**Day 4**

# PHP 5 Global Variables

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**

## 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:

## 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:

**<?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 page from which the current page was called

$\_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 $\_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

## Including a PHP File into Another PHP File

The include() and require() statement allow you to include the code contained in a PHP file within another PHP file. Including a file produces the same result as copying the script from the file specified and pasted into the location where it is called.

You can save a lot of time and work through including files — Just store a block of code in a separate file and include it wherever you want using the include() and require() statements instead of typing the entire block of code multiple times. A typical example is including the header, footer and menu file within all the pages of a website.

The basic syntax of the include() and require() statement can be given with:

**include("path/to/filename"); -Or- include "path/to/filename";  
require("path/to/filename"); -Or- require "path/to/filename";**

The following example will show you how to include the common header, footer and menu codes which are stored in separate 'header.php', 'footer.php' and 'menu.php' files respectively, within all the pages of your website. Using this technique, you can update all pages of the website at once by making the changes to just one file, this saves a lot of repetitive work.

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<title>Transflower Web</title>**

**</head>**

**<body> <?php include "header.php"; ?>**

**<?php include "menu.php"; ?>**

**<h1>Welcome to Our Website!</h1>**

**<p>Here you will find lots of useful information.</p>**

**<?php include "footer.php"; ?>**

**</body>**

**</html>**

## Difference Between include and require Statements

You might be thinking if we can include files using the **include()** statement then why we need **require().** Typically the **require()** statement operates like **include().**

The only difference is — the **include()** statement will only generate a PHP warning but allow script execution to continue if the file to be included can't be found, whereas the **require()** statement will generate a fatal error and stops the script execution.

**<?php require "my\_variables.php"; ?>**

**<?php require "my\_functions.php"; ?>**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<title>**

**<?php displayTitle($home\_page); ?>**

**</title>**

**</head>**

**<body>**

**<?php include "header.php"; ?>**

**<?php include "menu.php"; ?>**

**<h1>Welcome to Our Website!</h1>**

**<p>Here you will find lots of useful information.</p>**

**<?php include "footer.php"; ?>**

**</body>**

**</html>**

## The include\_once and require\_once Statements

If you accidentally include the same file (typically [functions](https://www.tutorialrepublic.com/php-tutorial/php-functions.php) or [classes](https://www.tutorialrepublic.com/php-tutorial/php-classes-and-objects.php) files) more than one time within your code using the **include** or **require** statements, it may cause conflicts. To prevent this situation, PHP provides **include\_once** and **require\_once** statements.

These statements behave in the same way as **include** and **require** statements with one exception.

The **include\_once** **and require\_once** statements will only include the file once even if asked to include it a second time i.e. if the specified file has already been included in a previous statement, the file is not included again. To better understand how it works, let's check out an example. Suppose we've a 'my\_functions.php' file with the following code:

**<?php**

**function multiplySelf($var)**

**{ $var \*= $var;**

**// multiply variable by itself**

**echo $var;**

**}**

**?>**

Here's is the PHP script within which we've included the 'my\_functions.php' file.

**<?php**

**// Including file**

**require "my\_functions.php";**

**// Calling the function**

**multiplySelf(2);**

**// Output:**

**echo "<br>";**

**// Including file once again**

**require "my\_functions.php"; // Calling the function**

**multiplySelf(5); // Doesn't execute**

**?>**

When you run the above script, you will see the error message something like this: **"Fatal error: Cannot redeclare multiplySelf()"**. This occurs because the 'my\_functions.php' included twice, this means the function **multiplySelf()** is defined twice, which caused PHP to stop script execution and generate fatal error. Now rewrite the above example with **require\_once**.

**<?php**

**// Including file**

**require\_once "my\_functions.php";**

**// Calling the function**

**multiplySelf(2); // Output: 4**

**echo "<br>"; // Including file once again**

**require\_once "my\_functions.php";**

**// Calling the function**

**multiplySelf(5); // Output: 25**

**?>**

As you can see, by using **require\_once** instead of **require**, the script works as we expected.

**PHP & MYSQL**

MySQL is one of the most popular relational database system being used on the Web today. It is freely available and easy to install, however if you have installed Wampserver it already there on your machine. MySQL database server offers several advantages:

* MySQL is easy to use, yet extremely powerful, fast, secure, and scalable.
* MySQL runs on a wide range of operating systems, including UNIX or Linux, Microsoft Windows, Apple Mac OS X, and others.
* MySQL supports standard SQL (Structured Query Language).
* MySQL is ideal database solution for both small and large applications.
* MySQL is developed, and distributed by Oracle Corporation.
* MySQL includes data security layers that protect sensitive data from intruders.

MySQL database stores data into tables like other relational database. A table is a collection of related data, and it is divided into rows and columns.

+----+------------+-----------+-----------------------+

| id | first\_name | last\_name | email |

+----+------------+-----------+-----------------------+

| 1 | Manisha | Parker | manishaparker@mail.com |

| 2 | Sagar | Rao | sagarrao@mail.com |

| 3 | Jai | Kumar | jaik@mail.com |

| 4 | Virendra | Vartak | virendrav@mail.com |

| 5 | Hari | Potale | hari.potale@mail.com |

+----+------------+-----------+----------------------+

## Talking to MySQL Databases with SQL

SQL, the Structured Query Language, is a simple, standardized language for communicating with relational databases like MySQL. With SQL you can perform any database-related task, such as creating databases and tables, saving data in database tables, query a database for specific records, deleting and updating data in databases.

Look at the following standard SQL query that returns the email address of a person whose first name is equal to 'Peter' in the *persons* table:

**SELECT email FROM persons WHERE first\_name="Manisha"**

if you execute the SQL query above it will return the following record:

**manishaparker@mail.com**

# PHP Connect to MySQL Server

## Ways of Connecting to MySQL through PHP

In order to store or access the data inside a MySQL database, you first need to connect to the MySQL database server. PHP offers two different ways to connect to MySQL server:

**MySQLi** (Improved MySQL) and **PDO** (PHP Data Objects) extensions.

While the PDO extension is more portable and supports more than twelve different databases, MySQLi extension as the name suggests supports MySQL database only. MySQLi extension however provides an easier way to connect to, and execute queries on, a MySQL database server. Both PDO and MySQLi offer an object-oriented API, but MySQLi also offers a procedural API which is relatively easy for beginners to understand.

## Connecting to MySQL Database Server

In PHP you can easily do this using the mysqli\_connect() function. All communication between PHP and the MySQL database server takes place through this connection. Here're the basic syntaxes for connecting to MySQL using MySQLi and PDO extensions:

#### Syntax: MySQLi, Procedural way

**$link = mysqli\_connect("hostname", "username", "password", "database");**

#### Syntax: MySQLi, Object Oriented way

**$mysqli = new mysqli("hostname", "username", "password", "database");**

#### Syntax: PHP Data Objects (PDO) way

**$pdo = new PDO("mysql:host=hostname;dbname=database", "username", "password");**

The hostname parameter in the above syntax specify the host name (e.g. localhost), or IP address of the MySQL server, whereas the username and password parameters specifies the credentials to access MySQL server, and the database parameter, if provided will specify the default MySQL database to be used when performing queries.

## Closing the MySQL Database Server Connection

The connection to the MySQL database server will be closed automatically as soon as the execution of the script ends. However, if you want to close it earlier you can do this by simply calling the PHP mysqli\_close() function.

## Creating MySQL Database Using PHP

Now that you've understood how to open a connection to the MySQL database server.

Before saving or accessing the data, we need to create a database first. The [CREATE DATABASE](https://www.tutorialrepublic.com/sql-tutorial/sql-create-database-statement.php)statement is used to create a new database in MySQL.

## Creating Tables inside MySQL Database Using PHP

In the previous chapter we've learned how to create a database on MySQL server. Now it's time to create some tables inside the database that will actually hold the data. A table organizes the information into rows and columns.

The SQL [CREATE TABLE](https://www.tutorialrepublic.com/sql-tutorial/sql-create-table-statement.php) statement is used to create a table in database.

## Inserting Data into a MySQL Database Table

Now that you've understood how to create database and tables in MySQL. In this tutorial you will learn how to execute SQL query to insert records into a table.

The [INSERT INTO](https://www.tutorialrepublic.com/sql-tutorial/sql-insert-statement.php) statement is used to insert new rows in a database table.

Let's make a SQL query using the INSERT INTO statement with appropriate values, after that we will execute this insert query through passing it to the PHP mysqli\_query() function to insert data in table. Here's an example, which insert a new row to the *persons* table by specifying values for the *first\_name*, *last\_name* and *email* fields.

## Inserting Data into a MySQL Database Table

Now that you've understood how to create database and tables in MySQL. In this tutorial you will learn how to execute SQL query to insert records into a table.

The [INSERT INTO](https://www.tutorialrepublic.com/sql-tutorial/sql-insert-statement.php) statement is used to insert new rows in a database table.

Let's make a SQL query using the INSERT INTO statement with appropriate values, after that we will execute this insert query through passing it to the PHP mysqli\_query() function to insert data in table. Here's an example, which insert a new row to the *persons* table by specifying values for the *first\_name*, *last\_name* and *email* fields.

**<?php**

**/\* Attempt MySQL server connection.**

**Assuming you are running MySQL server with default setting**

**(user 'root' with no password) \*/**

**$link = mysqli\_connect("localhost", "root", "", "demo");**

**// Check connection**

**if($link === false)**

**{**

**die("ERROR: Could not connect. " . mysqli\_connect\_error());**

**}**

**// Attempt insert query execution**

**$sql = "INSERT INTO persons (first\_name, last\_name, email) VALUES**

**('John', 'Rambo', 'johnrambo@mail.com'),**

**('Clark', 'Kent', 'clarkkent@mail.com'),**

**('John', 'Carter', 'johncarter@mail.com'),**

**('Harry', 'Potter', 'harrypotter@mail.com')";**

**if(mysqli\_query($link, $sql))**

**{**

**echo "Records added successfully.";**

**}**

**else**

**{**

**echo "ERROR: Could not able to execute $sql. " . mysqli\_error($link);**

**}**

**// Close connection**

**mysqli\_close($link); ?>**

**File Handling**

File handling is needed for any application. For some tasks to be done file needs to be processed. File handling in PHP is similar as file handling is done by using any programming language like C. PHP has many functions to work with normal files. Those functions are:

1) **fopen() –** PHP fopen() function is used to open a file. First parameter of fopen() contains name of the file which is to be opened and second parameter tells about mode in which file needs to be opened,

**<?php**

**$file = fopen(“demo.txt”,'w');**

**?>**

Files can be opened in any of the following modes :

* **“w”** – Opens a file for write only. If file not exist then new file is created and if file already exists then contents of file is erased.
* **“r”** – File is opened for read only.
* **“a”** – File is opened for write only. File pointer points to end of file. Existing data in file is preserved.
* **“w+”** – Opens file for read and write. If file not exist then new file is created and if file already exists then contents of file is erased.
* **“r+”** – File is opened for read/write.
* **“a+”** – File is opened for write/read. File pointer points to end of file. Existing data in file is preserved. If file is not there then new file is created.
* **“x”** – New file is created for write only.

2) **fread() –**– After file is opened using fopen() the contents of data are read using fread(). It takes two arguments. One is file pointer, and another is file size in bytes,

**<?php**

**$filename = "demo.txt";**

**$file = fopen( $filename, 'r' );**

**$size = filesize( $filename );**

**$filedata = fread( $file, $size );**

**?>**

3) **fwrite() –** New file can be created or text can be appended to an existing file using fwrite() function. Arguments for fwrite() function are file pointer and text that is to written to file. It can contain optional third argument where length of text to written is specified, e.g.,

**<?php**

**$file = fopen("demo.txt", 'w');**

**$text = "Hello world\n";**

**fwrite($file, $text);**

**?>**

4) **fclose() –** file is closed using fclose() function. Its argument is file which needs to be closed, e.g.,

**<?php**

**$file = fopen("demo.txt", 'r');**

**//some code to be executed**

**fclose($file);**

**?>**