

PHP Superglobals Form Data

Primary source: <https://www.w3schools.com/php/>

Internet Programming 2, Lesson 7

May 2024

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.

These provide information about server, environment, and user input.

The PHP superglobal variables are:

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

Array	Description
<code>\$_COOKIE</code>	An array of values passed to the current script as HTTP cookies
<code>\$_ENV</code>	An array of environment information
<code>\$_FILES</code>	An array of information about uploaded files
<code>\$_GET</code>	An array of values from a form submitted with the "get" method
<code>\$_POST</code>	An array of values from a form submitted with the "post" method
<code>\$_REQUEST</code>	An array of all the elements in the <code>\$_COOKIE</code> , <code>\$_GET</code> , and <code>\$_POST</code> arrays
<code>\$_SERVER</code>	An array of information about the Web server that served the current script
<code>\$_SESSION</code>	An array of session variables that are available to the current script
<code>\$GLOBALS</code>	An array of references to all variables that are defined with global scope

PHP \$_SERVER

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

The table below shows some of the elements in \$_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 (such as www.w3schools.com)
\$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
\$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
\$_SERVER['HTTP_HOST']	Returns the Host header from the current request
\$_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['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['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
\$_SERVER['SCRIPT_NAME']	Returns the path of the current script

\$_POST[] array

Files: *numbers.php* and *processNumbers.php*

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.

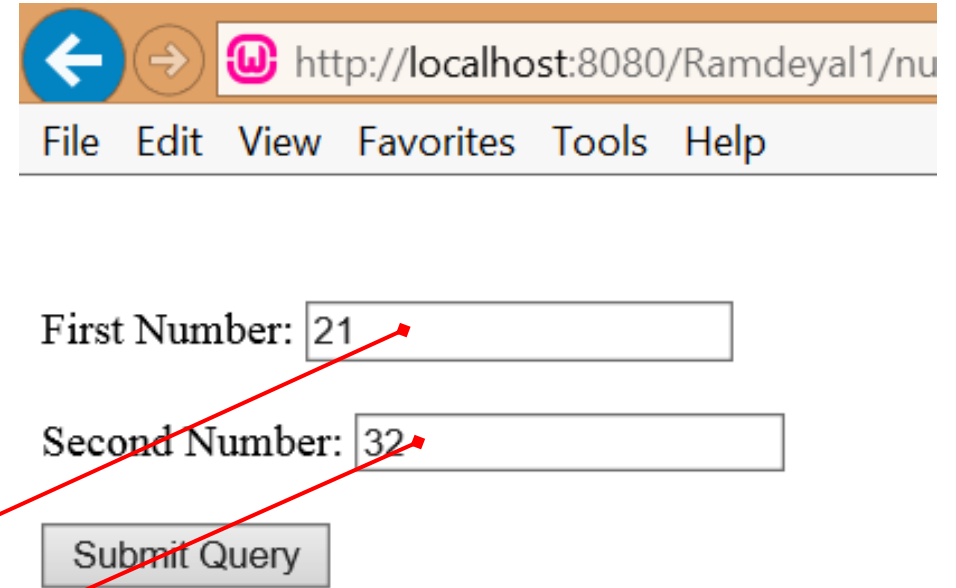
The example below shows a form with two input fields and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. Then, we can use the super global variable \$_POST to collect the value of the input field.

- The browser places the input data in an associative array called \$_POST and sends this to the server for processing
- The browser creates the \$_POST array as follows:
 - The browser places the value that the user enters in the first text box into the first element of the array and uses the name of the object as its index, the second element will contain the data from the next input element, and so on.
- When the server receives the request from the browser, it finds the file indicated in the action attribute and sends this, together with the \$_POST array to the PHP engine for processing
- Output from the PHP script is HTML which is then sent to the browser for display to the user in a new web page.

Form code

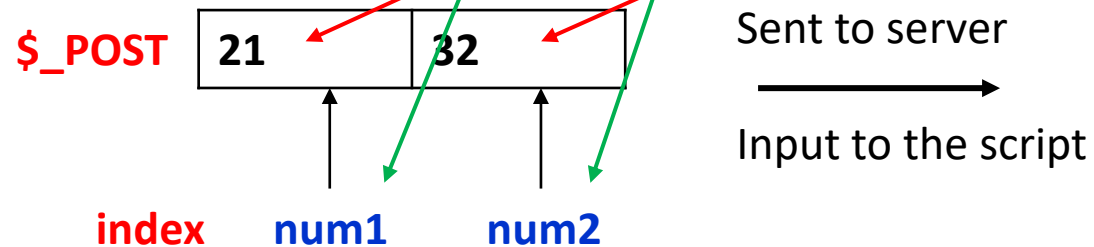
```
<form method="post" action="processNumbers.php">  
  First Number: <input type="text" name="num1" />  
  <br/><br />  
  Second Number: <input type="text" name="num2"/>  
  <br/><br />  
  <input type="submit" name="submit" />  
  <br/><br />  
</form>
```

Form display



The screenshot shows a web browser window with the address bar displaying `http://localhost:8080/Ramdeyal1/nu`. The browser's menu bar includes File, Edit, View, Favorites, Tools, and Help. The form displayed contains two text input fields. The first field is labeled "First Number:" and contains the value "21". The second field is labeled "Second Number:" and contains the value "32". Below the input fields is a button labeled "Submit Query".

`$_POST[]` array created by browser

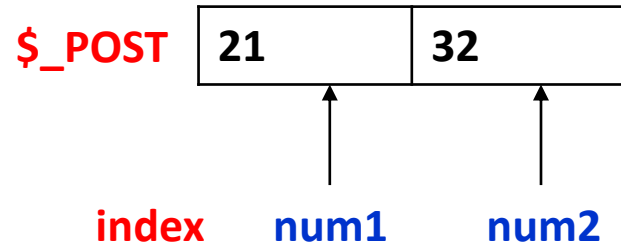


processNumbers.php

```
<?php  
$num1 = $num2 = $sum = 0;  
$num1 = $_POST["num1"];  
$num2 = $_POST["num2"];  
$sum = $num1 + $num2;  
echo $num1 . " + " . $num2 . " = " . $sum;  
?>
```

At the Server:

The server has access to the `$_POST` array and the code (`processNumbers.php`) to process the data



processNumbers.php

```
<?php
$num1 = $num2 = $sum = 0;
$num1 = $_POST["num1"];
$num2 = $_POST["num2"];
$sum = $num1 + $num2;
echo $num1 . " + " . $num2 . " = " . $sum;
?>
```

echo: Output placed
→
in output stream

\$_GET[] array

PHP \$_GET works in a similar way to the \$_POST array. However, this array is used when the developed uses the “get” method to sent data to the server.

It is important that you read and understand the differences between these two methods, and that you clearly understand why developers prefer the post method – see w3schools site.

Including the PHP script in the same file as the form

Note the “problem” with the previous solution was that after the php script executed, the output was displayed on a *new web page* (the original form disappeared from the page) – this is because the script was in a different file. Therefore, we now deal with including the script to process the data in the same (form) file. This means that we must set the action attribute of the form to point to itself. However, the full path and filename has to be indicated in the action attribute. If for some reason, the administrator moves the file to another folder, or renames the folder, the server will not be able to find the file, as its path will be incorrect – this is a general problem with absolute addressing (hard-coding the path and file name). Have a look at the numbers2.php file in the resources folder. To solve this problem, we use the PHP super global \$_SERVER["PHP_SELF"] to retrieve the path and file name of the currently executing script.

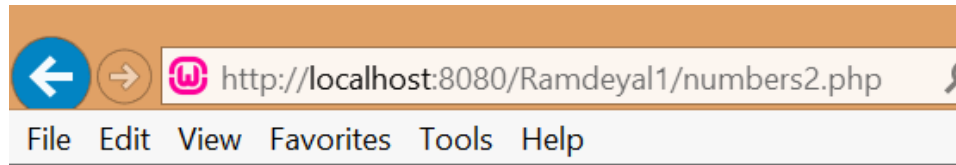
File: *numbers2.php*

```
<body>
<?php
$num1 = $num2 = $sum =0;
$num1 = $_POST["num1"];
$num2 = $_POST["num2"];
$sum = $num1 + $num2;
?>
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
  First Number: <input type="text" name="num1" />
  <br /><br />
  Second Number: <input type="text" name="num2"/>
  <br /><br />
  <input type="submit" name="submit" /> <br /><br />
  The answer is <span><?php echo $sum;?></span>
</form>
</body>
```

Returns the file name of the currently executing script

Note the use of a tag to display the output.

However, the above php code executes the first time the file is requested from the server (to display the form)...therefore, **it results in an error – undefined index (screenshot on next page)** – since the `$_POST` would not have been created as yet. However, if you ignore the error and input the numbers and submit the data to the server, the script works correctly.



(!) Notice: Undefined index: num1 in C:\wamp\www\Ramdeyal1\numbers2.php on line 13				
Call Stack				
#	Time	Memory	Function	Location
1	0.0011	133120	{main}()	..\numbers2.php:0

(!) Notice: Undefined index: num2 in C:\wamp\www\Ramdeyal1\numbers2.php on line 14				
Call Stack				
#	Time	Memory	Function	Location
1	0.0011	133120	{main}()	..\numbers2.php:0

First Number:

Second Number:

The answer is 0

The first time the script runs, the form has not been displayed and the user has not clicked submit – therefore any reference to the index of the array `$_POST` is an error, as the array has not been created yet.

If you ignore the error and provide the values and submit the data, the script works fine.

We can resolve the above problem by checking if the form data has been submitted, and if it has, execute the script, otherwise do not.

If the data has been submitted via the post method, then the super global `$_SERVER []` array element pointed to by the index “REQUEST_METHOD” will contain the value “POST”. Therefore, we will include an if statement to control the execution of the script by checking `if ($_SERVER["REQUEST_METHOD"] == "POST")`...next page

File: *numbers3.php*

```
<?php
    $num1 = $num2 = $sum =0;
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $num1 = $_POST["num1"];
        $num2 = $_POST["num2"];
        $sum = $num1 + $num2;
    }
?>
```

NOTE: While the IF statement resolved our current challenge of the warning in the previous version in which we were attempting to access a variable (\$_POST) that did not exist, a new observation is that the original numbers the user entered disappears from the form when the result is displayed. To solve this problem, we need to find a way to ensure that form data is retained in these types of circumstances (useful when the user makes an error in input) – next slide deck...

EXERCISE:

1. UserBio **version 1:**

Design a form containing input elements for the user to enter: surname, first Name(s), SA ID number, gender (cater only for M and F), and a submit button. Set the action attribute of the form to “processBio.php” and the method to “post”. Save the page as userBio.htm. Now write a PHP processBio.php to receive the user input, process and then display the following details:

Surname, First Name(s), SA ID Number and Gender)

2. UserBio **version 2:**

Rewrite the script from version 1, to include the following output:

Title, User Initials and surname [initials here simply mean the first letter of each first name the user has submitted, use Mr for Male and Ms for female]

Date of Birth

Age [Years and months as at today]

Save this version as processBio ver2.php

3. Personal DataAnalyser

In the next section, we will learn how to access data from a radio button and other form elements. Additionally, we perform some basic data validation in terms of checking whether the user has filled-in those fields which are required.

Examine the screen shot of a basic form on the right. The HTML code to display this form appears on the next slid. Save as personalData.php.

You may use this file to write the php script to process the data submitted by the user. In this case we are merely going to display the data back onto the form.

PHP Form Validation Example

** required field.*

Name: *

E-mail: *

Website:

Comment:

Gender: ☐ Female ☐ Male *

Your Input: