DOCUMENTATION

I. *Variable Super Global*

• Les variables super globales sont des variables particulières pour trois raisons :

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

Les principales variables super globales qui existent :

- ✓ $_SERVER : ce sont des valeurs renvoyées par le serveur. Elles sont nombreuses et quelques-unes d'entre elles peuvent nous être d'une grande utilité. Je vous propose de retenir au moins $_SERVER['REMOTE_ADDR']. Elle nous donne l'adresse IP du client qui a demandé à voir la page. Cela peut être utile pour l'identifier.
- ✓ $_ENV : ce sont des variables d'environnement, toujours données par le serveur. C'est le plus souvent sous des serveurs Linux que l'on retrouve des informations dans cette superglobale. Généralement, on ne trouvera rien de bien utile là-dedans pour notre site web.
- ✓ $_SESSION : on y retrouve les variables de session. Ce sont des variables qui restent stockées sur le serveur le temps de la visite d'un visiteur. Nous allons apprendre à nous en servir dans ce chapitre.
- ✓ $_COOKIE : contient les valeurs des cookies enregistrés sur l'ordinateur du visiteur. Cela nous permet de stocker des informations sur l'ordinateur du visiteur pendant plusieurs mois par exemple pour se souvenir de son nom.
- ✓ $_GET : vous la connaissez, elle contient les données envoyées en paramètre dans l'URL
- ✓ $_POST : de même, c'est une variable que vous connaissez qui contient les informations qui viennent d'être envoyées par un formulaire.
- ✓ $_FILES : elle contient la liste des fichiers qui ont été envoyés via le formulaire précédent.

Exemple de super globale :

❖ $_POST :

$_POST -- $HTTP_POST_VARS [deprecated] — HTTP POST variables

Tableau associatif de variables transmises au script actuel via la méthode HTTP POST lors de l'utilisation de *application / x-www-form-urlencoded* ou *multipart / form-data* comme HTTP Content-Type dans la demande.

Exemples

Exemple # 1 $ _POST exemple

```php
<?php
echo 'Hello ' . htmlspecialchars($_POST["name"]) . '!';
?>
```

En supposant que l'utilisateur POSTed nom = Hannes

L'exemple ci-dessus produira quelque chose de similaire à:

```
Bonjour Hannes!
```

Notes

Note:
Il s'agit d'une variable «superglobale», ou globale automatique. Cela signifie simplement qu'il est disponible dans toutes les étendues d'un script. Il n'est pas nécessaire de faire **une variable $ globale;** pour y accéder dans des fonctions ou des méthodes.

- $_GET :

$ _GET - $ HTTP_GET_VARS [obsolète] - Variables HTTP GET

Un tableau associatif de variables passées au script actuel via les paramètres URL (aka. Chaîne de requête). Notez que le tableau est non seulement rempli pour les demandes GET, mais plutôt pour toutes les demandes avec une chaîne de requête.

Exemple # 1 $ _GET exemple

```php
<?php
echo 'Hello ' . htmlspecialchars($_GET["name"]) . '!';
?>
```

En supposant que l'utilisateur a saisi http://example.com/?name=Hannes

L'exemple ci-dessus produira quelque chose de similaire à:

```
Bonjour Hannes!
```

Notes

Note:
Il s'agit d'une variable «superglobale», ou globale automatique. Cela signifie simplement qu'il est disponible dans toutes les étendues d'un script. Il n'est pas nécessaire de faire **une variable $ globale;** pour y accéder dans des fonctions ou des méthodes.

- $_SESSTION

DOCUMENTATION

- $ _SESSION - $ HTTP_SESSION_VARS [obsolète] - Variables de session
- Un tableau associatif contenant des variables de session disponibles pour le script actuel. Reportez-vous à la documentation des fonctions de session pour plus d'informations sur leur utilisation.
- $ HTTP_SESSION_VARS contient les mêmes informations initiales, mais n'est pas un super global . (Notez que $ HTTP_SESSION_VARS et $ _SESSION sont des variables différentes et que PHP les gère comme telles)
- Pour utiliser une session vous devez connaître 2 fonctions :

  1. session_start() : démarre le système de sessions. Si le visiteur vient d'arriver sur le site, alors un numéro de session est généré pour lui. Vous devez appeler cette fonction au tout début de chacune des pages où vous avez besoin des variables de session.

  2. session_destroy() : ferme la session du visiteur. Cette fonction est automatiquement appelée lorsque le visiteur ne charge plus de page de votre site pendant plusieurs minutes (c'est le timeout), mais vous pouvez aussi créer une page "Déconnexion" si le visiteur souhaite se déconnecter manuellement.

Note:

Il s'agit d'une variable «super globale», ou globale automatique. Cela signifie simplement qu'il est disponible dans toutes les étendues d'un script. Il n'est pas nécessaire de faire une variable $ globale; pour y accéder dans des fonctions ou des méthodes.

```
Creating New Session
========================
<?php
session_start();
/*session is started if you don't write this line can't use $_Session  global variable*/
$_SESSION["newsession"]=$value;
?>
Getting Session
========================
<?php
session_start();
/*session is started if you don't write this line can't use $_Session  global variable*/
$_SESSION["newsession"]=$value;
/*session created*/
echo $_SESSION["newsession"];
/*session was getting*/
?>
Updating Session
========================
<?php
session_start();
/*session is started if you don't write this line can't use $_Session  global variable*/
$_SESSION["newsession"]=$value;
/*it is my new session*/
$_SESSION["newsession"]=$updatedvalue;
/*session updated*/
?>
Deleting Session
========================
<?php
session_start();
/*session is started if you don't write this line can't use $_Session  global variable*/
$_SESSION["newsession"]=$value;
unset($_SESSION["newsession"]);
/*session deleted. if you try using this you've got an error*/
?>

Reference: http://gencbilgin.net/php-session-kullanimi.html
```

II. *Connexion de base de données (PHP et MYSQL) :*

a) Objet PDO pour la connexion à une base de données en PHP

PDO signifie PHP Data Objects. Il s'agit d'une interface qui permet au scripts PHP d'interroger une base de données via des requêtes SQL.

PDO est une extension qui s'ajoute au PHP pour que ses différentes fonctionnalités soient disponibles dans le langage. Il constitue une interface d'abstraction de la base de données, c'est à dire qu'on peut utiliser l'ensemble de ses fonctions pour exécuter des requêtes SQL quel que soit le SGBD. Autrement dit, si l'application Web repose sur le SGBD MySQL, on peut migrer vers le SGBD PostgreSQL sans modifier le code source (quelques modifications mineures sont requises).

L'abstraction de la base de données constitue un point fort par rapport aux anciennes méthodes d'accès à celles-ci. D'ailleurs, il constitue l'ultime avantage du PDO, sans en être le seul.

Le bon fonctionnement de PDO repose sur la disponibilité du pilote de la base de données. Il faut que celui-ci soit pris en charge pour pouvoir interroger le SGBD souhaité.

Pour déclarer le pilote du SBGD MySQL par exemple, il faut aller dans le fichier **php.ini** et ajouter la

b) Gestion d'erreurs éventuelles lors de l'instanciation (Exception PDOException)

Si les paramètres renseignés au constructeur de la classe PDO sont corrects et que le serveur de base de données fonctionne normalement alors l'objet $pdo sera créé sans encombre. Par contre si les paramètres ne sont pas bien déclarés ou que le serveur de base de données n'est pas accessible alors une exception de type **PDOException** (vue dans la page précédente) est lancée automatiquement. Il faut donc la rattraper comme ceci:

```
<?php
   try{
      $pdo=new
PDO("mysql:host=localhost;dbname=mabase","user","1234")
;
   }
   catch(PDOException $e){
      echo $e->getMessage();
   }
?>
```

On imagine que si notre projet Web est composé de plusieurs pages, alors la plupart d'entre elles aura besoin de dialoguer avec la base de données. Le code précédent sera donc déclaré dans chacune d'elles. Il faut alors songer à mettre ce code dans un fichier à part qu'on inclura au besoin (à l'aide de **include()** ou **require()**...).

c) Préparation et exécution des requêtes (Requêtes préparées)

L'un des points fort de PDO est les **requêtes préparées.** En effet, une requête préparée est plus rapide et, aussi, plus sûre (surtout contre les tentatives d'injections SQL).

Pour préparer une requête, il est recommandé de ne pas y spécifier les valeurs, mais plutôt remplacer celle-ci par des marqueurs interrogatifs (?) ou des paramètres nommés.

Pour mieux comprendre, supposons que l'on veut insérer dans la table "utilisateurs" les valeurs suivantes:
- Nom: Einstein
- Prénom: Albert
- Login: a.einstein
- Mot de passe: 2020

Nous allons donc préparer la requête puis l'exécuter comme ceci:

```
<?php
   $ins = $pdo->prepare("insert into utilisateurs
(nom,prenom,login,pass) values(?,?,?,?)");
   $ins->execute(array(
      "Einstein",
      "Albert",
      "a.einstein",
      md5("2020")
   ));
?>
```

La méthode **prepare()** de l'objet PDO permet de préparer une requête. Elle accepte comme paramètre une chaîne de caractères qui correspond à la requête souhaitée. Les marqueurs interrogatifs sont là pour désigner les valeurs à passer à la requête. Ces valeurs seront ensuite passées lors de l'exécution de la requête. $ins est l'objet correspondant à la requête préparée**.**

La méthode **execute()** est appelée par l'objet $ins qui correspond à la requête préparée. Elle accepte comme paramètre une variable de type tableau qui contient les valeurs qui remplaceront, dans l'ordre, les marqueurs interrogatif. Première entrée pour le premier marqueur, deuxième entrée pour le deuxième marqueur et ainsi de suite.

Si la requête préparée ne contient aucun marqueur interrogatif alors la méthode **execute()** est laissée vide.

Vous avez sans doute remarqué que le mot de passe a été haché par la fonction md5(). Il ne faut jamais stocker les mots de passe en claire (ni dans un fichier, ni dans la base de données).

Le traitement précédent peut être fait à l'aide des paramètres nommés au lieux des marqueurs interrogatifs comme ceci:

DOCUMENTATION

```
<?php
   $ins = $pdo->prepare("insert into utilisateurs
(nom,prenom,login,pass)
values(:nom,:prenom,:login,:pass)");
   $ins->execute(array(
      ":prenom"=>"Albert",
      ":pass"=>md5("2020"),
      ":nom"=>"Einstein",
      ":login"=>"a.einstein"
   ));
?>
```

Dans la requête préparée, on déclare à la place des valeurs des séquences qui commencent par deux points (:nom ou :prenom par exemple). Lors de l'exécution, on passe en paramètre un tableau associatif dont les clés sont les paramètres nommés (en tant que chaînes de caractères). Les valeurs du tableau seront les valeurs que l'on veut passer à la requête.

Cette technique ne nous oblige pas de déclarer les valeurs au sein du tableau passé lors de l'exécution dans l'ordre établi dans la requête. C'est utile quand la requête contient un nombre important de paramètres, ce qui rend le respect de l'ordre peu compliqué.

De la même manière on peut passer des requêtes de modification, suppression, vidage de la table...

Exemple de suppression de l'enregistrement inséré:

```
<?php
   $ins = $pdo->prepare("delete from utilisateurs where
id=? limit 1");
   $ins->execute(array(1));
?>
```

Exemple d'insertion:

```
<?php
   $ins = $pdo->prepare("select * from etidiant where
id=? limit 1");
   $ins->execute(array(1));
?>
```

Maintenant, il faut récupérer les enregistrements dans une variable PHP. En fait, on va récupérer toute la table dans une seule variable. Alors, il est évident que la variable sera un tableau à deux dimensions. Une pour désigner la ligne et l'autre pour désigner la colonne

```
<?php
   $tab = $ins->fetchAll();
?>
```

```
<?php
   while($tab = $ins->fetch()){};
?>
```

- La méthode **fetchAll()** appelée par l'objet requête retourne un tableau contenant toutes les lignes retournées par la requête.
- La méthode **fetch()** appelée par l'objet requête retourne un tableau contenant une ligne retournées par la requête, pour cela on fait la boucle pour récupérer toutes les lignes de la requête.

  Désormais que nous disposons de la variable $tab, on peut en faire ce qu'on veut. Nous avons choisi d'afficher toutes les lignes sur la page.

Le code ressemblerait donc à ceci:

```
<?php
   for($i=0;$i<count($tab);$i++){
      echo implode(" | ",$tab[$i])."<br />";
   }
?>

<?php
   while($tab = $ins->fetch()){
      echo $tab['id']. $tab['nom']."<br />";
   }
?>
```

III. *Fonctions de manipulation variables*

- Isset()

isset - Détermine si une variable est déclarée et est différente de NULL

```
isset ( mixte $var [, mixte $... ]): bool
```

Déterminer si une variable est considérée comme définie, cela signifie si une variable est déclarée et est différente de **NULL**.

Si une variable n'a pas été définie avec la fonction unset () , elle n'est plus considérée comme définie.

isset () retournera **FALSE** lors de la vérification d'une variable qui a été affectée à **NULL**. Notez également qu'un caractère nul ("\ 0") n'est pas équivalent à la **NULL** constante PHP.

Si plusieurs paramètres sont fournis, **isset ()** ne retournera **TRUE** que si tous les paramètres sont considérés comme définis. L'évaluation va de gauche à droite et s'arrête dès qu'une variable non définie est rencontrée.

DOCUMENTATION

Exemple # 1 isset () Exemples

```php
<?php

$var = '';

// This will evaluate to TRUE so the text will be printed.
if (isset($var)) {
    echo "This var is set so I will print.";
}

// In the next examples we'll use var_dump to output
// the return value of isset().

$a = "test";
$b = "anothertest";

var_dump(isset($a));      // TRUE
var_dump(isset($a, $b)); // TRUE

unset ($a);

var_dump(isset($a));     // FALSE
var_dump(isset($a, $b)); // FALSE

$foo = NULL;
var_dump(isset($foo));   // FALSE

?>
```

- Empty()

empty - Détermine si une variable est vide

```
vide ( mixte $var ): bool
```

Déterminez si une variable est considérée comme vide. Une variable est considérée comme vide si elle n'existe pas ou si sa valeur est égale **FALSE**. **empty ()** ne génère pas d'avertissement si la variable n'existe pas.

Exemple # 1 Une comparaison vide () / isset () simple .

```php
<?php
$var = 0;

// Evaluates to true because $var is empty
if (empty($var)) {
    echo '$var is either 0, empty, or not set at all';
}

// Evaluates as true because $var is set
if (isset($var)) {
    echo '$var is set even though it is empty';
}
?>
```

DOCUMENTATION

IV. Fonctions de chaine de caractères

- Strlen() :

Pour déterminer le nombre de caractères d'une chaîne, utilisez la fonction strlen() , dont la syntaxe est la suivante :

int strlen (string $ch)

Exemple : vérifier par la fonction strlen qu'un code postal saisi par un internaute comporte bien cinq caractères :

<?php $code = "7508" ; if (strlen($code) != 5) echo "Code erroné !" ; ?>

- Strtouper() strtolower()

string strtolower(string $ch) : retourne la chaîne avec tous les caractères en minuscules. ● string strtoupper(string $ch) : retourne la chaîne avec tous les caractères en majuscules

V. Fonction PCRE :

- Preg_math()

VI. Références

- https://www.php.net/manual/en/language.variables.superglobals
- https://www.chiny.me/objet-pdo-pour-la-connexion-a-une-base-de-donnees-en-php-8-12.php
- https://www.php.net/manual/en/ref.var.php
- https://www.php.net/manual/en/ref.strings.php
- https://www.php.net/manual/en/ref.pcre.php