Serrhini Mohammed
Faculté des Science Oujda
Département d'informatique

Licence SMI S6

Technologie du Web Avancée
Première Partie
Langage PHP-MYSQL

Polycopie version 2021-2022

Le polycopie suivant se trouve sur le site web :
www.emena.org/SMIS6

SOMMAIRE

1. Les cookies
2. Les sessions

VI PHP - Qu'est-ce que la POO?

1. PHP - Que sont les classes et les objets?
2. PHP POO - Classes et objets
3. PHP - Le mot-clé $ this, instanceof
4. PHP - La fonction __construct Destructor, Access Modifiers
5. PHP OOP – Héritage, Constantes de classe, Abstract Classes
6. PHP POO -
7. les interfaces, Traits, Static Methods Static Properties
8. Utilisation des espaces de noms Namespaces
9. PHP Iterables

VIII- Utilisation d'une base de données MySQL

1- Présentation
2- Qu'est-ce que MySQL? Accéder à MYSQL via PHP
3- Requêtes de base de données
4- Connexion PHP à MySQL
5- PHP Creation Database dans MySQL
6- PHP MySQL Création Table
7- PHP MySQL Insertion Data
8- La sécurité
9- Déclarations préparées en PDO
10- Sélection de Données PHP MySQL
11- PHP MySQL Supprimer les données
12- Mise à jour des Données de PHP MySQL
13- Limite Sélections de données de PHP MySQL

IX- L'évolution de PHP en PHP7

X- Webographie

I-Introduction

1- Qu'est-ce que PHP ?

PHP (officiellement "PHP: HyperText Preprocessor") est un langage de script qui est principalement utilisé pour être exécuté par un serveur HTTP, mais il peut fonctionner comme n'importe quel langage interprété en utilisant les scripts et son interpréteur sur un ordinateur. On désigne parfois PHP comme une plate-forme plus qu'un simple langage.

Sa syntaxe et sa construction ressemblent à celles des langages C++ et Perl, à la différence que le PHP peut être directement intégré dans du code HTML.

Exemple

```
<html>
<head>
<title>Exemple</title>
</head>
<body> <? echo "Bonjour, je suis un script PHP!";
?>
</body>
</html>
```

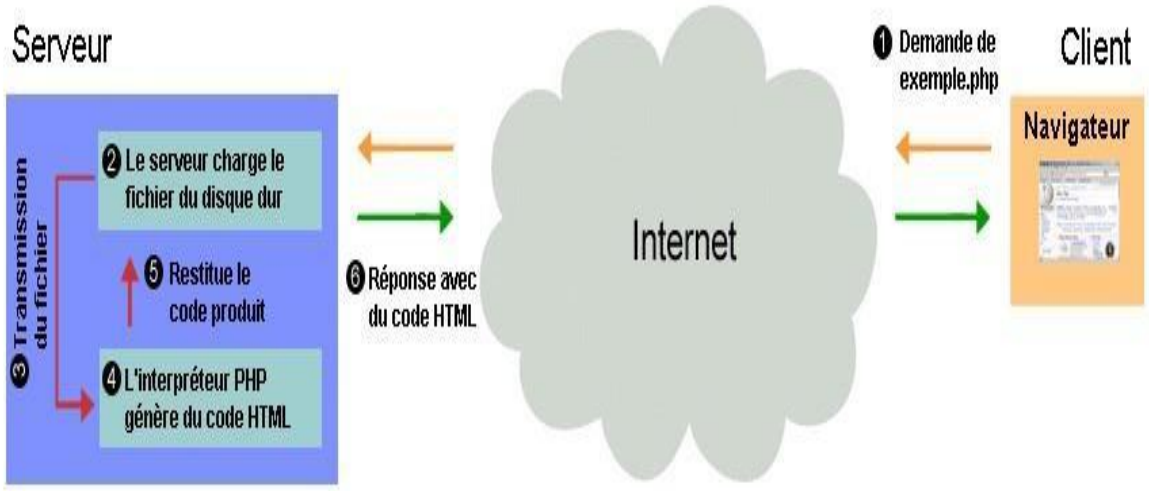
L'objet de ce langage est de permettre aux développeurs web d'écrire des pages dynamiques rapidement.

2- Fonctionnement :

PHP n'est pas un langage compilé, c'est un langage interprété par le serveur : le serveur lit le code PHP, le transforme et génère la page HTML. Pour fonctionner, il a donc besoin d'un serveur web. De ce fait une plateforme minimale de base pour l'exécution d'un site web développé en PHP comprend:

- l'interpréteur PHP (serveur PHP)
- un serveur web (Apache, IIS, …)

Lorsqu'un visiteur demande à consulter une page Web, son navigateur envoie une requête à un serveur HTTP. Si la page contient du code PHP, l'interpréteur PHP du serveur le traite et renvoie du code généré (HTML).



De ce fait le code PHP n'est jamais visible sur la page finale consultée par le client. Ainsi en éditant le source de la page on n'y trouvera que du code HTML.

3- Langages concurrents pour la génération des sites dynamiques.

JSP, ASP, PYTHON, Perl, coldfusion, CGI

4- Avantages, limitations.

Le langage PHP possède les même fonctionnalités que les autres langages permettant d'écrire des scripts CGI, comme collecter des données, générer dynamiquement des pages web ou bien envoyer et recevoir des cookies, … .

La plus grande qualité et le plus important avantage du langage PHP est le support d'un grand nombre de bases de données et la simplicité d'interfaçage avec eux.

PHP est utilisable sur la majorité des systèmes d'exploitation, comme Linux, de nombreuses variantes Unix (incluant HP-UX, Solaris et OpenBSD), Microsoft Windows, Mac OS X, RISC OS et d'autres encore.

PHP supporte aussi la plupart des serveurs web actuels : Apache, Microsoft Internet Information Server, Personal Web Server, Netscape et iPlanet servers, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd et beaucoup d'autres encore.

La gratuité et la disponibilité du code source (PHP est distribué sous licence GNU GPL) - La simplicité d'écriture de scripts (apprentissage rapide);

Limitations :

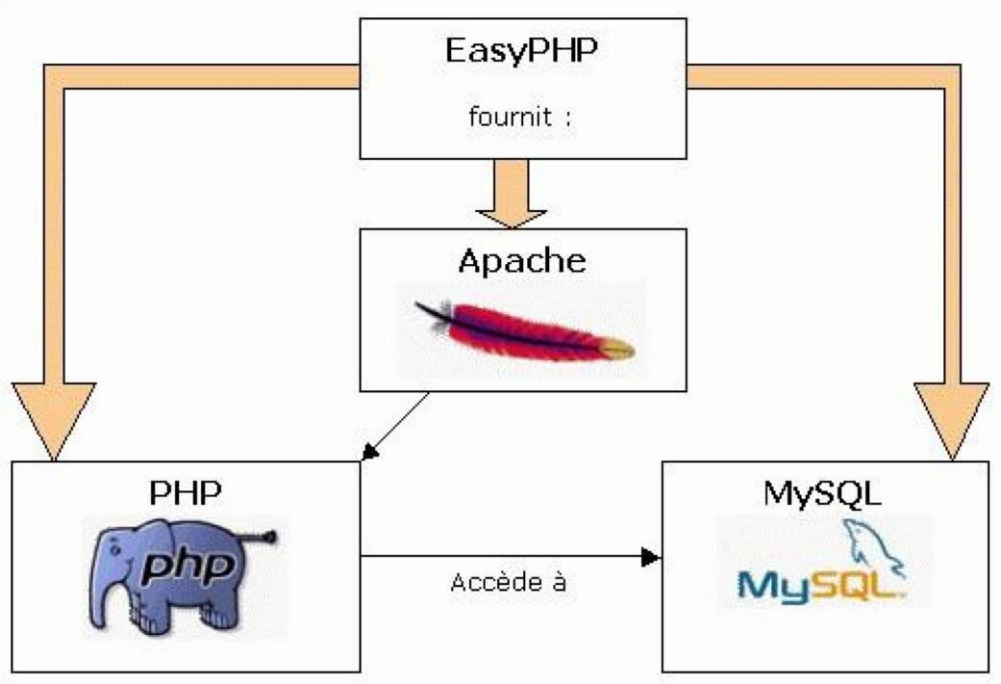
- Langage interprété
- L'orienté objet reste limité
- Pas adéquat en termes de rapidité et de maintenabilité pour des projets de grandes envergures

II-Mise en œuvre et déploiement

1- EasyPHP

EasyPHP est un paquetage contenant à la fois Apache, PHP et MySQL

EasyPHP permet d'installer automatiquement et facilement une plateforme permettant l'exploitation d'un site web en PHP qui éventuellement aurait besoin d'un accès a une base de données. De la sorte on se libère des complications d'une installation manuelle de chacune de composantes séparément.



Procédure d'installation :

- Télécharger EasyPHP sur le site www.easyphp.org
- Double cliquer sur l'exécutable téléchargé
- Sélectionner le répertoire d'installation et suivre la procédure

Procédure de lancement :

Lorsqu'EasyPHP est lancé, les serveurs Apache et MySQL sont automatiquement lancés (il est même possible de le faire automatiquement au démarrage de Windows). Une petite icône s'installe dans la barre des tâches,

à côté de l'horloge, permettant un accès rapide aux fonctions proposées par EasyPHP :

- Fichier Log : renvoie aux erreurs générées par Apache et MySQL
- Configuration : donne accès aux différentes configurations d'EasyPHP
- Administration : gestion des répertoires virtuels, des extensions, de PHPMyadmin
- Web local : ouvre la page http://localhost/ répertoire racine du site
- Démarrer/Arrêter : démarre/arrête Apache et MySQL
- Quitter : ferme EasyPHP

Pour que vos pages PHP soient interprétées, il est impératif de placer vos fichiers dans le répertoire www (sus le répertoire d'installation de easyphp). Ce répertoire est configuré par défaut comme répertoire racine de votre serveur web. Le serveur Apache est configuré pour ouvrir automatiquement un fichier index lorsque vous saisissez l'adresse 'http://localhost/' (à condition évidemment que le serveur Apache soit en route). Cette page sert de page d'accueil au web local et permet de vérifier le bon fonctionnement d'EasyPHP.

Il est conseillé de créer un répertoire par projet dans le répertoire www afin d'avoir une vision plus claire des développements.

Il est en outre possible de créer des répertoires virtuels. Ces répertoires virtuels peuvent être créé physiquement n'importe ou sur votre disque dur. Il suffira de leur associer un nom virtuel avec lequel le serveur apache le reconnaîtra.

2- Configuration et paramétrage (php.ini)

Lors de l'installation de PHP, un fichier de configuration nommé « php.ini » sera créé sur votre système. Le fichier php.ini est le fichier de configuration de serveur PHP. Ce fichier respecte la structure des fichiers INI bien connus de nombreuses applications Windows. Il s'agit d'un fichier texte ASCII divisé en plusieurs sections, chacune portant un nom et contenant des variables relatives à la section concernée.

Chaque section ressemble au fragment suivant :

```
[MaSection]
variable="valeur"
autrevariable="autrevaleur"
```

Le nom de la section figure au début entre crochets, suivi d'un certain nombre de paires nom/valeur, chaque paire figurant sur une ligne distincte. Comme pour tout code PHP, les noms des variables font la distinction entre majuscules et minuscules et ne peuvent pas contenir d'espace. Les valeurs peuvent quant à elles être numériques, booléennes ou correspondre à une chaîne.

Les lignes en commentaires commencent par un « ; »

Toute modification du fichier php.ini nécessite un redémarrage du serveur afin que la nouvelle configuration soit prise en compte.

Par défaut le fichier php.ini est configuré de sorte à être fonctionnel. Intervenir sur ce fichier peut être intéressant pour rajouter de nouveaux modules ou pour affiner certains paramètres de sécurité mais doit systématiquement faire l'objet d'un backup de sécurité au préalable.

Quelques exemples de paramétrage :

-Gérer l'affichage des messages d'erreur

Les erreurs PHP se répartissent en quatre catégories: erreurs du parseur, notifications de bugs dans le code (par exemple, des variables non initialisées), avertissements (erreurs non fatales) et erreurs fatales. Normalement, lorsque PHP détecte une erreur de parseur, non fatale ou fatale, il affiche l'erreur et, si elle est fatale, il arrête également de traiter le script à ce stade. Vous pouvez modifier ce comportement à l'aide de la variable error_reporting, qui accepte un champ binaire des codes d'erreurs et n'affiche que les erreurs correspondant à ces codes:

La configuration suivante par exemple permet l'affichage de tout type d'erreur.

error_reporting = E_ALL

Pour désactiver l'affichage des erreurs (recommandé en production), désactivez la variable display_errors et inscrivez plutôt les messages dans un journal des erreurs à l'aide de la variable log_errors.

Cette technique est également efficace pour des raisons de sécurité: en désactivant les erreurs, vous masquez des informations spécifiques du système que des utilisateurs peu scrupuleux pourraient exploiter pour essayer de nuire à votre site ou votre application. Définissez donc l'écriture de ces erreurs dans un fichier journal personnalisé ou dans le journal système, en paramétrant la variable error_log sur le nom d'un fichier .

display_errors = Off log_errors = On error_log = "error.log"

- **file_uploads**= On/Off permet d'autoriser ou non l'envoi de fichiers.

- **upload_tmp_dir** = répertoire permet de définir le répertoire temporaire permettant d'accueillir le fichier uploadé.

- **upload_max_filesize** = 2M permet de définir la taille maximale autorisée pour le fichier. Si cette limite est dépassée, le serveur enverra un code d'erreur.

- **post_max_size** indique la taille maximale des données envoyées par un formulaire. Cette directive prime sur upload_max_filesize, il faut donc s'assurer d'avoir post_max_size supérieure à upload_max_filesize.

- **register_globals** = Off [sécurité, performance]

Depuis la version 4.2.0 de PHP, la valeur par défaut de register_global est à Off dans le php.ini. Dorénavant une variable envoyée par un formulaire (méthode POST) n'est plus récupérée avec $variable mais avec $_POST["variable"]. Toutes les variables globales sont concernées (POST, GET, cookies, environnement et autres variables serveur : $_GET, $_POST, $_COOKIE, $_SERVER, $_ENV, $_REQUEST, $_SESSION). Ceci peut nécessiter la réécriture partielle de certains scripts.

Rq : il est vivement conseillé d'utiliser cette configuration qui est celle adoptée par défaut depuis PHP 4.2.0 et de coder vos scripts en conséquence. Cependant si vous souhaitez utiliser d'anciens scripts sans avoir à les réécrire, vous avez toujours la possibilité de remettre dans le fichier php.ini register_global à On.

3- Présentation des options les plus courantes.

Le Menu de EASYPHP offre les fonctionnalités suivantes :

- **Fichier Log**

Ce menu permet de visualiser l'ensemble des fichier logs que Apache et MYSQL génèrent. PHP ne générant pas de fichier log puisqu'il ne fonctionne qu'au travers d'Apache.

Configuration

Ce menu donne accès aux fichiers de configurations suivants :

-Apache : httpd.conf
-Extension PHP : php.ini
-PHP : php.ini
-MYSQL : mysql.ini
-PHPMyadmin

Administration

-Ce menu va permettre entre autre de créer des répertoires virtuels pour les sites web que vous allez créer.

-Il permettra en outre d'accéder à PHPMyadmin afin de gérer vos bases de données.

- **Web local**
- **Démarrer/Arrêter**
- **Quitter**

III-Les fonctionnalités du langage

1- Premiers éléments du langage.

- **Que peut faire PHP ?**

  PHP peut générer du contenu de page dynamique
  PHP peut créer, ouvrir, lire, écrire, supprimer et fermer des fichiers sur le serveur
  PHP peut collecter des données de formulaire
  PHP peut envoyer et recevoir des cookies
  PHP peut ajouter, supprimer, modifier des données dans votre base de données
  PHP peut être utilisé pour contrôler l'accès des utilisateurs
  PHP peut chiffrer les données

  Avec PHP, vous n'êtes pas limité à la sortie HTML. Vous pouvez produire des images, des fichiers PDF et même des films Flash. Vous pouvez également produire n'importe quel texte, tel que XHTML et XML.
  Quoi de neuf dans PHP 7

  PHP 7 est beaucoup plus rapide que la précédente version stable populaire (PHP 5.6)
  PHP 7 a amélioré la gestion des erreurs
  PHP 7 prend en charge des déclarations de type plus strictes pour les arguments de fonction
  PHP 7 prend en charge de nouveaux opérateurs (comme l'opérateur de vaisseau spatial : <=>)
  Sensibilité à la casse PHP

  En PHP, les mots-clés (par exemple if, else, while, echo, etc.), les classes, les fonctions et les fonctions définies par l'utilisateur ne sont pas sensibles à la casse.

  Dans l'exemple ci-dessous, les trois instructions d'écho ci-dessous sont égales et légales :

```php
<?php
ECHO "Bonjour<br>";
echo "HELLO!<br>";
EcHo "Salam<br>";
?>
```

- Exemple:

```php
<?php
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>
```

Regardez l'exemple ci-dessus; seule la première instruction affichera la valeur de la variable $color ! En effet, $color, $COLOR et $coLOR sont traités comme trois variables différentes :

Lorsque vous créez un code, il doit être placé entre balises php pour que celui- ci soit interprété, comme ceci:

<? *echo 'bonjour';* ***?>***

ou encore

<?PHP *echo 'bonjour';* ***?>***
Ce qui affichera à l'écran ' bonjour '.

Une syntaxe PHP se termine TOUJOURS par un point-virgule, si vous l'oubliez, vous verrez apparaître une PARSE ERROR lors de l'exécution de votre fichier.

Les commentaires peuvent se présenter sous deux formes :

```
<?
//ceci est // un
commentaire /*  ceci est un
commentaire */
?>
```

Utiliser des commentaires pour omettre des parties du code :

```
<?php
// Vous pouvez également utiliser des commentaires pour omettre des parties d'une ligne de code
$x = 5 /* + 15 */ + 5;
echo $x;
?>
```

Écho PHP et déclarations d'affichage et d'impression

2- Intégration de PHP dans une page HTML.

a Généralités

L'un des avantages du PHP est qu'il s'intègre facilement dans du code HTML classique. Chacun peut à sa guise inclure quelques parties en PHP dans des parties de code HTML.

Remarque importante :

Du moment que des parties de code PHP ont été intégrées dans une page HTML cette dernière doit impérativement être renommée en extension **.php**

Exemple

```
<html>
<body>

<font size="2" face="Arial">Le texte en HTML</font> <br>

<?  echo "<font size=\"2\" face=\"Arial\"> Le texte est en  PHP. </font>";
?>

<br>
<font size="2" face="Arial"> < ? echo 'Encore du texte en PHP' ?></font>

</body>
</html>
```

Le résultat obtenu sera :

Le texte est en HTML
Le texte est en PHP
Encore du texte en PHP

a

Les fonctions de sortie

Avec PHP, il existe deux méthodes de base pour obtenir une sortie : echo et print.
echo et print sont plus ou moins les mêmes. Ils sont tous deux utilisés pour afficher des données à l'écran.

Les différences sont minimes : echo n'a pas de valeur de retour tandis que print a une valeur de retour de 1, il peut donc être utilisé dans des expressions. echo peut prendre plusieurs paramètres (bien qu'une telle utilisation soit rare) tandis que print peut prendre un argument. echo est légèrement plus rapide que print.
L'instruction echo peut être utilisée avec ou sans parenthèses : echo ou echo().
L'instruction print peut être utilisée avec ou sans parenthèses : print ou print().

```php
<?php
echo "<h2>PHP</h2>";
echo "Pour SMIS6<br>";
echo "Attaalamo PHP!<br>";
echo "cette ", "chaine
caractere ", "est ", "faite ",
"avec plusieurs paramètres.";
?>
```

```php
<?php
$txt1 = "PHP";
$txt2 = "pour SMIS6";
$x = 5;
$y = 4;
echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>
```

<?php print " PHP is Fun! "; print "Hello world! "; print "I'm about to learn PHP!"; ?>	<?php $txt1 = "Apprendre PHP"; $txt2 = "avec Pr Serrhini"; $x = 5; $y = 4; print " " . $txt1 . " "; print "apprendre PHP avec " . $txt2 . " "; print $x + $y; ?>

b La fonction Require

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

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

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

Exemple :

include('fichier.php");

Cependant, il y a une grande différence entre inclure et require;

Lorsqu'un fichier est inclus avec l'instruction include et que PHP ne le trouve pas, le script continue à s'exécuter:

Les instructions include et require sont identiques, sauf en cas d'échec:

require produira une erreur fatale (E_COMPILE_ERROR) et arrêtera le script include produira seulement un avertissement (E_WARNING) et le script continuera

c La fonction include()

La fonction **include()** de PHP permet d'inclure un fichier dans un autre lors de son exécution. Un élément HTML répétitif qui apparaît dans toutes les pages de votre site (menu par exemple) pourra être isolé dans un fichier PHP. Un appel de ce fichier grâce à la fonction **include()** apparaîtra dans toutes les pages de votre site. Ainsi si le menu doit être par exemple modifié il suffira uniquement de changer le fichier contenant le menu.

Syntaxe

```
<?
Include ("nom_du_fichier_a_inclure") ;  ?>
```

Dans l'exemple suivant un fichier menu.php contiendra le code HTML nécessaire à la génération d'un menu en HTML.
Un second fichier page.php inclura ce fichier menu.php.

Exemple fichier menu.php

```
<a href= "menu1.php" > Menu1 </a><br>
<a href="menu2.php" > Menu2 </a><br>
<a href="menu3.php" > Menu3 </a><br>
```

Fichier page.php

```
<html> <body>
<?
Include ("menu.php") ;  / /inclusion du fichier contenant le menu
?>
</body>
</html>
```

Le résultat obtenu sera :

Menu1

Menu2

Menu3

Remarque : Il ne faut exécuter que le fichier page.php.

Ainsi toutes les pages du site sensées contenir ce menu intègreront grâce à la fonction include() le fichier menu.php.

Il suffira de changer le fichier menu.php afin que toutes les pages qui l'intègrent se trouvent automatiquement changées.

3- Variables, chaînes et concaténation

Une variable est définie sous la forme $variable_nom.

L'affectation d'une variable se fait de la manière suivante :

$variable_nom = variable_valeur.

L'appel de la valeur affectée à une variable se fait par son nom.

Syntaxe

```
<?
$variable1 = 'Bonjour 1';
```

```
//Affectation d'une chaîne avec quote simple
$variable2 = ”Bonjour 2”;
// Affectation d'une chaîne avec quote double
// Affectation d'un entier
$variable3 = 5;
 // Affectation d'un résultat d'opération
$variable4 = 2 + (3 * 5);
 // Affectation Booléenne
$variable5 = true; ?>
```

L'affichage des variables combinées à des chaînes de caractères peut se faire de plusieurs manières en utilisant les cotes simples (') ou les doubles cotes (").

```
<?
$nom = 'visiteur';
echo "bonjour $nom";
?>
```

Le résultat obtenu sera :

bonjour visiteur

Remarque : Il y'a un. Entre le ' et la variable $nom. Le point est un opérateur de concaténation pour les chaînes de caractères.

```
<?
$nom = 'visiteur';
echo 'bonjour $nom';
// La variable $nom ne sera pas interprétée
?>
```

Le résultat obtenu sera : bonjour nom

Remarques :

Si vous utilisez les ' au lieu des " , la variable n'est pas interprétée comme une variable mais comme une chaîne de caractère.

Remarque sur l'usage des cotes simples ou doubles

Lors de l'usage des ' pour l'affichage d'une chaîne de caractère contenant des apostrophes, vous devez impérativement faire précéder ces apostrophes

d'antislash. De la sorte l'apostrophe ne sera pas confondue avec le caractère de fin de la chaîne à afficher. Dans le cas contraire un message d'erreur sera affiché.

```
<?  echo 'vous n\'êtes pas inscrit';
?>
```
Le résultat obtenu sera : Vous n'êtes pas inscrit

Il en va de même lors de l'usage de '' pour l'affichage d'une chaîne de caractère contenant au préalable des doubles cotes.

```
<?
echo"<ahref=\"http://wwwetu\">lien.php</a>";
?>
```
Le résultat obtenu sera :
Lien.php

PHP est un langage faiblement typé

Dans l'exemple ci-dessus, notez que nous n'avons pas eu à indiquer à PHP le type de données de la variable.

PHP associe automatiquement un type de données à la variable, en fonction de sa valeur. Étant donné que les types de données ne sont pas définis au sens strict, vous pouvez par exemple ajouter une chaîne à un entier sans provoquer d'erreur.

En PHP 7, les déclarations de type ont été ajoutées. Cela donne une option pour spécifier le type de données attendu lors de la déclaration d'une fonction, et en activant l'exigence stricte, il lancera une "erreur fatale" sur une incompatibilité de type.

Portée des variables PHP

En PHP, les variables peuvent être déclarées n'importe où dans le script.
La portée d'une variable est la partie du script où la variable peut être référencée/utilisée.
PHP a trois portées de variables différentes :

- locale
- globale
- statique

Une variable déclarée en dehors d'une fonction a une SCOPE GLOBALE et n'est accessible qu'en dehors d'une fonction

```
<?php
$x = 5; // var globale scope
function myTest() {
 // l'utilisation de x dans cette fonction générera une erreur
```

```
  echo "<p> La variable x à l'intérieur de la fonction est : $x</p>";
}
myTest();
echo "<p> La variable x fonction extérieure est: $x</p>";
?>
```

Une variable déclarée dans une fonction a une PORTÉE LOCALE et n'est accessible qu'au sein de cette fonction

```
<?php
function myTest() {
  $x = 5; // locale scope
  echo "<p> La variable x à l'intérieur de la fonction est: $x</p>";
}
myTest();
// utiliser x en dehors de la fonction générera une erreur
echo "<p> La variable x fonction extérieure est: $x</p>";
?>
```

Vous pouvez avoir des variables locales portant le même nom dans différentes fonctions, car les variables locales ne sont reconnues que par la fonction dans laquelle elles sont déclarées. Le mot-clé global est utilisé pour accéder à une variable globale à partir d'une fonction.

```
<?php
$x = 5;
$y = 10;
function myTest() {
  global $x, $y;
  $y = $x + $y;
}
myTest();
echo $y; // affiche 15
?>
```

PHP stocke également toutes les variables globales dans un tableau appelé $GLOBALS[index]. L'index contient le nom de la variable. Ce tableau est également accessible depuis les fonctions et peut être utilisé pour mettre à jour directement les variables globales.

```
<?php
$x = 5;
$y = 10;
function myTest() {
  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}
myTest();
echo $y; //affiche 15
?>
```

Normalement, lorsqu'une fonction est terminée/exécutée, toutes ses variables sont supprimées. Cependant, nous voulons parfois qu'une variable locale ne soit PAS supprimée. Nous en avons besoin pour un autre travail.
Pour cela, utilisez le mot-clé static lors de la première déclaration de la variable :

```
<?php
function myTest() {
  static $x = 0;
  echo $x;
  $x++;
}
myTest();
myTest();
myTest();
?>
```

Valeur NULL PHP

Null est un type de données spécial qui ne peut avoir qu'une seule valeur : NULL.
Une variable de type de données NULL est une variable à laquelle aucune valeur n'est affectée.
Conseil : Si une variable est créée sans valeur, la valeur NULL lui est automatiquement affectée.
Les variables peuvent également être vidées en définissant la valeur sur NULL :

```
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
```

4- Conditions et boucles.

a- Conditions if

La syntaxe de base d'une instruction conditionnelle est la suivante :

```
<?  if($var ==
'condition')
{
// 'condition vérifiée';
} else
{
//'condition non vérifiée';
}
?>
```

==	strictement égal
!=	différent
>	plus grand que
<	inférieur à
>=	supérieur à
<=	inférieur à
&&	et
||	ou
AND	et
OR	ou
TRUE	1 ou oui
FALSE	0 ou non

Exemple en utilisant la fonction Date

```
<?php
$t = date("H");

if ($t < "20") {
  echo "Bonjour";
} else {
  echo "Bonsoir";
}
?>
```

Les opérateurs de contrôle sont les suivants :

Exemple

```
<?
$montant='100';
if
($montant <1000)
{  echo 'Montant inférieur à
1000';
}
else {
echo 'Montant supérieur à 1000';
}
?>
```

Le résultat obtenu sera : Montant inférieur à 1000

b- Conditions elseif

```
<?
If ($var == 'condition1')
{
        // 'condition1 vérifiée';
}  elseif  ($var ==
'condition2')
{
        // 'condition2 vérifiée';
} …  elseif  ($var ==
'conditionN)
{
        // 'conditionN vérifiée';
} …
else
{ echo 'Aucune condition n'est vérifiée'; }
?>
```

Les structures elseif pouvant se répéter autant de fois que des conditions prévues l'exigeront.

Exemple 1

```
<?
$montant='100' ;
 if ($montant >=0 &&
$montant<1000)
{ echo ' Votre montant '. $montant. ' est compris entre 0 et 1000';
} elseif ($montant >=1000 &&
$montant<5000)
{ echo ' Votre montant '. $montant. '  est compris entre 1000 et 5000';  } else
{ echo ' Votre montant '.$montant. ' est supérieur à 5000'; }  ?>
```

Le résultat obtenu sera :

Votre montant 100 est compris entre 0 et 1000

Exemple 2

```
<?
$variable = 3;


// serie de tests if
($variable == 1)
   {      echo 'La variable a pour
valeur 1';
   }  elseif ($variable
== 2)
   {      echo 'La variable a pour
valeur 2';
   }  elseif ($variable
== 3)
   {     echo 'La variable a pour
valeur 3';
   }  elseif ($variable
== 4)
   {     echo 'La variable a pour
valeur 4';
   }  elseif ($variable
== 5)
   {      echo 'La variable a pour
valeur 5';
   }  else       {      echo 'La variable n'appartient pas
à l'intervalle [1,5]';       }
?>
```

Le résultat obtenu sera :

La variable a pour valeur 3

c- Conditions SWITCH

Le switch est une structure qui permet d'éviter la lourdeur de l'écriture de l'exemple 2. Elle permet en outre une meilleure lisibilité.

Syntaxe

```
<?   switch
($variable)
   {      case
condition1:
       //Traitement de la condition
```

```
1        break;     case
condition2:
      //Traitement de la condition 2
break;
    ….

   case conditionN:
      //Traitement de la condition
N        break;      default:
       //Traitement par défaut
}
?>
```

Exemple

```
<?
$variable = 3;

switch ($variable)
{      case 1:         echo 'La variable a
pour valeur 1';          break;  case 2:
echo 'La variable a pour valeur 2';
break;  case 3:
      echo 'La variable a pour valeur 3';
break;  case 4:
      echo 'La variable a pour valeur 4';
break;  case 5:
      echo 'La variable a pour valeur 5';          break;  default:
echo 'La variable n'appartient pas à l'intervalle [1,5]';     }
?>
```

Le résultat obtenu sera :

La variable a pour valeur 3

d- Itération avec WHILE

Syntaxe

```
<?
While (condition)

{
//Traitements
}
?>
```

Exemple

```
<?
$nbre_maximum = 6;
$i = 0;      //initialisation de l'indice d'incrémentation while ($i < $nbre_maximum)
//condition      {      echo  $i.' est inférieur à '.
$nbre_maximum.'<br>';      $i++;  // $i++ est équivalent à ($i+1)       }  echo  $i.' est égal à '.
$nbre_maximum;
?>
```

Le résultat obtenu sera :

0 est inférieur à 6
1 est inférieur à 6
2 est inférieur à 6
3 est inférieur à 6
4 est inférieur à 6
5 est inférieur à 6 6 est égal à 6

e- Itération avec FOR

Syntaxe

```
<?
      for($i=0; $i != condition ; $i++)
  {
        //Traitements réalisés
        }
      ?>
```

Exemple

```
<?
$nbre_maximum = 6;

//----------DEBUT BOUCLE------------  for($i=0;
$i != $nbre_maximum ; $i++)
   {      echo  $i.'est inférieur à '.
$nbre_maximum.'<br>';     }
//------------FIN BOUCLE------------

echo  $i.' est égal à '. $nbre_maximum;
?>
```

Le résultat obtenu sera :

0 est inférieur à 6

```
1 est inférieur à 6
2 est inférieur à 6
3 est inférieur à 6
4 est inférieur à 6
5 est inférieur à 6
6 est égal à 6
```

Boucle foreach PHP

La boucle foreach - Parcourt un bloc de code pour chaque élément d'un tableau. La boucle foreach ne fonctionne que sur les tableaux et est utilisée pour parcourir chaque paire clé/valeur d'un tableau.

```
foreach ($array as $value) {
  code à exécuter ;
}
```

```
<?php
$colors = array("red", "green", "blue", "yellow");
foreach ($colors as $value) {
  echo "$value <br>";
}
?>
```

Ou pour L'exemple suivant affichera à la fois les clés et les valeurs du tableau donné ($age) :

```
<?php
$age = array("Sanae"=>"35", "Brahim"=>"37", "Jawad"=>"43");

foreach($age as $x => $val) {
  echo "$x = $val<br>";
}
?>
```

PHP Break et Continue

L'instruction **break** peut également être utilisée pour sortir d'une boucle.

```
<?php
for ($x = 0; $x < 10; $x++) {
  if ($x == 4) {
    break;
  }
  echo "ce nombre est: $x <br>";
}
?>
```

L'instruction **continue** interrompt une itération (dans la boucle), si une condition spécifiée se produit, et continue avec l'itération suivante dans la boucle.

```
<?php
for ($x = 0; $x < 10; $x++) {
```

```
 if ($x == 4) {
   continue;
 }
 echo "le nombre: $x <br>";
}
?>
```

5- Fonctions

La vraie puissance de PHP vient de ses fonctions. en plus vous pouvez créer vos propres fonctions personnalisées.

PHP propose une palette approximative de 2000 fonctions prédéfinies. Toutefois il vous est possible de créer vos propres bibliothèques de fonctions pour vos usages spécifiques, une meilleure lisibilité du code et une réutilisation.

a- Définition des fonctions

Les fonctions peuvent se distinguer en deux sous groupes :

-les fonctions qui effectuent un traitement (affichage par exemple)

-les fonctions qui effectuent un traitement et retournent un résultat

Syntaxe

```
function nom_de_la_fonction ($paramètres)
{
//traitement sur les paramètres effectué
}
L'appel de la fonction se fait de la manière suivante :
nom_de_la_fonction ($paramètres) ;
```

Exemple

```
<?  function afficher_nom_prenom
($nom,$prenom)
{  echo 'Bonjour '.$nom. '
'.$prenom ; }

afficher_nom_prenom ('Tarak','Joulak') ;
echo ' <br>' ;

$nom1='Mourad' ; $prenom1='Zouari' ;
afficher_nom_prenom
($nom1,$prenom1) ;
```

```
?>
Le résultat obtenu sera :
Bonjour Tarak Joulak
Bonjour Mourad Zouari
```

Dans le cas suivant un traitement est effectué à l'intérieur de la fonction puis retourné par cette dernière. De ce fait la fonction doit donc être affectée à une variable.

Syntaxe

```
function nom_de_la_fonction ($paramètres)
{
//traitement sur les paramètres effectué return
($resultat) ;
}
L'appel de la fonction se fait de la manière suivante :
$variable = nom_de_la_fonction ($paramètres)
;
```

Exemple

```
<?  function additionner ($variable1,$variable2)   {
$total = $variable1 + $variable2;  return ($total) ;
}
$resultat= additionner (1,2) ;
echo  $resultat.' <br>' ;

$var1=6 ;
$var2=7 ;

$resultat= additionner ($var1,$var2) ; echo  $resultat.' <br>' ;
?>
Le résultat obtenu sera :
3
13
```

b- Librairie de fonctions

Idéalement toutes les fonctions créées devraient être regroupées dans un même fichier créant ainsi une bibliothèque de fonctions. Ce fichier sera appelé à l'intérieur des autres fichiers par le biais de la fonction include.

Exemple fichier fonction.inc.php

```
<?
//Ce fichier contiendra l'ensemble des fonctions que vous développerez
function additionner ($variable1,$variable2)   {
$total = $variable1 + $variable2;
return ($total) ;  }
function afficher_nom_prenom ($nom,$prenom)
{  echo 'Bonjour '.$nom. '
'.$prenom ; }
?>
```

Exemple fichier **page.php**

```
<?
Include ("fonction.inc.php") ;
$resultat= additionner (1,2) ;  echo
$resultat.' <br>' ;
afficher_nom_prenom ('Tarik','Jalal') ;
?>
Le résultat obtenu sera :
3
Bonjour Tarak Joulak
```

PHP est un langage faiblement typé, PHP associe automatiquement un type de données à la variable, en fonction de sa valeur. Étant donné que les types de données ne sont pas définis au sens strict, vous pouvez par exemple ajouter une chaîne à un entier sans provoquer d'erreur.

En PHP 7, les déclarations de type ont été ajoutées. Cela nous donne la possibilité de spécifier le type de données attendu lors de la déclaration d'une fonction et, en ajoutant la déclaration stricte, une "erreur fatale" sera générée si le type de données ne correspond pas.

Dans l'exemple suivant, nous essayons d'envoyer à la fois un nombre et une chaîne à la fonction sans utiliser strict :

```
<?php
function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
// puisque strict n'est PAS activé "5 jours"
est changé en int(5), et il renverra 10//
?>
```

```
<?php declare(strict_types=1); // exigence strict
function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
/// puisque strict est activé et que "5 jours" n'est pas un
entier, une erreur sera renvoyée
?>
```

PHP Fatal error: Uncaught TypeError: Argument 2 passed to addNumbers() must be of the type integer, string given, called in /home/bWN0Ml/prog.php on line 6 and defined in

/home/bWN0Ml/prog.php:3 Stack trace: #0 /home/bWN0Ml/prog.php(6): addNumbers(5, '5 days') #1 {main} thrown in /home/bWN0Ml/prog.php on line 3

Pour éviter cette erreur, Pour spécifier strict, nous devons définir declare(strict_types=1);. Cela doit être sur la toute première ligne du fichier PHP.

Valeur d'argument par défaut PHP

fonction setHeight() sans arguments, elle prend la valeur par défaut comme argument

```
<?php declare(strict_types=1); // strict requirement
function setHeight(int $minheight = 50) {
  echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight(); // utilisera la valeur par défaut de 50
setHeight(135);
setHeight(80);
?>
```

6- Tableaux a- Tableaux numérotés et tableaux associatifs

En PHP, il existe trois types de tableaux :

Tableaux indexés - Tableaux avec un index numérique
Tableaux associatifs - Tableaux avec des clés nommées
Tableaux multidimensionnels - Tableaux contenant un ou plusieurs tableaux
La fonction array() est utilisée pour créer un tableau

-Les tableaux à index numériques (tableaux numérotés) dans lesquels l'accès à la valeur de la variable passe par un index numérique ex : $tableau[0], $tableau[1], $tableau[2], …

-Les tableaux à index associatifs (ou tableaux associatifs) dans lesquels l'accès à la valeur de la variable passe par un index nominatif

ex : $tableau[nom], $tableau[prénom], $tableau[adresse], …

Syntaxe

```
//Tableau à index numéroté
$tableau = array (valeur0,valeur1,valeur2, …) ;

//Accès à chacune des valeurs
$tableau[0] donnera valeur0
```

```
$tableau[1] donnera valeur1  …


//Tableau à index associatif

$tableau = array (variable1 => valeur1, variable2 => valeur2, …) ;


//Accès à chacune des valeurs
$tableau[variable1] donnera valeur1
$tableau[variable2] donnera valeur2
…
```

Exemple

```
<?
//Tableau à index numéroté
$tableau1 = array ('château','maison','bateau') ;
echo "Contenu du tableau 1
:<br>"; echo
$tableau1[0]."<br>"; echo
$tableau1[1]."<br>"; echo
$tableau1[2]."<br>";
 //Tableau à index associatif
$tableau2 = array ('prenom' =>'Tarak','nom' =>'Joulak','ville' =>'Oujda') ;


echo "Contenu du tableau 2 :<br>";
echo $tableau2['prenom']."<br>";
echo $tableau2['nom']."<br>";
echo $tableau2['ville']."<br>";
?>
```

Le résultat obtenu sera :

```
Contenu du tableau 1 :
château maison
bateau
Contenu du tableau 2 :
Tarak Joulak
Oujda
```

Pour Obtenir la longueur d'un tableau - La fonction count()

```
<?php
$Voiture = array("Volvo", "BMW", "Toyota");
echo count($Voiture);
?>
```

b- Parcours des tableaux

PHP intègre une structure de langage qui permet de parcourir un à un les élément d'un tableau :

foreach().

Syntaxe

```
$tableau = array (valeur0,valeur1,valeur2, …) ;

foreach ( $tableau as $valeur )
    {
    //Appeller ici la valeur courante  par $valeur
    …       }
```

Exemple

```
<?
//Cas du tableau numéroté
$tableau1 = array ('chateau','maison','bateau') ;
foreach ( $tableau1 as $valeur )
{   echo $valeur."<br>";
}
?>
```

Le résultat obtenu sera : château maison bateau

Exemple

```
<?  //Cas du tableau associatif
$tableau2 = array ('prenom' =>'Tarik','nom' =>'Jalal','ville' =>'Oujda') ;
foreach ( $tableau2 as $valeur )
{   echo
$valeur."<br>"; }
?>
```

Le résultat obtenu sera : Tarik
Jalal
Oujda

Dans l'exemple suivant et pour le cas des tableaux associatifs nous pouvons grâce à la boucle foreach() aussi bien énumérer le nom des variable que leur valeur.

Exemple

```
<?
$tableau2 = array ('prenom'
=>'Tarak','nom' =>'Joulak','ville' =>'Oujda')
; foreach ( $tableau2 as
$variable=>$valeur )
{
  echo $variable." a pour valeur
".$valeur."<br>"; }
?>
```

Le résultat obtenu sera : prenom a pour valeur Tarak

nom a pour valeur Joulak
ville a pour valeur Oujda

Tableaux associatifs PHP

Les tableaux associatifs sont des tableaux qui utilisent des clés nommées que vous leur affectez. Il existe deux manières de créer un tableau associatif :

```
$age = array("Asmae"=>"35", "nasssim"=>"37", "omar"=>"43");
$age['Haytam']="5";
$age['Akram']="11";
$age['Fahd'] = "9";
```

Boucle dans un tableau associatif

```
<?php
$age = array("Asmae"=>"35", " nasssim"=>"37", "omare"=>"43");
foreach($age as $x => $x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>
```

PHP - Tableaux multidimensionnels

```
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
<?php
echo $cars[0][0].": en stock: ".$cars[0][1].", sVendu: ".$cars[0][2].".<br>";
echo $cars[1][0].": en stock: ".$cars[1][1].", Vendu: ".$cars[1][2].".<br>";
echo $cars[2][0].": en stock: ".$cars[2][1].", Vendu: ".$cars[2][2].".<br>";
```

```
echo $cars[3][0].": En stock: ".$cars[3][1].", Vendu: ".$cars[3][2].".<br>";
?>
<?php
for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>
```

Tableaux de tri PHP

PHP - Fonctions de tri pour les tableaux
nous passerons en revue les fonctions de tri de tableau PHP suivantes :
sort() - trie les tableaux par ordre croissant
rsort() - trie les tableaux par ordre décroissant
asort() - trie les tableaux associatifs par ordre croissant, selon la valeur
ksort() - trie les tableaux associatifs par ordre croissant, selon la clé
arsort() - trie les tableaux associatifs par ordre décroissant, selon la valeur
krsort() - trie les tableaux associatifs par ordre décroissant, selon la clé

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
?>
```

c- Recherche dans un tableau

array_key_exists() permet de vérifier si dans un tableau associatif une variable associative (clef) existe ou non. Elle retourne donc une valeur booléenne (True ou False).

Syntaxe

$resultat= **array_key_exists**('nom_de_la variable', tableau_associatif) ;

Exemple

```
<?
$tableau2 = array ('prenom' =>'Tarak','nom' =>'Joulak','ville' =>'Oujda') ;

if (array_key_exists("nom", $tableau2))
```

```
{      echo 'La variable "Nom" se trouve dans le tableau et a pour
valeur: '.$tableau2['nom'];
}
else
{

    echo 'La variable "Nom" ne se trouve pas dans le
tableau'; }

?>
```

Le résultat obtenu sera :

La variable "Nom" se trouve dans le tableau et a pour valeur: Joulak

in_array() permet de déterminer si une valeur existe dans le tableau. Elle retourne donc une valeur booléenne (True ou False).

Syntaxe

```
$resultat=in_array ( nom_de_la_valeur, tableau) ;
```

Exemple

```
<?
$tableau2 = array ('prenom' =>'Tarak','nom' =>'Joulak','ville' =>'Oujda') ;

if (in_array("Tarak", $tableau2))
{      echo 'La valeur "Tarak" existe bien dans le
tableau';  } else
{
    echo 'La valeur "Tarak" ne se trouve pas dans le tableau';

}

?>
```

Le résultat obtenu sera :

La valeur "Tarak" existe bien dans le tableau

-**array_search** fonctionne comme in_array : il travaille sur les valeurs d'un tableau.

Si il a trouvé la valeur, array_search renvoit la clé correspondante (c'est-à-dire le numéro si c'est un tableau numéroté, ou le nom de la variable si c'est un tableau associatif). Si il n'a pas trouvé la valeur, array_search renvoit false (comme in_array).

Syntaxe

```
$resultat=array_search ( nom_de_la_valeur, tableau) ;
```

Exemple

```
<?
$tableau2 = array ('prenom' =>'Tarak','nom' =>'Joulak','ville'
=>'Oujda') ;

if ($position = array_search("Oujda", $tableau2))
{  echo '"Oujda" se trouve en position ' . $position . '<br />';
}
else
{
        echo '"Oujda" ne se trouve pas dans le tableau.';

}
?>
```

Le résultat obtenu sera :

"Oujda" se trouve en position ville

PHP Global Variables - Superglobals

Certaines variables prédéfinies en PHP sont des "superglobales", ce qui signifie qu'elles sont toujours accessibles, quelle que soit leur portée - et vous pouvez y accéder depuis n'importe quelle fonction, classe ou fichier sans avoir à faire quoi que ce soit de spécial.
Les variables PHP superglobales sont :

- $GLOBALS
- $_SERVER
- $_REQUEST

- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

PHP $GLOBALS

```
<?php
$x = 75;
$y = 25;
 function addition() {
  $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
 addition();
echo $z;
?>
```

PHP Superglobal - $_SERVER

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

PHP Superglobal - $_REQUEST

PHP $_REQUEST est une super variable globale PHP qui est utilisée pour collecter des données après avoir soumis un formulaire HTML.

L'exemple ci-dessous montre un formulaire avec un champ de saisie et un bouton d'envoi. Lorsqu'un utilisateur soumet les données en cliquant sur "Soumettre", les données du formulaire sont envoyées au fichier spécifié dans l'attribut action de la balise <form>. Dans cet exemple, nous pointons vers ce fichier lui-même pour le traitement des données du formulaire. Si vous souhaitez utiliser un autre fichier PHP pour traiter les données du formulaire, remplacez-le par le nom de fichier de votre choix. Ensuite, nous

pouvons utiliser la super variable globale $_REQUEST pour collecter la valeur du champ d'entrée :

```
<html>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  // collecter la valeur du champ de saisie
  $name = $_REQUEST['fname'];
  if (empty($name)) {
    echo "Nom vide";
  } else {
    echo $name;
  }
}
?>
</body>
</html>
```

PHP $_POST

PHP $_POST est une super variable globale PHP qui est utilisée pour collecter des données de formulaire après avoir soumis un formulaire HTML avec method="post". $_POST est également largement utilisé pour passer des variables.

PHP $_GET

PHP $_GET est une super variable globale PHP qui est utilisée pour collecter des données de formulaire après avoir soumis un formulaire HTML avec method="get".

$_GET peut également collecter des données envoyées dans l'URL.

```
<html>
<body>
<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>
```

```
</body>
</html>
```

7- Variables serveur

PHP propose l'accès à toute une série de variables comme les en-têtes, dossiers et chemins des scripts, sans que vous ayez à les créer, on les appelle les variables serveur. Ces variables sont créées par le serveur web.

Le tableau $_SERVER permet d'accéder à ces variables. Si la directive register_globals de PHP.INI est active, alors ces variables seront aussi rendues directement accessible dans le contexte d'exécution global c'est à dire séparément du tableau **$_SERVER**.

Description de certaines variables :

$_SERVER['SERVER_NAME'] Le nom du serveur hôte qui exécute le script

$_SERVER['PHP_SELF'] Le nom du fichier du script en cours d'exécution, par rapport à la racine web

$_SERVER['DOCUMENT_ROOT'] Racine du serveur

$_SERVER['REMOTE_ADDR'] Adresse IP du client

$_SERVER['QUERY_STRING'] Liste des paramètres passés au script

$_SERVER['REQUEST_METHOD'] Méthode d'appel du script

Exemple

```
<?
echo 'PHP_SELF: '.$_SERVER['PHP_SELF']; echo "<br>";
echo 'SERVER_NAME: '.$_SERVER['SERVER_NAME']; echo
"<br>"; echo 'DOCUMENT_ROOT:
'.$_SERVER['DOCUMENT_ROOT']; echo "<br>"; echo
'REMOTE_ADDR: '.$_SERVER['REMOTE_ADDR']; echo
"<br>"; echo 'QUERY_STRING:
'.$_SERVER['QUERY_STRING']; echo "<br>"; echo
'REQUEST_METHOD: '.$_SERVER['REQUEST_METHOD'];
echo "<br>";
?>
```

Le résultat obtenu sera :

PHP_SELF: /page.php

SERVER_NAME: localhost

DOCUMENT_ROOT: c:/program files/easyphp18/www

REMOTE_ADDR: 127.0.0.1 QUERY_STRING:

REQUEST_METHOD: GET

La fonction PHP permettant d'éditer l'intégralité des variables prédéfinies disponibles est phpinfo()

```
<?
 phpinfo() ;
?>
```

Expressions régulières PHP

Une expression régulière est une séquence de caractères qui forme un modèle de recherche. Lorsque vous recherchez des données dans un texte, vous pouvez utiliser ce modèle de recherche pour décrire ce que vous recherchez.
Une expression régulière peut être un caractère unique ou un modèle plus compliqué.

En PHP, les expressions régulières sont des chaînes composées de délimiteurs, d'un motif et de modificateurs facultatifs.

En PHP, les expressions régulières sont des chaînes composées de délimiteurs, d'un motif et de modificateurs facultatifs.

$exp = "/Cours SMI S6 TEC WEB AV/i";

Dans l'exemple ci-dessus, / est le délimiteur, Cours SMI S6 TEC WEB AV est le modèle recherché et i est un modificateur qui rend la recherche insensible à la casse.
Le délimiteur peut être n'importe quel caractère autre qu'une lettre, un chiffre, une barre oblique inverse ou un espace. Le délimiteur le plus courant est la barre oblique (/), mais lorsque votre modèle contient des barres obliques, il est pratique de choisir d'autres délimiteurs tels que # ou ~.

Fonctions d'expression régulière

PHP fournit une variété de fonctions qui vous permettent d'utiliser des expressions régulières. Les fonctions preg_match(), preg_match_all() et preg_replace() sont parmi les plus couramment utilisées :

Function	Description
preg_match()	Renvoie 1 si le motif a été trouvé dans la chaîne et 0 sinon
preg_match_all()	Renvoie le nombre de fois que le modèle a été trouvé dans la chaîne, qui peut également être 0
preg_replace()	Renvoie une nouvelle chaîne où les modèles correspondants ont été remplacés par une autre chaîne

La fonction preg_match() vous dira si une chaîne contient des correspondances d'un motif.

```php
<?php
$str = "Visite SMIS6-Cours";
$pattern = "/ SMIS6-Cours/i";
echo preg_match($pattern, $str); // sortie 1
?>
```

La fonction preg_match_all() vous dira combien de correspondances ont été trouvées pour un motif dans une chaîne.

```php
<?php
$str = "The rain in Morocco falls mainly on the plains.";
$pattern = "/ain/i";
echo preg_match_all($pattern, $str); // affiche 4
?>
```

La fonction preg_replace() remplacera toutes les correspondances du motif dans une chaîne par une autre chaîne.

```php
<?php
$str = "Visit Courstecweb";
$pattern = "/ Courstecweb /i";
echo preg_replace($pattern, "Emena.org", $str); // Outputs " Emena.org"
?>
```

IV- Accès fichiers

1- Ouverture / fermeture de fichier

Avant toute opération de lecture ou écriture sur un fichier il y a nécessité de l'ouvrir. Et à la fin de tout traitement d'un fichier il y a nécessité de le fermer.

Syntaxe

```
<?
// Ouverture de fichier
$monfichier = fopen("nom_du_fichier", "r+");
// Traitements sur le fichier
// .....
  // Fermeture du fichier fclose($monfichier);
?>
```

Fopen() prend en entrée :

-le nom du fichier (ou meme une url)

-le mode d'ouverture du fichier Il retourne un handle.

Fclose() prend en entrée le handle envoyé par le fopen.

Les modes d'ouverture d'un fichier sont les suivants :

'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.

Exemple

```
<?
$handle = fopen("http://www.example.com/", "r");
$handle = fopen("test/monfichier.txt", "r+");
?>
```

2- Lecture de fichier

a- fgets() string **fgets** (resource handle, int length)

fgets retourne la chaîne lue jusqu'à la longueur length - 1 octet depuis le pointeur de fichier handle , ou bien la fin du fichier, ou une nouvelle ligne (qui inclue la valeur retournée), ou encore un EOF (celui qui arrive en premier). Si aucune longueur n'est fournie, la longueur par défaut est de 1 ko ou 1024 octets.

Syntaxe

```
<?
$monfichier = fopen("nom_du_fichier", "r+");
fgets ($monfichier, $taille) ;

                fclose($monfichier);
        ?>
```

La fonction prend en paramètre :

le handle retourné par la fonction fopen()

La taille en octets de lecture (optionnel) Elle retourne en sortie une chaîne de caractères.

Exemple :

L'exemple suivant va permettre de lire dans un fichier texte (fichier.txt) ligne par ligne puis d'afficher le résultat à l'écran en mettant un numéro à chaque ligne.

Fichier

fichier.txt

```
Guest0
 Tarak Joulak
a
b
c
1
2
3
```

Fichier page.php

```
<?
$i=0;
$fichier = 'fichier.txt';
$fp = fopen($fichier,'r');
//ouverture du fichier en    lecture
seulement
```

```
while (!feof($fp))
// tant qu'on n'est pas a la fin du
fichier {
        $ligne = fgets($fp);
        //Lecture ligne par ligne
        echo  'Ligne '.$i++.'---
        >'.$ligne.'<br>';
}  fclose($fp);
?>
```

Le résultat obtenu sera :

Ligne 1 ---> Tarak Joulak
Ligne 2 ---> Guest0
Ligne 3 ---> abc123

b -fread() string fread (resource handle , int length)

Une autre manière de faire de la lecture dans un fichier est d'utiliser fread()

fread lit jusqu'à length octets dans le fichier référencé par handle . La lecture s'arrête lorsque length octets ont été lus, ou que l'on a atteint la fin du fichier, ou qu'une erreur survient (le premier des trois).

Syntaxe

```
<?
$monfichier = fopen("nom_du_fichier", "r+");
fread ($monfichier, $taille) ;
fclose($monfichier);
?>
```

Exemple Fichier page.php

```
        <?
// Lit un fichier, et le place dans une chaîne
$filename = "fichier.txt";
 $handle = fopen ($filename, "r");
$contents = fread ($handle, filesize ($filename));
```

```
 echo "-->".$contents; fclose ($handle);
?>
```

Le résultat obtenu sera :

--> Tarak Joulak Guest0 abc 123

c- file() array file (string filename)

Encore une autre manière de lire dans un fichier est d'utiliser la fonction file()

Identique à readfile, hormis le fait que file retourne le fichier dans un tableau. Chaque élément du tableau correspondant à une ligne du fichier, et les retour–chariots sont placés en fin de ligne.

Syntaxe

```
<?  file
($nom_de_fichier)
?>
```

Rq : la fonction file() ne nécessite pas l'usage de open() et close().

Exemple Fichier page.php

```
<?php
$filename = "test/fichier.txt";
$fcontents = file($filename);
echo $fcontents[0]."<br>";  echo
$fcontents[1]."<br>";
?>
```

Le résultat obtenu sera :

Tarak Joulak

Guest0

3. Ecriture dans fichier

La fonction pour l'écriture dans un fichier est **puts()**
int **fputs** (int handle , string string , int length
)

fputs écrit le contenu de la chaîne string dans le fichier pointé par handle . Si la longueur length est fournie, l'écriture s'arrêtera après length octets, ou à la fin de la chaîne (le premier des deux). fwrite retourne le nombre d'octets écrits ou FALSE en cas d'erreur.

Syntaxe

```
<?
$monfichier = fopen("nom_du_fichier", "r+");
fputs ($monfichier, "Texte à écrire");  fclose
($monfichier);
?>
```

Exemple

```
<?
$filename = "fichier.txt";
$monfichier = fopen ($filename, "a+");
$contenu="ceci est le texte a ecrire \r\n";
fputs($monfichier, $contenu);
fclose ($monfichier);
?>
```

Le résultat obtenu sera :
Le fichier fichier.txt aura une ligne supplémentaire contenant « ceci est le texte à écrire »

4. Fonctions diverses de traitement de fichier :

-feof ($handle) fonction qui retourne un booleen pour indiquer la fin du fichier parcouru. Elle prend en entrée le handle retourné par la fonction fopen().

-filesize ($nom_du_fichier) fonction qui retourne la taille du fichier en octets.

-file_exists ($nom_du_fichier) fonction qui retourne un booleen indiquant si le fichier existe ou non.

5. Fonctions PHP qui travaillent avec fichier Zip

Les fonctions des fichiers Zip vous permettent de lire les fichiers ZIP.

Conditions

L'extension ZIP nécessite libzip. https://libzip.org/

Installation

Systèmes Linux

Pour que ces fonctions fonctionnent, vous devez compiler PHP avec --enable-zip.
PHP 5.6: Utilisez l'option de configuration --with-libzip = DIR pour utiliser une installation système libzip. La version 0.11 de libzip est requise, avec la version 0.11.2 ou ultérieure recommandée.
PHP 7.3: Construire avec la libzip fournie est déconseillé, mais toujours possible en ajoutant --without-libzip à la configuration.

Systèmes Windows

Avant PHP 5.3: les utilisateurs doivent activer «php_zip.dll» dans «php.ini» pour que ces fonctions fonctionnent.

Depuis PHP 5.3: L'extension ZIP est intégrée.

PHP Zip Functions

zip_close () Ferme une archive de fichier ZIP
zip_entry_close () Ferme une entrée de répertoire ZIP
zip_entry_compressedsize () Renvoie la taille du fichier compressé d'une entrée de répertoire ZIP
zip_entry_compressionmethod () Renvoie la méthode de compression d'une entrée de répertoire ZIP
zip_entry_filesize () Renvoie la taille réelle du fichier d'une entrée de répertoire ZIP
zip_entry_name () Renvoie le nom d'une entrée de répertoire ZIP
zip_entry_open () Ouvre une entrée de répertoire dans un fichier ZIP pour lecture

zip_entry_read () Lit une entrée de répertoire ouvert dans le fichier ZIP
zip_open () Ouvre une archive de fichier ZIP
zip_read () Lit le fichier suivant dans une archive de fichier ZIP ouverte

Exemple

Ouvrez, lisez et fermez une archive de fichier ZIP:

```
<?php
$zip = zip_open("test.zip");
zip_read($zip);
// le code
zip_close($zip);
?>
```

Exemple

Ouvrez, lisez et fermez une archive de fichier ZIP:

```
<?php
$zip = zip_open("test.zip");
zip_read($zip);
// votre code cii
zip_close($zip);
?>
```

exemple

Ouvrez une archive de fichier ZIP et obtenez le nom et la taille de fichier des entrées du répertoire:

```
<?php
$zip = zip_open("test.zip");
if ($zip) {
  while ($zip_entry = zip_read($zip)) {
    echo "<p>";
    // le nom du dossier
    echo "Name: " . zip_entry_name($zip_entry) . "<br>";
    // Get filesize dossier la taille
    echo "Filesize: " . zip_entry_filesize($zip_entry);
    echo "</p>";
  }
  zip_close($zip);
}
?>
```

IV- Passage et transmission de variables

1-Passage et transmission de variables par formulaire

Quand dans un site web un formulaire est rempli et envoyé, le contenu des champs saisis est transféré à la page destination sous forme de variables. Ce passage de variables ou de paramètres peut se faire de deux manières : en GET ou en POST.

Syntaxe

```
<html>
<body>
        <!—Envoi d'un formulaire en POST  -->
        <form method="post" action="destination.php">
                <input type="text" name="nom" size="12"><br>
                <input type="submit" value="OK">
        </form>

        <!—Envoi d'un formulaire en GET  -->
        <form method="get" action=" destination.php">   …
        </form>
</body>
</html>
```

En GET les paramètres apparaissent associés à l'url sous formes de variables séparées par des & (http://localhost/destination.php?nom=Tarak&prenom=Joulak). En POST le passage de paramètre se fait de manière invisible.

Selon que la méthode d'envoi a été du GET ou du POST la récupération du contenu des variables est faite selon une syntaxe différente :

Syntaxe

```
<?
//Dans le cas d'un envoi des paramètres en POST
$variable1=$_POST['nom_du_champ'] ;

//Dans le cas d'un envoi des paramètres en GET
$variable1=$_GET['nom_du_champ'] ;
?>
```

Exemple 1 :

Dans l'exemple suivant le fichier formulaire.html contient le script html permettant d'afficher un formulaire et d'envoyer les résultats de la saisie à la page resultat.php qui elle les affichera.

Fichier formulaire.html

```
<html>
<body>
        <form method="post" action="resultat.php">
        Nom : <input type="text" name="nom" size="12"><br>
        Prénom : <input type="text" name="prenom" size="12">
        <input type="submit" value="OK">
        </form>
</body>
</html>
```

Fichier resultat.php

```
<? //Récupération des paramètres passés
$prenom = $_POST['prenom'];
$nom = $_POST['nom'];
//affichage des paramètres
echo "<center>Bonjour $prenom $nom</center>";   ?>
```

En exécutant à travers le serveur web le fichier formulaire.html, en remplissant le formulaire et en cliquant sur OK, nous sommes emmenés vers la page resultat.php qui nous affiche une phrase composée des champs saisis dans le formulaire. Les champs saisis sont donc passé de formulaire.html vers resultat.php.

Exemple 2 :

L'exemple précédent peut tenir dans un seul fichier qui s'enverrait à lui même les paramètres

Fichier formulaire.php

```
 <? if ($_POST['submit'] ==
"OK")
{
```

```
        //Le formulaire a été transmis
        //Récupération des paramètres passés
        $prenom = $_POST['prenom'];
        $nom = $_POST['nom'];

        //affichage des paramètres
        echo "<center>Bonjour $prenom
        $nom</center>";
        } else
{
?>
        <!-- Affichage du formulaire de saisie  -->

        <form method="post"
        action="formulaire.php">
        Nom : <input type="text" name="nom"
        size="12"><br>  Prénom : <input
        type="text" name="prenom"
size="12">  <input type="submit" name="submit"
value="OK">  </form> <?
}
?>
```

Remarque :

La définition de la destination du formulaire (action="formulaire.php") aurait pu ne pas être nominative mais exploiter une variable serveur PHP_SELF puisque les paramètres sont envoyées à la page elle même. Ainsi $_SERVER['PHP_SELF'] retournera naturellement formulaire.php. Cela donnera donc :

<form method="**post**" action=**"<? echo $_SERVER['PHP_SELF'];
?>"** > à la place de
<form method="**post**" action="**formulaire.php**">

2-Passage et transmission de variables par hyperlien

Des paramètres ou variables peuvent passer d'une page source vers une page destination sans transiter par un formulaire pour leur envoi. Les hyperliens peuvent être des vecteurs de passage de paramètre.

Syntaxe

```
<!—Syntaxe d'envoi -->
<a href=destination.php
variable1=contenu1&variable2=contenu2&...> Lien </a>
```

La récupération des paramètres dans la page destination se fait par le tableau $_GET

```
<?
$variable1=$_GET['variable1'] ;
$variable2=$_GET['variable2'] ;
…
?>
```

Exemple :

Dans l'exemple suivant nous allons créer un fichier menu.php contenant un menu fait d'hyperliens.

Chacun de ces hyperliens enverra des paramètres différents.

Ce menu sera appelé dans une page qui réagira différemment selon le paramètre envoyé.

Fichier menu.php

```
<table width="200" border="0" cellspacing="0" cellpadding="1">  <tr>
<td> <a href="page.php?menu=1">Menu1</a> </td></tr> <tr>
<td> <a href="page.php?menu=2">Menu2</a> </td></tr>
<tr>
<td> <a href="page.php?menu=3">Menu3</a> </td>
<td> <a href="page.php?menu=3">Menu3</a> </td></tr> </table>
```

Fichier page.php

```
<?
Include("menu.php") ;
echo "<br><br>" ;
$menu= $_GET['menu'] ;
switch ($menu)
   {     case 1:         echo  "Ceci est la page obtenue
du Menu1" ;          break;       case 2:           echo
"Ceci est la page obtenue du Menu2" ;         break;
case 3:
          echo  "Ceci est la page obtenue du Menu3" ;
   break;
   default:
       echo  "Ceci est la page d'accueil" ;
}
?>
```

En exécutant à travers le serveur web page.php, la sélection de chacun des menus chargera à nouveau la page en envoyant des paramètres différents qui seront traités par la page.

3- Redirection

La fonction essentielle de la redirection consiste à sitôt que l'instruction a été trouvée de l'exécuter en redirigeant l'utilisateur vers la page spécifiée.

Header()

int header(« Location :string destination)

Syntaxe

```
<? header("Location:$page_destination");
exit() ;
?>
```

Exemples

```
<?
header("Location:http://www.google.com");
exit() ;
```

```
?>
```

Le résultat obtenu sera :

Vous serez automatiquement redirectionné vers le site de google

```
<?
header("Location:menu.php");
exit() ;
?>
```

Le résultat obtenu sera :

Vous serez automatiquement redirectionné vers la page menu.php de votre répertoire web

Remarque importante

La fonction header doit être appelée avant la première balise HTML, et avant n'importe quel envoi de commande PHP. C'est une erreur très courante que de lire du code avec la fonction include et d'avoir des espaces ou des lignes vides dans ce code qui produisent un début de sortie avant que header n'ait été appelé.

PHP et Les formulaires

Les filtres

2019/2020

Gestion des formulaires PHP

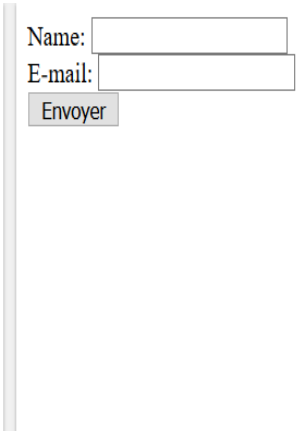
Les superglobaux PHP $ _GET et $ _POST sont utilisés pour collecter les données de formulaire.

PHP - Un formulaire HTML simple:

```
<!DOCTYPE HTML>
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```



HTTP POST

- Lorsque l'utilisateur remplit le formulaire ci-dessus et clique sur le bouton Soumettre, les données du formulaire sont envoyées pour traitement dans un fichier PHP nommé "welcome.php". Les données du formulaire sont envoyées avec la méthode HTTP POST.

  Pour afficher les données soumises, vous pouvez simplement faire écho à toutes les variables. Le "welcome.php" ressemble à ceci:

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

HTTP GET

- Le même résultat pourrait également être obtenu en utilisant la méthode HTTP GET:

```
<html>
<body>
<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```

Le PHP traitement

- "welcome_get.php" est ceci:

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>
```

```
</body>
</html>
```

- Le code ci-dessus est assez simple. Cependant, la chose la plus importante manque. Vous devez valider les données du formulaire pour pro
- Pensez à la SÉCURITÉ lors du traitement des formulaires PHP!
  Cette page ne contient aucune validation de formulaire, elle montre simplement comment envoyer et récupérer des données de formulaire. comment traiter les formulaires PHP avec la sécurité à l'esprit! Une validation correcte des données du formulaire est importante pour protéger votre formulaire contre les pirates et les spammeurs! téger votre script contre le code malveillant

GET vs. POST

- GET et POST créent un tableau (par exemple, tableau (clé1 => valeur1, clé2 => valeur2, clé3 => valeur3, ...)). Ce tableau contient des paires clé / valeur, où les clés sont les noms des contrôles de formulaire et les valeurs sont les données d'entrée de l'utilisateur.
- GET et POST sont traités comme $ _GET et $ _POST. Ce sont des superglobaux, ce qui signifie qu'ils sont toujours accessibles, quelle que soit leur portée - et vous pouvez y accéder depuis n'importe quelle fonction, classe ou fichier sans rien faire de spécial.

  $ _GET est un tableau de variables passées au script actuel via les paramètres URL.

  $ _POST est un tableau de variables passées au script actuel via la méthode HTTP POST.

Quand utiliser GET?

- Les informations envoyées à partir d'un formulaire avec la méthode GET sont visibles par tous (tous les noms et valeurs de variables sont affichés dans l'URL). GET a également des limites sur la quantité d'informations à envoyer. La limitation est d'environ 2000 caractères. Cependant, comme les variables sont affichées dans l'URL, il est possible de mettre la page en signet. Cela peut être utile dans certains cas.

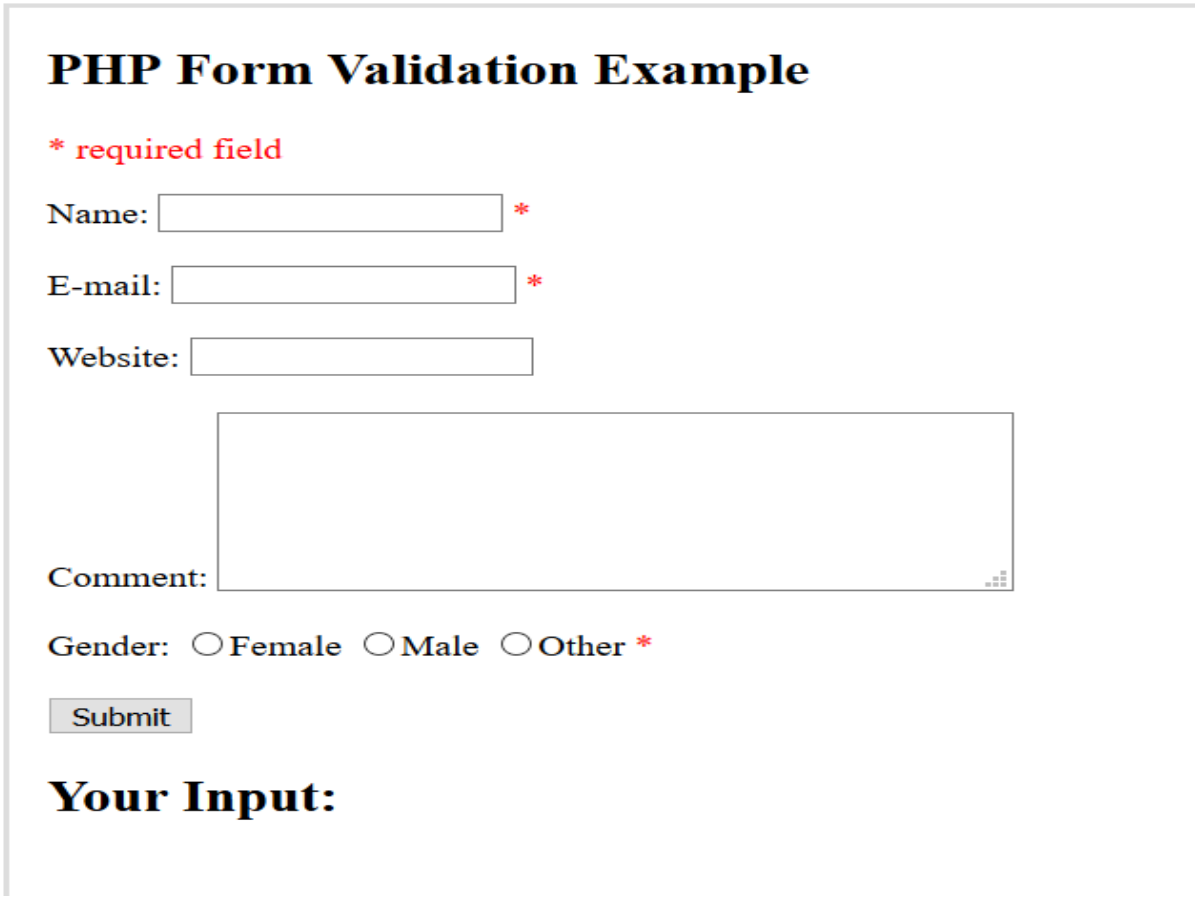
  GET peut être utilisé pour envoyer des données non sensibles.

Remarque: GET ne doit JAMAIS être utilisé pour envoyer des mots de passe ou d'autres informations sensibles!

-
- Quand utiliser POST?
- Les informations envoyées à partir d'un formulaire avec la méthode POST sont invisibles pour les autres (tous les noms / valeurs sont intégrés dans le corps de la requête HTTP) et n'ont pas de limites sur la quantité d'informations à envoyer.

  De plus, POST prend en charge des fonctionnalités avancées telles que la prise en charge de l'entrée binaire en plusieurs parties lors du téléchargement de fichiers sur le serveur.

  Cependant, étant donné que les variables ne sont pas affichées dans l'URL, il n'est pas possible de mettre la page en signet.
- Les développeurs préfèrent le POST pour l'envoi de données de formulaire.
- Validation de formulaire PHP
- les suivants montrent comment utiliser PHP pour valider les données du formulaire.
- Le formulaire HTML sur lequel nous travaillerons dans ces chapitres contient divers champs de saisie: champs de texte obligatoires et facultatifs, boutons radio et bouton d'envoi:



HTML brut pour le formulaire:

- Champs de texte :

Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>

- Boutons radio
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <input type="radio" name="gender" value="other">Other
- L'élément de formulaire

<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">

Qu'est-ce que la variable $ _SERVER ["PHP_SELF"]?

$ _SERVER ["PHP_SELF"] est une variable super globale qui renvoie le nom de fichier du script en cours d'exécution.

Ainsi, $ _SERVER ["PHP_SELF"] envoie les données de formulaire soumises à la page elle-même, au lieu de passer à une autre page. De cette façon, l'utilisateur recevra des messages d'erreur sur la même page que le formulaire.

Qu'est-ce que la fonction htmlspecialchars ()?

- La fonction htmlspecialchars () convertit les caractères spéciaux en entités HTML. Cela signifie qu'il remplacera les caractères HTML comme <et> par <et>. Cela empêche les attaquants d'exploiter le code en injectant du code HTML ou Javascript (attaques Cross-site Scripting) dans les formulaires.

Grande note sur la sécurité des formulaires PHP

- La variable $ _SERVER ["PHP_SELF"] peut être utilisée par les pirates!

  Si PHP_SELF est utilisé dans votre page, un utilisateur peut entrer une barre oblique (/) puis quelques commandes Cross Site Scripting (XSS) à exécuter.

Sécurité

Supposons que nous ayons le formulaire suivant dans une page nommée "test_form.php":

<form method="post" action="<?php echo _SERVER["PHP_SELF"];?>">

Maintenant, si un utilisateur entre l'URL normale dans la barre d'adresse comme "http://www.example.com/test_form.php", le code ci-dessus sera traduit en:

<form method="post" action="test_form.php">

Jusqu'ici tout va bien.
Cependant, considérez qu'un utilisateur entre l'URL suivante dans la barre d'adresse:

http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

Dans ce cas, le code ci-dessus sera traduit en:

<form method="post" action="test_form.php/"><script>alert('hacked')</script>

- Ce code ajoute une balise de script et une commande d'alerte. Et lorsque la page se charge, le code JavaScript sera exécuté (l'utilisateur verra une boîte d'alerte). Ceci est juste un exemple simple et inoffensif de la façon dont la variable PHP_SELF peut être exploitée.

Sachez que tout code JavaScript peut être ajouté à l'intérieur de la balise <script>! Un pirate peut rediriger l'utilisateur vers un fichier sur un autre serveur, et ce fichier peut contenir du code malveillant qui peut modifier les variables globales ou soumettre le formulaire à une autre adresse pour enregistrer les données utilisateur, par exemple.

Comment éviter les exploits de $_SERVER ["PHP_SELF"]?

- Les exploits de $ _SERVER ["PHP_SELF"] peuvent être évités en utilisant la f
- Le code du formulaire devrait ressembler à ceci:onction htmlspecialchars ().

```
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

- La fonction htmlspecialchars () convertit les caractères spéciaux en entités HTML. Maintenant, si l'utilisateur essaie d'exploiter la variable PHP_SELF, cela se traduira par la sortie suivante:

```
<form method="post"
action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">
```

La tentative d'exploitation échoue et aucun mal n'est fait!

Valider les données du formulaire avec PHP

- La première chose que nous ferons est de passer toutes les variables via la fonction htmlspecialchars () de PHP. Lorsque nous utilisons la fonction htmlspecialchars (); puis si un utilisateur essaie de soumettre ce qui suit dans un champ de texte:
  <script> location.href ('http://www.hacked.com') </script>
  - cela ne serait pas exécuté, car il serait enregistré en tant que code d'échappement HTML, comme ceci:
  <script> location.href ('http://www.hacked.com') </script>

  Le code peut désormais être affiché en toute sécurité sur une page ou dans un e-mail. Nous ferons également deux autres choses lorsque l'utilisateur soumettra le formulaire:

  Supprimez les caractères inutiles (espace supplémentaire, tabulation, nouvelle ligne) des données d'entrée de l'utilisateur (avec la fonction PHP trim ())

  Supprimez les barres obliques inverses (\) des données d'entrée de

l'utilisateur (avec la fonction PHP stripslashes ())

L'étape suivante consiste à créer une fonction qui fera toutes les vérifications pour nous (ce qui est beaucoup plus pratique que d'écrire encore et encore le même code).

La validation

- Nous nommerons la fonction test_input ().
  Maintenant, nous pouvons vérifier chaque variable $ _POST avec la fonction test_input (), et le script ressemble à ceci:

```
<?php
// définir des variables et définir des valeurs vides
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

- Notez qu'au début du script, nous vérifions si le formulaire a été envoyé à l'aide de $ _SERVER ["REQUEST_METHOD"]. Si REQUEST_METHOD est POST, le formulaire a été soumis - et il doit être validé. S'il n'a pas été soumis, ignorez la validation et affichez un formulaire vierge.
- Cependant, dans l'exemple ci-dessus, tous les champs de saisie sont facultatifs. Le script fonctionne correctement même si l'utilisateur n'entre aucune donnée.

L'étape suivante consiste à rendre les champs de saisie obligatoires et à créer des messages d'erreur si nécessaire.

Formulaires PHP - Champs obligatoires

- Dans le code suivant, nous avons ajouté de nouvelles variables:

$ nameErr, $ emailErr, $ genderErr et $ websiteErr.

- Ces variables d'erreur contiendront des messages d'erreur pour les champs obligatoires.
- Nous avons également ajouté une instruction if else pour chaque variable $ _POST.
- Ceci vérifie si la variable $ _POST est vide (avec la fonction PHP empty ()).
- S'il est vide, un message d'erreur est stocké dans les différentes variables d'erreur, et s'il n'est pas vide, il envoie les données d'entrée de l'utilisateur via la fonction test_input ():

```
<?php
// définir des variables et définir des valeurs vides
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }
```

```
  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP - Afficher les messages d'erreur

- Ensuite, dans le formulaire HTML, nous ajoutons un petit script après chaque champ obligatoire, ce qui génère le message d'erreur correct si nécessaire (c'est-à-dire si l'utilisateur essaie de soumettre le formulaire sans remplir les champs obligatoires):

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">

Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
Website:
<input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">

</form>
```

Formulaires PHP - Valider l'e-mail et l'URL

- L'étape suivante consiste à valider les données d'entrée, à savoir "Le champ Nom contient-il uniquement des lettres et des espaces?", Et "Le

champ E-mail contient-il une syntaxe d'adresse de messagerie valide?", Et s'il est rempli, " Le champ Site Web contient-il une URL valide? ".

- **PHP - Valider le nom**

Le code ci-dessous montre un moyen simple de vérifier si le champ de nom ne contient que des lettres et des espaces. Si la valeur du champ de nom n'est pas valide, stockez un message d'erreur:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Seules les lettres et les espaces blancs sont autorisés";
}
```

La fonction preg_match () recherche un motif dans une chaîne, renvoyant vrai si le motif existe et faux sinon.

- **PHP - Valider l'e-mail**

Le moyen le plus simple et le plus sûr de vérifier si une adresse e-mail est bien formée est d'utiliser la fonction filter_var () de PHP.
Dans le code ci-dessous, si l'adresse e-mail n'est pas bien formée, stockez un message d'erreur:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

- **PHP - Validateur URL**

Le code ci-dessous montre un moyen de vérifier si une syntaxe d'adresse URL est valide (cette expression régulière autorise également les tirets dans l'URL). Si la syntaxe d'adresse URL n'est pas valide, stockez un message d'erreur:

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

- **PHP - Valider le nom, l'e-mail et l'URL**

```
<?php
// définir des variables et définir des valeurs vides
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
```

```
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
// vérifie si le nom ne contient que des lettres et des espaces
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
// vérifier si l'adresse e-mail est bien formée
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
// vérifie si la syntaxe d'adresse URL est valide (cette expression régulière autorise
également les tirets dans l'URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }
  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }
  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP - Gardez les valeurs dans le formulaire

- L'étape suivante consiste à montrer comment empêcher le formulaire de vider tous les champs de saisie lorsque l'utilisateur soumet le formula
- Pour afficher les valeurs dans les champs de saisie après que l'utilisateur a cliqué sur le bouton Soumettre, nous ajoutons un petit script PHP à l'intérieur de l'attribut de valeur des champs de saisie suivants: nom, e-mail et site Web. Dans le champ de zone de commentaire, nous plaçons le script entre les balises <textarea> et </textarea>. Le petit script affiche la valeur des variables $ name, $ email, $ website et $ comment.
- nous devons également afficher le bouton radio coché. Pour cela, nous devons manipuler l'attribut vérifié (pas l'attribut value pour les boutons radio):

- Name: <input type="text" name="name" value="<?php echo $name;?>">

  E-mail: <input type="text" name="email" value="<?php echo $email;?>">

  Website: <input type="text" name="website" value="<?php echo $website;?>">

  Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comment;?></textarea>

  Gender:
  <input type="radio" name="gender"
  <?php if (isset($gender) && $gender=="female") echo "checked";?>
  value="female">Female
  <input type="radio" name="gender"
  <?php if (isset($gender) && $gender=="male") echo "checked";?>
  value="male">Male
  <input type="radio" name="gender"
  <?php if (isset($gender) && $gender=="other") echo "checked";?>
  value="other">Other

  Le code

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
```

```
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }
    if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression also allows dashes in
the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-
z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
```

```
  }
  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field</span></p>
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name" value="<?php echo $name;?>">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email" value="<?php echo $email;?>">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website" value="<?php echo $website;?>">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" <?php if (isset($gender) && $gender=="female")
echo "checked";?> value="female">Female
  <input type="radio" name="gender" <?php if (isset($gender) && $gender=="male")
echo "checked";?> value="male">Male
  <input type="radio" name="gender" <?php if (isset($gender) && $gender=="other")
echo "checked";?> value="other">Other
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
```

```
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

6. Fonctions de filtrage PHP

Ce filtre PHP est utilisé pour valider et filtrer les données provenant de sources Depuis PHP 5.2.0, les fonctions de filtrage sont activées par défaut. Aucune installation n'est nécessaire pour utiliser ces fonctions. non sécurisées, comme les entrées utilisateur.

Fonctions de filtrage PHP:

filter_has_var(): Vérifie s'il existe une variable d'un type d'entrée spécifié,

```
<?php
if (!filter_has_var(INPUT_GET, "email")) {
    echo("Email not found");
} else {
    echo("Email found");
}
?>
```

Vérifiez si la variable d'entrée "email" est envoyée à la page PHP, via la méthode "get"

AUSSI

- filter_var(): Vérifiez si $ email est une adresse e-mail valide:

```
<?php
$email = "jamal.driss@example.com";

if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
  echo("$email is a valid email address");
} else {
```

```
  echo("$email is not a valid email address");
}
?>
```

Aussi

```
<?php
$email = " jamal.driss@example.com ";
// Supprimer tous les caractères illégaux de l'e-mail
$email = filter_var($email, FILTER_SANITIZE_EMAIL);
// Valider e-mail
if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo("$email is a valid email address");
} else {
    echo("$email is not a valid email address");
}
?>
```

Filtre PHP FILTER_VALIDATE_FLOAT

```
<?php
$var=12.3;
var_dump(filter_var($var, FILTER_VALIDATE_FLOAT));
?>
```

La sortie du code sera: float(12.3)

PHP FILTER_VALIDATE_BOOLEAN Filter:

- ```
  <?php
  $var1="yes";
  $var2="off";
  var_dump(filter_var($var1, FILTER_VALIDATE_BOOLEAN));
  echo "<br>";
  var_dump(filter_var($var2, FILTER_VALIDATE_BOOLEAN));
  ?>
  ```
- ```
  bool(true)
  bool(false)
  ```

PHP FILTER_SANITIZE_STRING Filter

- ```
  <?php
  $str = "<h1>Hello World!</h1>";

  $newstr = filter_var($str, FILTER_SANITIZE_STRING);
  echo $newstr;
  ?>
  ```
- Supprimez toutes les balises HTML et tous les caractères avec une valeur ASCII> 127, d'une chaîne:

```
<?php
$str = "<h1>Hello WorldÆØÅ!</h1>";
```
```

```
$newstr = filter_var($str, FILTER_SANITIZE_STRING, FILTER_FLAG_STRIP_HIGH);
echo $newstr;
?>
```

Téléchargement de fichiers PHP

- Avec PHP, il est facile de télécharger des fichiers sur le serveur. Cependant, avec facilité vient le danger, alors soyez toujours prudent lorsque vous autorisez les téléchargements de fichiers!
- Configurer le fichier "php.ini"
- Tout d'abord, assurez-vous que PHP est configuré pour autoriser les téléchargements de fichiers.
- Dans votre fichier "php.ini", recherchez la directive file_uploads et réglez-la sur On:
- file_uploads = On

Créer le formulaire HTML

- Ensuite, créez un formulaire HTML qui permet aux utilisateurs de choisir le fichier image qu'ils souhaitent télécharger:
- ```
  <!DOCTYPE html>
  <html>
  <body>
  <form action="upload.php" method="post" enctype="multipart/form-data">
      Select image to upload:
      <input type="file" name="fileToUpload" id="fileToUpload">
      <input type="submit" value="Upload Image" name="submit">
  </form>
  </body>
  </html>
  ```

Créer le formulaire HTML

- Quelques règles à suivre pour le formulaire HTML ci-dessus:

Assurez-vous que le formulaire utilise method = "post"

Le formulaire a également besoin de l'attribut suivant: enctype = "multipart / form-data". Il spécifie le type de contenu à utiliser lors de la soumission du formulaire Sans les exigences ci-dessus, le téléchargement de fichiers ne fonctionnera pas.

- Autres choses à noter:
  ```

L'attribut type = "file" de la balise <input> affiche le champ de saisie comme un contrôle de sélection de fichier, avec un bouton "Parcourir" à côté du contrôle d'entrée

- Le formulaire ci-dessus envoie des données vers un fichier appelé "upload.php", que nous allons créer ensuite.

Créer le script PHP de téléchargement de fichier

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Vérifiez si le fichier image est une image réelle ou une fausse image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "Le fichier est une image- " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "Le fichier n'est pas une image.";
        $uploadOk = 0;
    }
}
?>
```

Le script PHP a expliqué:

- $target_dir = "uploads/" - spécifie le répertoire où le fichier va être placé
- $target_file -spécifie le chemin du fichier à télécharger
- $uploadOk=1 n'est pas encore utilisé (sera utilisé plus tard)
- $imageFileType contient l'extension du fichier (en minuscules)
- Next, Ensuite, vérifiez si le fichier image est une image réelle ou une fausse image

  Vérifiez si le fichier existe déjà :
- Nous pouvons maintenant ajouter quelques restrictions.

  Tout d'abord, nous vérifierons si le fichier existe déjà dans le dossier "uploads". Si c'est le cas, un message d'erreur s'affiche et $ uploadOk est remie sur 0:

  // Vérifier si le fichier existe déjà

```
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
```

Limiter la taille du fichier :

- Le champ de saisie de fichier dans notre formulaire HTML ci-dessus est nommé "fileToUpload".
  Maintenant, nous voulons vérifier la taille du fichier. Si le fichier est supérieur à 500 Ko, un message d'erreur s'affiche et $ uploadOk est remise à 0:

```
// Vérifier la taille du fichier
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
```

Type de fichier limite :

- Le code ci-dessous permet uniquement aux utilisateurs de télécharger des fichiers JPG, JPEG, PNG et GIF. Tous les autres types de fichiers donnent un message d'erreur avant de mettre $ uploadOk à 0:

```
// Autoriser certains formats de fichiers
if($imageFileType != "jpg" && $imageFileType != "png" &&
$imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
```

Script PHP de téléchargement de fichier complet

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Vérifier si le fichier image est une image réelle ou une fausse image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
```

```
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
// Vérifier si le fichier existe déjà
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
// Vérifier la taille du fichier
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
// Autoriser certains formats de fichiers
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
// Vérifiez si $ uploadOk est mis à 0 par une erreur
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
// si tout va bien, essayez de télécharger le fichier
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file ". basename( $_FILES["fileToUpload"]["name"]). " has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>
```

7. Filtres PHP avancés

Valider un entier dans une plage

- L'exemple suivant utilise la fonction filter_var () pour vérifier si une variable est à la fois de type INT et comprise entre 1 et 200:

```
<?php
$int = 122;
$min = 1;
$max = 200;
```

```
if (filter_var($int, FILTER_VALIDATE_INT, array("options" => array("min_range"=>$min,
"max_range"=>$max))) === false) {
    echo(" La valeur variable n'est pas dans la plage légale ");
} else {
    echo(" La valeur variable se situe dans la plage légale ");
}
?>
```

Valider l'adresse IPv6

- L'exemple suivant utilise la fonction filter_var () pour vérifier si la variable $ ip est une adresse IPv6 valide:

```
<?php
$ip = "2001:0db8:85a3:08d3:1319:8a2e:0370:7334";

if (!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6) === false) {
    echo(" $ ip est une adresse IPv6 valide ");
} else {
    echo(" $ ip n'est pas une adresse IPv6 valide ");
}
?>
```

Valider l'adresse IPv4

```
<?php
$ip = "127.0.0.1";

if (filter_var($ip, FILTER_VALIDATE_IP)) {
    echo("$ip is a valid IP address");
} else {
    echo("$ip is not a valid IP address");
}
?>
```

Fonction PHP filter_var_array ()

Utilisez la fonction filter_var_array () pour obtenir plusieurs variables:

```
<?php
$data = array(
  'fullname' => ALI,
  'age' => '41',
  'email' => 'ALI@example.com',
);

$mydata = filter_var_array($data);
```

```
var_dump($mydata);

?>
```

La sortie du code doit être:

```
array(3) {
  ["fullname"]=>
  string(13) "ALI"
  ["age"]=>
  string(2) "41"
  ["email"]=>
  string(17) "ALI@example.com"
}
```

Astuce: filter_var () et problème avec 0

- Dans l'exemple ci-dessus, si $ int a été défini sur 0, la fonction ci-dessus retournera "Integer is not valid". Pour résoudre ce problème, utilisez le code ci-dessous:

```
<?php
$int = 0;

if (filter_var($int, FILTER_VALIDATE_INT) === 0 || !filter_var($int,
FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

VI-Variables persistantes: Cookies et Session

Les variables ont une durée de vie limitée : celle du script qui les appelle. Ainsi que nous l'avons vu dans le chapitre précédent l'unique moyen de transmettre ces variables de pages en pages consiste à effectuer un passage de paramètres (méthode GET ou POST) ce qui d'une manière générale est contraignant à plus d'un titre :

- Contraignant pour le développeur puisque il doit gérer par code ces passages de paramètres, - Contraignant pour la sécurité si l'on ne désire pas que le client accède à certaines informations.

PHP offre un mécanisme de stockage d'informations de manière persistante. Autrement dit tout au long de la navigation à l'intérieur d'un site des variables seront accessibles sans avoir pour autant à les passer en paramètres.

Deux types de variables persistantes existent :

- Les variables persistantes côté client : les cookies
- Les variables persistantes côté serveur : la session

1- les Cookies

Les cookies sont un mécanisme d'enregistrement d'informations sur le client, et de lecture de ces informations. Ce système permet d'authentifier et de suivre les visiteurs d'un site. PHP supporte les cookies de manière transparente.

a- setcookie() : Cette fonction permet de définir un cookie qui sera envoyé avec le reste des en−têtes.

int setcookie (string nom_variable ,[string valeur_variable],[int expiration],[string chemin],[string domaine],[int securité])

nom_variable : nom de la variable a stocker valeur_variable : Valeur de la variable a stocker expiration : durée pour l'expiration du cookie chemin :

le chemin du répertoire ou doit être lu le cookie

domaine : le nom

domaine securité : le
type d'entête (http, https)

Tous les arguments sauf nom_variable sont optionnels.

A l'instar de la fonction header(), setcookie() doit impérativement être appelée avant tout affichage de texte.

Syntaxe

```
<? setcookie("variable1",$valeur1,
$durée,$chemin,$domaine,$securite);
?>
```

Exemples

```
<?
$valeur= "Ceci est la valeur de la variable" ; setcookie
("TestCookie",$value,time()+3600); /* expire dans une heure */  ?>
```

b- lire un cookie

Syntaxe

```
<?
$HTTP_COOKIE_VARS["$nom_de_la_variable"] ;
?>
```

Exemple

```
<? echo
$HTTP_COOKIE_VARS["TestCookie"];
?>
Le résultat obtenu sera :
Ceci est la valeur de la variable
```

c- Tableau de variables dans un cookie

Il est possible d'envoyer un tableau de variables à stocker dans un cookie

Exemple

```
<?

setcookie( "tcookie[trois]", "troisième cookie" );
```

```
setcookie( "tcookie[deux]", "second cookie" ); setcookie(
"tcookie[un]", "premier cookie" );
 }
?>
```

Exemple de lecture d'un tableau stocké dans un cookie

```
<?
//récupération du tableau cookie
dans la variable
$cookie
$cookie=$HTTP_COOKIE_VARS["t
cookie"];
//Vérification si la variable est vide
ou non if ( isset( $cookie ) )
{ while( list( $name, $value ) =
each( $cookie ) )
        { echo "$name ==
        $value<br>\n";
        }
} ?>
```

Le résultat obtenu sera : trois == troisième cookie deux == deuxième cookie un == premier cookie

d-Suppression d'un cookie

Pour supprimer un cookie envoyez un cookie avec une variable sans valeur et un délai dépassé.

Exemple

```
<?
Setcookie ("TestCookie","",time()-100);
?>
```

e- Limitation des cookies

Le problème majeur du cookie c'est que le client a le pouvoir de le refuser (en configurant spécifiquement son browser). Votre application risque donc de ne pas pouvoir fonctionner.

Il y a aussi des risques plus graves quant à la sécurité. L'usurpation d'identité, car ce fichier peut être recopié facilement sur un autre ordinateur et modifié puisque ce n'est qu'un fichier texte.

2- les sessions

La gestion des sessions avec PHP est un moyen de sauver des informations entre deux accès. Cela permet notamment de construire des applications personnalisées, et d'accroître l'attrait de votre site. Chaque visiteur qui accède à votre site se voit assigner un numéro d'identifiant, appelé plus loin "identifiant de session". Celui−ci est enregistré soit dans un cookie, chez le client, soit dans l'URL. Les sessions vous permettront d'enregistrer des variables pour les préserver et les réutiliser tout au long de la visites de votre site. Lorsqu'un visiteur accède à votre site, PHP vérifiera si une session a déjà été ouverte. Si une telle session existe déjà, l'environnement précédent sera recréé.

L'inconvénient précédemment évoqué concernant les cookies est dépassé dans la mesure ou tout est stocké sur le serveur même.

a- session_start() session_start() permet de démarrer une session pour le client .

Syntaxe

```
<? session_start()
; ?>
```

Cette commande doit figurer dans toutes les pages elle perpétue le transfert des variables de session au cours de la navigation dans le site.

Le compilateur PHP va alors créer alors dans le répertoire de sauvegarde des sessions, un fichier dont le nom commence par sess_ et se termine par un identifiant généré de manière aléatoire.

L'identifiant de session peut être affiché par la commande session_id(). Vous pouvez également gérer vous-même ce nom de session en utilisant session_name() avant le démarrage de la session. La durée de vie d'une session est paramétrée par session.cache_expire

La session est perdue définitivement pour l'utilisateur lorsque :

1- Il n'a exécuté aucune action (POST ou GET) au delà de la durée de vie définie .

2- Il ferme son navigateur.

3- La commande session_destroy est appelée.

b- Ajouter des variables dans la session

L'ajout de variable dans l'environnement de session se fait au travers d'une affectation classique.

Toutefois la variable session définie doit être appelée vie la tableau **$_SESSION[]**

Syntaxe

```
<?
session_start() ;
$_SESSION['nom_variable'] = $valeur;
?>
```

Exemple

```
<?
session_start() ;
$_SESSION['ville'] = "Oujda";
?>
```

Dans cet exemple une variable du non de ville contenant la valeur Oujda a été enregistrée dans la session courante.

c- Lire une variable dans la session

Syntaxe

```
<?
session_start() ;
$variable = $_SESSION['nom_variable'] ;  ?>
```

Exemple

```
<?  session_start() ; echo  ' Le contenu de la variable VILLE de la session
est : '. $_SESSION['ville'] ;
?>
```

Le résultat obtenu sera :

Le contenu de la variable VILLE de la session est : Oujda

d- Supprimer une variable de la session

Syntaxe

```
<?
session_start() ;  unset
($_SESSION['nom_de_le_variable']);  ?>
```

Exemple

```
<?
session_start() ; unset
($_SESSION['ville']);
//Verificatin de la
suppression
if ( isset ($_SESSION['ville']))
{
        $resultat = "La suppression a échoué .";
}
else
{
        $resultat = "La ville a été effacée.";
}  echo
$resultat;
?>
```

Le résultat obtenu sera :

La ville a été effacée.

e- Supprimer l'environnement de session

Dans l'exemple précédent nous avons vu comment supprimer une variable de l'environnement de session.

Il reste toutefois possible de supprimer tout un environnement de session donné. Pour cela il suffit de vider le tableau global des session en le réinitialisant.

Synta

xe <?

```
session_start() ;
$_SESSION = array();?>
```

VI PHP - Qu'est-ce que la POO?

Depuis PHP5, vous pouvez également écrire du code PHP dans un style orienté objet.

La programmation orientée objet est plus rapide et plus facile à exécuter.
PHP Qu'est-ce que la POO?

OOP signifie programmation orientée objet.

La programmation procédurale consiste à écrire des procédures ou des fonctions qui effectuent des opérations sur les données, tandis que la programmation orientée objet consiste à créer des objets contenant à la fois des données et des fonctions.

La programmation orientée objet présente plusieurs avantages par rapport à la programmation procédurale:

- La POO est plus rapide et plus facile à exécuter
- La POO fournit une structure claire pour les programmes
- La POO aide à garder le code PHP DRY 'Don't Repeat Yourself', et facilite la maintenance, la modification et le débogage du code
- La POO permet de créer des applications entièrement réutilisables avec moins de code et un temps de développement plus court

Conseil: Le principe «Ne vous répétez pas» (DRY) consiste à réduire la répétition du code. Vous devez extraire les codes communs à l'application, les placer à un seul endroit et les réutiliser au lieu de le répéter.

PHP - Que sont les classes et les objets?

Les classes et les objets sont les deux principaux aspects de la programmation orientée objet.

Regardez l'illustration suivante pour voir la différence entre la classe et les objets:

Another example:

Ainsi, une classe est un modèle pour les objets et un objet est une instance d'une classe.

Lorsque les objets individuels sont créés, ils héritent de toutes les propriétés et comportements de la classe, mais chaque objet aura des valeurs différentes pour les propriétés.

Consultez les chapitres suivants pour en savoir plus sur la POO.

PHP POO - Classes et objets

Une classe est un modèle pour les objets et un objet est une instance de classe.
Cas de POO

Supposons que nous ayons une classe nommée Fruit. Un Fruit peut avoir des propriétés

comme le nom, la couleur, le poids, etc. Nous pouvons définir des variables comme $ name, $ color et $ weight pour contenir les valeurs de ces propriétés.

Définir une classe

Une classe est définie à l'aide du mot-clé class, suivi du nom de la classe et d'une paire d'accolades ({}). Toutes ses propriétés et méthodes vont à l'intérieur des accolades:
Syntaxe :

```
<?php
class Fruit {
  // code goes here...
}
?>
```

Ci-dessous, nous déclarons une classe nommée Fruit composée de deux propriétés ($ name et $ color) et de deux méthodes set_name () et get_name () pour définir et obtenir la propriété $ name:

Example

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

Define Objects

Les cours ne sont rien sans objets! Nous pouvons créer plusieurs objets à partir d'une classe. Chaque objet possède toutes les propriétés et méthodes définies dans la classe, mais elles auront

des valeurs de propriété différentes.

Les objets d'une classe sont créés à l'aide du nouveau **new** mot-clé.

Dans l'exemple ci-dessous, $ apple et $ banana sont des instances de la classe Fruit:

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

Dans l'exemple ci-dessous, nous ajoutons deux méthodes supplémentaires à la classe Fruit, pour définir et obtenir la propriété $ color:

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
```

```
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>
```

PHP - Le mot-clé $ this

Le mot-clé $ this fait référence à l'objet courant et n'est disponible que dans les méthodes.

Regardez l'exemple suivant:

```
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
?>
```

Alors, où pouvons-nous changer la valeur de la propriété $ name? Il y a deux manières:

1. À l'intérieur de la classe (en ajoutant une méthode set_name () et en utilisant $ this):

```php
<?php
class Fruit {
  public $name;
  function set_name($name) {
    $this->name = $name;
  }
}
$apple = new Fruit();
$apple->set_name("Apple");
?>
```

PHP – instanceof

Vous pouvez utiliser le mot-clé instanceof pour vérifier si un objet appartient à une classe spécifique:

```php
<?php
$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>
```

PHP - La fonction __construct

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function get_name() {
    return $this->name;
  }
  function get_color() {
    return $this->color;
  }
}
```

```
$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>
```

PHP OOP - Destructor

Un destructeur est appelé lorsque l'objet est détruit ou que le script est arrêté ou quitté.

Si vous créez une fonction __destruct (), PHP appellera automatiquement cette fonction à la fin du script.

Notez que la fonction de destruction commence par deux traits de soulignement (__)!

L'exemple ci-dessous a une fonction __construct () qui est automatiquement appelée lorsque vous créez un objet à partir d'une classe, et une fonction __destruct () qui est automatiquement appelée à la fin du script:

```
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function __destruct() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

$apple = new Fruit("Apple", "red");
?>
```

REMARQUES: comme les constructeurs et les destructeurs aident à réduire la quantité de code, ils sont très utiles.

PHP OOP - Access Modifiers

Les propriétés et les méthodes peuvent avoir des modificateurs d'accès qui contrôlent où ils sont accessibles.

Il existe trois modificateurs d'accès:

public - la propriété ou la méthode est accessible de partout. C'est par défaut
protected - la propriété ou la méthode est accessible dans la classe et par les classes dérivées de cette classe
private - la propriété ou la méthode est UNIQUEMENT accessible dans la classe

Dans l'exemple suivant, nous avons ajouté trois modificateurs d'accès différents aux trois propriétés. Ici, si vous essayez de définir la propriété name, cela fonctionnera correctement (car la propriété name est publique). Cependant, si vous essayez de définir la propriété color ou weight, cela entraînera une erreur fatale (car les propriétés color et weight sont protégées et privées):

```php
<?php
class Fruit {
  public $name;
  public $color;
  public $weight;

  function set_name($n) {  // a public function (default)
    $this->name = $n;
  }
  protected function set_color($n) { // a protected function
    $this->color = $n;
  }
  private function set_weight($n) { // a private function
    $this->weight = $n;
  }
}

$mango = new Fruit();
$mango->set_name('Mango'); // OK
$mango->set_color('Yellow'); // ERROR
$mango->set_weight('300'); // ERROR
?>
```

PHP POO – Héritage

Héritage en POO = Quand une classe dérive d'une autre classe.

La classe enfant héritera de toutes les propriétés et méthodes publiques et protégées de la classe parent. De plus, il peut avoir ses propres propriétés et méthodes.

Une classe héritée est définie à l'aide du mot clé extend.

Regardons un exemple:

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

// Strawberry is inherited from Fruit
class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

La classe Strawberry est héritée de la classe Fruit.

Cela signifie que la classe Strawberry peut utiliser les propriétés publiques $ name et $ color ainsi que les méthodes publiques __construct () et intro () de la classe Fruit en raison de l'héritage.

La classe Strawberry a également sa propre méthode: message ().

PHP POO - Constantes de classe

Les constantes ne peuvent pas être modifiées une fois qu'elles ont été déclarées.

Les constantes de classe peuvent être utiles si vous devez définir des données constantes dans une classe.

Une constante de classe est déclarée à l'intérieur d'une classe avec le mot-clé **const.**

Les constantes de classe sont sensibles à la casse. Cependant, il est recommandé de nommer les constantes dans toutes les lettres majuscules.

Nous pouvons accéder à une constante depuis l'extérieur de la classe en utilisant le nom de la classe suivi de l'opérateur de résolution de portée (: :) suivi du nom de la constante, comme ici:

```
<?php
class Goodbye {
  const LEAVING_MESSAGE = " Corona-virus est dangereux ";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

Ou, nous pouvons accéder à une constante depuis l'intérieur de la classe en utilisant le mot-clé **self** suivi de l'opérateur de résolution de portée **(: :)** suivi du nom de la constante, comme ici:

```
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Corona-virus est dangereux";
  public function byebye() {
    echo self::LEAVING_MESSAGE;
  }
}

$goodbye = new Goodbye();
$goodbye->byebye();
?>
```

PHP OOP - Abstract Classes

Les classes et méthodes abstraites se produisent lorsque la classe parente a une méthode nommée, mais a besoin de sa (ses) classe (s) enfant (s) pour remplir les tâches.

Une classe abstraite est une classe qui contient au moins une méthode abstraite. Une méthode abstraite est une méthode déclarée, mais non implémentée dans le code.

Une classe ou méthode abstraite est définie avec le mot-clé **abstract**:

```php
<?php
// Parent class
abstract class Car {
  public $name;
  public function __construct($name) {
    $this->name = $name;
  }
  abstract public function intro() : string;
}

// Child classes
class Audi extends Car {
  public function intro() : string {
    return "La qualité allemande! Je suis $this->name!";
  }
}

class Volvo extends Car {
  public function intro() : string {
    return "La securité Suedoise! Je suis a $this->name!";
  }
}

class Citroen extends Car {
  public function intro() : string {
    return "La fancaise mauvaise! Je suis $this->name!";
  }
}

// Créer des objets à partir des classes enfants
$audi = new audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new volvo("Volvo");
echo $volvo->intro();
```

```
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();
?>
```

PHP - Que sont les interfaces Interfaces?

Les interfaces vous permettent de spécifier les méthodes qu'une classe doit implémenter.
Les interfaces facilitent l'utilisation d'une variété de classes différentes de la même manière. Lorsqu'une ou plusieurs classes utilisent la même interface, on parle de «polymorphisme».
Les interfaces sont déclarées avec le mot-clé interface:

```
<?php
// Interface definition
interface Animal {
  public function makeSound();
}

// Class definitions
class Cat implements Animal {
  public function makeSound() {
    echo " Miyaw ";
  }
}

class Dog implements Animal {
  public function makeSound() {
    echo " HOWHOW ";
  }
}

class Mouse implements Animal {
  public function makeSound() {
    echo " ZEZEZE ";
  }
}
```

```php
// Créez une liste des animaux
$cat = new Cat();
$dog = new Dog();
$mouse = new Mouse();
$animals = array($cat, $dog, $mouse);

// Dites aux animaux de faire un son
foreach($animals as $animal) {
  $animal->makeSound();
}
?>
```

Exemple expliqué

Cat, Dog et Mouse sont toutes des classes qui implémentent l'interface Animal, ce qui signifie qu'elles sont toutes capables de produire un son en utilisant la méthode makeSound (). Pour cette raison, nous pouvons parcourir tous les animaux et leur dire de faire un son même si nous ne savons pas de quel type d'animal chacun est.

Puisque l'interface ne dit pas aux classes comment implémenter la méthode, chaque animal peut émettre un son à sa manière.

PHP OOP - Traits

PHP ne prend en charge que l'héritage unique: une classe enfant ne peut hériter que d'un seul parent.Alors, que se passe-t-il si une classe doit hériter de plusieurs comportements? Les traits POO résolvent ce problème.
Les traits sont utilisés pour déclarer des méthodes qui peuvent être utilisées dans plusieurs classes. Les traits peuvent avoir des méthodes et des méthodes abstraites qui peuvent être utilisées dans plusieurs classes, et les méthodes peuvent avoir n'importe quel modificateur d'accès (public, privé ou protégé).

Les traits sont déclarés avec le mot-**clé trait**:

```php
<?php
trait message1 {
  public function msg1() {
    echo "OOP est BIEN ";
  }
```

```
}
trait message2 {
  public function msg2() {
    echo "OOP réduit la duplication du code!";
  }
}

class Welcome {
  use message1;
}

class Welcome2 {
  use message1, message2;
}

$obj = new Welcome();
$obj->msg1();
echo "<br>";

$obj2 = new Welcome2();
$obj2->msg1();
$obj2->msg2();
?>
```

PHP OOP - Static Methods

Les méthodes statiques peuvent être appelées directement - sans créer d'abord une instance de la classe.Les méthodes statiques sont déclarées avec le mot-clé static:
Les méthodes statiques peuvent également être appelées à partir de méthodes d'autres classes. Pour ce faire, la méthode statique doit **être public**:

```
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}

class SomeOtherClass {
  public function message() {
```

```
    greeting::welcome();
  }
}
?>
```

Pour appeler une méthode statique à partir d'une classe enfant, utilisez le mot clé parent à l'intérieur de la classe enfant. Ici, la méthode statique peut être publique ou protégée public or protected.

```
<?php
class domain {
  protected static function getWebsiteName() {
    return "www.emena.org/SMIS6";
  }
}

class domainW3 extends domain {
  public $websiteName;
  public function __construct() {
    $this->websiteName = parent::getWebsiteName();
  }
}

$domainW3 = new domainW3;
echo $domainW3 -> websiteName;
?>
```

PHP OOP - Static Properties

Les propriétés statiques peuvent être appelées directement - sans créer d'instance de classe.

Les propriétés statiques sont déclarées avec le mot-clé **static:**

Une classe peut avoir à la fois des propriétés statiques et non statiques. Une propriété statique est accessible à partir d'une méthode de la même classe en utilisant le mot-clé **self** et le double deux-points (: :):

```
<?php
class pi {
  public static $value=3.14159;
}

class x extends pi {
```

```
  public function xStatic() {
    return parent::$value;
  }
}

// Récupère la valeur de la propriété statique directement via la classe enfant
echo x::$value;

// ou obtenir la valeur de la propriété statique via la méthode xStatic via xStatic() method
$x = new x();
echo $x->xStatic();
?>
```

Espaces de noms PHP Namespaces

Les espaces de noms sont des qualificatifs qui résolvent deux problèmes différents:

Ils permettent une meilleure organisation en regroupant des classes qui travaillent ensemble pour effectuer une tâche Ils permettent d'utiliser le même nom pour plus d'une classe. Par exemple, vous pouvez avoir un ensemble de classes qui décrivent un tableau HTML, tel que Table, Row et Cell, tout en ayant également un autre ensemble de classes pour décrire les meubles, tels que Table, Chair et Bed. Les espaces de noms peuvent être utilisés pour organiser les classes en deux groupes différents tout en empêchant également les deux classes Table et Table d'être mélangées.
Déclaration d'un espace de noms
Les espaces de noms sont déclarés au début d'un fichier à l'aide du mot-clé namespace:
Syntaxe

Déclarez un espace de noms appelé Html:

Remarque: Une déclaration d'espace de noms namespace doit être la première chose dans le fichier PHP. Le code suivant serait invalide:

```
<?php
echo "Hello World!"; Faux
namespace Html;
```

```
...
?>
```

Créez une classe Table dans l'espace de noms Html

```
<?php
namespace Html;
class Table {
  public $title = "";
  public $numRows = 0;
  public function message() {
    echo "<p>Table '{$this->title}' has {$this->numRows} rows.</p>";
  }
}
$table = new Table();
$table->title = "Ma table";
$table->numRows = 5;
?>

<!DOCTYPE html>
<html>
<body>

<?php
$table->message();
?>

</body>
</html>
```

Utilisation des espaces de noms

Tout code qui suit une déclaration d'espace de noms fonctionne à l'intérieur de l'espace de noms, de sorte que les classes qui appartiennent à l'espace de noms peuvent être instanciées sans aucun qualificatif. Pour accéder aux classes depuis l'extérieur d'un espace de noms, la classe doit avoir l'espace de noms qui lui est attaché.

Index.php

```
<?php
include "Html.php";

$table = new Html\Table();
$table->title = "My table";
$table->numRows = 5;

$row = new Html\Row();
$row->numCells = 3;
?>

<html>
<body>

<?php $table->message(); ?>
<?php $row->message(); ?>

</body>
</html>
```

HTML.php

```
<?php
namespace Html;
class Table {
  public $title = "";
  public $numRows = 0;

  public function message() {
    echo "<p>Table '{$this->title}' has {$this->numRows} rows.</p>";
  }
}

class Row {
  public $numCells = 0;
  public function message() {
    echo "<p>The row has {$this->numCells} cells.</p>";
  }
}
?>
```

PHP Iterables

Un iterable est toute valeur qui peut être bouclée avec une boucle foreach (). Le pseudo-type itérable a été introduit dans PHP 7.1, et il peut être utilisé comme type de données pour les arguments de fonction et les valeurs de retour de fonction.PHP - Utiliser Iterables

Le mot clé iterable peut être utilisé comme type de données d'un argument de fonction ou comme type de retour d'une fonction:

```
<?php
function printIterable(iterable $myIterable) {
  foreach($myIterable as $item) {
    echo $item;
  }
}

$arr = ["a", "b", "c"];
printIterable($arr);
?>
```

PHP - Création d'itérables

Arrays : Tous les tableaux sont itérables, donc n'importe quel tableau peut être utilisé comme argument d'une fonction qui nécessite un itérable.

Iterators : Tout objet qui implémente l'interface **Iterator** peut être utilisé comme argument d'une fonction qui nécessite un itérable.

Un itérateur contient une liste d'éléments et fournit des méthodes pour les parcourir. Il garde un pointeur sur l'un des éléments de la liste. Chaque élément de la liste doit avoir une clé qui peut être utilisée pour trouver l'élément.

Un itérateur doit avoir ces méthodes:

current () - Renvoie l'élément sur lequel le pointeur pointe actuellement. Il peut s'agir de n'importe quel type de données
key () Renvoie la clé associée à l'élément courant dans la liste. Il ne peut s'agir que d'un entier, d'un flottant, d'un booléen ou d'une chaîne
next () Déplace le pointeur vers l'élément suivant de la liste
rewind () Déplace le pointeur sur le premier élément de la liste
valid () Si le pointeur interne ne pointe vers aucun élément (par exemple, si next () a été appelé à la fin de la liste), cela doit renvoyer false. Il renvoie vrai dans tous les autres cas.

Example

```php
<?php
// Créer un itérateur
class MyIterator implements Iterator {
  private $items = [];
  private $pointer = 0;
  public function __construct($items) {
    // array_values()assure que les clés sont des nombres
    $this->items = array_values($items);
  }
  public function current() {
    return $this->items[$this->pointer];
  }
  public function key() {
    return $this->pointer;
  }
  public function next() {
    $this->pointer++;
  }
  public function rewind() {
    $this->pointer = 0;
  }
  public function valid() {
    // count()indique le nombre du éléments dans la liste
    return $this->pointer < count($this->items);
  }
}
// A function qui utilise des itérables
function printIterable(iterable $myIterable) {
  foreach($myIterable as $item) {
    echo $item;
  }
}
// Utilisez itérateur comme itérateur
$iterator = new MyIterator(["a", "b", "c"]);
printIterable($iterator);
?>
```

Partie 2 : PHP 5 & MySQLi

Système de base de données PHP + MySQL

Présentation

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

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

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

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

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

Qu'est-ce que MySQL?

MySQL est un système de base de données utilisé sur le web

MySQL est un système de base de données qui fonctionne sur un serveur

MySQL est idéal pour les petites et grandes applications

MySQL est très rapide, fiable et facile à utiliser

MySQL utilise SQL standard

MySQL compile sur un certain nombre de plateformes

MySQL est gratuit à télécharger et à utiliser

MySQL est développé, distribué et pris en charge par Oracle Corporation

MySQL porte le nom de la fille de Co-fondateur Monty Widenius: My

Les données d'une base de données MySQL sont stockées dans des tables. Une table est une collection de données connexes, et elle se compose de colonnes et de lignes.

Les bases de données sont utiles pour stocker des informations de manière catégorique. Une entreprise peut avoir une base de données avec les tables suivantes:

Des employés, Des produits ,Les clients ,Ordres

Requêtes de base de données

Une requête est une question ou une requête.

Nous pouvons interroger une base de données pour des informations spécifiques et avoir un jeu d'enregistrements retourné.

Regardez la requête suivante (en utilisant SQL standard): SELECT LastName FROM Employés

La requête ci-dessus sélectionne toutes les données de la colonne "Nom" de la table "Employés".

Pour en savoir plus sur SQL, visitez didacticiel SQL. Http://www.mysql.com/

MySQL est le système de base de données standard de facto pour les sites Web avec d'énormes volumes de données et d'utilisateurs finaux (comme Facebook, Twitter et Wikipedia).

Connexion PHP à MySQL

PHP 5 et versions ultérieures peuvent fonctionner avec une base de données MySQL en utilisant:
Extension **MySQLi** (**le 'i' signifie amélioré**)
PDO (objets de données PHP)
Les versions antérieures de PHP utilisaient l'extension MySQL. Cependant, cette extension est devenue obsolète en 2012.

Dois-je utiliser MySQLi ou PDO?

Si vous avez besoin d'une réponse courte, ce serait «Tout ce que vous voulez».
MySQLi et PDO ont tous deux leurs avantages:
PDO fonctionnera sur 12 systèmes de bases de données différents, alors que MySQLi ne fonctionnera qu'avec des bases de données MySQL.
Ainsi, si vous devez changer votre projet pour utiliser une autre base de données, PDO facilite le processus. Il vous suffit de modifier la chaîne de connexion et quelques requêtes. Avec MySQLi, vous devrez réécrire tout le code - requêtes incluses.
Les deux sont orientés objet, mais MySQLi propose également une API procédurale.
Les deux prennent en charge les déclarations préparées. Les instructions préparées protègent de l'injection SQL et sont très importantes pour la sécurité des applications Web.

Exemples dans ce cours seront MySQL dans la syntaxe MySQLi et PDO

Dans ce document, et dans les chapitres suivants, nous présentons trois façons de travailler avec PHP et MySQL:

MySQLi (orienté objet)
MySQLi (procédural)
PDO

Installation de MySQLi

Pour Linux et Windows: L'extension MySQLi est automatiquement installée dans la plupart des cas, lorsque le package php5 mysql est installé.

Pour plus d'informations sur l'installation, accédez à:
http://php.net/manual/en/mysqli.installation.php
Installation PDO

Ouvrez une connexion à MySQL

Avant de pouvoir accéder aux données de la base de données MySQL, nous devons pouvoir nous connecter au serveur:

Example (MySQLi Object-Oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Créer une connexion

$conn = new mysqli($servername, $username, $password);

// Vérifier la connexion

if ($conn->connect_error) {
  die("Connection echec: " . $conn->connect_error);
}
echo "Connection avec succès ";
?>
```

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Créer une connexion
$conn = mysqli_connect($servername, $username, $password);

// Vérifier la connexion
if (!$conn) {
  die("Connection echec: " . mysqli_connect_error());
}
echo " Connection avec succès ";
?>
```

Example (PDO)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
  $conn = new PDO("mysql:host=$servername;dbname=myDB", $username,
$password);
  // définir le mode erreur du PDO sur exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo " Connection avec succès ";
} catch(PDOException $e) {
  echo " Connection echec: " . $e->getMessage();
}
?>
```

Close the Connection

La connexion sera fermée automatiquement à la fin du script. Pour fermer la connexion avant, utilisez ce qui suit:

MySQLi Object-Oriented:

```
$conn->close();
```

MySQLi Procedural:

```
mysqli_close($conn);
```

PDO:

```
$conn = null;
```

PHP Create a MySQL Database

Une base de données se compose d'une ou de plusieurs tables.
Vous aurez besoin de privilèges CREATE spéciaux pour créer ou supprimer une base de données MySQL.

Créer une base de données MySQL à l'aide de MySQLi et PDO

L'instruction CREATE DATABASE est utilisée pour créer une base de données dans MySQL.
Les exemples suivants créent une base de données nommée 'myDB':

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// Creation de la connection
$conn = new mysqli($servername, $username, $password);
// verifier la connection
if ($conn->connect_error) {
  die("Connection echec " . $conn->connect_error);
}
// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
  echo "Database creation success";
```

```
} else {
  echo "Erreur creation database: " . $conn->error;
}
$conn->close();
?>
```

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Example (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
  $conn = new PDO("mysql:host=$servername", $username, $password);
  // PDO erreur mode pour exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "CREATE DATABASE myDBPDO";
```

```
  // utilise exec () car aucun résultat n'est renvoyé
  $conn->exec($sql);
  echo "Database creation success<br>";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}
$conn = null;
?>
```

PHP MySQL Create Table

Une table de base de données a son propre nom unique et se compose de colonnes et de lignes.

Créer une table MySQL à l aide de MySQLi et PDO

L'instruction CREATE TABLE est utilisée pour créer une table dans MySQL.
Nous allons créer une table nommée 'MyGuests', avec cinq colonnes:

'id', 'firstname', 'lastname', 'email' et 'reg_date':

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
```

Notes sur le tableau ci-dessus:

Le type de données spécifie le type de données que la colonne peut contenir.
Pour une référence complète de tous les types de données disponibles, consultez notre référence sur les types de données.
Après le type de données, vous pouvez spécifier d'autres attributs facultatifs pour chaque colonne:

NOT NULL - Chaque ligne doit contenir une valeur pour cette colonne, les valeurs nulles ne sont pas autorisées
V DEFAULT value valeur par défaut - Définit une valeur par défaut qui est ajoutée lorsqu'aucune autre valeur n'est transmise
UNSIGNED - Utilisé pour les types de nombres, limite les données stockées aux

nombres positifs et à zéro
INCREMENT AUTO - MySQL augmente automatiquement la valeur du champ de 1 à chaque fois qu'un nouvel enregistrement est ajouté
PRIMARY KEY - Utilisé pour identifier de manière unique les lignes d'une table. La colonne avec le paramètre PRIMARY KEY est souvent un numéro d'identification et est souvent utilisée avec AUTO_INCREMENT

Chaque table doit avoir une colonne de clé primaire (dans ce cas: la colonne «id»). Sa valeur doit être unique pour chaque enregistrement de la table.

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Creation connection
$conn = new mysqli($servername, $username, $password, $dbname);
// verifier connection
if ($conn->connect_error) {
  die("Connection echec: " . $conn->connect_error);
}
// sql de creation de la table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)";
if ($conn->query($sql) === TRUE) {
  echo "Table MyGuests creation success";
} else {
  echo "Erreur creation de la table: " . $conn->error;
}
$conn->close();
?>
```

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Creation connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// verifier la connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
// sql creation la table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)";
if (mysqli_query($conn, $sql)) {
  echo "Table MyGuests creation success";
} else {
  echo "Erreur creation la table: " . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Example (PDO)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  // PDO erreur mode pour l' exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  // sql to creation la table
```

```
  $sql = "CREATE TABLE MyGuests (
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  firstname VARCHAR(30) NOT NULL,
  lastname VARCHAR(30) NOT NULL,
  email VARCHAR(50),
  reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
  )";
  $conn->exec($sql);
  echo "Table MyGuests creation avec success";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}
$conn = null;
?>
```

PHP MySQL Insert Data

Insert Data dans MySQL en utilisant MySQLi et PDO

Une fois qu'une base de données et une table ont été créées, nous pouvons commencer à y ajouter des données.

Voici quelques règles de syntaxe à suivre:

La requête SQL doit être citée en PHP
Les valeurs de chaîne à l'intérieur de la requête SQL doivent être entre guillemets, Les valeurs numériques ne doivent pas être entre guillemets
Le mot NULL ne doit pas être entre guillemets, L'instruction INSERT INTO est utilisée pour ajouter de nouveaux enregistrements à une table MySQL:

```
INSERT INTO table_name (column1, column2, column3,...) VALUES (value1, value2, value3,...)
```

Dans le chapitre précédent, nous avons créé une table vide nommée «MyGuests» avec cinq colonnes: «id», «firstname», «lastname», «email» et «reg_date». Maintenant, remplissons le tableau avec des données.

Example (MySQLi Object-oriented)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Creation connection
$conn = new mysqli($servername, $username, $password, $dbname);
// verifier connection
if ($conn->connect_error) {
  die("Connection echeck: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Jamal', 'Selami', 'jamal@example.com')";
if ($conn->query($sql) === TRUE) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>
```

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Creation connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
//verifier connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Jamal', 'Selami', 'jamal@example.com')";
if (mysqli_query($conn, $sql)) {
  echo " creation avec success";
} else {
```

```
  echo "Erreurr: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Example (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  // PDO erreur mode exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('Jamal', 'Selami', 'jamal@example.com')";  $conn->exec($sql);
  echo "Ajout  creation success";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}
$conn = null;
?>
```

PHP MySQL Obtenir le dernier ID inséré

Si nous effectuons un INSERT ou UPDATE sur une table avec un champ AUTO_INCREMENT, nous pouvons obtenir immédiatement l'ID du dernier enregistrement inséré / mis à jour.

Dans la table 'MyGuests', la colonne 'id' est un champ AUTO_INCREMENT:

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
```

```php
// Creation de la  connection
$conn = new mysqli($servername, $username, $password, $dbname);
// verification de laconnection
if ($conn->connect_error) {
  die("Connection faute: " . $conn->connect_error);
}
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Jamal', 'Selami', 'jamal@example.com')";
if ($conn->query($sql) === TRUE) {
  $last_id = $conn->insert_id;
  echo " creation success. Last inserted ID is: " . $last_id;
} else {
  echo "Erreur: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>
```

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Creaion de la connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// test de la connection
if (!$conn) {
  die("Connection echec: " . mysqli_connect_error());
}
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Jamal', 'Selami', 'jamal@example.com')";
if (mysqli_query($conn, $sql)) {
  $last_id = mysqli_insert_id($conn);
  echo " creation success. Last inserted ID is: " . $last_id;
} else {
  echo "Erreur: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Example (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  // PDO exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('Jamal', 'Selami', 'jamal@example.com')";
  //
  $conn->exec($sql);
  $last_id = $conn->lastInsertId();
  echo "creation  success. Last inserted ID is: " . $last_id;
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}
$conn = null;
?>
```

PHP MySQL Insérer plusieurs enregistrements

Plusieurs instructions SQL doivent être exécutées avec la fonction mysqli_multi_query ().

Les exemples suivants ajoutent trois nouveaux enregistrements à la table 'MyGuests':

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
```

```
$username = "username";
$password = "password";
$dbname = "myDB";

// Creation la connection
$conn = new mysqli($servername, $username, $password, $dbname);
// verifier la  connection
if ($conn->connect_error) {
  die("Connection faute: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Jamal', 'Selami', 'jamal@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mohamed', 'Moussa', 'moussamohamed@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('HAssane', 'Dolmi', 'dolmi@example.com')";

if ($conn->multi_query($sql) === TRUE) {
  echo "success";
} else {
  echo "Erreur: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

Notez que chaque instruction SQL doit être séparée par un point-virgule

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Creer la connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// verifier la connection
if (!$conn) {
```

```
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Jamal', 'Selami', 'jamal@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mohamed', 'Moussa', 'moussamohamed@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('HAssane', 'Dolmi', 'dolmi@example.com')";

if (mysqli_multi_query($conn, $sql)) {
  echo " success";
} else {
  echo "Erreur: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

La méthode PDO est un peu différente::

Example (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);  $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
// commence la transaction
  $conn->beginTransaction();
// nos instructions SQL
  $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('Jamal', 'Selami', 'jamal@example.com')");
  $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('Mohamed', 'Moussa', 'moussamohamed@example.com')");
```

```
  $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('HAssane', 'Dolmi', 'dolmi@example.com')");

  // commence la transaction

  $conn->commit();
  echo " success";
} catch(PDOException $e) {
// annule la transaction si quelque chose a échoué
  $conn->rollback();
  echo "Erreur: " . $e->getMessage();
}

$conn = null;
?>
```

Instructions préparées par PHP MySQL

Les instructions préparées sont très utiles contre **les injections SQL.**

Instructions préparées et paramètres liés.

Une instruction préparée est une fonctionnalité utilisée pour exécuter les mêmes instructions SQL (ou similaires) à plusieurs reprises avec une grande efficacité.

Les instructions préparées fonctionnent essentiellement comme ceci:

Préparation: un modèle d'instruction SQL est créé et envoyé à la base de données. Certaines valeurs ne sont pas spécifiées, appelées paramètres (étiquetés «?»). Exemple: INSÉRER DANS LES VALEURS MyGuests (?,?,?)
La base de données analyse, compile et exécute l'optimisation des requêtes sur le modèle d'instruction SQL et stocke le résultat sans l'exécuter
Exécuter: ultérieurement, l'application lie les valeurs aux paramètres et la base de données exécute l'instruction. L'application peut exécuter l'instruction autant de fois qu'elle le souhaite avec des valeurs différentes

Par rapport à l'exécution directe d'instructions SQL, les instructions préparées présentent trois avantages principaux:

Les instructions préparées réduisent le temps d'analyse car la préparation de la requête n'est effectuée qu'une seule fois (bien que l'instruction soit exécutée plusieurs fois)
Les paramètres liés minimisent la bande passante vers le serveur car vous ne devez envoyer que les paramètres à chaque fois, et non la requête entière
Les instructions préparées sont très utiles contre les injections SQL, car les valeurs de paramètres, qui sont transmises ultérieurement à l'aide d'un protocole différent, n'ont pas besoin d'être correctement échappées. Si le modèle d'instruction d'origine n'est pas dérivé d'une entrée externe, l'injection SQL ne peut pas se produire.

La sécurité

La faille SQLi, abréviation de SQL Injection, soit injection SQL en français, est un groupe de méthodes d'exploitation de faille de sécurité d'une application interagissant avec une base de données. Elle permet d'injecter dans la requête SQL en cours un morceau de requête non prévu par le système et pouvant en compromettre la sécurité.

Considérons un site web dynamique (programmé en PHP dans cet exemple) qui dispose d'un système permettant aux utilisateurs possédant un nom d'utilisateur et un mot de passe valides de se connecter. Ce site utilise la requête SQL suivante pour identifier un utilisateur :

```
SELECT uid FROM Users WHERE name = '(nom)' AND password = '(mot de
passe hashé)';
```

L'utilisateur Dupont souhaite se connecter avec son mot de passe « truc » hashé en MD5. La requête suivante est exécutée :

```
SELECT uid FROM Users WHERE name = 'Dupont' AND password =
'45723a2af3788c4ff17f8d1114760e62';
```

Attaquer la requête

Imaginons à présent que le script PHP exécutant cette requête ne vérifie pas les données entrantes pour garantir sa sécurité. Un hacker pourrait alors fournir les informations suivantes :

- Utilisateur : `Dupont';--`
- Mot de passe : n'importe lequel

La requête devient :

```
SELECT uid FROM Users WHERE name = 'Dupont';--' AND password =
'4e383a1918b432a9bb7702f086c56596e';
```

Les caractères -- marquent le début d'un commentaire en SQL. La requête est donc équivalente à :

```
SELECT uid FROM Users WHERE name = 'Dupont';
```

L'attaquant peut alors se connecter sous l'utilisateur Dupont avec n'importe quel mot de passe. Il s'agit d'une injection de SQL réussie, car l'attaquant est parvenu à injecter les caractères qu'il voulait pour modifier le comportement de la requête.

Supposons maintenant que l'attaquant veuille non pas tromper le script SQL sur le nom d'utilisateur, mais sur le mot de passe. Il pourra alors injecter le code suivant :

- Utilisateur : Dupont
- Mot de passe : `' or 1 --`

L'apostrophe indique la fin de la zone de frappe de l'utilisateur, le code « `or 1` » demande au script si 1 est vrai, or c'est toujours le cas, et -- indique le début d'un commentaire.

La requête devient alors :

```
SELECT uid FROM Users WHERE name = 'Dupont' AND password = '' or 1 --';
```

Ainsi, le script programmé pour vérifier si ce que l'utilisateur tape est vrai, il verra que 1 est vrai, et l'attaquant sera connecté sous la session Dupont.

Example (MySQLi avec déclarations préparées)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// connection
$conn = new mysqli($servername, $username, $password, $dbname);
// verifier connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
// préparer et lier
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// définir les paramètres et exécuter
$firstname = "Jamal";
$lastname = "Selami";
$email = "jamal@example.com";
$stmt->execute();

$firstname = "Mohamed";
$lastname = "Moussa";
$email = "moussamohamed@example.com";
$stmt->execute();

$firstname = "Majid";
$lastname = "Dolmi";
$email = "dolmi@example.com";
$stmt->execute();

echo " success";

$stmt->close();
$conn->close();
?>
```

Lignes de code à expliquer à partir de l'exemple ci-dessus:

"Dans notre SQL, nous insérons un point d'interrogation (?) Où nous voulons substituer une valeur entière, chaîne, double ou blob.

Ensuite, jetez un œil à la fonction bind_param ():INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)"

```
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

Cette fonction lie les paramètres à la requête SQL et indique à la base de données quels sont les paramètres. L'argument 'sss' répertorie les types de données que sont les paramètres. Le caractère s indique à mysql que le paramètre est une chaîne.

L'argument peut être de quatre types:

- i - integer
- d - double
- s - string
- b - BLOB

Nous devons en avoir un pour chaque paramètre.

En indiquant à mysql à quel type de données s'attendre, nous minimisons le risque d'injections SQL.

Remarque: Si nous voulons insérer des données provenant de sources externes (comme l'entrée utilisateur), il est très important que les données soient nettoyées et validées.

Prepared Statements in PDO

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
```

```
  // PDO  exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // preparer sql parameters lies
  $stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES (:firstname, :lastname, :email)");
  $stmt->bindParam(':firstname', $firstname);
  $stmt->bindParam(':lastname', $lastname);
  $stmt->bindParam(':email', $email);

  // insert a row
  $firstname = "Jamal";
  $lastname = "Selami";
  $email = "selami@example.com";
  $stmt->execute();

  // insert autre row
  $firstname = "Moussa";
  $lastname = "Mohamede";
  $email = "mmoussa@example.com";
  $stmt->execute();

  // insert another row
  $firstname = "Majid";
  $lastname = "Dolmi";
  $email = "dolmi@example.com";
  $stmt->execute();

   echo " success";
} catch(PDOException $e) {
   echo "Erreur: " . $e->getMessage();
}
$conn = null;
?>
```

Sélection de Données PHP MySQL

L'instruction SELECT est utilisée pour sélectionner des données dans une ou plusieurs tables: SELECT column_name(s) FROM table_name

Ou nous pouvons utiliser le caractère * pour sélectionner TOUTES les colonnes d'une table: SELECT * FROM table_name

Sélectionnez des données avec MySQLi

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Creation de la connection
$conn = new mysqli($servername, $username, $password, $dbname);
// verification de la  connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // sortie des donneés pour chaque row
  while($row = $result->fetch_assoc()) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"].
"<br>";
  }
} else {
  echo "0 results";
}
$conn->close();
?>
```

Lignes de code à expliquer à partir de l'exemple ci-dessus:

Tout d'abord, nous configurons une requête SQL qui sélectionne les colonnes id, firstname et lastname de la table MyGuests. La ligne de code suivante exécute la requête et place les données résultantes dans une variable appelée $ result.
Ensuite, la fonction num_rows () vérifie s'il y a plus de zéro ligne retournée. S'il y a plus de zéro ligne retournée, la fonction fetch_assoc () place tous les résultats dans un tableau associatif que nous pouvons parcourir. La boucle while () parcourt l'ensemble de résultats et génère les données des colonnes id, firstname et lastname.

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Creation de  connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// verifier la  connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
  // sortie pour chaque row
  while($row = mysqli_fetch_assoc($result)) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"].
"<br>";
  }
} else {
  echo "0 results";
}

mysqli_close($conn);
?>
```

Sélectionner les données avec PDO + instructions préparées

Example (PDO)

```
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";
class TableRows extends RecursiveIteratorIterator {
  function __construct($it) {
    parent::__construct($it, self::LEAVES_ONLY);
  }
  function current() {
    return "<td style='width:150px;border:1px solid black;'>" . parent::current(). "</td>";
  }
  function beginChildren() {
    echo "<tr>";
  }
  function endChildren() {
    echo "</tr>" . "\n";
  }
}
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests");
  $stmt->execute();
  // définir le tableau résultant sur associatif
  $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
  foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v) {
    echo $v;
  }
} catch(PDOException $e) {
  echo "Erreur: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

PHP MySQL utilise la clause WHERE

La clause WHERE est utilisée pour filtrer les enregistrements. La clause WHERE est utilisée pour extraire uniquement les enregistrements qui remplissent une condition spécifiée.

SELECT column_name(s) FROM table_name WHERE column_name operator value

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Creation de la  connection
$conn = new mysqli($servername, $username, $password, $dbname);
// verification de la connection
if ($conn->connect_error) {
  die("Connection faute: " . $conn->connect_error);
}
$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Selami'";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
  // sortie donnée de chaque  row
  while($row = $result->fetch_assoc()) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"].
"<br>";
  }
} else {
  echo "0 resultats";
}
$conn->close();
?>
```

function num_rows () vérifie s'il y a plus de zéro ligne retournée

S'il y a plus de zéro ligne retournée, la fonction fetch_assoc () place tous les résultats dans un tableau associatif que nous pouvons parcourir. La boucle while () parcourt l'ensemble de résultats et génère les données des colonnes id, firstname et lastname.

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Dolme'";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
  // output data of each row
  while($row = mysqli_fetch_assoc($result)) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"].
"<br>";
  }
} else {
  echo "0 results";
}

mysqli_close($conn);
?>
```

Example (PDO)

```
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator {
  function __construct($it) {
    parent::__construct($it, self::LEAVES_ONLY);
  }
```

```
  function current() {
    return "<td style='width:150px;border:1px solid black;'>" . parent::current(). "</td>";
  }

  function beginChildren() {
    echo "<tr>";
  }

  function endChildren() {
    echo "</tr>" . "\n";
  }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests WHERE
lastname='Dolmi'");
  $stmt->execute();

  // définir le tableau résultant sur associatif
  $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
  foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v) {
    echo $v;
  }
}
catch(PDOException $e) {
  echo "Erreur: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

PHP MySQL utilise la clause ORDER BY

La clause ORDER BY est utilisée pour trier le jeu de résultats par ordre croissant ou décroissant.
La clause ORDER BY trie les enregistrements dans l'ordre croissant par défaut.
Pour trier les enregistrements par ordre décroissant, utilisez le mot clé DESC.

SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC

(MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Creation de la connection
$conn = new mysqli($servername, $username, $password, $dbname);
// test connection
if ($conn->connect_error) {
  die("Connection faute: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests ORDER BY lastname";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // sortie raw
  while($row = $result->fetch_assoc()) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"].
"<br>";
  }
} else {
  echo "0 resultats";
}
$conn->close();
?>
```

MySQLi Procedural

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = mysqli_connect($servername, $username, $password, $dbname);
if (!$conn) {
  die("Connection faute: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests ORDER BY lastname";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {  while($row = mysqli_fetch_assoc($result)) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"].
"<br>";
  }
} else {
  echo "0 results";
}
mysqli_close($conn);
?>
```

PDO

```
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator {
  function __construct($it) {
    parent::__construct($it, self::LEAVES_ONLY);
  }

  function current() {
    return "<td style='width:150px;border:1px solid black;'>" . parent::current(). "</td>";
  }
```

```
  function beginChildren() {
    echo "<tr>";
  }

  function endChildren() {
    echo "</tr>" . "\n";
  }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests ORDER
BY lastname");
  $stmt->execute();

  $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
  foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v) {
    echo $v;
  }
} catch(PDOException $e) {
  echo "Errurr: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

PHP MySQL Supprimer les données

L'instruction DELETE est utilisée pour supprimer des enregistrements d'une table:

```
DELETE FROM table_name
WHERE some_column = some_value
```

Notez la clause WHERE dans la syntaxe DELETE: La clause WHERE spécifie le ou les enregistrements à supprimer. Si vous omettez la clause WHERE, tous les enregistrements seront supprimés!

Les exemples suivants suppriment l'enregistrement avec id = 3 dans la table 'MyGuests':

MySQLi Object-oriented

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// sql pour supprimer l'enregistrement
 $sql = "DELETE FROM MyGuests WHERE id=3";

if ($conn->query($sql) === TRUE) {
  echo "Enregistrement supprimé avec succès";
} else {
  echo "Erreur lors de la suppression de l' enregistrement: " . $conn->error;
}

$conn->close();
?>
```

MySQLi Procedural

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
```

```php
// Creation connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
if (!$conn) {
  die("Connection faute: " . mysqli_connect_error());
}
// sql supprime les données
$sql = "DELETE FROM MyGuests WHERE id=3";

if (mysqli_query($conn, $sql)) {
  echo "suppression avec success";
} else {
  echo "Erreur suppression: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (PDO)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);  $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
  $sql = "DELETE FROM MyGuests WHERE id=3";

  $conn->exec($sql);
  echo "suppression success";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

Mise à jour des Données de PHP MySQL

L'instruction UPDATE est utilisée pour mettre à jour les enregistrements existants dans une table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

MySQLi Object-oriented

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
  die("Connection faute: " . $conn->connect_error);
}

$sql = "UPDATE MyGuests SET lastname='Dolmi' WHERE id=2";

if ($conn->query($sql) === TRUE) {
  echo " updated success";
} else {
  echo "Erreur de updating: " . $conn->error;
}

$conn->close();
?>
```

MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
```

```php
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if (mysqli_query($conn, $sql)) {
  echo "Record updated successfully";
} else {
  echo "Error updating record: " . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Example (PDO)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);  $conn->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
  $sql = "UPDATE MyGuests SET lastname='Dolmie' WHERE id=2";
  // Preparer instruction statement
  $stmt = $conn->prepare($sql);
  // executer la requete
  $stmt->execute();
  // echo un message pour dire que la mise à jour a réussi
  echo $stmt->rowCount() . " records UPDATED successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}
$conn = null;
?>
```

Limite de Sélections de données de PHP MySQL

MySQL fournit une clause LIMIT qui est utilisée pour spécifier le nombre d'enregistrements à renvoyer.

La clause LIMIT facilite le codage des résultats de plusieurs pages ou de la pagination avec SQL, et est très utile sur les grandes tables. Le renvoi d'un grand nombre d'enregistrements peut avoir un impact sur les performances.

Supposons que nous souhaitons sélectionner tous les enregistrements de 1 à 30 (inclus) dans une table appelée «Commandes». La requête SQL ressemblerait alors à ceci:

```
$sql = "SELECT * FROM Orders LIMIT 30";
```

Lorsque la requête SQL ci-dessus est exécutée, elle renverra les 30 premiers enregistrements.
Que faire si nous voulons sélectionner les enregistrements 16 - 25 (inclus)?
Mysql fournit également un moyen de gérer cela: en utilisant OFFSET.
La requête SQL ci-dessous dit `` ne renvoie que 10 enregistrements, commencez à l'enregistrement 16 (OFFSET 15) '':

```
$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";
```

On peut également utiliser une syntaxe plus courte pour obtenir le même résultat:

```
$sql = "SELECT * FROM Orders LIMIT 15, 10";
```

Notez que les nombres sont inversés lorsque vous utilisez une virgule.