PHP 5 Global Variables - Superglobals

Superglobals were introduced in PHP 4.1.0, and are built-in variables that are always available in all scopes.

PHP Global Variables - Superglobals

Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special. The PHP superglobal variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

This chapter will explain some of the superglobals, and the rest will be explained in later chapters.

PHP $GLOBALS

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods). PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.

The example below shows how to use the super global variable $GLOBALS:

Example

```
<?php
$x = 75;
$y = 25;

function addition() {
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}

addition();
echo $z;
?>
```

In the example above, since z is a variable present within the $GLOBALS array, it is also accessible from outside the function!

PHP $_SERVER

$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

The example below shows how to use some of the elements in $_SERVER:

Example

```
<?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'];
?>
```

The following table lists the most important elements that can go inside $_SERVER:

Element/Code	**Description**
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script

$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)

$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_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_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)

$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

PHP $_REQUEST

PHP $_REQUEST is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

Example

```
<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") {
    // collect value of input field
    $name = $_REQUEST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

PHP $_POST

PHP $_POST is widely used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_POST to collect the value of the input field:

Example

```
<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") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

PHP $_GET

PHP $_GET can also be used to collect form data after submitting an HTML form with method="get". $_GET can also collect data sent in the URL. Assume we have an HTML page that contains a hyperlink with parameters:

```
<html>
<body>

<a href="test_get.php?subject1=PHP&subject2=MySQL">Test $GET</a>

</body>
</html>
```

When a user clicks on the link "Test $GET", the parameters "subject1" and "subject2" are sent to "test_get.php", and can then access their values in "test_get.php" with $_GET.

The example below shows the code in "test_get.php":

Example

```
<html>
<body>

<?php
echo "Study " . $_GET['subject1'] . " and " . $_GET['subject2'];
?>

</body>
</html>
```

PHP Forms

PHP 5 Form Handling

The PHP superglobals $_GET and $_POST are used to collect form-data.

PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

Example

```
<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>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

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

The output could be something like this:

```
Welcome John
Your email address is john.doe@example.com
```

The same result could also be achieved using the HTTP GET method:

and "welcome_get.php" looks like this:

```
<?php

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

print_r($_GET);

echo "<br>";

foreach($_GET as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}

?>
```

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

Think SECURITY when processing PHP forms!

This page does not contain any form validation, it just shows how you can send an form data. However, the next pages will show how to process PHP forms with security Proper validation of form data is important to protect your form from hackers and spammers!

GET vs. POST

Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$_GET is an array of variables passed to the current script via the URL parameters.

$_POST is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data. بيانات غير حساسة

Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

Developers prefer POST for sending form data.

Next, lets see how we can process PHP forms the secure way!

PHP 5 Form Validation

This and the next chapters show how to use PHP to validate form data.

PHP Form Validation

Think SECURITY when processing PHP forms!

These pages will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!

The validation rules for the form above are as follows:

- Preventing blank values (name, email address …etc).
- Ensuring the type of value: integer, real, phone number, email address, name …etc.
- Ensuring the format and range of values (phon number must be 9-digit integer).
- Ensuring that values fit together (user types email twice, and the two must match).

However, in the example below, all input fields are optional. The script works fine even if the user does not enter any data.

Example

```
<form action="form1.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit" value="Send">
</form>
```

```
<hr>

<?php

$name  =$_POST['name'];
$email =$_POST['email'];

echo "your  Name is $name and your E-mail is $email";

?>
```

The next step is to make input fields required and create error messages if needed.

In the example below, examine empty filed, and if they are show an error message.

Example

```
<form action="<?php  echo  $PHP_SELVE ; ?>" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit" value="Send">
</form>

<hr>

<?php
// examine empty filed, and if they are show an error message.
// sends the submitted form data to the page itself using $PHP_SELVE
$name  =$_POST['name'];
$email =$_POST['email'];

if ($name=="" )
   echo "please enter your name";
else
echo "your  Name is $name ";

echo "<br>";

if ($email=="" )
   echo "please enter your email";
else
echo "your  E-mail is $email ";

?>
```

This chapter shows how to validate names, e-mails, and URLs.

PHP - Validate Name

The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = $_POST["name"];
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  echo " Name : Only letters and white space allowed";
}
```

The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP - Validate E-mail

The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function.

In the code below, if the e-mail address is not well-formed, then store an error message:

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

PHP - Validate URL

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

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

PHP - Validate Name and E-mail

Now, the script looks like this:

Example

```
<form action="<?php  echo  $PHP_SELVE ; ?>" method="post">
Name: <input type="text" name="name"><br>
```

```
E-mail: <input type="text" name="email"><br>
<input type="submit" value="Send">
</form>
<hr>
<?php

// sends the submitted form data to the page itself using $PHP_SELVE
$name  =$_POST['name'];   $email =$_POST['email'];

if ($name=="" )
   echo "Name is required";
// check if name only contains letters and whitespace
elseif (!preg_match("/^[a-zA-Z ]*$/",$name))
    echo "Name : Only letters and white space allowed";
else
   echo "your  Name is $name ";

echo "<br> ----------------- <br>";

if ($email=="" )
   echo "E-mail is required";
// check if e-mail address is well-formed
elseif (!filter_var($email, FILTER_VALIDATE_EMAIL))
   echo "please enter correct email";
else
   echo "your  E-mail is $email ";

?>
```

PHP 5 Complete Form Example

Here is the complete code for the PHP Form Validation Example:

Example

```
<h2>PHP Form Validation Example</h2>
<form action="<?php  echo  $PHP_SELVE ; ?>" method="post">
Name: <input type="text" name="name" > <br><br>
E-mail: <input type="text" name="email"> <br><br>
Website: <input type="text" name="website" > <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 <br><br>
Your Foverat Languag : <br>
```

```
<input type="checkbox" name="lang[]" value="Arabic"> Arabic <br>
<input type="checkbox" name="lang[]" value="English"> English <br>
<input type="checkbox" name="lang[]" value="French"> French <br>

Select Website:
<select name="site[]">
<option value="http://Google.com"> Google </option>
<option value="http://Yahoo.com"> Yahoo </option>
<option value="http://Bing.com"> Bing </option>
<option value="http://Ask.com"> Ask </option>
</select>  <br><br>

<input type="submit" value="Send">
</form>

<hr>


<?php

// define variables
$name    =$_POST[name];
$email   =$_POST[email];
$website =$_POST[website];
$comment =$_POST[comment];
$gender  =$_POST[gender];

if ($name=="" )
   echo "Name is required";
// check if name only contains letters and whitespace
elseif (!preg_match("/^[a-zA-Z ]*$/",$name))
    echo "Name : Only letters and white space allowed";
else
   echo "your  Name is $name ";

echo "<br> ----------------- <br>";

if ($email=="" )
   echo "E-mail is required";
// check if e-mail address is well-formed
elseif (!filter_var($email, FILTER_VALIDATE_EMAIL))
   echo "please enter correct email";
else
   echo "your  E-mail is $email ";

echo "<br> ----------------- <br>";

if ($website == '')
   echo 'Website= "" ';
// check if URL address syntax is valid
elseif (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website))
  echo "Invalid URL";
else
```

```
   echo "your  Website is $website ";

echo "<br> ----------------- <br>";

if ($comment == "")
   echo 'Comment ="" ';
else
   echo "your  Comment is $comment ";

echo "<br> ----------------- <br>";

if ($gender == '')
   echo "Gender is required";
else
   echo "your  Gender is $gender ";

echo "<br> ----------------- <br>";

$chek=array('Arabic' , 'English' , 'French');
if (isset($_POST['lang'])){
    $ice=$_POST['lang'];
    foreach ($ice as $ch){
        if(in_array($ch , $chek))
           echo $ch;
           echo "<br>";
       }
}

echo " ----------------- <br>";

$site=$_POST['site'][0];
echo "<a href= '$site'>". $site." </a>" ;
echo "<br>";

?>
```

PHP 5 Multidimensional Arrays

Earlier in this tutorial, we have described arrays that are a single list of key/value pairs.

However, sometimes you want to store values with more than one key.

This can be stored in multidimensional arrays.

PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP understands multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

The dimension of an array indicates the number of indices you need to select an element.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays). First, take a look at the following table:

Name	**Stock**	**Sold**
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );
```

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.

To get access to the elements of the $cars array we must point to the two indices (row and column):

Example

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
```

We can also put a For loop inside another For loop to get the elements of the $cars array (we still have to point to the two indices):

Example

```
<?php

$arr=array(
            array( "ali"   , 70 , "aa@ya"),
            array( "ahmad" , 30 , "hh@ya"),
            array( "saad"  , 50 , "ss@ya"),
            );
$arr[]=array("huda"  ,20 , "hu@ya");
$arr[]=array("farah" ,55 , "ff@ya");


for ($row = 0; $row < 5; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$arr[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>
```

Example

```
<?php

$arr=array(array(
                   'name'  =>'Ali',
                   'age'   =>30,
                   'email' =>'www'
```

```
                  ),

            array(
                    'name'  =>'Ahmed',
                    'age'   =>20,
                    'email' =>'sss'
                  ),

            array(
                    'name'  =>'Saad',
                    'age'   =>40,
                    'email' =>"bbb"
                  )
           );

for ($row = 0; $row < 3; $row++) {
      echo "<p><b>Row number $row</b></p>";
      echo "<ul>";
      foreach ($arr[$row] as $key=>$value) {
            echo "<li>".$key."=>".$value."</li>";
      }
      echo "</ul>";
}
?>
```