aucun enregitrement ne correspond\n";
    }
    else
    {
        while($row = mysql_fetch_row($result))
        {
            $Nom = $row[0];
            $Url = $row[1];

            echo "<tr>\n
            <td><a href=\"\$Url\">$Nom</a></td>\n
            <td>$Url</td>\n
            </tr>\n";
        }
    }
    mysql_close();
?>
```

Trouvez l'erreur dans ce code : le premier enregistrement ne sera jamais affiché. ☺

LES SESSIONS

5.1 Introduction

Il existe deux moyens de recueillir les informations adéquates concernant les utilisateurs accédant à votre site :

- les cookies (pas toujours très appréciés par les internautes)
- l'implémentation de suivi de session (plus populaire)

Le support des sessions de PHP est un moyen de préserver des données entre plusieurs accès. Cela vous permet de créer des applications personnalisées, et d'augmenter l'attractivité de votre site.

Chaque visiteur accédant à votre page Web se voit assigner un identifiant unique, appelé 'identifiant de session'. Il peut être stocké soit dans un cookie, soit propagé dans l'URL.

Le support des sessions vous permet d'enregistrer un nombre illimité de variables qui doivent être préservées entre les requêtes. Lorsqu'un visiteur accède à votre site, PHP va vérifier automatiquement (si ***session.auto_start*** est activé¹) ou sur demande (explicitement avec ***session_start*** ou implicitement avec ***session_register***) s'il existe une session du même nom. Si c'est le cas, l'environnement précédemment sauvé sera recréé.

Toutes les variables sont sérialisées après l'exécution du script PHP. Les variables qui sont indéfinies sont marquées comme telles. Lors des accès ultérieurs, elles ne seront pas définies, jusqu'à ce que l'utilisateur le fasse.

Attention : La notion de session a été introduite dans la version 4.0, donc si vous utilisez une version antérieure, il vous faudra télécharger un paquetage supplémentaire

5.2 Session et la sécurité

Attention : Lorsque vous utilisez les sessions, les données de session pourront être visibles par plus d'un utilisateur.

Il est important de garder cela en tête, lorsque vous stockez et affichez des données importantes. Lorsque vous stockez des données dans une session, il faut se demander quels seront les problèmes posés si quelqu'un d'autre accède à cette information, ou comment votre application est affectée si la session est en fait celle d'un autre.

Par exemple, si quelqu'un usurpe une session, il peut alors poster un message dans un forum sous ne fausse identité. Quelle est la gravité de ce problème? Ou bien, il peut accéder aux commandes d'un client, et même, modifier son panier d'achats. A priori, c'est moins problématique pour un fleuriste que pour un pharmacien.

Par conséquent, lorsque vous manipulez des données importantes, il faut exploiter d'autres méthodes pour décider si une session est valide ou pas. Les sessions ne fournissent pas une méthode fiable d'authentification.

Il existe toutefois un moyen d'accroître un peu la sécurité de ces variables session. Il s'agit du *codage* et *décodage* des données. En effet, PHP fournit la fonction ***session_encode()*** pour encoder les informations de la session courante. Cette chaîne

peut alors être décodée et il est possible de récupérer les variables initiales par le biais de la fonction `session_decode()`. Vous trouverez un exemple d'utilisation au chapitre 6.3

Les sessions reposent sur un identifiant de session, ce qui signifie que quelqu'un peut voler cet identifiant, rien qu'en volant l'ID. Ce vol peut être rendu très difficile, comme par exemple en utilisant les cookies, mais en aucun cas cela sera impossible. Les sessions dépendent aussi de la discipline de l'utilisateur qui referme son navigateur à la fin de la session pour tout clore proprement. De plus, même les cookies de session peuvent être surveillés sur un réseau, ou bien notés par un Proxy.

5.3 Utilisation de session

Une session commence lorsqu'un surfeur arrive sur votre page Web et se termine à son départ.

Toutes les pages utilisant les sessions doivent appeler la fonction `session_start()` pour indiquer au moteur PHP4 de charger les informations relatives à la session dans la mémoire.

L'utilisation des sessions entraîne l'enregistrement de fichiers contenant des informations sur vos variables de sessions.

session.name définit le nom de session qui sera utilisé comme nom de cookie. Il ne peut contenir que des caractères alphanumériques. Par défaut : PHPSESSID.

session.lifetime définit le temps de vie du cookie en secondes, celui-ci est envoyé au navigateur. La valeur 0 signifie que le navigateur est ouvert. Valeur par défaut : 0.

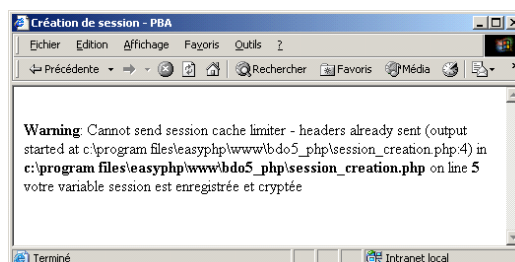
session_start() démarre l'utilisation des sessions.

Pour enregistrer une variable de session, assignez une valeur à une variable qui va devenir une variable de session et appelez la fonction **session_register("nom_de_la_var")**.

Sur toutes les prochaines pages utilisant les sessions (en appelant la fonction `session_start()`), la variable `nom_de_la_var` aura la valeur qui lui a été assignée plus tôt quand elle a été enregistrée comme une variable de session.

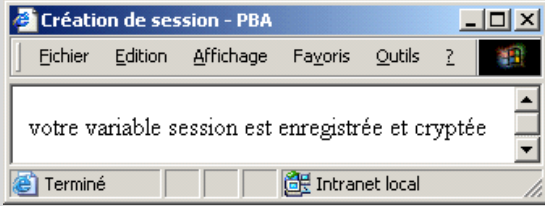
Des changements de la valeur de la variable seront automatiquement enregistrés dans la session et sauvegardés pour les prochaines utilisations.

Attention : Il ne doit **pas** y avoir **de code HTML ou d'instruction d'affichage avant l'écriture de la session** sinon le parser vous générera une erreur comme montré ci-dessous. Les sessions étant passées en en-tête HTTP vers le serveur, elles ne peuvent être traitées qu'avant l'affichage du moindre caractère.



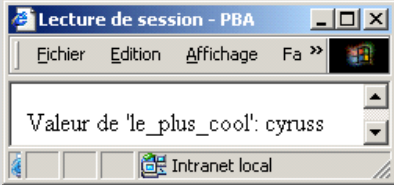
Exemple : Page 1 :

```
<?
// Initialisation de la session
session_start();
$le_plus_cool = "cyruss";
// Enregistrement de la donnée
session_register( "le_plus_cool");
// Cryptage de la donnée
session_encode();
echo "votre variable session est enregistrée et cryptée";
?>
```



Page 2 :

```
<?
session_start();
// Décryptage de la donnée session
session_decode($le_plus_cool);
print "Valeur de 'le_plus_cool':    *
    $_SESSION['le_plus_cool']";
// Destruction de la session
session_destroy()
?>
* : saut de ligne uniquement utilisé pour l'affichage (pas dans le code)
```



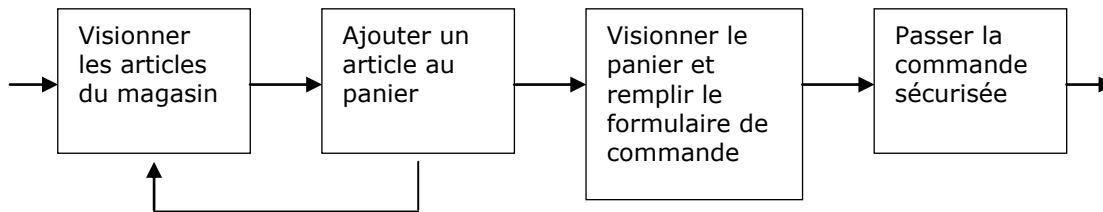
Résultat : la création de la variable de session : var_session.

Cela veut dire que cette variable sera maintenue en vie à travers tous les accès aux pages aussi longtemps que la fonction session_start() sera appelée.

La destruction de la session ne se fait que si les données de la session n'ont plus aucune utilité pour l'administrateur du site.

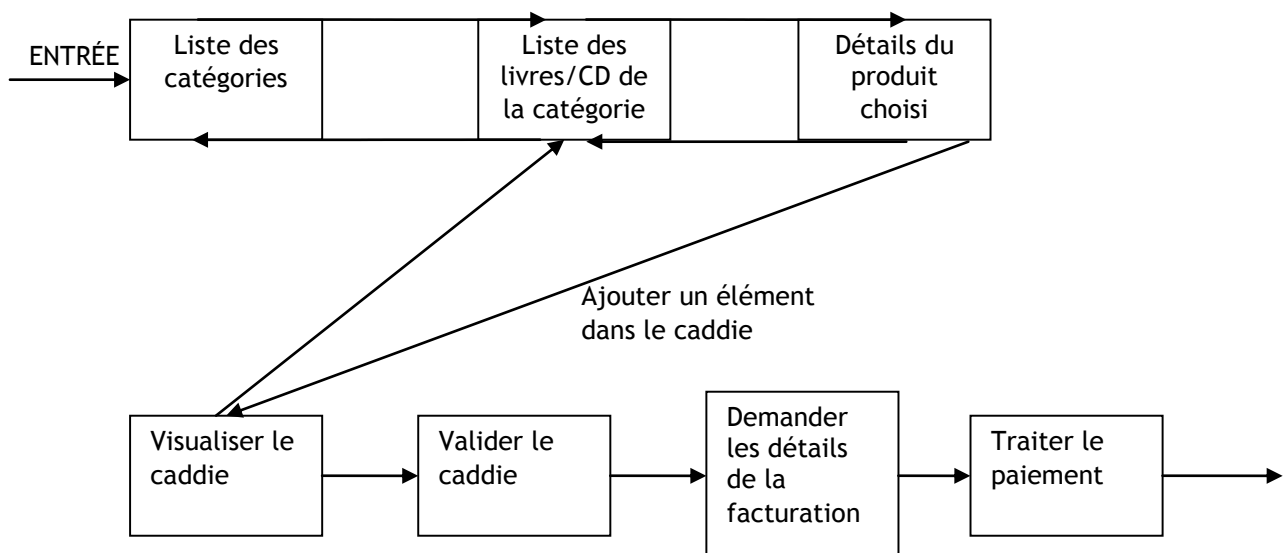
Implémentation d'un caddie

6.1 Schéma du pipeline suivi par l'utilisateur.



6.2.- Présentation d'un exemple d'architecture

Le site et l'implémentation d'un caddie seront interprétés différemment en fonction de la personne qui traite celui-ci. Dans la figure qui suit, l'architecture du site du point de vue utilisateur est représentée.

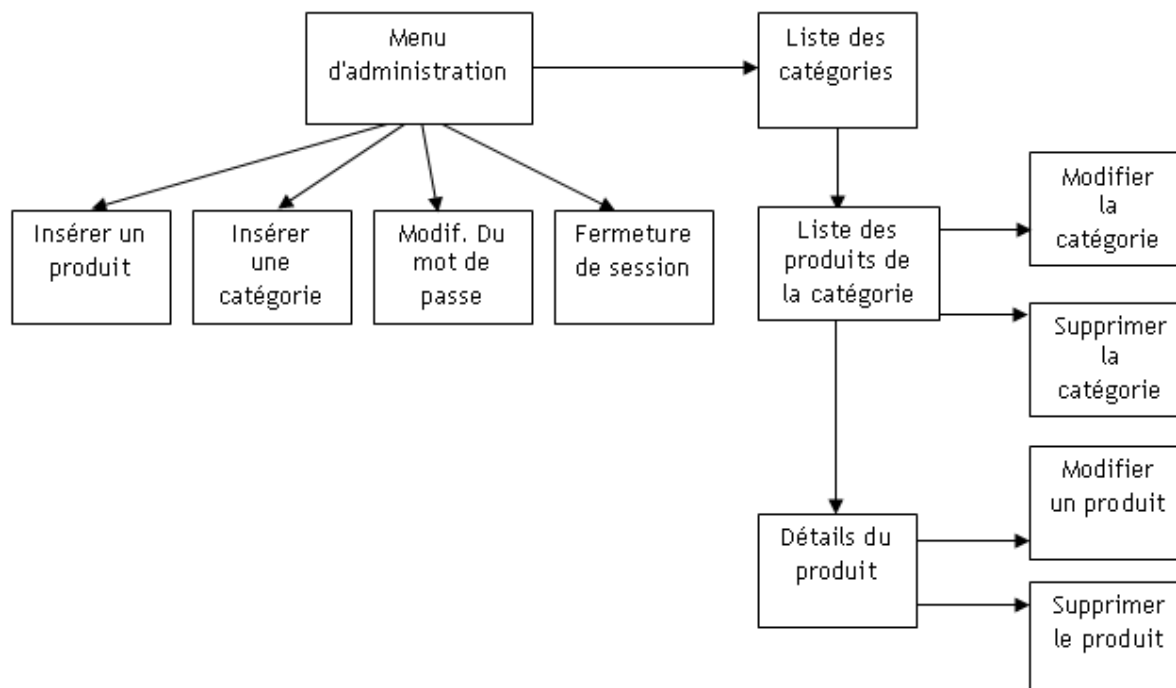


Le client aura la possibilité de gérer son caddie, d'y ajouter (nombre d'articles du même type et/ou nouveaux articles) ou supprimer des articles. Il va de soi qu'après la validation de celui-ci il pourra lors de la facturation modifier ses données personnelles et l'adresse d'envoi. Un traitement comme le calcul du prix total, de la TVA et des frais de port.

Il va de soi que pour la réalisation du caddie (côté client) plusieurs solutions sont envisageables avec l'utilisation de différentes techniques comme :

- Les variables session et/ou les querystring
- Les classes
- La sauvegarde dans une base de données
- Les cookies

Point de vue administrateur la représentation se fera comme suit :



6.3.- Techniques d'implémentation d'un caddie

Généralement la fonctionnalité de caddie fait intervenir des variables **session** sous forme de tableau associatif avec les clés, les descriptions et les valeurs de quantités. Toutefois, il est souvent conseillé d'utiliser la notion de **classe** avec une structure permettant de regrouper tous les paramètres du caddie comme montré dans le paragraphe 9.5.

Le souci premier sera de conserver les données envoyées par le visiteur pendant sa présence sur le site (ou "session client") : en effet, il ne fait que passer de page en page, sélectionnant ici ou là un ou plusieurs produits : il s'attend logiquement à tous les retrouver dans son panier. Pour ce faire, il est possible d'utiliser les **cookies** et/ou les **sessions**

Il est possible d'utiliser une table spécifique dans une **base de données**, mais cela reste une solution peu viable du fait qu'il est nécessaire de stocker des données temporaires et que dans certains cas cela peut se révéler plus lent.

6.3.1.- Technique des cookies

Un cookie peut contenir jusqu'à 4 Ko d'informations, et un serveur ne peut normalement envoyer que 20 cookies par client. Par ailleurs, un ordinateur n'est pas sensé accepter plus de 300 cookies. Sa durée de vie peut varier de la durée de la session client (jusqu'à la fermeture du navigateur) à plusieurs années...

Une taille de 4 Ko permet de stocker un bon nombre d'informations, mais il peut toujours arriver une commande dépassant cette taille. Pire : les clients peuvent parfaitement choisir de ne pas accepter certains cookies...

6.3.2.- technique des sessions

Les sessions sont probablement un meilleur choix. Lorsqu'un site utilise des sessions, chaque visiteur reçoit un identifiant unique, qui correspond au nom du fichier, stocké sur

le serveur, contenant les informations enregistrées par celui-ci. La taille du fichier est (en théorie) illimitée, il le suit durant la totalité de la session, et même après : la durée de vie d'une session est généralement de 24 minutes, donc si l'utilisateur ferme le navigateur par inadvertance, il pourra retrouver sa session en se reconnectant...

Notre session, tout du long du cheminement du (futur) client, contiendra une variable dédiée uniquement à la commande en train de se construire : un tableau associatif, \$achats[x][y]. Le tableau "x" ne contiendra qu'une valeur, le numéro du produit dans la commande, tandis que le tableau "y" contiendra l'id du produit, et la quantité voulue.

6.4.- Exemple de caddie avec des sessions

L'exemple qui suit montre comment gérer (création, ajout, suppression et mise à jour) un caddie avec des sessions.

Par exemple, prenons un caddie pour un seul produit :

```
<?php
$achats = array ( "1" => array("id" => 135423, "qte" => 2) );
?>
```

En travaillant avec les sessions, cela prendrait la forme suivante :

```
session_start();
$_SESSION['achats'] = array ("id" => 135423, "qte" => 2);
```

L'utilisation de ces variables de session se fait ensuite via la variable \$_SESSION:

```
$id = $_SESSION['achats']['id'];
$qte = $_SESSION['achats']['qte'];
```

Lors de l'utilisation d'un caddie, l'utilisateur à quelques commandes de base : **ajouter un produit dans le caddie** (utilisé pendant le parcours du catalogue), **retirer un produit de la commande** (utilisé lors de l'affichage final), **actualiser le caddie** (idem, affichage final - pour les clients qui voudraient ajouter ou enlever des produits) et enfin, **valider la commande**. Chacune de ses commandes fera l'objet d'un script au sein de notre application.

1) Ajouter

Nous partons du principe que nous sommes dans le catalogue, et que plusieurs produits sont affichés. L'affichage des produits est bien entendu dynamique.

Nos produits sont présentés ainsi :

Produit	Prix	Quantité	
PremierProduit	127	2 ▼	Ajouter
AutreProduit	230	1 ▼	Ajouter
UnBonProduit	450	1 ▼	Ajouter
UnProduitPasCher	12	1 ▼	Ajouter

On veut que toute l'interaction se fasse à partir du lien situé sous le texte *Ajouter*: quand on le clique. On veut de plus que les références du produit de ce champ (son id et la quantité souhaitée) soient ajoutées à notre variable \$liste[[]], et que le catalogue s'affiche à nouveau, à la même page, avec la valeur des quantités mises à jour. Ce lien renvoie donc sur la même page *catalogue.php*.

Le code PHP gérant l'ajout prendra cette forme :

```
if ($_GET['ajout'] == "AJOUT")
{
    $_SESSION['achats'][] = array( "id"=>$_GET['id'], "qte" =>$_GET['qte']);
}
```

Le lien Ajouter correspondra donc à :

```
catalogue.php?ajout="AJOUT"&id=<?=$row['id'];?>&qte=<?=$qte123456;?>
```

...c'est à dire, après compilation par PHP, donnerait :

```
catalogue.php?ajout="AJOUT"&id=123456&qte=3
```

2) Supprimer

Nous travaillons maintenant dans notre caddie (caddie.php). Il se présente ainsi:

Produit	Prix	Quantité	Total		
PremierProduit	127	3 ▼	381	Mettre à jour	Retirer
UnBonProduit	450	5 ▼	2250	Mettre à jour	Retirer
Prix total		8 produit(s)	2631	Valider la commande	

Nous laissons les calculs et affichages des totaux, qui ne font qu'utiliser les mêmes méthodes que pour l'affichage du catalogue, l'arithmétique en plus. L'affichage des quantités dans le menu déroulant est intelligemment géré par l'utilisation de "selected".

```
for ($i=0; $i < count($_SESSION['achats']); $i++)
{
    // code MySQL allant chercher les informations pour chaque produit
    // en fonction de $_SESSION['achats'][$i]['id']
    ?>

    <select name="qte<?=$i"?>">
        <option value="1"
            <?=(($_SESSION['achats'][$i]['qte'] == 1 ? "SELECTED" : null);?>
        </option>
        <option value="12"
            <?=(($_SESSION['achats'][$i]['qte'] == 2 ? "SELECTED" : null);?>
        <!-- etc. -->
        </option>
    </select>
    <?>
```

Nous travaillons toujours avec nos variables de session : pour supprimer un élément de \$_SESSION['achats'], il nous faut la parcourir, à la recherche de l'id du produit à supprimer. Notre lien [Mettre A Jour](#) sera de la forme suivante :

```
caddie.php?retirer=RETIRER&id_produit=<?=$row['id'];?>
```

...c'est à dire...

```
caddie.php?retirer=RETIRER&id_produit=123456
```

Nous ferons cela à l'aide d'une boucle `for` et de la fonction `array_splice()`, qui permet d'effacer une portion d'un tableau :

```
if ($_GET['retirer'] == "RETIRER")
{
```

```

for ($i=0; $i < count($_SESSION['achats']); $i++)
{
    if ($i == $_GET['id_produit'] )
        array_splice($_SESSION['achats'], $i, 1);
}
}

```

3) Mettre à jour

Sur le même modèle que Supprimer : on charge la même page, en mettant simplement à jour les valeurs de session...

4) Valider

Cette dernière commande renvoie la page d'affichage final, où l'utilisateur ne peut plus modifier sa liste d'achats, ni revenir en arrière. Les valeurs sont sauveées dans la base MySQL, sur le schéma suivant :

```

$req = "INSERT INTO commandes (id_client, date, total, etat)
VALUES (xborderie, " date(); . " , " . $_GET['total'] . " , "En cours..");"

mysql_query($req);
$commande_id = mysql_insert_id();

for ($i=0; $i < count($_SESSION['achats']); $i++)
{
    $req2 = "INSERT INTO achats (id_produit,id_commande,quantite,prix_total)
VALUES (" . $_SESSION['achats'][$i]['id'] . " , " . $commande_id . " . " , "
$_SESSION['achats'][$i]['qte'] . " , " . $_SESSION['achats'][$i]['qte'] *
$_SESSION['achats'][$i]['prix'] . " , ";
    mysql_query($req2);
}

```

6.5.- Classe pour un caddie (panier virtuel)

Dans l'exemple de caddie, l'utilisation d'une classe Caddie en php montre comment avec quelques méthodes et propriétés liées à cette classe, il est possible de gérer aisément un caddie :

```

<?php

class Caddie
{
    var $produits; //Tableau des produits du caddie
    var $nbproduits; //Nombre de produits dans le caddie
    var $date; //Date de la création du caddie
    var $idclient; //Identifiant du client à qui appartient le caddie
    //var $montant; //Montant total du caddie

    //Constructeur initialise le tableau de produits du caddie et montant
    function Caddie()
    {
        $this->produits = array();
        $this->nbproduits = 0;
        $this->date = date("d/m/Y");
        $this->idclient = "";
        //$this->montant = 0;
    }

    //Renvoie la reference du produit $i
    function getRefProduits($i)
    { return ($this->produits[$i]['id']); }
}

```

```
//Renvoie la quantité du produit $i
function getQteProduits($i)
{ return ($this->produits[$i]['qte']); }

//Renvoie le nombre de produits contenus dans le caddie
function nombreDeProduits()
{ return $this->nbproduits; }

//Renvoi la date de la création du caddie
function getDateCaddie()
{ return $this->date; }

//Définie l'identifiant du client à qui appartient le caddie
function setIdClientCaddie($id)
{ $this->idclient = $id; }

//Renvoi l'identifiant du client à qui appartient le caddie
function getIdClientCaddie()
{ return $this->idclient; }

//Ajoute un produit dans le caddie
function ajouter($refproduit,$quantite)
{
    if (!empty($refproduit))
    {
        $this->produits[] = array("id"=>$refproduit,"qte"=>$quantite);
        $this->nbproduits++;
    }
}

//Supprime un produit du caddie
function supprimer($refproduit)
{
    if (!empty($refproduit))
    {
        for($i=0; $i<$this->nbproduits; $i++)
        {
            if ($this->produits[$i]['id'] == $refproduit)
            {
                array_splice($this->produits, $i, 1);
                $this->nbproduits--;
                break;
            }
        }
    }
}

//Met à jour la quantité d'un produit sélectionné dans le caddie
function miseAJour($refproduit,$quantite)
{
    if (!empty($refproduit))
    {
        for($i=0; $i<$this->nbproduits; $i++)
        {
            if ($this->produits[$i]['id'] == $refproduit)
            {
                $this->produits[$i]['qte'] = $quantite;
                break;
            }
        }
    }
}
?>
```

Bibliographie

Livres

Php & MySQL 2^{ème} édition – CampusPress – ISBN : 2-7440-1860-0

Sites

<http://www.easyphp.org>

<http://www.selfhtml.org>

<http://developpeur.journaldunet.com/tutoriel>

<http://www.phpcs.com>

<http://www.expreg.com/presentation.php>

<http://www.siteduzero.com/tuto-3-150-1-memo-pour-les-regex.html>

Références

Support de cours de M. André Mottier

Support de cours de M. Pascal Bonzonana