PHP 3

$_SERVER Variables: Script Name

```
<?php
    echo $_SERVER['SCRIPT_NAME'];
    //Outputs "/somefile.php"
?>
```

Notre exemple a été **écrit dans un fichier** appelé **somefile.php**, qui se **trouve** dans le **racine du serveur web**.

$ _SERVER

$ _SERVER est un **tableau** qui inclut des informations telles que des **en-tête**, des **chemins** et des **emplacements** du **script**. Les **entrées** de ce tableau sont **créées par le serveur Web**.
$ _SERVER ['SCRIPT_NAME'] renvoie le chemin du script courant:

Variable prédéfinie

Un **superglobal** est une **variable prédéfinie** qui est toujours accessible, quelle que soit la portée. Vous pouvez **accéder aux superglobals PHP** à travers **n'importe** quelle **fonction**, **classe** ou **fichier**.

Les variables superglobales de PHP sont **$ _SERVER**, **$ GLOBALS**, **$ _REQUEST**, **$ _POST**, **$ _GET**, **$ _FILES**, **$ _ENV**, **$ _COOKIE**, **$ _SESSION**.

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

Ce **graphique** montre les **principaux éléments** de **$ _SERVER**.

$_SERVER Variables: Host Name

Le **chemin vers vos images** est maintenant **dynamique**. Il **changera automatiquement**, en fonction de l'en-tête de l'hôte.

```
<?php
    $host = $_SERVER['HTTP_HOST'];
    $image_path = $host."/images/";
?>
```

Créez un **fichier config.php**, qui contient le **chemin d'accès à vos images**:

```
<?php
require 'config.php';
echo "<img src=".$image_path."header.png".">";
?>
```

Utilisez le fichier config.php dans vos scripts:

$ _SERVER

$ _SERVER ['HTTP_HOST'] renvoie **l'en-tête de l'hôte de la requête** en cours. Cette méthode peut être **utile lorsque vous avez beaucoup d'images sur votre serveur et que vous devez transférer le site vers un autre hôte**. Au lieu de modifier le chemin pour chaque image, vous pouvez effectuer les opérations suivantes:

```
<?php
echo $_SERVER['HTTP_HOST'];
//Outputs "localhost"
?>
```

PHP Forms

```
<form action="first.php" method="post">
<p>Name: <input type="text" name="name" /></p>
<p>Age: <input type="text" name="age" /></p>
<p><input type="submit" name="submit" value="Submit" /></p>
</form>
```

Formulaire

Le but des PHP superglobals **$ _GET** et **$ _POST** est de **collecter des données** qui ont été entrées dans un **formulaire**.
L'exemple ci-dessus montre un **formulaire HTML simple** qui comprend **deux champs de saisie** et un **bouton de soumission**.

Resultat

L'attribut **action** spécifie que **lorsque le formulaire est soumis**, les **données** sont **envoyées à un fichier PHP nommé first.php**.
Les **éléments de formulaire HTML** ont des **noms**, qui seront utilisés lors de l'accès aux données avec PHP. **L'attribut de méthode** sera discuté dans la **leçon suivante**. Pour l'instant, nous allons définir la **valeur de "post"**.

Nous avons utilisé le tableau **$ _POST**, car la **méthode** = **"post"** a été spécifiée dans le formulaire.

```
<form>
<body>

Welcome <?php echo $_POST["name"]; ?><br />
Your age: <?php echo $_POST["age"]; ?>

</body>
</form>
```

Maintenant, quand nous avons un **formulaire HTML** avec l'**attribut d'action mis à notre fichier PHP**, nous pouvons **accéder** aux **données de formulaire** affichées en **utilisant le tableau** associatif **$ _POST**.
Dans le fichier **first.php**:

Le **tableau superglobal $ _POST** contient des **paires clé / valeur**. Dans les **paires**, les **clés sont les noms des contrôles de formulaire** et les **valeurs sont les données saisies** par l'utilisateur.

GET and POST

Les deux méthodes de soumission des formulaires sont **GET** et **POST**.
Les **informations envoyées à partir d'un formulaire via la méthode POST sont invisibles pour les autres**, car **tous les noms et / ou valeurs** sont **intégrés** dans le **corps de la requête HTTP**. En outre, il n'y a **pas de limites quant à la quantité d'informations à envoyer**. De plus, **POST** prend en charge des **fonctionnalités avancées** telles que le **support d'entrées binaires multi-parties** lors du **chargement de fichiers sur le serveur**.
Cependant, il n'est **pas possible de mettre la page en signet** car **les valeurs soumises ne sont pas visibles**.

POST est la méthode préférée pour **envoyer des données de formulaire**.

OBTENEZ

Les **informations envoyées** via un **formulaire** à l'aide de la **méthode GET** sont **visibles pour tout le monde (tous les noms et valeurs des variables sont affichées dans l'URL)**. **GET** définit également des **limites sur la quantité d'informations pouvant être envoyées** - environ **2000 caractères**.
Toutefois, comme les **variables sont affichées dans l'URL**, il est possible de **mettre en signet la page**, ce qui peut-être utile dans certaines situations.

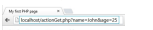
GET ne doit **JAMAIS** être utilisé pour l'envoi de **mots de passe** ou d'autres **informations sensibles** !
Lors de l'utilisation de **POST** ou **GET**, une **validation appropriée des données de formulaire** par le **filtrage** et le **traitement** est d'une importance vitale pour protéger votre formulaire contre les pirates et les exploits!

For example:

```
<form action="actionGet.php" method="get">
Name: <input type="text" name="name" /><br /><br />
Age: <input type="text" name="age" /><br /><br />
<input type="submit" name="submit" value="Submit" />
</form>
```

actionGet.php

```
<?php
echo "Hi ".$_GET['name']." ";
echo "You are ".$_GET['age'].' years old ";
?>
```



Hi John. You are 25 years old.

Maintenant, le formulaire est **soumis à actionGet.php**, et vous pouvez **voir les données soumises dans l'URL**.

$_SESSION

La fonction **session_start()** doit être la **première chose dans votre document. Avant toute balise** HTML.

```
<?php
// Start the session
session_start();

$_SESSION['color'] = "red";
$_SESSION['name'] = "John";
?>
```

Démarrer une session PHP

Maintenant, les variables de couleur et de session de nom sont **accessibles sur plusieurs pages**, **tout au long de la session**.

À l'aide d'une **session**, vous pouvez **stocker des informations dans des variables**, à **utiliser sur plusieurs pages**.
Les informations **ne sont pas stockées sur l'ordinateur** du **navigateur**, comme c'est le cas avec les cookies.
Par défaut, les variables de session durent jusqu'à ce que **l'utilisateur ferme le navigateur**.

Une session est lancée à l'aide de la fonction **session_start()**.
Utilisez le PHP global **$ _SESSION** pour **définir les variables de session**.

Vos variables de session restent **disponibles** dans le superglobal $ _SESSION **jusqu'à la fermeture de votre session**.
Toutes les variables de session globale peuvent être **supprimées manuellement** en utilisant **session_unset()**. Vous pouvez également **détruire la session** avec **session_destroy()**.

```
<?php
// Start the session
session_start();

<!DOCTYPE html>
<html>
<body>
<?php
echo "Your name is " . $_SESSION['name'] . "<br>";
echo "Your favorite color is " . $_SESSION['color'] . ".";
?>

</body>
</html>
```

Variables de session

Une **autre page peut être créée pour accéder aux variables de session** que nous avons **définies à la page précédente**.

$_COOKIE

Le paramètre **nom** est le **seul** qui est **requis**. Tous les **autres paramètres** sont **facultatifs**.

Name: Spécifie le **nom** du cookie
Value: Spécifie la **valeur** du cookie
Expire: Spécifie (en secondes) quand le cookie doit **expirer**. La valeur **time() + 86400*30**, définit le cookie à expirer dans **30 jours**. Si ce paramètre est **omis ou mis à 0**, le cookie **expire à la fin de la session** (lorsque le navigateur se ferme). La **valeur par défaut est 0**. 13 mois maximum au niveau légal.
Path: Spécifie le **chemin du serveur du cookie**. Si cette option est réglée sur **"/"**, le cookie sera **disponible dans tout le domaine**. Si'il est réglé sur **"/ php /"**, le cookie ne sera disponible que dans le **répertoire php et tous les sous-répertoires de php**. La valeur **par défaut** est le **répertoire courant** dans lequel le cookie est configuré.
Domain: Spécifie le **nom de domaine** du cookie. Pour rendre le cookie **disponible sur tous les sous-domaines de example.com**, définissez le domaine sur **"example.com"**.
Secure: Spécifie si le cookie doit uniquement être transmis via une **connexion HTTPS sécurisée**. **TRUE** indique que le cookie **ne sera défini que si** une connexion sécurisée existe. La valeur **par défaut** est **FALSE**.
Httponly: Si cette option est définie sur **TRUE**, le cookie ne sera accessible **que via le protocole HTTP** (le cookie ne sera **pas accessible aux langages de script**). L'utilisation de httponly aide à **réduire le vol d'identité en utilisant des attaques XSS**. La valeur par défaut est **FALSE**.

Les **cookies** sont souvent utilisés pour **identifier l'utilisateur**. Un cookie est un **petit fichier que le serveur intègre sur l'ordinateur de l'utilisateur**. Chaque fois que le même ordinateur demande une page via un navigateur, **il enverra le cookie, aussi**. Avec PHP, vous pouvez à la fois **créer et récupérer des valeurs de cookie**.

Créer des cookies à l'aide de la fonction **setcookie()**:

```
<?php
$value = "John";
setcookie("user", $value, time() + (86400 * 30), "/");

if(isset($_COOKIE["user"])) {
  echo "Value is: ".$_COOKIE["user"];
}
//Outputs "Value is: John"
?>
```

La fonction **setcookie()** doit apparaître **AVANT la balise html**.
La valeur du cookie est **automatiquement encodée** lorsque le cookie est **envoyé**, et est **décodée automatiquement** quand elle est **reçue**. Néanmoins, **NE JAMAIS stocker des informations sensibles** dans les cookies.

L'exemple suivant **crée un cookie** nommé **"user"** avec la valeur **"John"**. Le cookie expire après **30 jours**, ce qui est écrit comme 86.400 * 30, dans lequel **86.400 secondes = un jour**. Le **"/"** signifie que le cookie est disponible sur tout le site Web.

Nous **récupérons alors la valeur** du cookie "user" (en utilisant la variable **globale $ _COOKIE**). Nous utilisons également la fonction **isset()** pour savoir si le cookie est défini:

User-Defined Functions

Les noms de fonction ne sont PAS sensibles à la casse.

```
function functionName() {
//code to be executed
}
```

Une fonction est un **bloc d'instructions** qui peuvent être utilisées de façon **répétée** dans un programme.
Une fonction ne s'exécute pas immédiatement lorsqu'une page est chargée. Il sera exécuté par un **appel à la fonction**.
Une **déclaration de fonction définie** par l'utilisateur **commence avec le mot fonction**. Un nom de fonction peut commencer par une **lettre** ou un **trait de soulignement**, mais pas avec un numéro ou un symbole spécial.

```
<?php
function sayHello() {
    echo "Hello!";
}

sayHello(); //call the function
//Outputs "Hello!"
?>
```

Dans l'exemple ci-dessus, nous créons la fonction **sayHello()**. L'accolade d'ouverture { indique que c'est le **début** du code de fonction, tandis que l'accolade de fermeture } indique que c'est la **fin**.
Pour **appeler la fonction**, il suffit d'**écrire son nom()**;

Function Parameters

```
<?php
function multiplyByTwo($number) {
    $answer = $number * 2;
    echo $answer;
}
multiplyByTwo(3);
?>
```

L'**information** peut être **transmise aux fonctions** par des **arguments**, qui sont comme des **variables**.
Les arguments sont **spécifiés après le nom de la fonction et entre parenthèses**.
Ici, notre fonction prend un nombre, le multiplie par deux et imprime le résultat: //Outputs 6

```
<?php
function multiply($num1, $num2) {
    $result = $num1 * $num2;
    return $result;
}

multiply(2, 3);
//Outputs 6
?>
```

Lorsque vous **DEFINISSEZ une fonction**, les **variables qui représentent les valeurs** qui lui seront transmises pour le traitement **sont appelées paramètres**. Toutefois, lorsque vous **UTILISEZ une fonction**, la **valeur** que vous passez à elle est **appelée un argument**.

Vous pouvez **ajouter autant d'arguments que vous le souhaitez**, à **condition** qu'ils soient **séparés par des virgules**.

```
<?php
function setCounter($num=10) {
    echo "Counter is ".$num."<br />";
}
setCounter(42);  //Counter is 42
setCounter();  //Counter is 10
?>
```

Les **arguments par défaut peuvent être définis** pour les arguments de fonction.
Dans l'exemple ci-dessous, nous appelons la fonction **setCounter()**. Il n'y a **pas d'arguments**, donc il **prendra les valeurs par défaut** qui ont été définies.

Lors de l'**utilisation d'arguments par défaut**, les **valeurs par défaut** doivent être sur le **côté droit** de tous les arguments non par défaut. Sinon, les choses ne fonctionneront pas comme prévu.

The Return Statement

```
<?php
function mult($num1, $num2) {
    $res = $num1 * $num2;
    return $res;
}

echo mult(4, 6); // Outputs 24
?>
```

Une fonction peut **retourner une valeur** à l'aide de l'instruction **return**.
Return arrête l'exécution de la fonction et **renvoie la valeur** au code appelant.
Par exemple:

En laissant les résultats de retour dans une valeur NULL retourner.
Une **fonction ne peut pas retourner plusieurs valeurs**, mais **renvoyer un tableau produira des résultats similaires**.

Writing to a File

Manipulation des fichiers
PHP offre un certain nombre de **fonctions** à utiliser lors de la **création**, la **lecture**, le **téléchargement** et l'**édition** de fichiers.
La fonction **fopen() crée ou ouvre un fichier**. Si vous utilisez **fopen()** avec un fichier qui n'existe pas, le **fichier sera créé**, étant donné que le fichier a été ouvert pour **l'écriture(w)** ou **l'ajout(a)**.

Utilisez l'un des **modes suivants pour ouvrir le fichier**.
r **Ouvre** le fichier en **lecture seule**.
w: **Ouvre** le fichier pour l'**écriture seulement**. **Efface le contenu du fichier** ou **crée un nouveau fichier s'il n'existe pas**.
a: **Ouvre** le fichier pour l'**écriture seulement**.
x: **Crée** un nouveau fichier pour l'**écriture seulement**.
r+ **Ouvre** le fichier pour **lecture / écriture**.
w+ **Ouvre** le fichier pour **lecture / écriture**. **Efface le contenu du fichier** ou **crée un nouveau fichier s'il n'existe pas**.
a+ **Ouvre** le fichier pour **lecture / écriture**. **Crée** un nouveau fichier **si** le fichier **n'existe pas**.
x+ **Crée** un nouveau fichier pour **lecture / écriture**.

```
$myfile = fopen("file.txt", "w");
```

L'exemple ci-dessous **crée un nouveau fichier**, **"file.txt"**, qui sera **créé dans le même répertoire qui héberge le code PHP**.

Le symbole \n est utilisé lors de l'écriture de **nouvelles lignes**.

```
<?php
$myfile = fopen("names.txt", "w");

$txt = "Jehn\n";
fwrite($myfile, $txt);
$txt = "Jhaiah\n";
fwrite($myfile, $txt);

fclose($myfile);

// File contents:
// John
// Jhaiah
?>
```

Ecrire au fichier
Lors de l'**écriture dans un fichier**, utilisez la fonction **fwrite()**.
Le premier paramètre de **fwrite()** est le **fichier à écrire**, le second paramètre est la **chaîne à écrire**.

L'exemple ci-dessus **écrit deux noms** dans un nouveau fichier **appelé "names.txt"**. Notez que nous avons écrit **deux fois au fichier "names.txt"**, puis nous avons utilisé la fonction **fclose()** pour **fermer le fichier**.

Fclose ()
La fonction **fclose()** ferme un fichier ouvert et retourne **TRUE en cas de succès** ou **FALSE en cas d'échec**.
C'est une bonne pratique de **fermer tous les fichiers après avoir terminé de travailler avec eux**.

Appending to a File

Lors de l'ajout à un fichier utilisant le **mode 'a'**, le **pointeur** du fichier est **placé à la fin du fichier**, ce qui garantit que **toutes les nouvelles données** sont ajoutées à la fin du fichier.

```
$myFile = "test.txt";
$fh = fopen($myFile, 'a');
fwrite($fh, "Some text");
fclose($fh);
```

Ajout à un fichier
Si vous souhaitez **ajouter du contenu à un fichier**, vous devez **ouvrir le fichier** en **mode append**.

```
<?php
if(isset($_POST['text')) {
  $name = $_POST['text'];
  $handle = fopen('names.txt', 'a');
  fwrite($handle, $name."\n");
  fclose($handle);
}
?>

<form method="post">
  Name: <input type="text" name="text" />
  <input type="submit" name="submit" />
</form>
```

Créons un **exemple d'un formulaire** qui ajoute des **données remplies à un fichier**.

Maintenant, **chaque fois qu'un nom est entré** et soumis, il est **ajouté au fichier "names.txt"**, avec une nouvelle ligne.

La fonction **isset()** a déterminé **si le formulaire avait été soumis**, ainsi que **si le texte contenait une valeur**.

Nous n'avons **pas spécifié d'attribut d'action pour le formulaire**, donc il **se soumet à lui-même**.

Reading a File

```
$read = file('names.txt');
foreach ($read as $line) {
  echo $line ".";
}
```

Lecture d'un fichier
La fonction **file()** lit le fichier entier dans un tableau.
Chaque élément du tableau correspond à **une ligne dans le fichier**.

```
<?php
$myfile = fopen("names.txt", "w");

$txt = "Jehn\n";
fwrite($myfile, $txt);
$txt = "Jhaiah\n";
fwrite($myfile, $txt);

fclose($myfile);

$read = file("names.txt");
foreach ($read as $line) {
  echo $line ".";
}

$read = file("names.txt");
foreach ($read as $line) {
  echo $line ".";
}
?>
```

Cela **imprime toutes les lignes du fichier** et les **sépare par des virgules**.
Nous avons utilisé **la boucle foreach**, car la variable **$read est un tableau**.

```
<?php
$myfile = fopen("names.txt", "w");

$txt = "Jehn\n";
fwrite($myfile, $txt);
$txt = "Jhaiah\n";
fwrite($myfile, $txt);

fclose($myfile);

$read = file("names.txt");
$count = count($read);
$i = 1;
foreach ($read as $line) {
  echo $line;
  if ($i < $count) {
    echo ", ";
  }
  $i++;
}
?>
```

À la fin de la sortie dans l'exemple précédent, **nous aurions une virgule**, comme nous **l'imprimons après chaque élément du tableau**.
Le **code suivant** nous permet **d'éviter l'impression de cette virgule finale**.

Cela évite l'impression de la virgule finale, comme pour la dernière ligne, **Si est égal à $count**.

La variable **$count** utilise la **fonction count** pour obtenir le nombre d'éléments dans le **tableau $read**.
Ensuite, dans **la boucle foreach**, après chaque ligne imprime, nous déterminons **si la ligne courante est inférieure au nombre total de lignes**, et **imprimer une virgule si elle est**.