Predefined Variables

Predefined Variables

A **superglobal** is a predefined variable that is always accessible, regardless of scope. You can access the PHP superglobals through any function, class, or file.

PHP's superglobal variables are $_SERVER, $GLOBALS, $_REQUEST, $_POST, $_GET, $_FILES, $_ENV, $_COOKIE, $_SESSION.

$_SERVER

$_SERVER is an array that includes information such as headers, paths, and script locations. The entries in this array are created by the web server.
$_SERVER['SCRIPT_NAME'] returns the path of the current script:

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

Try It Yourself

Our example was written in a file called **somefile.php**, which is located in the root of the web server.

$_SERVER

$_SERVER['HTTP_HOST'] returns the Host header from the current request.

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

This method can be useful when you have a lot of images on your server and need to transfer the website to another host. Instead of changing the path for each image, you can do the following:
Create a **config.php** file, that holds the path to your images:

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

Use the **config.php** file in your scripts:

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

The path to your images is now dynamic. It will change automatically, based on the Host header.

This graphic shows the main elements of $_SERVER.

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

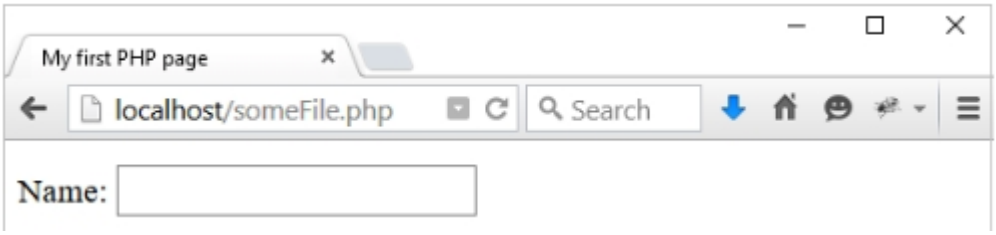
$_SERVER['HTTP_HOST'] returns the Host header from the current request.

Forms

The purpose of the PHP superglobals **$_GET** and **$_POST** is to collect data that has been entered into a form.
The example below shows a simple HTML form that includes two input fields and a submit button:

```
<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>
```

Result:



Age:

Submit

The purpose of the PHP superglobals $_GET and $_POST is to collect data that has been entered into a form.

Forms

The **action** attribute specifies that when the form is submitted, the data is sent to a PHP file named **first.php**.
HTML form elements have **names**, which will be used when accessing the data with PHP.

The **method** attribute will be discussed in the next lesson. For now, we'll set the value to "**post**".

Forms

Now, when we have an HTML form with the **action** attribute set to our PHP file, we can access the posted form data using the **$_POST** associative array.
In the **first.php** file:

```
<html>
<body>

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

</body>
</html>
```

The **$_POST** superglobal array holds key/value pairs. In the pairs, keys are the **names** of the form controls and values are the **input data** entered by the user.

We used the **$_POST** array, as the **method="post"** was specified in the form.
To learn more about the form methods, press **Continue**!

POST

The two methods for submitting forms are **GET** and **POST**.
Information sent from a form via the **POST** method is invisible to others, since all names and/or values are embedded within the body of the HTTP request. Also, there are no limits on the amount of information to be sent.
Moreover, POST supports advanced functionality such as support for multi-part binary input while uploading files to the server.
However, it is not possible to bookmark the page, as the submitted values are not visible.

POST is the preferred method for sending form data.

GET

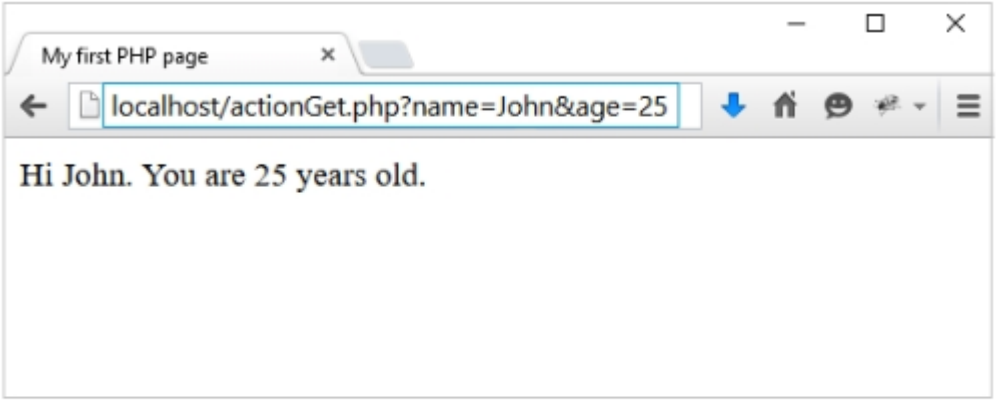
Information sent via a form using the **GET** method is visible to everyone (all variable names and values are displayed in the **URL**). **GET** also sets limits on the amount of information that can be sent - about 2000 characters.
However, because the variables are displayed in the URL, it is possible to bookmark the page, which can be useful in some situations.
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.";
?>
```

Now, the form is submitted to the **actionGet.php**, and you can see the submitted data in the **URL**:



GET should **NEVER** be used for sending passwords or other sensitive information!
When using POST or GET, proper validation of form data through filtering and processing is vitally important to protect your form from hackers and exploits!

Sessions

Using a **session**, you can store information in variables, to be used across multiple pages.
Information is not stored on the user's computer, as it is with **cookies**.
By default, session variables last until the user closes the browser.

Start a PHP Session

A session is started using the **session_start()** function.
Use the PHP global **$_SESSION** to set session variables.

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

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

Now, the **color** and **name** session variables are accessible on multiple pages, throughout the entire session.

> The **session_start()** function must be the very first thing in your document. Before any HTML tags.

Session Variables

Another page can be created that can access the session variables we set in the previous page:

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
echo "Your name is " . $_SESSION['name'];
// Outputs "Your name is John"
?>
</body>
</html>
```

> Your session variables remain available in the **$_SESSION** superglobal until you close your session.
> All global session variables can be removed manually by using **session_unset()**. You can also destroy the session with **session_destroy()**.

Cookies

Cookies are often used to identify the user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page through a browser, it will send the cookie, too. With PHP, you can both create and retrieve cookie values.

Create cookies using the **setcookie()** function:

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

name: Specifies the cookie's name
value: Specifies the cookie's value
expire: Specifies (in seconds) when the cookie is to expire. The value: time()+86400*30, will set the cookie to expire in 30 days. If this parameter is omitted or set to 0, the cookie will expire at the end of the session (when the browser closes). Default is 0.
path: Specifies the server path of the cookie. If set to "/", the cookie will be available within the entire domain. If set to "/php/", the cookie will only be available within the php directory and all sub-directories of php. The default value is the current directory in which the cookie is being set.
domain: Specifies the cookie's domain name. To make the cookie available on all subdomains of example.com, set the domain to "example.com".
secure: Specifies whether or not the cookie should only be transmitted over a secure, HTTPS connection. TRUE indicates that the cookie will only be set if a secure connection exists. Default is FALSE.

httponly: If set to TRUE, the cookie will be accessible only through the HTTP protocol (the cookie will not be accessible to scripting languages). Using httponly helps reduce identity theft using XSS attacks. Default is FALSE.

> The **name** parameter is the only one that's required. All of the other parameters are optional.

Cookies

The following example creates a cookie named "user" with the value "John". The cookie will expire after 30 days, which is written as 86,400 * 30, in which 86,400 seconds = one day. The '/' means that the cookie is available throughout the entire website.

We then retrieve the value of the cookie "user" (using the global variable **$_COOKIE**). We also use the **isset()** function to find out if the cookie is set:

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

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

> The **setcookie()** function must appear BEFORE the <html> tag.
> The value of the cookie is automatically encoded when the cookie is sent, and is automatically decoded when it's received. Nevertheless, **NEVER** store sensitive information in cookies.

End.