

Site Web Dynamique : Développement PHP

1) Introduction à PHP

a. Les sites statiques et dynamiques

On considère qu'il existe deux types de sites web : les **sites statiques** et les **sites dynamiques**.

Les sites dynamiques

Plus complexes, ils utilisent d'autres langages en plus de HTML et CSS, tels que PHP et MySQL. Le contenu de ces sites web est dit « dynamique » parce qu'il peut changer sans l'intervention du webmaster !

Le seul prérequis pour apprendre à créer ce type de sites est de savoir réaliser des sites statiques en HTML et CSS.

PHP est un langage interprété (un langage de script) exécuté du côté serveur et non du côté client (un script écrit en Javascript ou une applet Java s'exécute sur votre ordinateur...).

La syntaxe du langage provient de celles du langage C, du Perl et de Java.

Les pages web contenant du PHP ont l'extension **.php**.

b. Architecture Client-Serveur :

Architecture "3-tiers" car on peut décomposer fonctionnellement notre application en 3 couches distinctes :

↻ Couche présentation :

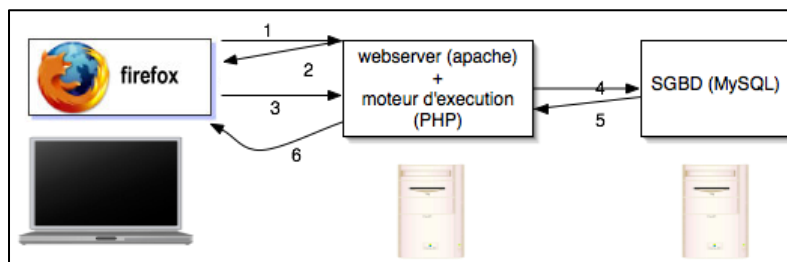
C'est la partie de l'application visible par les utilisateurs.

↻ Couche métier :

C'est la "logique" de l'application elle-même, c'est-à-dire les algorithmes implémentés pour remplir les fonctions spécifiées.

↻ Couche de données :

C'est l'ensemble des données de l'application.



c. PHP pour le WEB :

Pour que l'ordinateur puisse lire du PHP, il faut qu'il se comporte comme un serveur. Il suffit simplement d'installer les mêmes programmes que ceux que l'on trouve sur les serveurs qui délivrent les sites web aux internautes.

Ces programmes dont nous allons avoir besoin, sont :

- Apache.
- PHP.
- MySQL.

⌘ Apache : c'est un serveur web.

Il s'agit du plus important de tous les programmes, car c'est lui qui est chargé de délivrer les pages web aux visiteurs.

Cependant, Apache ne gère que les sites web statiques (il ne peut traiter que des pages HTML). Il faut donc le compléter avec d'autres programmes.

⌘ PHP :

C'est un plug-in pour Apache qui le rend capable de traiter des pages web dynamiques en PHP. En combinant Apache et PHP, notre ordinateur sera capable de lire des pages web en PHP.

⌘ MySQL :

C'est le logiciel de gestion de bases de données. Il permet d'enregistrer des données de manière organisée (comme la liste des membres de votre site).

⌘ Outils de développement

Il existe plusieurs paquetages tout prêts pour Windows, nous allons utiliser WampServer sachant qu'il existe d'autres paquetages (easyphp, xampp) qui peuvent rendre les mêmes services que notre choix.

⌘ Exemple de Script PHP

On peut distinguer certaines similitudes avec le HTML, par exemple l'utilisation d'une balise de début "<?php" et d'une balise de fin ">".

Toute portion de code PHP doit donc être placée entre une balise d'ouverture et une balise de fermeture.

Si votre code n'est pas correctement écrit, PHP arrête l'exécution du script en lançant une erreur :

```
<html >
  <head>
    <title> </title>
  </head>
  <body>
    <?php
      ..... //code PHP ;
    ?>
  </body>
</html>
```

Parse error: syntax error, unexpected ';' in C:\.....cheminphp on line 2

d. Les commentaires

Un commentaire est un texte que vous mettez pour vous dans le code PHP. Ce texte est ignoré, c'est-à-dire qu'il disparaît complètement lors de la génération de la page. Il n'y a que vous qui voyez ce texte.

Il existe deux types de commentaires :

⌘ Les commentaires mon lignes.

```
<?php
echo "J'habite en Chine."; // Cette ligne indique où j'habite

// La ligne suivante indique mon âge
echo "J'ai 92 ans.";
?>
```

⌘ Les commentaires multilignes.

```
<?php
/* La ligne suivante indique mon âge
Si vous ne me croyez pas...
... vous avez raison ;o) */
echo "J'ai 92 ans.";
?>
```

e. Méthode d'affichage en PHP :

Les différences sont faibles : **echo** n'a pas de valeur de retour tandis que **print** a une valeur de retour de 1, donc il peut être utilisé dans les expressions.

L'écho peut prendre plusieurs paramètres (bien que cette utilisation se voit rare) alors que print peut prendre un argument. L'écho est marginalement plus rapide que l'impression.

```
<?php
    echo "Hello world!";
?>

Ou

<?php
    print "Hello world!";
?>
```

2) PHP : les variables.

Une variable est toujours constituée de deux éléments :

∞ Son nom :

Pour pouvoir la reconnaître, vous devez donner un nom à votre variable. Par exemple `age_du_visiteur` ;




∞ Sa valeur :

C'est l'information qu'elle contient, et qui peut changer. Par exemple : `17`.

a. Les différents types de variables

Les variables sont capables de stocker différents types d'informations. On parle de types de données. Voici les principaux types à connaître :

- **Les chaînes de caractères (string)** : les chaînes de caractères sont le nom informatique qu'on donne au texte.
- **Les nombres entiers (int)** : ce sont les nombres du type 1, 2, 3, 4, etc.
- **Les nombres décimaux (float)** : ce sont les nombres à virgule
- **Les booléens (bool)** : c'est un type très important qui permet de stocker soit vrai soit faux.
- **Rien (NULL)** : on indique qu'une variable ne contient rien.

Type de données	Exemple de valeur
string	"Du texte"
int	42
float	14.738
bool	true  false 
NULL	

b. Déclaration et affectation d'une variable :

Soit le code PHP suivant :

```
<?php
$age_du_visiteur = 17;
?>
```

On vient en fait de créer une variable dont :

- Son nom est `age_du_visiteur` ;
- Et sa valeur est `17`.

Remarques :

Le nom de la variable est sensible à la case :

`$ma_variable` # `$mA_VaRiable`,

Ce sont 2 variables différentes.

Vous n'avez pas à spécifier le type de variable en PHP : Car il prendrait directement le nom de la variable.

On peut utiliser une variable dans une variable ;

Exemple :

```
$var1 = 'bonne journée!';  
$var2 = "Passez une $var1";  
echo $var2;
```

c. Conventions pour nommer les Variables

- Les variables doivent commencer par une lettre ou par l'underscore "_".
- Les variables peuvent être composées seulement par des caractères alpha-numériques et des underscores. a-z, A-Z, 0-9, ou _.
- Les variables de plus d'un mot devront être séparées par des underscores : \$ma_variable.
- Les variables de plus d'un mot peuvent aussi être différenciées avec des majuscules : \$maVariable.
- Il n'y a pas de limite à la taille des variables.
- Il existe des mots de variables réservés.

d. Chaîne de caractères en PHP :

i. Création de chaînes :

Il est possible de créer une chaîne de caractères en utilisant des apostrophes (des guillemets simples) ' ou des guillemets ".

Le point est utilisé pour concaténer des chaînes, variables ...

```
echo 'Mon nom est ' . $nom;
```

ii. Caractères spéciaux

Il existe un problème avec les chaînes de caractères, quand on veut afficher une chaîne contenant un ' et que celle-ci est délimitée par des '. En effet, cela donne une ligne d'erreur comme :

Parse error: parse error, unexpected T_STRING, expecting ',' or ';' in votrefichier.php on ...

La solution, c'est l'antislash (\) qui permet de faire comprendre à PHP qu'il ne faut pas s'arrêter sur ce caractère-là. Ce qui donne :

```
echo 'j\'utilise php ';      affiche : j'utilise php
```

e. Les opérateurs :

PHP dispose des opérateurs classiques pour effectuer des calculs :

```
<?php  
echo 1+1; //addition  
echo 1-1; //soustraction  
echo 1*1; //multiplication  
echo 1/1; //division  
echo 1%1; //modulo  
?>
```

Les opérateurs d'affectation permettent de donner une valeur à une variable, on cite parmi eux :

=, +=, -=, *=, /=, .=

Exemple :

```
<?php
    $x = 5;
    $x += 1;
    $x *= 3;
    $x /= 4;
    echo '$x';
?>
```

3) Les variables superglobales

Les variables superglobales sont mises en place par PHP lors du début du traitement d'une demande par Apache.

Ces variables n'obéissent pas aux limites habituelles des variables en termes de visibilité à l'intérieur d'une fonction.

Elles sont accessibles de partout, c'est pourquoi elles portent le nom de "superglobales".

Les variables superglobales sont des variables un peu particulières pour trois raisons :

- Elles sont écrites en majuscules et commencent toutes, à une exception près, par un underscore (_). **\$_GET** et **\$_POST** en sont des exemples que vous connaissez ;
- Les superglobales sont des array car elles contiennent généralement de nombreuses informations ;
- Ces variables sont automatiquement créées par PHP à chaque fois qu'une page est chargée. Elles existent donc sur toutes les pages et sont accessibles partout : au milieu de votre code, au début, dans les fonctions, etc.

Les variables ***\$HTTP_*_VARS*** ne sont disponibles que si l'option de configuration *track_vars* a été activée.

Les superglobales :

- **\$_GET** : Les valeurs provenant de l'URL ;
- **\$_POST** : Les valeurs envoyées par formulaire ;
- **\$_FILE** : Les fichiers envoyés par formulaire ;
- **\$_SERVER** : Les valeurs mises en place par le serveur Web (elles peuvent donc changer d'une configuration à l'autre) ;
- **\$_ENV** : Les variables d'environnement (système d'exploitation) ;
- **\$_SESSION** : Les valeurs mises dans le magasin des sessions ;
- **\$_COOKIE** : Les valeurs transmises au moyen de cookies par le navigateur ;
- **\$GLOBALS** : L'ensemble des variables du script.

4) Les boucles :

Les boucles sont l'un des attraits des langages de script, car c'est quelque chose que le HTML ne peut pas faire.

a. FOR:

```
<?php
    for($i=0; $i<1000; ++$i)
    {
        echo $i;
    }
?>
```

b. While :

```
<?php
    While ($continuer_boucle == true)
    {
        // instructions à exécuter dans la boucle
    }
?>
```

En résumé

Les boucles demandent à PHP de répéter des instructions plusieurs fois. Les deux principaux types de boucles sont :

While : à utiliser de préférence lorsqu'on ne sait pas par avance combien de fois la boucle doit être répétée ;

For : à utiliser lorsqu'on veut répéter des instructions un nombre précis de fois.

- L'incrémentement est une technique qui consiste à ajouter 1 à la valeur d'une variable (++\$i).
- La décrémentation retire au contraire 1 à cette variable. On trouve souvent des incrémentations au sein de boucles for (--\$i).

5) Les conditions

Une condition peut être écrite en PHP sous différentes formes. On parle de structures conditionnelles.

a. if... else

```
<?php
    if(<expression>)
    {
        <instructions>
    }
    else
    {.....}
?>
```

b. switch

```
<?php
switch(<expression>)
```

```
{  
case <valeur 1>:  
<instructions>  
break;  
case <valeur 2>:  
<instructions>  
break;  
...  
default:  
<instructions>  
}
```

c. Symboles de comparaison

Symbole	Signification
=	Est égal à
>	Est supérieur à
<	Est inférieur à
>=	Est supérieur ou égal à
<=	Est inférieur ou égal à
!=	Est différent de

6) Les tableaux :

Un tableau est une variable contenant plusieurs valeurs. En PHP, les variables étant faiblement typées, les tableaux sont très simples à manipuler.

On accède au tableau entier en utilisant le nom de la variable, ou bien à un élément concret au moyen des crochets [et].

Ce qui se situe entre les crochets est appelé "index" ou encore "clef" de l'élément du tableau, et un même index est unique dans un tableau.

Il est possible d'enregistrer de nombreuses informations dans une seule variable grâce aux tableaux. On distingue deux types de tableaux :

- Les tableaux *numérotés* ;
- Les tableaux *associatifs*.

a. Les tableaux numérotés

Dans un tableau numéroté les valeurs sont rangées dans des « cases » différentes, *c'est-à-dire* que chaque case est identifiée par un numéro. Ce numéro est appelé **clé**.

Remarque :

Un tableau numéroté commence toujours à la case n°0 !

Créer un tableau numéroté :

Pour créer un tableau numéroté en PHP, on utilise la fonction **array**.

Exemple :

```
<?php
$preNoms = array ('mohamed', 'ali', 'sara', 'Véronique', 'Benoît');
echo $preNoms[1];
?>
```

b. Les tableaux associatifs

Les tableaux associatifs fonctionnent sur le même principe, sauf qu'au lieu de numéroter les cases, on va les étiqueter en leur donnant à chacune un nom différent.

Créer un tableau associatif

Comme pour les tableaux numérotés Pour en créer un tableau associatif, on utilisera la fonction array mais on va mettre « **l'étiquette** » devant chaque information :

```
<?php
$coordonnees = array (
    'prenom' => 'amine',
    'nom' => 'Otmani',
    'adresse' => '3 Rue de la marine',
    'ville' => 'laayoune');

echo $coordonnees ['nom'];
?>
```

c. Parcourir un tableau

Il existe trois moyens d'explorer un array :

- La boucle **for** ;
- La boucle **foreach** ;
- La fonction **print_r**.

∞ La boucle foreach

La boucle **foreach** est une sorte de boucle for spécialisée dans les tableaux.

foreach va passer en revue chaque ligne du tableau, et lors de chaque passage, elle va mettre la valeur de cette ligne dans une variable temporaire (par exemple \$enregistrement).

L'avantage de **foreach** est qu'il permet aussi de parcourir les tableaux associatifs.

Syntaxe de la boucle Foreach :

```
<?php
    foreach($var_array as $enregistrement)
    {
        //Traitement sur $enregistrement;
    }
?>
```

Remarque :

On peut aussi récupérer la clé de l'élément. On doit dans ce cas écrire *foreach* comme ceci :

```
<?php
    foreach($coordonnees as $cle => $enregistrement)
?>
```

7) Les fonctions

Une fonction permet de réutiliser facilement du code PHP. Une fonction se déclare en utilisant le mot clef "**function**" suivi du nom de la fonction (mêmes règles que pour les noms de variables), d'un couple de parenthèses et d'un couple d'accolades.

Les parenthèses contiennent la liste des paramètres (de zéro à l'infini) et les accolades contiennent les actions de la fonction.

La valeur de retour d'une fonction est définie par le mot clef "return". Dès que ce mot clef est exécuté, PHP sort de la fonction.

a. Include :

L'instruction de langage **include** inclut et exécute le fichier spécifié en argument. Les fichiers sont inclus suivant le chemin du fichier fourni, l'**include_path** sera vérifié. Si le fichier n'est pas trouvé dans l'**include_path**, **include** vérifiera dans le dossier du script appelant et dans le dossier de travail courant avant d'échouer.

L'instruction **include** enverra une erreur de type warning si elle ne peut trouver le fichier, ce comportement est différent de require, qui enverra une erreur de niveau fatal. **Include** retourne **FALSE** et émet une alerte. Les inclusions avec succès, y compris si elles sont écrasées par le fichier inclus, retourne **1**.

b. Les fonctions créées par l'utilisateur :

Une fonction peut être définie en utilisant la syntaxe suivante :

```
<?php
function <nom de la fonction> (<noms des paramètres>)
{
    <instructions>
    return <valeur>; (optionnel)
}
```

Exemple :

```
1 <?php
2 function foo($arg_1, $arg_2, /* ..., */ $arg_n)
3 {
4     echo "Exemple de fonction.\n";
5     return $retval;
6 }
7 ?>
```

Remarques :

Le nom de la fonction suit les mêmes règles que les noms de variables :

- Le nom doit commencer par une lettre
- Un nom de fonction peut comporter des lettres, des chiffres et les caractères `_` et `&` (les espaces ne sont pas autorisés !)

- Le nom de la fonction, comme celui des variables est sensible à la casse (différenciation entre les minuscules et majuscules)

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

c. Appel de fonction

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

Nom_De_La_Fonction (argument_1, argument_2, ..., argument_n);

d. Travailler sur des variables dans les fonctions : la portée des variables.

Il existe plusieurs niveaux de définition de variables :

- **Global.**
- **Static**
- **Local**

⌘ 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

⌘ Static :

Permet de définir une variable locale à la fonction, qui persiste durant tout le temps d'exécution du script

Par défaut, **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

Exemple :

```

1  <?php
2      $chaine = "Nombre de camions : ";
3      function ajoute_camion($mode='')
4      {
5          global $chaine;
6          static $nb=0;
7          $nb++;
8          if($mode == "affiche")
9          {
10             echo $chaine.$nb;
11          }
12      }
13      ajoute_camion();
14      ajoute_camion();
15      ajoute_camion();
16      ajoute_camion("affiche");
17  ?>

```

e. Les fonctions sur les chaînes de caractères

Fonction	Description	Code PHP	Résultat
addslashes()	Ajoute des anti-slashes devant les caractères spéciaux	\$res = addslashes ("L'a");	L\'a
stripslashes()	Retire les anti-slashes devant les caractères spéciaux.	\$res = stripslashes ("L'a");	L'a
dechex()	Retourne la valeur hexadécimale d'un nombre (ici 2548).	\$res = dechex ("2548");	9f4
ceil()	Retourne le nombre entier supérieur (utiliser floor() pour le nombre entier inférieur et round() pour le nombre entier le plus proche).	\$res = ceil ("12.1"); *	13
chunk_split()	Permet de scinder une chaîne en plusieurs morceaux.	\$res = chunk_split ("DGDFEF", "2", "-");	DG-DF-EF-
htmlentities()	Remplace les caractères par leur équivalent HTML (si ils existent).	\$res = htmlentities ("&");	&
strstr()	Recherche le premier caractère 'p' dans la chaîne et affiche le reste de la chaîne y compris le 'p'.	\$res = strstr ("webmaster@phpdebutant.org", "p");	phpdebutant.org
strlen()	Retourne la longueur de la chaîne	\$res = strlen ("lachainedecaracteres");	20
strtolower()	Passe tous les caractères en minuscules.	\$res = strtolower ("LA CHAINE dE caRActERes");	la chaine de caracteres
strtoupper()	Passe tous les caractères en MAJUSCULES.	\$res = strtoupper ("LA CHAINE dE caRActERes");	LA CHAINE DE CARACTERES
str_shuffle	mélanger les lettres	\$chaine = str_shuffle (' Cette chaîne va être mélangée !');	
str_replace()	Remplace un caractère par un autre dans une chaîne. Tiens compte de la casse.	\$res = str_replace ("a", "o", "LAlala");	LAlolo
trim()	Efface les espaces blancs (\n, \r, etc) au début et à la fin d'une chaîne (pas au milieu).	\$res = trim (" Salut le monde ");	Salut le monde
ucfirst()	Met la première lettre de chaque chaîne en Majuscule.	\$res = ucfirst ("salut le monde. ca va ?");	Salut le monde. ca va ?
ucwords()	Met la première lettre de chaque mot d'une chaîne en Majuscule.	\$res = ucwords ("salut le monde");	Salut Le Monde
strpos()	Recherche la position du premier caractère trouvé. Retourne le nombre de caractères placés avant lui (ici 4).	\$res = strpos ("abcdef", "e");	4
ereg()	Recherche si une chaîne de caractère est contenue dans une autre (ex. recherche si "ABCDE" contient "BCD").	if (ereg ("BCD", "ABCDE")) {echo "oui"; } else {echo "non"; }	oui
La virgule sous PHP est représentée par un point "." ainsi 12,1 s'écrit : 12.1 !			

f. Les fonctions d'heure et de date

PHP fournit un vaste éventail de fonctions liées à l'heure et la date.

Selon son paramétrage, la fonction date renvoie l'heure ou la date courantes dans beaucoup de formats différents.

Voici quelques-uns des paramètres les plus utiles :

- **date("Y")** Retourne l'année courante d'une date ;
- **date("m")** Retourne le mois courant d'une date ;
- **date("F")** Retourne le nom du mois courant d'une date ;
- **date("d")** Retourne le jour courant du mois d'une date ;
- **date("l")** Retourne le nom du jour de la semaine d'une date ;
- **date("w")** Retourne le jour de la semaine courant d'une date
- **date("H")** Retourne l'heure courante d'un temps ;
- **date("i")** Retourne les minutes courantes d'un temps ;
- **date("s")** Retourne les secondes courantes d'un temps ;

g. La fonction time()

Renvoie l'heure courante comme le nombre de secondes écoulées depuis le 1er janvier 1970 à 0:00 GMT.

8) Transmettre des données avec les formulaires

Les sites web interactifs ont besoin que les utilisateurs saisissent quelque chose. La méthode la plus courante pour recueillir ces entrées est d'utiliser un formulaire.

a. La balise FORM

Les formulaires sont délimités par la balise `<FORM> ... </FORM>`, une balise qui permet de regrouper plusieurs éléments de formulaire (boutons, champs de saisie,...) et qui possède les attributs obligatoires suivants :

- **ACTION** indique l'adresse d'envoi
- **METHOD** indique sous quelle forme seront envoyées les réponses :
 - ✓ « **POST** » est la valeur qui correspond à un envoi de données stockées dans le corps de la requête. avec "post" elles sont envoyées en bloc via le service d'entrée standard (STDIN) ,
 - ✓ « **GET** » correspond à un envoi des données codées dans l'URL, et séparées de l'adresse du script par un point d'interrogation.

b. Gestion de formulaire avec PHP : Les méthodes de transmissions

Il existe plusieurs moyens d'envoyer les données du formulaire. Vous pouvez en employer l'un des deux:

get : les données transiteront par l'URL comme on l'a appris précédemment. On pourra les récupérer grâce à l'array `$ _GET`. Cette méthode est assez peu utilisée car on ne peut pas envoyer beaucoup d'informations dans l'URL.

post : les données ne transiteront pas par l'URL, l'utilisateur ne les verra donc pas passer dans la barre d'adresse. Cette méthode permet d'envoyer autant de données que l'on veut, ce qui fait qu'on la privilégie le plus souvent.

i. Transmettre des données avec l'URL (GET)

URL:Uniform Resource Locator, sert à représenter une adresse sur le web; parfois les URL sont assez longues et comportent des caractères spéciaux.

Exemple:

<https://www.google.co.ma/search?q=php&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:fr:official&client=firefox-a&channel=fflb>

les informations après le point d'interrogation sont des données que l'on fait transiter d'une page à une autre.

ii. Récupérer les paramètres en PHP envoyés par un URL (\$ _GET).

Un tableau associatif des valeurs passées au script courant via les paramètres d'URL.

`$HTTP_GET_VARS` contient les mêmes informations, mais n'est pas superglobale. (Notez que `$HTTP_GET_VARS` et `$ _GET` sont des variables différentes et que PHP les traite comme telles.)

```
<a href="bonjour.php? nom=OTMANI;prenom=Amine">Dis-moi bonjour!</a>
```

c. Vérification des champs d'un formulaire: ISSET & EMPTY

i. isset

Détermine si une variable est définie et est différente de **NULL**. Si une variable a été détruite avec la fonction unset(), la fonction isset() renverra **FALSE**.

Isset() renverra **FALSE** lors du test d'une variable de valeur **NULL**. Notez aussi que le caractère nul ("0") n'est pas équivalent à la constante **PHP**.

```
<?php
    $var = "";
    if (isset($var))
    {
        echo 'Cette variable existe, donc je peux l'afficher.';
    }
?>
```

ii. unset

unset() détruit la ou les variables dont le nom a été passé en argument var. Le comportement de unset() à l'intérieur d'une fonction peut varier suivant le type de variable que vous voulez détruire.

Si une variable globale est détruite avec unset() depuis une fonction, seule la variable locale sera détruite. La variable globale gardera la valeur acquise avant l'appel à unset().

```
<?php
    function destroy_foo()
    {
        global $foo;
        unset($foo);
    }
    $foo = 'bar';
    destroy_foo();
    echo $foo;
?>
```

iii. empty

Détermine si une variable est considérée comme vide. Une variable est considérée comme vide si elle n'existe pas, ou si sa valeur équivaut à **FALSE**.

La fonction empty() ne génère pas d'alerte si la variable n'existe pas.

```
<?php
    $var = 0;
    // Évalué à vrai car $var est vide
    if (empty($var)) {
        echo 'Svar vaut soit 0, vide, ou pas définie du tout!';
    }
    // Évalué à vrai car $var est défini
    if (isset($var))
    {
        echo 'Svar est définie même si elle est vide';
    }
?>
```


9) les Sessions et les cookies

a. Header

Header() permet de spécifier l'en-tête HTTP string lors de l'envoi des fichiers HTML. N'oubliez jamais que **header()** doit être appelée avant que le moindre contenu ne soit envoyé, soit par des lignes html habituelles dans le fichier, soit par des affichages php.

Une erreur très classique est de lire un fichier avec include ou require, et de laisser des espaces ou des lignes vides, qui produiront un affichage avant que la fonction **header()** ne soit appelée.

```
1. <html>
2. <?php
3. /* Ceci produira une erreur. Notez la sortie ci-dessus,
4.  * qui se trouve avant l'appel à la fonction header() */
5. header('Location: http://www.example.com/');
6. exit;
7. ?>
```

b. Session : Principe

Un mécanisme permettant de mettre en relation les différentes requêtes du même client sur une période de temps donnée. Les sessions permettent de conserver des informations relatives à un utilisateur lors de son parcours sur un site web et des données spécifiques à un visiteur pourront être transmises de page en page afin d'adapter personnellement les réponses d'une application PHP. Chaque visiteur en se connectant à un site reçoit un numéro d'identification dénommé identifiant de session (SID)

La fonction `session_start()` se charge de générer automatiquement cet identifiant unique de session et de créer un répertoire. Elle doit être placée au début de chaque page afin de démarrer ou de continuer une session.

```
<?php
session_start();
$_SESSION_ID = session_id();
// $_SESSION_ID = 7edf48ca359ee24dbc5b3f6ed2557e90
?>
```

Un répertoire est créé sur le serveur à l'emplacement désigné par le fichier de configuration `php.ini`, afin de recueillir les données de la nouvelle session.

La session en cours peut être détruite par la fonction

`session_destroy()`.

Cette commande supprime toutes les informations relatives à l'utilisateur.

Une fois la variable enregistrée, elle est accessible à travers le tableau associatif

`$_SESSION["nom_variable"]`

Les fonctions de sessions

- **`session_start()`** : Initialise les données de session
- **`session_id()`** : Affecte et/ou retourne l'identifiant de session courante
- **`session_name()`** : Affecte et/ou retourne le nom de la session courante
- **`session_register()`** : Enregistre une variable dans la session courante

- **session_destroy()** : Détruit toutes les données enregistrées d'une session
- **session_is_registered()** : Indique si une variable a été enregistrée dans la session ou pas
- **session_unregister()** : Supprime une variable dans la session courante
- **session_unset()** : Détruit toutes les variables de session

Pour effacer toutes les variables de sessions, il suffit de faire `$_SESSION=array();`

- **session_cache_expire()** : Retourne la date d'expiration du cache de la session
- **session_save_path()** : Affecte et/ou retourne le chemin de sauvegarde de la session courante
- **session_decode()** : Décode les données de session à partir d'une chaîne
- **session_encode()** : Encode les données de session dans une chaîne

c. Les cookies

Principe

Un cookie est un fichier texte créé par un script et stocké sur l'ordinateur des visiteurs d'un site. Les cookies permettent de conserver des renseignements utiles sur chaque utilisateur, et de les réutiliser lors de sa prochaine visite.

Exemple : personnaliser la page d'accueil ou les autres pages du site

Les cookies étant stockés sur le poste client, l'identification est immédiate et ne concernent que les renseignements qui le concernent. Pour des raisons de sécurité, les cookies ne peuvent être lus que par des pages issues du serveur qui les a créés. Le nombre de cookies qui peuvent être définis sur le même poste client est limité à 20 et la taille de chacun est limitée à 4ko. Un navigateur peut stocker un maximum 300 cookies

La date d'expiration des cookies est définie de manière explicite par le serveur web chargé de les mettre en place.

Les cookies disponibles sont importés par PHP sous forme de variables identifiées sous les noms utilisés par ces cookies

La variable globale du serveur `$_COOKIES` enregistre tous les cookies qui ont été définis

i. Création des cookies

L'écriture de cookies est possible grâce à la fonction `setcookie()`. Il faut appeler cette fonction dès le début du script avant l'envoi d'aucune autre information de la part du serveur vers le poste client.

Setcookie("nom_var", "valeur_var", date_expiration, "chemin", "domain", "secure")

- **Nom_var** : nom de la variable qui va stocker l'information sur le poste client et qui sera utilisée pour récupérer cette information dans la page qui lira le cookie.
- *C'est la seule indication obligatoire pour un cookie*
- **Valeur_var** : valeur stockée dans la variable. Par exemple une chaîne de caractères ou une variable chaîne, en provenance d'un formulaire
- **Date_expiration** : la date à laquelle le cookie ne sera plus lisible et sera effacé du poste client

Chemin : définit la destination (incluant les sous-répertoire) à laquelle le navigateur doit envoyer le cookie.

Domain set : le nom du domaine à partir duquel peuvent être lus les cookies.

On peut aussi utiliser la variable d'environnement `$SERVER_NAME` à la place.

- **Secure** : un nombre qui vaut 0 si la connexion n'est pas sécurisée, sinon, il vaut 1 pour une connexion sécurisée

On utilise en général la date du jour, définie avec la fonction `time()` à laquelle on ajoute la durée de validité désirée

Si l'attribut n'est pas spécifié, le cookie expire à l'issue de la session.

```
<?php
    setcookie ("PremierCookie", "Salut", time() +3600*24*7) ;
    ...
    if (!$PremierCookie) {
        echo "le cookie n'a pas été défini";
    }
    else {
        echo $premierCookie, "<br>";
    }
?>
```

ii. Lecture de cookies

Les cookies sont accessibles dans le tableau associatif `$_COOKIE`

Si celui-ci visite une des pages PHP de ce même domaine dont le chemin est inclut dans le paramètre chemin (si le chemin est / le cookie est valide pour toutes les pages de ce site).

Il faut d'abord vérifier l'existence des variables dont les noms et les valeurs ont été définis lors de la création du cookie.

Cette vérification s'effectue grâce à la fonction `isset($_COOKIE["nom_var"])` qui renvoie true si la variable `$nom_var` existe et false sinon.

```
<?php
if (isset($_COOKIE["nom_var"])
{
    echo "<h2> Bonjour    $_COOKIE["nom_var"]</h2>" ;
}
else echo "pas de cookie" ;
?>
```

10) La gestion des fichiers avec PHP

Les fichiers sont des entités très importantes.

En effet, conserver des informations dans le but de les restituer, faire des statistiques, des fichiers d'accès, ... sont des éléments importants d'utilisation des fichiers.

Il existe une multitude de fonctions dédiées à l'utilisation des fichiers.

La communication entre le script PHP et le fichier est repérée par une variable, indiquant l'état du fichier et que l'on peut passer en paramètre aux fonctions spécialisées pour le manipuler.

a. file_exists

Cette fonction permet de déterminer si un fichier existe et est accessible.

Elle renvoie la valeur true si le fichier est présent.

```
$fichier="donnees.dat";  
if(file_exists($fichier))  
{  
    print " Le fichier $fichier existe";  
}  
else  
print "Le fichier $fichier n'existe pas sur la machine ";
```

b. is_file et is_dir

Vous permet de définir si l'élément sur lequel on travaille est un fichier.

La fonction *is_dir* permet de tester si c'est un répertoire. Ces fonctions renvoient true si c'est un fichier ou si c'est un répertoire.

```
If (is_file("fichier.data"))  
    print "l'élément est un fichier";  
else  
    print "L'élément n'est pas un fichier";
```

c. Ouvrir un fichier : la fonction fopen()

La fonction de base est la fonction *fopen()*. C'est elle qui permet d'ouvrir un fichier, que ce soit pour le lire, le créer, ou y écrire.

Syntaxe :

```
entier fopen (chaîne nomdufichier, chaîne mode);  
Ex:  
$fp = fopen("f.txt","r");
```

Le mode indique le type d'opération qu'il sera possible d'effectuer sur le fichier après ouverture. Il s'agit d'une lettre indiquant l'opération possible :

- ✓ **r** (comme *read*) indique une ouverture en lecture seulement

- ✓ **w** (comme *write*) indique une ouverture en écriture seulement (la fonction crée le fichier s'il n'existe pas)
- ✓ **a** (comme *append*) indique une ouverture en écriture seulement avec ajout du contenu à la fin du fichier (la fonction crée le fichier s'il n'existe pas).

Remarque :

Lorsque le mode est suivi du caractère + celui-ci peut être lu et écrit.

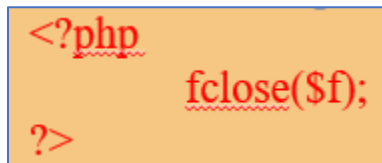
Le fait de faire suivre le mode par la lettre b entre crochets indique que le fichier est traité de façon binaire.

Modes d'ouverture :

- ✓ **'r'** Ouvre en lecture seule, et place le pointeur de fichier au début du fichier.
- ✓ **'r+'** Ouvre en lecture et écriture, et place le pointeur de fichier au début du fichier.
- ✓ **'w'** Ouvre en écriture seule ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
- ✓ **'w+'** Ouvre en lecture et écriture ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
- ✓ **'a'** Ouvre en écriture seule ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.
- ✓ **'a+'** Ouvre en lecture et écriture ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.
- ✓ **'x'** Crée et ouvre le fichier en lecture seule : Le pointeur est placé au début du fichier. Si le fichier existe déjà, fopen va échouer.
- ✓ **'x+'** Crée et ouvre le fichier en lecture/écriture : Le pointeur est placé au début du fichier. Si le fichier existe déjà, fopen va échouer.

d. Fermer un fichier : fclose()

La fonction fclose() est utilisé pour fermer un fichier.



```
<?php
    fclose($f);
?>
```

e. Lire une ligne du fichier texte : fgets

La fonction **fgets**, nous permet lire une ligne du fichier texte. Cette méthode effectue une lecture jusqu'au premier saut de ligne (celui-ci exclus). Elle retourne une chaîne de caractères contenant les *n* premiers caractères, moins 1 octet depuis le pointeur de fichier **\$fich**.

FALSE est retourné s'il n'y a plus de données à lire.

Syntaxe :

String fgets (source \$fich [, int \$length])

Exemple :

\$ligne = fgets(\$monfichier) ;

Remarque :

La fonction fgets() est utilisé pour lire les caractères dans un fichier

Écrire dans un fichier :

fwrite() alias **fputs()** permet d'écrire une chaîne de caractères dans le fichier :

Entier fputs(entier Etat_du_fichier, chaine Sortie) ;

fwrite() retourne le nombre d'octets écrits, ou **FALSE** si une erreur survient.

Exemple :

```
<?php
    fputs($monfichier, 'Texte à écrire');
?>
```

Pour stocker des informations dans le fichier, il faut dans un premier temps ouvrir le fichier en écriture en le créant s'il n'existe pas. Deux choix : le mode 'w' et le mode 'a'.

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

f. La fin d'un fichier :

La fonction **feof()** est utilisé pour déterminer si le pointeur est placé à la fin du fichier ou lister un fichier jusqu'au dernier caractère.

```
<?php
    If (feof($f))
        echo'Fin du fichier';
?>
```

g. fseek

Modifie le curseur de position dans le fichier. La nouvelle position, mesurée en octets, à partir du début du fichier, est obtenue en additionnant la distance à la position.

Retourne 0 en cas de succès, et sinon -1.

```
<?php
    $fp = fopen('somefile.txt', 'r');
    $data = fgets($fp, 4096);
    fseek($fp, 0);
?>
```

11) PHP et les bases de données

a. Introduction

MYSQL dérive directement de SQL (Structured Query Language) qui est un langage de requêtes vers les bases de données relationnelles.

Il en reprend la syntaxe mais n'en a pas toute la puissance.

Le serveur de base de données MySQL est très souvent utilisé avec le langage de création de pages web dynamiques : PHP.

Les données manipulées par le site seront stockées dans une base de données MySQL. Le script PHP, se servira de commandes MySQL pour gérer et extraire des informations de la base de données afin d'engendrer les pages HTML qui seront interprétées par le navigateur.

MySQL

Pour pouvoir utiliser un serveur MySQL, il faut qu'il soit lancé. Cela se fait dans notre cas en lançant EasyPHP qui n'a d'autre effet que de démarrer les serveurs Apache et MySQL.

On peut ensuite dialoguer avec le serveur Mysql, soit en mode commande (dans une fenêtre invite de commande), soit via l'interface PhpMyadmin.

PHPMyAdmin

Un serveur MySQL, une interface graphique de gestion des bases de données MySQL : PHPMyAdmin et le langage PHP.

EasyPHP offre un utilitaire : PHPMyAdmin qui donne une interface graphique pour Mysql.

i. Gérer la base de données MySQL avec PHP

Pour utiliser une base de données écrite en MySQL, il faut :

- Connexion au serveur
 - Création d'une ressource
 - Authentification
 - Sélection de la base de données
- **Utilisation des tables MySQL (via PHP)**
 - Lecture
 - Écriture
 - Modification
- **Déconnexion du serveur**

ii. API recommandée :

PHP offre 3 APIs différentes pour se connecter à MySQL.

Les APIs sont fournies par les extensions *mysql*, *MySQLi* et *PDO*.

Il est recommandé d'utiliser soit l'extension **MySQL**, soit l'extension **PDO MySQL**.

Il n'est pas recommandé d'utiliser l'ancienne extension *mysql* pour de nouveaux développements sachant qu'elle est obsolète depuis PHP 5.5.0, et sera supprimée dans un futur proche.

b. MySqli :

i. Connection à une BD.

```
1 <?php
2 $bdd = mysqli_connect('serveur', 'utilisateur', 'mot_de_passe', 'base');
3 ?>
```

- **\$bdd** correspond à une variable où seront stockées les informations de la base de données.
- **serveur** correspond au serveur SQL.
- **utilisateur** correspond au nom d'utilisateur pour se connecter au serveur SQL.
- **mot_de_passe** correspond au mot de passe pour le serveur SQL !
- **base** correspond à votre base de données du serveur SQL.

```
1 <?php
2 if($bdd = mysqli_connect('localhost', 'root', '', 'base'))
3 {
4
5 }
6 else
7 {
8     echo 'Erreur';
9 }
10 ?>
```

ii. Exécution des requêtes

Les requêtes peuvent être exécutées avec les fonctions **mysqli_query()**, **mysqli_real_query()** et **mysqli_multi_query()**.

La fonction **mysqli_query()** est la plus commune, et combine l'exécution de la requête avec une récupération de son jeu de résultats en mémoire tampon, s'il y en a un, en un seul appel.

Appeler la fonction **mysqli_query()** est identique à appeler la fonction **mysqli_real_query()** suivie d'un appel à la fonction **mysqli_store_result()**.

∞ La requête

Le code de la requête est le suivant :

```
1 <?php
2 $resultat = mysqli_query($bdd, 'requete');
3 ?>
```

- **\$resultat** est la variable où seront stockées les données.
- **\$bdd** est la variable retournée par la fonction de connexion à la base de données, .
- **'requete'** est une requête SQL.

Exemple :

\$resultat = mysqli_query(\$bdd, 'SELECT * FROM classe') ;

iii. Afficher le résultat d'une requête simple

mysqli_fetch_row() : tableau d'indices numériques (\$array[1])

mysqli_fetch_assoc() : tableau associatif (\$array['nom_du_champ'])

mysqli_fetch_array() : tableau d'indices numériques + tableau associatif

```
1 <?php
2 while($donnees = mysqli_fetch_assoc($resultat))
3 {
4     echo $donnees['id'];
5     echo "\n";
6     echo $donnees['pseudo'];
7 }
8 ?>
```

\$donnees est transformé en un tableau avec les noms des colonnes demandées dans la requête. Cette transformation est réalisée grâce à la fonction *mysqli_fetch_assoc(\$resultat)*, *\$resultat* étant le résultat brut de la requête. Cette fonction sert aussi et surtout à passer au résultat suivant, d'où l'utilisation de la boucle *while*.

Il ne faut pas oublier de fermer le curseur avec la fonction suivante :

```
<?php
mysqli_free_result($resultat);
?>
```

iv. Les requêtes préparées

La base de données MySQL supporte les requêtes préparées. Une requête préparée ou requête paramétrable est utilisée pour exécuter la même requête plusieurs fois, avec une grande efficacité. L'exécution d'une requête préparée se déroule en deux étapes :

- La préparation et l'exécution. Lors de la préparation, un template de requête est envoyé au serveur de base de données.
- Le serveur effectue une vérification de la syntaxe, et initialise les ressources internes du serveur pour une utilisation ultérieure.

Le serveur MySQL supporte le mode anonyme, avec des marqueurs de position utilisant le caractère *?* Pour travailler avec les requêtes préparées, il y a un certain ordre à suivre :

- Préparer la requête ;
- Lier les variables à la requête ;
- Exécuter la requête ;
- Si la requête renvoie un résultat, on continue :
 - Lier le résultat à des variables ;
- *Fetcher* le résultat.

✎ Préparer la requête

On prépare la requête avec cette fonction :

```
$req_pre = mysqli_prepare($bdd, 'SELECT * FROM membres WHERE id = ?');
```

∞ Lier les variables à la requête préparée

On les lie avec cette fonction

```
mysqli_stmt_bind_param($req_pre, "i", $id);
```

i est le type de la variable qui suit.

La variable correspond à ce qui va remplacer le point d'interrogation dans la requête.

Lettre	Correspond à
<i>i</i>	un nombre entier
<i>d</i>	un nombre décimal
<i>s</i>	une chaîne de caractères

∞ Exécuter la requête

```
mysqli_stmt_execute($req_pre);
```

Si la requête ne renvoie pas de résultat (les requêtes d'insertion, de mise à jour, de suppression), on s'arrête là. Sinon, il faut lier le résultat de l'exécution à des variables.

Exemple avec une requête qui ne renvoie pas de résultat

```
<?php
// include de la connexion SQL.
$req_pre = mysqli_prepare($bdd, 'INSERT INTO classe (id_classe, nom_classe) VALUES (?, ?)');
mysqli_stmt_bind_param($req_pre, "is", $id_classe, $nom_classe);
mysqli_stmt_execute($req_pre);
?>
```

Exemple avec une requête qui renvoie des résultats

```
<?php
// include de la connexion SQL.
$req_pre = mysqli_prepare($bdd, 'SELECT * FROM classe WHERE id_classe = ? ');
mysqli_stmt_bind_param($req_pre, "i", $id_c, $nom_c);
mysqli_stmt_execute($req_pre);
mysqli_stmt_bind_result($req_pre, $donnees['id_classe'], $donnees['nom_classe']);
while(mysqli_stmt_fetch($req_pre))
{
    echo $donnees['id_classe'] . " , " . $donnees['nom_classe'];
}
?>
```

c. Les exceptions

PHP a une gestion des exceptions similaire à ce qu'offrent les autres langages de programmation. Une exception peut être lancée ("*throw*") et attrapée ("*catch*") dans PHP.

Le code devra être entouré d'un bloc *try* pour faciliter la saisie d'une exception potentielle. Chaque *try* doit avoir au moins un bloc *catch* ou *finally* correspondant.

L'objet lancé doit être une instance de la classe Exception ou une sous-classe de la classe Exception. Tenter de lancer un objet qui ne correspond pas à cela résultera en une erreur fatale émise par PHP.

i. catch

L'exécution normale (lorsqu'aucune exception n'est lancée dans le bloc try) continue après le dernier bloc catch défini dans la séquence. Les exceptions peuvent être lancées (ou relancées) dans un bloc catch.

Lorsqu'une exception est lancée, le code suivant le traitement ne sera pas exécuté et PHP tentera de trouver le premier bloc catch correspondant.

Si une exception n'est pas attrapée, une erreur fatale issue de PHP sera envoyée avec un message "Uncaught Exception ..." indiquant que l'exception n'a pu être attrapée à moins qu'un gestionnaire d'exceptions ne soit défini avec la fonction set_exception_handler().

ii. finally

En PHP 5.5 et suivant, un bloc finally peut aussi être spécifié après des blocs catch. Le code à l'intérieur du bloc finally sera toujours exécuté après les blocs try et catch, indépendamment du fait qu'une exception a été lancée, avant de continuer l'exécution normale.

Exemple :

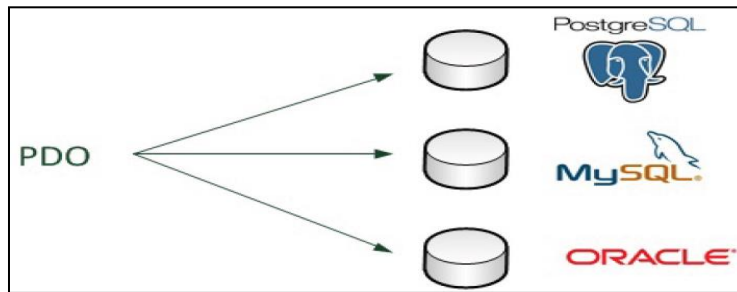
```
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Division par zéro.');
```

d. Base de données et PHP : PDO

L'extension *PHP Data Objects* (PDO) définit une excellente interface pour accéder à une base de données depuis PHP. Chaque pilote de base de données implémenté dans l'interface PDO peut utiliser des fonctionnalités spécifiques de chacune des bases de données en utilisant des extensions de fonctions. PDO est ce qu'on appelle une extension **orientée objet**.

Notez que vous ne pouvez exécuter aucune fonction de base de données en utilisant l'extension PDO par elle-même ; vous devez utiliser un **driver PDO spécifique à la base de données** pour accéder au serveur de base de données.

Les fonctions **mysql_** ne sont plus à utiliser (on dit qu'elles sont « obsolètes »). Il reste à choisir entre **MySQL_** et **PDO**.



PDO permet de se connecter à n'importe quel type de base de données

i. Se connecter à MySQL avec PDO

- **Le nom de l'hôte** : c'est l'adresse de l'ordinateur où MySQL est installé (comme une adresse IP). Le plus souvent
- **La base** : c'est le nom de la base de données à laquelle vous voulez vous connecter.
- **Le login** : il permet de vous identifier.
- **Le mot de passe** : pour confirmer l'accès et gérer les droits d'accès

```

1 <?php
2 $bdd = new PDO('mysql:host=localhost;dbname=mabase', 'root', '');
3 ?>

```

Tester la présence d'erreurs

En cas d'erreur, PDO renvoie une **exception** qui permet de **capturer** l'erreur.

```

1 <?php
2 try
3 {
4     $bdd = new PDO('mysql:host=localhost;dbname=test', 'root', '');
5 }
6 catch (Exception $e)
7 {
8     die('Erreur : ' . $e->getMessage());
9 }
10 ?>

```

ii. Récupérer les données : créer une requête

Pour récupérer des informations de la base de données, nous avons besoin de notre objet \$bdd représentant la connexion à la base.

Code : PHP

\$reponse = \$bdd->query('Tapez votre requête SQL ici');

\$reponse contient quelque chose d'inexploitable. MySQL nous renvoie beaucoup d'informations qu'il faut organiser. Pour récupérer une entrée, on prend la réponse de MySQL et on y exécute fetch(), ce qui nous renvoie la première ligne. Il faut faire une boucle pour parcourir les entrées une à une.

\$donnees = \$reponse->fetch()

Le fetch renvoie faux (false) dans \$donnees lorsqu'il est arrivé à la fin des données, c'est-à-dire que toutes les entrées ont été passées en revue.

Dans ce cas, la condition du **while** vaut faux et la boucle s'arrête.

```
while ($donnees = $reponse->fetch())  
{  
    .....;  
}
```

Code PHP arrêt et fin du traitement :

```
<?php  
    $reponse->closeCursor();  
?>
```