PHP is a server-side scripting language.

**What You Should Already Know**

Before you continue you should have a basic understanding of the following:

* HTML/XHTML
* JavaScript

If you want to study these subjects first, find the tutorials on our [Home page](http://www.w3schools.com/default.asp).

**What is PHP?**

* PHP stands for **P**HP: **H**ypertext **P**reprocessor
* PHP is a server-side scripting language, like ASP
* PHP scripts are executed on the server
* PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
* PHP is an open source software
* PHP is free to download and use

**What is a PHP File?**

* PHP files can contain text, HTML tags and scripts
* PHP files are returned to the browser as plain HTML
* PHP files have a file extension of ".php", ".php3", or ".phtml"

**What is MySQL?**

* MySQL is a database server
* MySQL is ideal for both small and large applications
* MySQL supports standard SQL
* MySQL compiles on a number of platforms
* MySQL is free to download and use

**PHP + MySQL**

* PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

**Why PHP?**

* PHP runs on different platforms (Windows, Linux, Unix, etc.)
* PHP is compatible with almost all servers used today (Apache, IIS, etc.)
* PHP is FREE to download from the official PHP resource: [www.php.net](http://www.php.net/)
* PHP is easy to learn and runs efficiently on the server side

**Where to Start?**

To get access to a web server with PHP support, you can:

* Install Apache (or IIS) on your own server, install PHP, and MySQL
* Or find a web hosting plan with PHP and MySQL support

## What do you Need?

If your server supports PHP you don't need to do anything.

Just create some .php files in your web directory, and the server will parse them for you. Because it is free, most web hosts offer PHP support.

However, if your server does not support PHP, you must install PHP.

Here is a link to a good tutorial from PHP.net on how to install PHP5: <http://www.php.net/manual/en/install.php>

## Download PHP

Download PHP for free here: <http://www.php.net/downloads.php>

## Download MySQL Database

Download MySQL for free here: <http://www.mysql.com/downloads/>

## Download Apache Server

Download Apache for free here: <http://httpd.apache.org/download.cgi>

PHP code is executed on the server, and the plain HTML result is sent to the browser.

**Basic PHP Syntax**

A PHP scripting block always starts with **<?php** and ends with **?>**. A PHP scripting block can be placed anywhere in the document.

On servers with shorthand support enabled you can start a scripting block with <? and end with ?>.

For maximum compatibility, we recommend that you use the standard form (<?php) rather than the shorthand form.

|  |
| --- |
| <?php ?> |

A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.

Below, we have an example of a simple PHP script which sends the text "Hello World" to the browser:

|  |
| --- |
| <html> <body>  <?php echo "Hello World"; ?>  </body> </html> |

Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.

There are two basic statements to output text with PHP: **echo** and **print**. In the example above we have used the echo statement to output the text "Hello World".

**Note:** The file must have a .php extension. If the file has a .html extension, the PHP code will not be executed.

**Comments in PHP**

In PHP, we use // to make a single-line comment or /\* and \*/ to make a large comment block.

|  |
| --- |
| <html> <body>  <?php //This is a comment  /\* This is a comment block \*/ ?>  </body> </html> |

A variable is used to store information.

**Variables in PHP**

Variables are used for storing values, like text strings, numbers or arrays.

When a variable is declared, it can be used over and over again in your script.

All variables in PHP start with a $ sign symbol.

The correct way of declaring a variable in PHP:

|  |
| --- |
| $var\_name = value; |

New PHP programmers often forget the $ sign at the beginning of the variable. In that case it will not work.

Let's try creating a variable containing a string, and a variable containing a number:

|  |
| --- |
| <?php $txt="Hello World!"; $x=16; ?> |

**PHP is a Loosely Typed Language**

In PHP, a variable does not need to be declared before adding a value to it.

In the example above, you see that you do not have to tell PHP which data type the variable is.

PHP automatically converts the variable to the correct data type, depending on its value.

In a strongly typed programming language, you have to declare (define) the type and name of the variable before using it.

In PHP, the variable is declared automatically when you use it.

**Naming Rules for Variables**

* A variable name must start with a letter or an underscore "\_"
* A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and \_ )
* A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore ($my\_string), or with capitalization ($myString)
* A string variable is used to store and manipulate text.
* **String Variables in PHP**
* String variables are used for values that contain characters.
* In this chapter we are going to look at the most common functions and operators used to manipulate strings in PHP.
* After we create a string we can manipulate it. A string can be used directly in a function or it can be stored in a variable.
* Below, the PHP script assigns the text "Hello World" to a string variable called $txt:

|  |
| --- |
| <?php $txt="Hello World"; echo $txt; ?> |

* The output of the code above will be:

|  |
| --- |
| Hello World |

* Now, lets try to use some different functions and operators to manipulate the string.
* **The Concatenation Operator**
* There is only one string operator in PHP.
* The concatenation operator (.)  is used to put two string values together.
* To concatenate two string variables together, use the concatenation operator:

|  |
| --- |
| <?php $txt1="Hello World!"; $txt2="What a nice day!"; echo $txt1 . " " . $txt2; ?> |

* The output of the code above will be:

|  |
| --- |
| Hello World! What a nice day! |

* If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string (a space character), to separate the two strings.
* **The strlen() function**
* The strlen() function is used to return the length of a string.
* Let's find the length of a string:

|  |
| --- |
| <?php echo strlen("Hello world!"); ?> |

* The output of the code above will be:

|  |
| --- |
| 12 |

* The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string).
* **The strpos() function**
* The strpos() function is used to search for a character/text within a string.
* If a match is found, this function will return the character position of the first match. If no match is found, it will return FALSE.
* Let's see if we can find the string "world" in our string:

|  |
| --- |
| <?php echo strpos("Hello world!","world"); ?> |

* The output of the code above will be:

|  |
| --- |
| 6 |

* The position of the string "world" in the example above is 6. The reason that it is 6 (and not 7), is that the first character position in the string is 0, and not 1.
* **Complete PHP String Reference**
* For a complete reference of all string functions, go to our [complete PHP String Reference](http://www.w3schools.com/php/php_ref_string.asp).
* The reference contains a brief description, and examples of use, for each function!
* Operators are used to operate on values.
* **PHP Operators**
* This section lists the different operators used in PHP.
* **Arithmetic Operators**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Result** |
| + | Addition | x=2 x+2 | 4 |
| - | Subtraction | x=2 5-x | 3 |
| \* | Multiplication | x=4 x\*5 | 20 |
| / | Division | 15/5 5/2 | 3 2.5 |
| % | Modulus (division remainder) | 5%2 10%8 10%2 | 1 2 0 |
| ++ | Increment | x=5 x++ | x=6 |
| -- | Decrement | x=5 x-- | x=4 |

* **Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Is The Same As** |
| = | x=y | x=y |
| += | x+=y | x=x+y |
| -= | x-=y | x=x-y |
| \*= | x\*=y | x=x\*y |
| /= | x/=y | x=x/y |
| .= | x.=y | x=x.y |
| %= | x%=y | x=x%y |

* **Comparison Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | is equal to | 5==8 returns false |
| != | is not equal | 5!=8 returns true |
| <> | is not equal | 5<>8 returns true |
| > | is greater than | 5>8 returns false |
| < | is less than | 5<8 returns true |
| >= | is greater than or equal to | 5>=8 returns false |
| <= | is less than or equal to | 5<=8 returns true |

* **Logical Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | and | x=6 y=3  (x < 10 && y > 1) returns true |
| || | or | x=6 y=3  (x==5 || y==5) returns false |
| ! | not | x=6 y=3  !(x==y) returns true |

Conditional statements are used to perform different actions based on different conditions.

## Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

* **if statement** - use this statement to execute some code only if a specified condition is true
* **if...else statement** - use this statement to execute some code if a condition is true and another code if the condition is false
* **if...elseif....else statement** - use this statement to select one of several blocks of code to be executed
* **switch statement** - use this statement to select one of many blocks of code to be executed

## The if Statement

Use the if statement to execute some code only if a specified condition is true.

### Syntax

|  |
| --- |
| if (*condition*) *code to be executed if condition is true;* |

The following example will output "Have a nice weekend!" if the current day is Friday:

|  |
| --- |
| <html> <body>  <?php $d=date("D"); if ($d=="Fri") echo "Have a nice weekend!"; ?>  </body> </html> |

Notice that there is no ..else.. in this syntax. The code is executed **only if the specified condition is true**.

## The if...else Statement

Use the if....else statement to execute some code if a condition is true and another code if a condition is false.

### Syntax

|  |
| --- |
| if (*condition*)   *code to be executed if condition is true;* else   *code to be executed if condition is false;* |

### Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

|  |
| --- |
| <html> <body>  <?php $d=date("D"); if ($d=="Fri")   echo "Have a nice weekend!"; else   echo "Have a nice day!"; ?>  </body> </html> |

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

|  |
| --- |
| <html> <body>  <?php $d=date("D"); if ($d=="Fri")   {   echo "Hello!<br />";   echo "Have a nice weekend!";   echo "See you on Monday!";   } ?>  </body> </html> |

## The if...elseif....else Statement

Use the if....elseif...else statement to select one of several blocks of code to be executed.

### Syntax

|  |
| --- |
| if (*condition*)   *code to be executed if condition is true;* elseif (*condition*)   *code to be executed if condition is true;* else   *code to be executed if condition is false;* |

### Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

|  |
| --- |
| <html> <body>  <?php $d=date("D"); if ($d=="Fri")   echo "Have a nice weekend!"; elseif ($d=="Sun")   echo "Have a nice Sunday!"; else   echo "Have a nice day!"; ?>  </body> </html> |

Conditional statements are used to perform different actions based on different conditions.

## The PHP Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

### Syntax

|  |
| --- |
| switch (*n*) { case *label1:*   *code to be executed if n=label1;*   break; case *label2:*   *code to be executed if n=label2;*   break; default:   *code to be executed if n is different from both label1 and label2;* } |

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically. The default statement is used if no match is found.

### Example

|  |
| --- |
| <html> <body>  <?php switch ($x) { case 1:   echo "Number 1";   break; case 2:   echo "Number 2";   break; case 3:   echo "Number 3";   break; default:   echo "No number between 1 and 3"; } ?>  </body> </html> |

An array stores multiple values in one single variable.

## What is an Array?

A variable is a storage area holding a number or text. The problem is, a variable will hold only one value.

An array is a special variable, which can store multiple values in one single variable.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

|  |
| --- |
| $cars1="Saab"; $cars2="Volvo"; $cars3="BMW"; |

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array!

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own index so that it can be easily accessed.

In PHP, there are three kind of arrays:

* **Numeric array** - An array with a numeric index
* **Associative array** - An array where each ID key is associated with a value
* **Multidimensional array** - An array containing one or more arrays

## Numeric Arrays

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

|  |
| --- |
| $cars=array("Saab","Volvo","BMW","Toyota"); |

2. In the following example we assign the index manually:

|  |
| --- |
| $cars[0]="Saab"; $cars[1]="Volvo"; $cars[2]="BMW"; $cars[3]="Toyota"; |

### Example

In the following example you access the variable values by referring to the array name and index:

|  |
| --- |
| <?php $cars[0]="Saab"; $cars[1]="Volvo"; $cars[2]="BMW"; $cars[3]="Toyota";  echo $cars[0] . " and " . $cars[1] . " are Swedish cars."; ?> |

The code above will output:

|  |
| --- |
| Saab and Volvo are Swedish cars. |

## Associative Arrays

An associative array, each ID key is associated with a value.

When storing data about specific named values, a numerical array is not always the best way to do it.

With associative arrays we can use the values as keys and assign values to them.

### Example 1

In this example we use an array to assign ages to the different persons:

|  |
| --- |
| $ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34); |

### Example 2

This example is the same as example 1, but shows a different way of creating the array:

|  |
| --- |
| $ages['Peter'] = "32"; $ages['Quagmire'] = "30"; $ages['Joe'] = "34"; |

The ID keys can be used in a script:

|  |
| --- |
| <?php $ages['Peter'] = "32"; $ages['Quagmire'] = "30"; $ages['Joe'] = "34";  echo "Peter is " . $ages['Peter'] . " years old."; ?> |

The code above will output:

|  |
| --- |
| Peter is 32 years old. |

## Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

### Example

In this example we create a multidimensional array, with automatically assigned ID keys:

|  |
| --- |
| $families = array   (   "Griffin"=>array   (   "Peter",   "Lois",   "Megan"   ),   "Quagmire"=>array   (   "Glenn"   ),   "Brown"=>array   (   "Cleveland",   "Loretta",   "Junior"   )   ); |

The array above would look like this if written to the output:

|  |
| --- |
| Array ( [Griffin] => Array   (   [0] => Peter   [1] => Lois   [2] => Megan   ) [Quagmire] => Array   (   [0] => Glenn   ) [Brown] => Array   (   [0] => Cleveland   [1] => Loretta   [2] => Junior   ) ) |

### Example 2

Lets try displaying a single value from the array above:

|  |
| --- |
| echo "Is " . $families['Griffin'][2] .  " a part of the Griffin family?"; |

The code above will output:

|  |
| --- |
| Is Megan a part of the Griffin family? |

## Complete PHP Array Reference

For a complete reference of all array functions, go to our [complete PHP Array Reference](http://www.w3schools.com/php/php_ref_array.asp).

The reference contains a brief description, and examples of use, for each function!

Loops execute a block of code a specified number of times, or while a specified condition is true.

## PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In PHP, we have the following looping statements:

* **while** - loops through a block of code while a specified condition is true
* **do...while** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
* **for** - loops through a block of code a specified number of times
* **foreach** - loops through a block of code for each element in an array

## The while Loop

The while loop executes a block of code while a condition is true.

### Syntax

|  |
| --- |
| while (*condition*)   { *code to be executed*;   } |

### Example

The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

|  |
| --- |
| <html> <body>  <?php $i=1; while($i<=5)   {   echo "The number is " . $i . "<br />";   $i++;   } ?>  </body> </html> |

Output:

|  |
| --- |
| The number is 1 The number is 2 The number is 3 The number is 4 The number is 5 |

## The do...while Statement

The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true.

### Syntax

|  |
| --- |
| do   { *code to be executed;* } while (*condition*); |

### Example

The example below defines a loop that starts with i=1. It will then increment i with 1, and write some output. Then the condition is checked, and the loop will continue to run as long as i is less than, or equal to 5:

|  |
| --- |
| <html> <body>  <?php $i=1; do   {   $i++;   echo "The number is " . $i . "<br />";   } while ($i<=5); ?>  </body> </html> |

Output:

|  |
| --- |
| The number is 2 The number is 3 The number is 4 The number is 5 The number is 6 |

The for loop and the foreach loop will be explained in the next chapter.

Loops execute a block of code a specified number of times, or while a specified condition is true.

## The for Loop

The for loop is used when you know in advance how many times the script should run.

### Syntax

|  |
| --- |
| for (*init; condition; increment*)   {   *code to be executed;*   } |

Parameters:

* *init*: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
* *condition*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
* *increment*: Mostly used to increment a counter (but can be any code to be executed at the end of the loop)

**Note:** Each of the parameters above can be empty, or have multiple expressions (separated by commas).

### Example

The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

|  |
| --- |
| <html> <body>  <?php for ($i=1; $i<=5; $i++)   {   echo "The number is " . $i . "<br />";   } ?>  </body> </html> |

Output:

|  |
| --- |
| The number is 1 The number is 2 The number is 3 The number is 4 The number is 5 |

## The foreach Loop

The foreach loop is used to loop through arrays.

### Syntax

|  |
| --- |
| foreach ($*array* as$*value*)   {   *code to be executed;*   } |

For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

### Example

The following example demonstrates a loop that will print the values of the given array:

|  |
| --- |
| <html> <body>  <?php $x=array("one","two","three"); foreach ($x as $value)   {   echo $value . "<br />";   } ?>  </body> </html> |

Output:

|  |
| --- |
| one two three |

The real power of PHP comes from its functions.

In PHP, there are more than 700 built-in functions.

## PHP Built-in Functions

For a complete reference and examples of the built-in functions, please visit our [PHP Reference.](http://www.w3schools.com/php/default.asp)

## PHP Functions

In this chapter we will show you how to create your own functions.

To keep the script from being executed when the page loads, you can put it into a function.

A function will be executed by a call to the function.

You may call a function from anywhere within a page.

## Create a PHP Function

A function will be executed by a call to the function.

### Syntax

|  |
| --- |
| function *functionName*() { *code to be executed*; } |

PHP function guidelines:

* Give the function a name that reflects what the function does
* The function name can start with a letter or underscore (not a number)

### Example

A simple function that writes my name when it is called:

|  |
| --- |
| <html> <body>  <?php function writeName() { echo "Kai Jim Refsnes"; }  echo "My name is "; writeName(); ?>  </body> </html> |

Output:

|  |
| --- |
| My name is Kai Jim Refsnes |

## PHP Functions - Adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable.

Parameters are specified after the function name, inside the parentheses.

### Example 1

The following example will write different first names, but equal last name:

|  |
| --- |
| <html> <body>  <?php function writeName($fname) { echo $fname . " Refsnes.<br />"; }  echo "My name is "; writeName("Kai Jim"); echo "My sister's name is "; writeName("Hege"); echo "My brother's name is "; writeName("Stale"); ?>  </body> </html> |

Output:

|  |
| --- |
| My name is Kai Jim Refsnes. My sister's name is Hege Refsnes. My brother's name is Stale Refsnes. |

### Example 2

The following function has two parameters:

|  |
| --- |
| <html> <body>  <?php function writeName($fname,$punctuation) { echo $fname . " Refsnes" . $punctuation . "<br />"; }  echo "My name is "; writeName("Kai Jim","."); echo "My sister's name is "; writeName("Hege","!"); echo "My brother's name is "; writeName("Ståle","?"); ?>  </body> </html> |

Output:

|  |
| --- |
| My name is Kai Jim Refsnes. My sister's name is Hege Refsnes! My brother's name is Ståle Refsnes? |

## PHP Functions - Return values

To let a function return a value, use the return statement.

### Example

|  |
| --- |
| <html> <body>  <?php function add($x,$y) { $total=$x+$y; return $total; }  echo "1 + 16 = " . add(1,16); ?>  </body> </html> |

Output:

|  |
| --- |
| 1 + 16 = 17 |

The PHP $\_GET and $\_POST variables are used to retrieve information from forms, like user input.

## PHP Form Handling

The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will **automatically** be available to your PHP scripts.

### Example

The example below contains an HTML form with two input fields and a submit button:

|  |
| --- |
| <html> <body>  <form action="welcome.php" method="post"> Name: <input type="text" name="fname" /> Age: <input type="text" name="age" /> <input type="submit" /> </form>  </body> </html> |

When a user fills out the form above and click on the submit button, the form data is sent to a PHP file, called "welcome.php":

"welcome.php" looks like this:

|  |
| --- |
| <html> <body>  Welcome <?php echo $\_POST["fname"]; ?>!<br /> You are <?php echo $\_POST["age"]; ?> years old.  </body> </html> |

Output could be something like this:

|  |
| --- |
| Welcome John! You are 28 years old. |

The PHP $\_GET and $\_POST functions will be explained in the next chapters.

## Form Validation

User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and reduces the server load.

You should consider server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

The built-in $\_GET function is used to collect values in a form with method="get".

## The $\_GET Function

The built-in $\_GET function is used to collect values from a form sent with method="get".

Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

### Example

|  |
| --- |
| <form action="welcome.php" method="get"> Name: <input type="text" name="fname" /> Age: <input type="text" name="age" /> <input type="submit" /> </form> |

When the user clicks the "Submit" button, the URL sent to the server could look something like this:

|  |
| --- |
| http://www.w3schools.com/welcome.php?fname=Peter&age=37 |

The "welcome.php" file can now use the $\_GET function to collect form data (the names of the form fields will automatically be the keys in the $\_GET array):

|  |
| --- |
| Welcome <?php echo $\_GET["fname"]; ?>.<br /> You are <?php echo $\_GET["age"]; ?> years old! |

## When to use method="get"?

When using method="get" in HTML forms, all variable names and values are displayed in the URL.

**Note:** This method should not be used when sending passwords or other sensitive information!

However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

**Note:** The get method is not suitable for very large variable values. It should not be used with values exceeding 2000 characters.

The built-in $\_POST function is used to collect values in a form with method="post".

## The $\_POST Function

The built-in $\_POST function is used to collect values from a form sent with method="post".

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

**Note:** However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the post\_max\_size in the php.ini file).

### Example

|  |
| --- |
| <form action="welcome.php" method="post"> Name: <input type="text" name="fname" /> Age: <input type="text" name="age" /> <input type="submit" /> </form> |

When the user clicks the "Submit" button, the URL will look like this:

|  |
| --- |
| http://www.w3schools.com/welcome.php |

The "welcome.php" file can now use the $\_POST function to collect form data (the names of the form fields will automatically be the keys in the $\_POST array):

|  |
| --- |
| Welcome <?php echo $\_POST["fname"]; ?>!<br /> You are <?php echo $\_POST["age"]; ?> years old. |

## When to use method="post"?

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

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

## The PHP $\_REQUEST Function

The PHP built-in $\_REQUEST function contains the contents of both $\_GET, $\_POST, and $\_COOKIE.

The $\_REQUEST function can be used to collect form data sent with both the GET and POST methods.

### Example

|  |
| --- |
| Welcome <?php echo $\_REQUEST["fname"]; ?>!<br /> You are <?php echo $\_REQUEST["age"]; ?> years old. |

The PHP date() function is used to format a time and/or date.

## The PHP Date() Function

The PHP date() function formats a timestamp to a more readable date and time.

TipA timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

### Syntax

|  |
| --- |
| date(*format*,*timestamp*) |

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| Format | Required. Specifies the format of the timestamp |
| timestamp | Optional. Specifies a timestamp. Default is the current date and time |

## PHP Date() - Format the Date

The required *format* parameter in the date() function specifies how to format the date/time.

Here are some characters that can be used:

* d - Represents the day of the month (01 to 31)
* m - Represents a month (01 to 12)
* Y - Represents a year (in four digits)

A list of all the characters that can be used in the *format* parameter, can be found in our [PHP Date reference](http://www.w3schools.com/php/php_ref_date.asp).

Other characters, like"/", ".", or "-" can also be inserted between the letters to add additional formatting:

|  |
| --- |
| <?php echo date("Y/m/d") . "<br />"; echo date("Y.m.d") . "<br />"; echo date("Y-m-d"); ?> |

The output of the code above could be something like this:

|  |
| --- |
| 2009/05/11 2009.05.11 2009-05-11 |

## PHP Date() - Adding a Timestamp

The optional *timestamp* parameter in the date() function specifies a timestamp. If you do not specify a timestamp, the current date and time will be used.

The mktime() function returns the Unix timestamp for a date.

The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

### Syntax for mktime()

|  |
| --- |
| mktime(hour,minute,second,month,day,year,is\_dst) |

To go one day in the future we simply add one to the day argument of mktime():

|  |
| --- |
| <?php $tomorrow = mktime(0,0,0,date("m"),date("d")+1,date("Y")); echo "Tomorrow is ".date("Y/m/d", $tomorrow); ?> |

The output of the code above could be something like this:

|  |
| --- |
| Tomorrow is 2009/05/12 |

## Complete PHP Date Reference

For a complete reference of all date functions, go to our [complete PHP Date Reference](http://www.w3schools.com/php/php_ref_date.asp).

The reference contains a brief description, and examples of use, for each function!

## Server Side Includes (SSI)

You can insert the content of one PHP file into another PHP file before the server executes it, with the include() or require() function.

The two functions are identical in every way, except how they handle errors:

* include() generates a warning, but the script will continue execution
* require() generates a fatal error, and the script will stop

These two functions are used to create functions, headers, footers, or elements that will be reused on multiple pages.

Server side includes saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. When the header needs to be updated, you can only update the include file, or when you add a new page to your site, you can simply change the menu file (instead of updating the links on all your web pages).

## PHP include() Function

The include() function takes all the content in a specified file and includes it in the current file.

If an error occurs, the include() function generates a warning, but the script will continue execution.

### Example 1

Assume that you have a standard header file, called "header.php". To include the header file in a page, use the include() function:

|  |
| --- |
| <html> <body>  <?php include("header.php"); ?> <h1>Welcome to my home page!</h1> <p>Some text.</p>  </body> </html> |

### Example 2

Assume we have a standard menu file, called "menu.php", that should be used on all pages:

|  |
| --- |
| <a href="/default.php">Home</a> <a href="/tutorials.php">Tutorials</a> <a href="/references.php">References</a> <a href="/examples.php">Examples</a>  <a href="/about.php">About Us</a>  <a href="/contact.php">Contact Us</a> |

All pages in the Web site should include this menu file. Here is how it can be done:

|  |
| --- |
| <html> <body>  <div class="leftmenu"> <?php include("menu.php"); ?> </div>  <h1>Welcome to my home page.</h1> <p>Some text.</p>  </body> </html> |

If you look at the source code of the page above (in a browser), it will look like this:

|  |
| --- |
| <html> <body>  <div class="leftmenu"> <a href="/default.php">Home</a> <a href="/tutorials.php">Tutorials</a> <a href="/references.php">References</a> <a href="/examples.php">Examples</a>  <a href="/about.php">About Us</a>  <a href="/contact.php">Contact Us</a> </div>  <h1>Welcome to my home page!</h1> <p>Some text.</p>  </body> </html> |

## PHP require() Function

The require() function is identical to include(), except that it handles errors differently.

If an error occurs, the include() function generates a warning, but the script will continue execution. The require() generates a fatal error, and the script will stop.

### Error Example include() Function

|  |
| --- |
| <html> <body>  <?php include("wrongFile.php"); echo "Hello World!"; ?>  </body> </html> |

Error message:

|  |
| --- |
| **Warning:** include(wrongFile.php) [function.include]: failed to open stream: No such file or directory in C:\home\website\test.php on line 5  **Warning:** include() [function.include]: Failed opening 'wrongFile.php' for inclusion (include\_path='.;C:\php5\pear') in C:\home\website\test.php on line 5  Hello World! |

Notice that the echo statement is executed! This is because a Warning does not stop the script execution.

### Error Example require() Function

Now, let's run the same example with the require() function.

|  |
| --- |
| <html> <body>  <?php require("wrongFile.php"); echo "Hello World!"; ?>  </body> </html> |

Error message:

|  |
| --- |
| **Warning:** require(wrongFile.php) [function.require]: failed to open stream: No such file or directory in C:\home\website\test.php on line 5  **Fatal error:** require() [function.require]: Failed opening required 'wrongFile.php' (include\_path='.;C:\php5\pear') in C:\home\website\test.php on line 5 |

The echo statement is not executed, because the script execution stopped after the fatal error.

It is recommended to use the require() function instead of include(), because scripts should not continue after an error.

The fopen() function is used to open files in PHP.

## Opening a File

The fopen() function is used to open files in PHP.

The first parameter of this function contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened:

|  |
| --- |
| <html> <body>  <?php $file=fopen("welcome.txt","r"); ?>  </body> </html> |

The file may be opened in one of the following modes:

|  |  |
| --- | --- |
| **Modes** | **Description** |
| r | Read only. Starts at the beginning of the file |
| r+ | Read/Write. Starts at the beginning of the file |
| w | Write only. Opens and clears the contents of file; or creates a new file if it doesn't exist |
| w+ | Read/Write. Opens and clears the contents of file; or creates a new file if it doesn't exist |
| a | Append. Opens and writes to the end of the file or creates a new file if it doesn't exist |
| a+ | Read/Append. Preserves file content by writing to the end of the file |
| x | Write only. Creates a new file. Returns FALSE and an error if file already exists |
| x+ | Read/Write. Creates a new file. Returns FALSE and an error if file already exists |

**Note:** If the fopen() function is unable to open the specified file, it returns 0 (false).

### Example

The following example generates a message if the fopen() function is unable to open the specified file:

|  |
| --- |
| <html> <body>  <?php $file=fopen("welcome.txt","r") or exit("Unable to open file!"); ?>  </body> </html> |

## Closing a File

The fclose() function is used to close an open file:

|  |
| --- |
| <?php $file = fopen("test.txt","r");  //some code to be executed  fclose($file); ?> |

## Check End-of-file

The feof() function checks if the "end-of-file" (EOF) has been reached.  
  
The feof() function is useful for looping through data of unknown length.

**Note:** You cannot read from files opened in w, a, and x mode!

|  |
| --- |
| if (feof($file)) echo "End of file"; |

## Reading a File Line by Line

The fgets() function is used to read a single line from a file.

**Note:** After a call to this function the file pointer has moved to the next line.

### Example

The example below reads a file line by line, until the end of file is reached:

|  |
| --- |
| <?php $file = fopen("welcome.txt", "r") or exit("Unable to open file!"); //Output a line of the file until the end is reached while(!feof($file))   {   echo fgets($file). "<br />";   } fclose($file); ?> |

## Reading a File Character by Character

The fgetc() function is used to read a single character from a file.

**Note:** After a call to this function the file pointer moves to the next character.

### Example

The example below reads a file character by character, until the end of file is reached:

|  |
| --- |
| <?php $file=fopen("welcome.txt","r") or exit("Unable to open file!"); while (!feof($file))   {   echo fgetc($file);   } fclose($file); ?> |

## PHP Filesystem Reference

For a full reference of the PHP filesystem functions, visit our [PHP Filesystem Reference](http://www.w3schools.com/php/php_ref_filesystem.asp).

With PHP, it is possible to upload files to the server.

**Create an Upload-File Form**

To allow users to upload files from a form can be very useful.

Look at the following HTML form for uploading files:

|  |
| --- |
| <html> <body>  <form action="upload\_file.php" method="post" enctype="multipart/form-data"> <label for="file">Filename:</label> <input type="file" name="file" id="file" />  <br /> <input type="submit" name="submit" value="Submit" /> </form>  </body> </html> |

Notice the following about the HTML form above:

* The enctype attribute of the <form> tag specifies which content-type to use when submitting the form. "multipart/form-data" is used when a form requires binary data, like the contents of a file, to be uploaded
* The type="file" attribute of the <input> tag specifies that the input should be processed as a file. For example, when viewed in a browser, there will be a browse-button next to the input field

**Note:** Allowing users to upload files is a big security risk. Only permit trusted users to perform file uploads.

**Create The Upload Script**

The "upload\_file.php" file contains the code for uploading a file:

|  |
| --- |
| <?php if ($\_FILES["file"]["error"] > 0)   {   echo "Error: " . $\_FILES["file"]["error"] . "<br />";   } else   {   echo "Upload: " . $\_FILES["file"]["name"] . "<br />";   echo "Type: " . $\_FILES["file"]["type"] . "<br />";   echo "Size: " . ($\_FILES["file"]["size"] / 1024) . " Kb<br />";   echo "Stored in: " . $\_FILES["file"]["tmp\_name"];   } ?> |

By using the global PHP $\_FILES array you can upload files from a client computer to the remote server.

The first parameter is the form's input name and the second index can be either "name", "type", "size", "tmp\_name" or "error". Like this:

* $\_FILES["file"]["name"] - the name of the uploaded file
* $\_FILES["file"]["type"] - the type of the uploaded file
* $\_FILES["file"]["size"] - the size in bytes of the uploaded file
* $\_FILES["file"]["tmp\_name"] - the name of the temporary copy of the file stored on the server
* $\_FILES["file"]["error"] - the error code resulting from the file upload

This is a very simple way of uploading files. For security reasons, you should add restrictions on what the user is allowed to upload.

**Restrictions on Upload**

In this script we add some restrictions to the file upload. The user may only upload .gif or .jpeg files and the file size must be under 20 kb:

|  |
| --- |
| <?php if ((($\_FILES["file"]["type"] == "image/gif") || ($\_FILES["file"]["type"] == "image/jpeg") || ($\_FILES["file"]["type"] == "image/pjpeg")) && ($\_FILES["file"]["size"] < 20000))   {   if ($\_FILES["file"]["error"] > 0)     {     echo "Error: " . $\_FILES["file"]["error"] . "<br />";     }   else     {     echo "Upload: " . $\_FILES["file"]["name"] . "<br />";     echo "Type: " . $\_FILES["file"]["type"] . "<br />";     echo "Size: " . ($\_FILES["file"]["size"] / 1024) . " Kb<br />";     echo "Stored in: " . $\_FILES["file"]["tmp\_name"];     }   } else   {   echo "Invalid file";   } ?> |

**Note:** For IE to recognize jpg files the type must be pjpeg, for FireFox it must be jpeg.

**Saving the Uploaded File**

The examples above create a temporary copy of the uploaded files in the PHP temp folder on the server.

The temporary copied files disappears when the script ends. To store the uploaded file we need to copy it to a different location:

|  |
| --- |
| <?php if ((($\_FILES["file"]["type"] == "image/gif") || ($\_FILES["file"]["type"] == "image/jpeg") || ($\_FILES["file"]["type"] == "image/pjpeg")) && ($\_FILES["file"]["size"] < 20000))   {   if ($\_FILES["file"]["error"] > 0)     {     echo "Return Code: " . $\_FILES["file"]["error"] . "<br />";     }   else     {     echo "Upload: " . $\_FILES["file"]["name"] . "<br />";     echo "Type: " . $\_FILES["file"]["type"] . "<br />";     echo "Size: " . ($\_FILES["file"]["size"] / 1024) . " Kb<br />";     echo "Temp file: " . $\_FILES["file"]["tmp\_name"] . "<br />";      if (file\_exists("upload/" . $\_FILES["file"]["name"]))       {       echo $\_FILES["file"]["name"] . " already exists. ";       }     else       {       move\_uploaded\_file($\_FILES["file"]["tmp\_name"],       "upload/" . $\_FILES["file"]["name"]);       echo "Stored in: " . "upload/" . $\_FILES["file"]["name"];       }     }   } else   {   echo "Invalid file";   } ?> |

The script above checks if the file already exists, if it does not, it copies the file to the specified folder.

**Note:** This example saves the file to a new folder called "upload"

A cookie is often used to identify a user.

## What is a Cookie?

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

## How to Create a Cookie?

The setcookie() function is used to set a cookie.

**Note:** The setcookie() function must appear BEFORE the <html> tag.

### Syntax

|  |
| --- |
| setcookie(name, value, expire, path, domain); |

### Example 1

In the example below, we will create a cookie named "user" and assign the value "Alex Porter" to it. We also specify that the cookie should expire after one hour:

|  |
| --- |
| <?php setcookie("user", "Alex Porter", time()+3600); ?>  <html> ..... |

**Note:** The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

### Example 2

You can also set the expiration time of the cookie in another way. It may be easier than using seconds.

|  |
| --- |
| <?php $expire=time()+60\*60\*24\*30; setcookie("user", "Alex Porter", $expire); ?>  <html> ..... |

In the example above the expiration time is set to a month (*60 sec \* 60 min \* 24 hours \* 30 days*).

## How to Retrieve a Cookie Value?

The PHP $\_COOKIE variable is used to retrieve a cookie value.   
  
In the example below, we retrieve the value of the cookie named "user" and display it on a page:

|  |
| --- |
| <?php // Print a cookie echo $\_COOKIE["user"];  // A way to view all cookies print\_r($\_COOKIE); ?> |

In the following example we use the isset() function to find out if a cookie has been set:

|  |
| --- |
| <html> <body>  <?php if (isset($\_COOKIE["user"]))   echo "Welcome " . $\_COOKIE["user"] . "!<br />"; else   echo "Welcome guest!<br />"; ?>  </body> </html> |

## How to Delete a Cookie?

When deleting a cookie you should assure that the expiration date is in the past.

Delete example:

|  |
| --- |
| <?php // set the expiration date to one hour ago setcookie("user", "", time()-3600); ?> |

## What if a Browser Does NOT Support Cookies?

If your application deals with browsers that do not support cookies, you will have to use other methods to pass information from one page to another in your application. One method is to pass the data through forms (forms and user input are described earlier in this tutorial).

The form below passes the user input to "welcome.php" when the user clicks on the "Submit" button:

|  |
| --- |
| <html> <body>  <form action="welcome.php" method="post"> Name: <input type="text" name="name" /> Age: <input type="text" name="age" /> <input type="submit" /> </form>  </body> </html> |

Retrieve the values in the "welcome.php" file like this:

|  |
| --- |
| <html> <body>  Welcome <?php echo $\_POST["name"]; ?>.<br /> You are <?php echo $\_POST["age"]; ?> years old.  </body> </html> |

A PHP session variable is used to store information about, or change settings for a user session. Session variables hold information about one single user, and are available to all pages in one application.

**PHP Session Variables**

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

**Starting a PHP Session**

Before you can store user information in your PHP session, you must first start up the session.

**Note:** The session\_start() function must appear BEFORE the <html> tag:

|  |
| --- |
| <?php session\_start(); ?>  <html> <body>  </body> </html> |

The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

**Storing a Session Variable**

The correct way to store and retrieve session variables is to use the PHP $\_SESSION variable:

|  |
| --- |
| <?php session\_start(); // store session data $\_SESSION['views']=1; ?>  <html> <body>  <?php //retrieve session data echo "Pageviews=". $\_SESSION['views']; ?>  </body> </html> |

Output:

|  |
| --- |
| Pageviews=1 |

In the example below, we create a simple page-views counter. The isset() function checks if the "views" variable has already been set. If "views" has been set, we can increment our counter. If "views" doesn't exist, we create a "views" variable, and set it to 1:

|  |
| --- |
| <?php session\_start();  if(isset($\_SESSION['views'])) $\_SESSION['views']=$\_SESSION['views']+1; else $\_SESSION['views']=1; echo "Views=". $\_SESSION['views']; ?> |

**Destroying a Session**

If you wish to delete some session data, you can use the unset() or the session\_destroy() function.

The unset() function is used to free the specified session variable:

|  |
| --- |
| <?php unset($\_SESSION['views']); ?> |

You can also completely destroy the session by calling the session\_destroy() function:

|  |
| --- |
| <?php session\_destroy(); ?> |

**Note:** session\_destroy() will reset your session and you will lose all your stored session data.

PHP allows you to send e-mails directly from a script.

**The PHP mail() Function**

The PHP mail() function is used to send emails from inside a script.

**Syntax**

|  |
| --- |
| mail(to,subject,message,headers,parameters) |

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| To | Required. Specifies the receiver / receivers of the email |
| subject | Required. Specifies the subject of the email. **Note:** This parameter cannot contain any newline characters |
| message | Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters |
| headers | Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n) |
| parameters | Optional. Specifies an additional parameter to the sendmail program |

**Note:** For the mail functions to be available, PHP requires an installed and working email system. The program to be used is defined by the configuration settings in the php.ini file. Read more in our [PHP Mail reference](http://www.w3schools.com/php/php_ref_mail.asp).

**PHP Simple E-Mail**

The simplest way to send an email with PHP is to send a text email.

In the example below we first declare the variables ($to, $subject, $message, $from, $headers), then we use the variables in the mail() function to send an e-mail:

|  |
| --- |
| <?php $to = "someone@example.com"; $subject = "Test mail"; $message = "Hello! This is a simple email message."; $from = "someonelse@example.com"; $headers = "From:" . $from; mail($to,$subject,$message,$headers); echo "Mail Sent."; ?> |

**PHP Mail Form**

With PHP, you can create a feedback-form on your website. The example below sends a text message to a specified e-mail address:

|  |
| --- |
| <html> <body>  <?php if (isset($\_REQUEST['email'])) //if "email" is filled out, send email   {   //send email   $email = $\_REQUEST['email'] ;   $subject = $\_REQUEST['subject'] ;   $message = $\_REQUEST['message'] ;   mail("someone@example.com", "$subject",   $message, "From:" . $email);   echo "Thank you for using our mail form";   } else //if "email" is not filled out, display the form   {   echo "<form method='post' action='mailform.php'>   Email: <input name='email' type='text' /><br />   Subject: <input name='subject' type='text' /><br />   Message:<br />   <textarea name='message' rows='15' cols='40'>   </textarea><br />   <input type='submit' />   </form>";   } ?>  </body> </html> |

This is how the example above works:

* First, check if the email input field is filled out
* If it is not set (like when the page is first visited); output the HTML form
* If it is set (after the form is filled out); send the email from the form
* When submit is pressed after the form is filled out, the page reloads, sees that the email input is set, and sends the email

**Note:** This is the simplest way to send e-mail, but it is not secure. In the next chapter of this tutorial you can read more about vulnerabilities in e-mail scripts, and how to validate user input to make it more secure.

**PHP Mail Reference**

For more information about the PHP mail() function, visit our [PHP Mail Reference](http://www.w3schools.com/php/php_ref_mail.asp).

There is a weakness in the PHP e-mail script in the previous chapter.

**PHP E-mail Injections**

First, look at the PHP code from the previous chapter:

|  |
| --- |
| <html> <body>  <?php if (isset($\_REQUEST['email'])) //if "email" is filled out, send email   {   //send email   $email = $\_REQUEST['email'] ;   $subject = $\_REQUEST['subject'] ;   $message = $\_REQUEST['message'] ;   mail("someone@example.com", "Subject: $subject",   $message, "From: $email" );   echo "Thank you for using our mail form";   } else //if "email" is not filled out, display the form   {   echo "<form method='post' action='mailform.php'>   Email: <input name='email' type='text' /><br />   Subject: <input name='subject' type='text' /><br />   Message:<br />   <textarea name='message' rows='15' cols='40'>   </textarea><br />   <input type='submit' />   </form>";   } ?>  </body> </html> |

The problem with the code above is that unauthorized users can insert data into the mail headers via the input form.

What happens if the user adds the following text to the email input field in the form?

|  |
| --- |
| someone@example.com%0ACc:person2@example.com %0ABcc:person3@example.com,person3@example.com, anotherperson4@example.com,person5@example.com %0ABTo:person6@example.com |

The mail() function puts the text above into the mail headers as usual, and now the header has an extra Cc:, Bcc:, and To: field. When the user clicks the submit button, the e-mail will be sent to all of the addresses above!

**PHP Stopping E-mail Injections**

The best way to stop e-mail injections is to validate the input.

The code below is the same as in the previous chapter, but now we have added an input validator that checks the email field in the form:

|  |
| --- |
| <html> <body> <?php function spamcheck($field)   {   //filter\_var() sanitizes the e-mail   //address using FILTER\_SANITIZE\_EMAIL   $field=filter\_var($field, FILTER\_SANITIZE\_EMAIL);    //filter\_var() validates the e-mail   //address using FILTER\_VALIDATE\_EMAIL   if(filter\_var($field, FILTER\_VALIDATE\_EMAIL))     {     return TRUE;     }   else     {     return FALSE;     }   }  if (isset($\_REQUEST['email']))   {//if "email" is filled out, proceed    //check if the email address is invalid   $mailcheck = spamcheck($\_REQUEST['email']);   if ($mailcheck==FALSE)     {     echo "Invalid input";     }   else     {//send email     $email = $\_REQUEST['email'] ;     $subject = $\_REQUEST['subject'] ;     $message = $\_REQUEST['message'] ;     mail("someone@example.com", "Subject: $subject",     $message, "From: $email" );     echo "Thank you for using our mail form";     }   } else   {//if "email" is not filled out, display the form   echo "<form method='post' action='mailform.php'>   Email: <input name='email' type='text' /><br />   Subject: <input name='subject' type='text' /><br />   Message:<br />   <textarea name='message' rows='15' cols='40'>   </textarea><br />   <input type='submit' />   </form>";   } ?>  </body> </html> |

In the code above we use PHP filters to validate input:

* The FILTER\_SANITIZE\_EMAIL filter removes all illegal e-mail characters from a string
* The FILTER\_VALIDATE\_EMAIL filter validates value as an e-mail address

You can read more about filters in our [PHP Filter](http://www.w3schools.com/php/php_filter.asp) chapter.

The default error handling in PHP is very simple. An error message with filename, line number and a message describing the error is sent to the browser.

**PHP Error Handling**

When creating scripts and web applications, error handling is an important part. If your code lacks error checking code, your program may look very unprofessional and you may be open to security risks.

This tutorial contains some of the most common error checking methods in PHP.

We will show different error handling methods:

* Simple "die()" statements
* Custom errors and error triggers
* Error reporting

**Basic Error Handling: Using the die() function**

The first example shows a simple script that opens a text file:

|  |
| --- |
| <?php $file=fopen("welcome.txt","r"); ?> |

If the file does not exist you might get an error like this:

|  |
| --- |
| **Warning**: fopen(welcome.txt) [function.fopen]: failed to open stream: No such file or directory in **C:\webfolder\test.php** on line **2** |

To avoid that the user gets an error message like the one above, we test if the file exist before we try to access it:

|  |
| --- |
| <?php if(!file\_exists("welcome.txt"))   {   die("File not found");   } else   {   $file=fopen("welcome.txt","r");   } ?> |

Now if the file does not exist you get an error like this:

|  |
| --- |
| File not found |

The code above is more efficient than the earlier code, because it uses a simple error handling mechanism to stop the script after the error.

However, simply stopping the script is not always the right way to go. Let's take a look at alternative PHP functions for handling errors.

**Creating a Custom Error Handler**

Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP.

This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

**Syntax**

|  |
| --- |
| error\_function(error\_level,error\_message, error\_file,error\_line,error\_context) |

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| error\_level | Required. Specifies the error report level for the user-defined error. Must be a value number. See table below for possible error report levels |
| error\_message | Required. Specifies the error message for the user-defined error |
| error\_file | Optional. Specifies the filename in which the error occurred |
| error\_line | Optional. Specifies the line number in which the error occurred |
| error\_context | Optional. Specifies an array containing every variable, and their values, in use when the error occurred |

**Error Report levels**

These error report levels are the different types of error the user-defined error handler can be used for:

|  |  |  |
| --- | --- | --- |
| **Value** | **Constant** | **Description** |
| 2 | E\_WARNING | Non-fatal run-time errors. Execution of the script is not halted |
| 8 | E\_NOTICE | Run-time notices. The script found something that might be an error, but could also happen when running a script normally |
| 256 | E\_USER\_ERROR | Fatal user-generated error. This is like an E\_ERROR set by the programmer using the PHP function trigger\_error() |
| 512 | E\_USER\_WARNING | Non-fatal user-generated warning. This is like an E\_WARNING set by the programmer using the PHP function trigger\_error() |
| 1024 | E\_USER\_NOTICE | User-generated notice. This is like an E\_NOTICE set by the programmer using the PHP function trigger\_error() |
| 4096 | E\_RECOVERABLE\_ERROR | Catchable fatal error. This is like an E\_ERROR but can be caught by a user defined handle (see also set\_error\_handler()) |
| 8191 | E\_ALL | All errors and warnings, except level E\_STRICT (E\_STRICT will be part of E\_ALL as of PHP 6.0) |

Now lets create a function to handle errors:

|  |
| --- |
| function customError($errno, $errstr)   {   echo "<b>Error:</b> [$errno] $errstr<br />";   echo "Ending Script";   die();   } |

The code above is a simple error handling function. When it is triggered, it gets the error level and an error message. It then outputs the error level and message and terminates the script.

Now that we have created an error handling function we need to decide when it should be triggered.

**Set Error Handler**

The default error handler for PHP is the built in error handler. We are going to make the function above the default error handler for the duration of the script.

It is possible to change the error handler to apply for only some errors, that way the script can handle different errors in different ways. However, in this example we are going to use our custom error handler for all errors:

|  |
| --- |
| set\_error\_handler("customError"); |

Since we want our custom function to handle all errors, the set\_error\_handler() only needed one parameter, a second parameter could be added to specify an error level.

**Example**

Testing the error handler by trying to output variable that does not exist:

|  |
| --- |
| <?php //error handler function function customError($errno, $errstr)   {   echo "<b>Error:</b> [$errno] $errstr";   }  //set error handler set\_error\_handler("customError");  //trigger error echo($test); ?> |

The output of the code above should be something like this:

|  |
| --- |
| **Error:** [8] Undefined variable: test |

**Trigger an Error**

In a script where users can input data it is useful to trigger errors when an illegal input occurs. In PHP, this is done by the trigger\_error() function.

**Example**

In this example an error occurs if the "test" variable is bigger than "1":

|  |
| --- |
| <?php $test=2; if ($test>1) { trigger\_error("Value must be 1 or below"); } ?> |

The output of the code above should be something like this:

|  |
| --- |
| **Notice**: Value must be 1 or below in **C:\webfolder\test.php** on line **6** |

An error can be triggered anywhere you wish in a script, and by adding a second parameter, you can specify what error level is triggered.

Possible error types:

* E\_USER\_ERROR - Fatal user-generated run-time error. Errors that can not be recovered from. Execution of the script is halted
* E\_USER\_WARNING - Non-fatal user-generated run-time warning. Execution of the script is not halted
* E\_USER\_NOTICE - Default. User-generated run-time notice. The script found something that might be an error, but could also happen when running a script normally

**Example**

In this example an E\_USER\_WARNING occurs if the "test" variable is bigger than "1". If an E\_USER\_WARNING occurs we will use our custom error handler and end the script:

|  |
| --- |
| <?php //error handler function function customError($errno, $errstr)   {   echo "<b>Error:</b> [$errno] $errstr<br />";   echo "Ending Script";   die();   }  //set error handler set\_error\_handler("customError",E\_USER\_WARNING);  //trigger error $test=2; if ($test>1)   {   trigger\_error("Value must be 1 or below",E\_USER\_WARNING);   } ?> |

The output of the code above should be something like this:

|  |
| --- |
| **Error:** [512] Value must be 1 or below Ending Script |

Now that we have learned to create our own errors and how to trigger them, lets take a look at error logging.

**Error Logging**

By default, PHP sends an error log to the servers logging system or a file, depending on how the error\_log configuration is set in the php.ini file. By using the error\_log() function you can send error logs to a specified file or a remote destination.

Sending errors messages to yourself by e-mail can be a good way of getting notified of specific errors.

**Send an Error Message by E-Mail**

In the example below we will send an e-mail with an error message and end the script, if a specific error occurs:

|  |
| --- |
| <?php //error handler function function customError($errno, $errstr)   {   echo "<b>Error:</b> [$errno] $errstr<br />";   echo "Webmaster has been notified";   error\_log("Error: [$errno] $errstr",1,   "someone@example.com","From: webmaster@example.com");   }  //set error handler set\_error\_handler("customError",E\_USER\_WARNING);  //trigger error $test=2; if ($test>1)   {   trigger\_error("Value must be 1 or below",E\_USER\_WARNING);   } ?> |

The output of the code above should be something like this:

|  |
| --- |
| **Error:** [512] Value must be 1 or below Webmaster has been notified |

And the mail received from the code above looks like this:

|  |
| --- |
| Error: [512] Value must be 1 or below |

This should not be used with all errors. Regular errors should be logged on the server using the default PHP logging system.

Exceptions are used to change the normal flow of a script if a specified error occurs

**What is an Exception**

With PHP 5 came a new object oriented way of dealing with errors.

Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.  
  
This is what normally happens when an exception is triggered:

* The current code state is saved
* The code execution will switch to a predefined (custom) exception handler function
* Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

We will show different error handling methods:

* Basic use of Exceptions
* Creating a custom exception handler
* Multiple exceptions
* Re-throwing an exception
* Setting a top level exception handler

**Note:** Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point.

**Basic Use of Exceptions**

When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.

If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

Lets try to throw an exception without catching it:

|  |
| --- |
| <?php //create function with an exception function checkNum($number)   {   if($number>1)     {     throw new Exception("Value must be 1 or below");     }   return true;   }  //trigger exception checkNum(2); ?> |

The code above will get an error like this:

|  |
| --- |
| **Fatal error**: Uncaught exception 'Exception' with message 'Value must be 1 or below' in C:\webfolder\test.php:6 Stack trace: #0 C:\webfolder\test.php(12): checkNum(28) #1 {main} thrown in **C:\webfolder\test.php** on line **6** |

**Try, throw and catch**

To avoid the error from the example above, we need to create the proper code to handle an exception.

Proper exception code should include:

1. Try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
2. Throw - This is how you trigger an exception. Each "throw" must have at least one "catch"
3. Catch - A "catch" block retrieves an exception and creates an object containing the exception information

Lets try to trigger an exception with valid code:

|  |
| --- |
| <?php //create function with an exception function checkNum($number)   {   if($number>1)     {     throw new Exception("Value must be 1 or below");     }   return true;   }  //trigger exception in a "try" block try   {   checkNum(2);   //If the exception is thrown, this text will not be shown   echo 'If you see this, the number is 1 or below';   }  //catch exception catch(Exception $e)   {   echo 'Message: ' .$e->getMessage();   } ?> |

The code above will get an error like this:

|  |
| --- |
| Message: Value must be 1 or below |

**Example explained:**

The code above throws an exception and catches it:

1. The checkNum() function is created. It checks if a number is greater than 1. If it is, an exception is thrown
2. The checkNum() function is called in a "try" block
3. The exception within the checkNum() function is thrown
4. The "catch" block retrives the exception and creates an object ($e) containing the exception information
5. The error message from the exception is echoed by calling $e->getMessage() from the exception object

However, one way to get around the "every throw must have a catch" rule is to set a top level exception handler to handle errors that slip through.

**Creating a Custom Exception Class**

Creating a custom exception handler is quite simple. We simply create a special class with functions that can be called when an exception occurs in PHP. The class must be an extension of the exception class.

The custom exception class inherits the properties from PHP's exception class and you can add custom functions to it.

Lets create an exception class:

|  |
| --- |
| <?php class customException extends Exception   {   public function errorMessage()     {     //error message     $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()     .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';     return $errorMsg;     }   }  $email = "someone@example...com";  try   {   //check if   if(filter\_var($email, FILTER\_VALIDATE\_EMAIL) === FALSE)     {     //throw exception if email is not valid     throw new customException($email);     }   }  catch (customException $e)   {   //display custom message   echo $e->errorMessage();   } ?> |

The new class is a copy of the old exception class with an addition of the errorMessage() function. Since it is a copy of the old class, and it inherits the properties and methods from the old class, we can use the exception class methods like getLine() and getFile() and getMessage().

**Example explained:**

The code above throws an exception and catches it with a custom exception class:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is not a valid e-mail address
4. The "try" block is executed and an exception is thrown since the e-mail address is invalid
5. The "catch" block catches the exception and displays the error message

**Multiple Exceptions**

It is possible for a script to use multiple exceptions to check for multiple conditions.

It is possible to use several if..else blocks, a switch, or nest multiple exceptions. These exceptions can use different exception classes and return different error messages:

|  |
| --- |
| <?php class customException extends Exception { public function errorMessage() { //error message $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile() .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address'; return $errorMsg; } }  $email = "someone@example.com";  try   {   //check if   if(filter\_var($email, FILTER\_VALIDATE\_EMAIL) === FALSE)     {     //throw exception if email is not valid     throw new customException($email);     }   //check for "example" in mail address   if(strpos($email, "example") !== FALSE)     {     throw new Exception("$email is an example e-mail");     }   }  catch (customException $e)   {   echo $e->errorMessage();   }  catch(Exception $e)   {   echo $e->getMessage();   } ?> |

**Example explained:**

The code above tests two conditions and throws an exception if any of the conditions are not met:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block is executed and an exception is not thrown on the first condition
5. The second condition triggers an exception since the e-mail contains the string "example"
6. The "catch" block catches the exception and displays the correct error message

If there was no customException catch, only the base exception catch, the exception would be handled there

**Re-throwing Exceptions**

Sometimes, when an exception is thrown, you may wish to handle it differently than the standard way. It is possible to throw an exception a second time within a "catch" block.

A script should hide system errors from users. System errors may be important for the coder, but is of no interest to the user. To make things easier for the user you can re-throw the exception with a user friendly message:

|  |
| --- |
| <?php class customException extends Exception   {   public function errorMessage()     {     //error message     $errorMsg = $this->getMessage().' is not a valid E-Mail address.';     return $errorMsg;     }   }  $email = "someone@example.com";  try   {   try     {     //check for "example" in mail address     if(strpos($email, "example") !== FALSE)       {       //throw exception if email is not valid       throw new Exception($email);       }     }   catch(Exception $e)     {     //re-throw exception     throw new customException($email);     }   }  catch (customException $e)   {   //display custom message   echo $e->errorMessage();   } ?> |

**Example explained:**

The code above tests if the email-address contains the string "example" in it, if it does, the exception is re-thrown:

1. The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
2. The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
3. The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
4. The "try" block contains another "try" block to make it possible to re-throw the exception
5. The exception is triggered since the e-mail contains the string "example"
6. The "catch" block catches the exception and re-throws a "customException"
7. The "customException" is caught and displays an error message

If the exception is not caught in its current "try" block, it will search for a catch block on "higher levels".

**Set a Top Level Exception Handler**

The set\_exception\_handler() function sets a user-defined function to handle all uncaught exceptions.

|  |
| --- |
| <?php function myException($exception) { echo "<b>Exception:</b> " , $exception->getMessage(); }  set\_exception\_handler('myException');  throw new Exception('Uncaught Exception occurred'); ?> |

The output of the code above should be something like this:

|  |
| --- |
| **Exception:** Uncaught Exception occurred |

In the code above there was no "catch" block. Instead, the top level exception handler triggered. This function should be used to catch uncaught exceptions.

**Rules for exceptions**

* Code may be surrounded in a try block, to help catch potential exceptions
* Each try block or "throw" must have at least one corresponding catch block
* Multiple catch blocks can be used to catch different classes of exceptions
* Exceptions can be thrown (or re-thrown) in a catch block within a try block

A simple rule: If you throw something, you have to catch it.

PHP filters are used to validate and filter data coming from insecure sources, like user input.

**What is a PHP Filter?**

A PHP filter is used to validate and filter data coming from insecure sources.

To test, validate and filter user input or custom data is an important part of any web application.

The PHP filter extension is designed to make data filtering easier and quicker.

**Why use a Filter?**

Almost all web applications depend on external input. Usually this comes from a user or another application (like a web service). By using filters you can be sure your application gets the correct input type.

**You should always filter all external data!**

Input filtering is one of the most important application security issues.

What is external data?

* Input data from a form
* Cookies
* Web services data
* Server variables
* Database query results

**Functions and Filters**

To filter a variable, use one of the following filter functions:

* filter\_var() - Filters a single variable with a specified filter
* filter\_var\_array() - Filter several variables with the same or different filters
* filter\_input - Get one input variable and filter it
* filter\_input\_array - Get several input variables and filter them with the same or different filters

In the example below, we validate an integer using the filter\_var() function:

|  |
| --- |
| <?php $int = 123;  if(!filter\_var($int, FILTER\_VALIDATE\_INT))   {   echo("Integer is not valid");   } else   {   echo("Integer is valid");   } ?> |

The code above uses the "FILTER\_VALIDATE\_INT"  filter to filter the variable. Since the integer is valid, the output of the code above will be: "Integer is valid".

If we try with a variable that is not an integer (like "123abc"), the output will be: "Integer is not valid".

For a complete list of functions and filters, visit our [PHP Filter Reference.](http://www.w3schools.com/php/php_ref_filter.asp)

**Validating and Sanitizing**

There are two kinds of filters:

Validating filters:

* Are used to validate user input
* Strict format rules (like URL or E-Mail validating)
* Returns the expected type on success or FALSE on failure

Sanitizing filters:

* Are used to allow or disallow specified characters in a string
* No data format rules
* Always return the string

**Options and Flags**

Options and flags are used to add additional filtering options to the specified filters.

Different filters have different options and flags.

In the example below, we validate an integer using the filter\_var() and the "min\_range" and "max\_range" options:

|  |
| --- |
| <?php $var=300;  $int\_options = array( "options"=>array   (   "min\_range"=>0,   "max\_range"=>256   ) );  if(!filter\_var($var, FILTER\_VALIDATE\_INT, $int\_options))   {   echo("Integer is not valid");   } else   {   echo("Integer is valid");   } ?> |

Like the code above, options must be put in an associative array with the name "options". If a flag is used it does not need to be in an array.

Since the integer is "300" it is not in the specified range, and the output of the code above will be: "Integer is not valid".

For a complete list of functions and filters, visit our [PHP Filter Reference.](http://www.w3schools.com/php/php_ref_filter.asp) Check each filter to see what options and flags are available.

**Validate Input**

Let's try validating input from a form.

The first thing we need to do is to confirm that the input data we are looking for exists.

Then we filter the input data using the filter\_input() function.

In the example below, the input variable "email" is sent to the PHP page:

|  |
| --- |
| <?php if(!filter\_has\_var(INPUT\_GET, "email"))   {   echo("Input type does not exist");   } else   {   if (!filter\_input(INPUT\_GET, "email", FILTER\_VALIDATE\_EMAIL))     {     echo "E-Mail is not valid";     }   else     {     echo "E-Mail is valid";     }   } ?> |

**Example Explained**

The example above has an input (email) sent to it using the "GET" method:

1. Check if an "email" input variable of the "GET" type exist
2. If the input variable exists, check if it is a valid e-mail address

**Sanitize Input**

Let's try cleaning up an URL sent from a form.

First we confirm that the input data we are looking for exists.

Then we sanitize the input data using the filter\_input() function.

In the example below, the input variable "url" is sent to the PHP page:

|  |
| --- |
| <?php if(!filter\_has\_var(INPUT\_POST, "url"))   {   echo("Input type does not exist");   } else   {   $url = filter\_input(INPUT\_POST,    "url", FILTER\_SANITIZE\_URL);   } ?> |

**Example Explained**

The example above has an input (url) sent to it using the "POST" method:

1. Check if the "url" input of the "POST" type exists
2. If the input variable exists, sanitize (take away invalid characters) and store it in the $url variable

If the input variable is a string like this "http://www.W3ååSchøøools.com/", the $url variable after the sanitizing will look like this:

|  |
| --- |
| http://www.W3Schools.com/ |

**Filter Multiple Inputs**

A form almost always consist of more than one input field. To avoid calling the filter\_var or filter\_input functions over and over, we can use the filter\_var\_array or the filter\_input\_array functions.

In this example we use the filter\_input\_array() function to filter three GET variables. The received GET variables is a name, an age and an e-mail address:

|  |
| --- |
| <?php $filters = array   (   "name" => array     (     "filter"=>FILTER\_SANITIZE\_STRING     ),   "age" => array     (     "filter"=>FILTER\_VALIDATE\_INT,     "options"=>array       (       "min\_range"=>1,       "max\_range"=>120       )     ),   "email"=> FILTER\_VALIDATE\_EMAIL,   );  $result = filter\_input\_array(INPUT\_GET, $filters);  if (!$result["age"])   {   echo("Age must be a number between 1 and 120.<br />");   } elseif(!$result["email"])   {   echo("E-Mail is not valid.<br />");   } else   {   echo("User input is valid");   } ?> |

**Example Explained**

The example above has three inputs (name, age and email) sent to it using the "GET" method:

1. Set an array containing the name of input variables and the filters used on the specified input variables
2. Call the filter\_input\_array() function with the GET input variables and the array we just set
3. Check the "age" and "email" variables in the $result variable for invalid inputs. (If any of the input variables are invalid, that input variable will be FALSE after the filter\_input\_array() function)

The second parameter of the filter\_input\_array() function can be an array or a single filter ID.

If the parameter is a single filter ID all values in the input array are filtered by the specified filter.

If the parameter is an array it must follow these rules:

* Must be an associative array containing an input variable as an array key (like the "age" input variable)
* The array value must be a filter ID or an array specifying the filter, flags and options

**Using Filter Callback**

It is possible to call a user defined function and use it as a filter using the FILTER\_CALLBACK filter. This way, we have full control of the data filtering.

You can create your own user defined function or use an existing PHP function

The function you wish to use to filter is specified the same way as an option is specified. In an associative array with the name "options"

In the example below, we use a user created function to convert all  "\_" to whitespaces:

|  |
| --- |
| <?php function convertSpace($string) { return str\_replace("\_", " ", $string); }  $string = "Peter\_is\_a\_great\_guy!";  echo filter\_var($string, FILTER\_CALLBACK, array("options"=>"convertSpace")); ?> |

The result from the code above should look like this:

|  |
| --- |
| Peter is a great guy! |

**Example Explained**

The example above converts all "\_" to whitespaces:

1. Create a function to replace "\_" to whitespaces
2. Call the filter\_var() function with the FILTER\_CALLBACK filter and an array containing our function
3. MySQL is the most popular open-source database system.
4. **What is MySQL?**
5. MySQL is a database.
6. The data in MySQL is stored in database objects called tables.
7. A table is a collection of related data entries and it consists of columns and rows.
8. Databases are useful when storing information categorically. A company may have a database with the following tables: "Employees", "Products", "Customers" and "Orders".
9. **Database Tables**
10. A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.
11. Below is an example of a table called "Persons":

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Hansen | Ola | Timoteivn 10 | Sandnes |
| Svendson | Tove | Borgvn 23 | Sandnes |
| Pettersen | Kari | Storgt 20 | Stavanger |

1. The table above contains three records (one for each person) and four columns (LastName, FirstName, Address, and City).
2. **Queries**
3. A query is a question or a request.
4. With MySQL, we can query a database for specific information and have a recordset returned.
5. Look at the following query:

|  |
| --- |
| SELECT LastName FROM Persons |

1. The query above selects all the data in the "LastName" column from the "Persons" table, and will return a recordset like this:

|  |
| --- |
| **LastName** |
| Hansen |
| Svendson |
| Pettersen |

1. **Download MySQL Database**
2. If you don't have a PHP server with a MySQL Database, you can download MySQL for free here: <http://www.mysql.com/downloads/index.html>
3. **Facts About MySQL Database**
4. One great thing about MySQL is that it can be scaled down to support embedded database applications. Perhaps it is because of this reputation that many people believe that MySQL can only handle small to medium-sized systems.
5. The truth is that MySQL is the de-facto standard database for web sites that support huge volumes of both data and end users (like Friendster, Yahoo, Google).
6. Look at <http://www.mysql.com/customers/> for an overview of companies using MySQL.
7. The free MySQL database is very often used with PHP.

## Create a Connection to a MySQL Database

1. Before you can access data in a database, you must create a connection to the database.
2. In PHP, this is done with the mysql\_connect() function.

### Syntax

|  |
| --- |
| mysql\_connect(servername,username,password); |

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| servername | Optional. Specifies the server to connect to. Default value is "localhost:3306" |
| username | Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process |
| password | Optional. Specifies the password to log in with. Default is "" |

1. **Note:** There are more available parameters, but the ones listed above are the most important. Visit our full [PHP MySQL Reference](http://www.w3schools.com/php/php_ref_mysql.asp) for more details.

### Example

1. In the following example we store the connection in a variable ($con) for later use in the script. The "die" part will be executed if the connection fails:

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  // some code ?> |

## Closing a Connection

1. The connection will be closed automatically when the script ends. To close the connection before, use the mysql\_close() function:

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  // some code  mysql\_close($con); ?> |

A database holds one or multiple tables.

## Create a Database

The CREATE DATABASE statement is used to create a database in MySQL.

### Syntax

|  |
| --- |
| CREATE DATABASE database\_name |

To learn more about SQL, please visit our [SQL tutorial](http://www.w3schools.com/sql/default.asp).

To get PHP to execute the statement above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

### Example

The following example creates a database called "my\_db":

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  if (mysql\_query("CREATE DATABASE my\_db",$con))   {   echo "Database created";   } else   {   echo "Error creating database: " . mysql\_error();   }  mysql\_close($con); ?> |

## Create a Table

The CREATE TABLE statement is used to create a table in MySQL.

### Syntax

|  |
| --- |
| CREATE TABLE table\_name( column\_name1 data\_type,column\_name2 data\_type,column\_name3 data\_type, *....* ) |

To learn more about SQL, please visit our [SQL tutorial](http://www.w3schools.com/sql/default.asp).

We must add the CREATE TABLE statement to the mysql\_query() function to execute the command.

### Example

The following example creates a table named "Persons", with three columns. The column names will be "FirstName", "LastName" and "Age":

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  // Create database if (mysql\_query("CREATE DATABASE my\_db",$con))   {   echo "Database created";   } else   {   echo "Error creating database: " . mysql\_error();   }  // Create table mysql\_select\_db("my\_db", $con); $sql = "CREATE TABLE Persons ( FirstName varchar(15), LastName varchar(15), Age int )";  // Execute query mysql\_query($sql,$con);  mysql\_close($con); ?> |

**Important:** A database must be selected before a table can be created. The database is selected with the mysql\_select\_db() function.

**Note:** When you create a database field of type varchar, you must specify the maximum length of the field, e.g. varchar(15).

The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MySQL, go to our complete [Data Types reference](http://www.w3schools.com/sql/sql_datatypes.asp).

## Primary Keys and Auto Increment Fields

Each table should have a primary key field.

A primary key is used to uniquely identify the rows in a table. Each primary key value must be unique within the table. Furthermore, the primary key field cannot be null because the database engine requires a value to locate the record.

The following example sets the personID field as the primary key field. The primary key field is often an ID number, and is often used with the AUTO\_INCREMENT setting. AUTO\_INCREMENT automatically increases the value of the field by 1 each time a new record is added. To ensure that the primary key field cannot be null, we must add the NOT NULL setting to the field.

### Example

|  |
| --- |
| $sql = "CREATE TABLE Persons  ( personID int NOT NULL AUTO\_INCREMENT,  PRIMARY KEY(personID), FirstName varchar(15), LastName varchar(15), Age int )";  mysql\_query($sql,$con); |

The INSERT INTO statement is used to insert new records in a table.

## Insert Data Into a Database Table

The INSERT INTO statement is used to add new records to a database table.

### Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

|  |
| --- |
| INSERT INTO table\_nameVALUES (value1, value2, value3,...) |

The second form specifies both the column names and the values to be inserted:

|  |
| --- |
| INSERT INTO table\_name (column1, column2, column3,...) VALUES (value1, value2, value3,...) |

To learn more about SQL, please visit our [SQL tutorial](http://www.w3schools.com/sql/default.asp).

To get PHP to execute the statements above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

### Example

In the previous chapter we created a table named "Persons", with three columns; "Firstname", "Lastname" and "Age". We will use the same table in this example. The following example adds two new records to the "Persons" table:

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  mysql\_select\_db("my\_db", $con);  mysql\_query("INSERT INTO Persons (FirstName, LastName, Age) VALUES ('Peter', 'Griffin', '35')");  mysql\_query("INSERT INTO Persons (FirstName, LastName, Age)  VALUES ('Glenn', 'Quagmire', '33')");  mysql\_close($con); ?> |

## Insert Data From a Form Into a Database

Now we will create an HTML form that can be used to add new records to the "Persons" table.

Here is the HTML form:

|  |
| --- |
| <html> <body>  <form action="insert.php" method="post"> Firstname: <input type="text" name="firstname" /> Lastname: <input type="text" name="lastname" /> Age: <input type="text" name="age" /> <input type="submit" /> </form>  </body> </html> |

When a user clicks the submit button in the HTML form in the example above, the form data is sent to "insert.php".

The "insert.php" file connects to a database, and retrieves the values from the form with the PHP $\_POST variables.

Then, the mysql\_query() function executes the INSERT INTO statement, and a new record will be added to the "Persons" table.

Here is the "insert.php" page:

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  mysql\_select\_db("my\_db", $con);  $sql="INSERT INTO Persons (FirstName, LastName, Age) VALUES ('$\_POST[firstname]','$\_POST[lastname]','$\_POST[age]')";  if (!mysql\_query($sql,$con))   {   die('Error: ' . mysql\_error());   } echo "1 record added";  mysql\_close($con) ?> |

The SELECT statement is used to select data from a database.

## Select Data From a Database Table

The SELECT statement is used to select data from a database.

### Syntax

|  |
| --- |
| SELECT column\_name(s) FROM table\_name |

To learn more about SQL, please visit our [SQL tutorial](http://www.w3schools.com/sql/default.asp).

To get PHP to execute the statement above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

### Example

The following example selects all the data stored in the "Persons" table (The \* character selects all the data in the table):

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  mysql\_select\_db("my\_db", $con);  $result = mysql\_query("SELECT \* FROM Persons");  while($row = mysql\_fetch\_array($result))   {   echo $row['FirstName'] . " " . $row['LastName'];   echo "<br />";   }  mysql\_close($con); ?> |

The example above stores the data returned by the mysql\_query() function in the $result variable.

Next, we use the mysql\_fetch\_array() function to return the first row from the recordset as an array. Each call to mysql\_fetch\_array() returns the next row in the recordset. The while loop loops through all the records in the recordset. To print the value of each row, we use the PHP $row variable ($row['FirstName'] and $row['LastName']).

The output of the code above will be:

|  |
| --- |
| Peter Griffin Glenn Quagmire |

## Display the Result in an HTML Table

The following example selects the same data as the example above, but will display the data in an HTML table:

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  mysql\_select\_db("my\_db", $con);  $result = mysql\_query("SELECT \* FROM Persons");  echo **"**<table border='1'> **<**tr> <th>Firstname</th> <th>Lastname</th> </tr>";  while($row = mysql\_fetch\_array($result))   {   echo "<tr>";   echo "<td>" . $row['FirstName'] . "</td>";   echo "<td>" . $row['LastName'] . "</td>";   echo "</tr>";   } echo "</table>";  mysql\_close($con); ?> |

The output of the code above will be:

|  |  |
| --- | --- |
| **Firstname** | **Lastname** |
| Glenn | Quagmire |
| Peter | Griffin |

The WHERE clause is used to filter records.

## The WHERE clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

### Syntax

|  |
| --- |
| SELECT column\_name(s) FROM table\_name WHERE column\_name operator value |

To learn more about SQL, please visit our [SQL tutorial](http://www.w3schools.com/sql/default.asp).

To get PHP to execute the statement above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

### Example

The following example selects all rows from the "Persons" table where "FirstName='Peter':

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  mysql\_select\_db("my\_db", $con);  $result = mysql\_query("SELECT \* FROM Persons WHERE FirstName='Peter'");  while($row = mysql\_fetch\_array($result))   {   echo $row['FirstName'] . " " . $row['LastName'];   echo "<br />";   } ?> |

The output of the code above will be:

|  |
| --- |
| Peter Griffin |

The ORDER BY keyword is used to sort the data in a recordset.

## The ORDER BY Keyword

The ORDER BY keyword is used to sort the data in a recordset.

The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

### Syntax

|  |
| --- |
| SELECT column\_name(s) FROM table\_name ORDER BY column\_name(s) ASC|DESC |

To learn more about SQL, please visit our [SQL tutorial](http://www.w3schools.com/sql/default.asp).

### Example

The following example selects all the data stored in the "Persons" table, and sorts the result by the "Age" column:

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  mysql\_select\_db("my\_db", $con);  $result = mysql\_query("SELECT \* FROM Persons ORDER BY age");  while($row = mysql\_fetch\_array($result))   {   echo $row['FirstName'];   echo " " . $row['LastName'];   echo " " . $row['Age'];   echo "<br />";   }  mysql\_close($con); ?> |

The output of the code above will be:

|  |
| --- |
| Glenn Quagmire 33 Peter Griffin 35 |

## Order by Two Columns

It is also possible to order by more than one column. When ordering by more than one column, the second column is only used if the values in the first column are equal:

|  |
| --- |
| SELECT column\_name(s) FROM table\_name ORDER BY column1, column2 |

The UPDATE statement is used to modify data in a table.

## Update Data In a Database

The UPDATE statement is used to update existing records in a table.

### Syntax

|  |
| --- |
| UPDATE table\_name SET column1=value, column2=value2,... WHERE some\_column=some\_value |

**Note:** Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

To learn more about SQL, please visit our [SQL tutorial](http://www.w3schools.com/sql/default.asp).

To get PHP to execute the statement above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

### Example

Earlier in the tutorial we created a table named "Persons". Here is how it looks:

|  |  |  |
| --- | --- | --- |
| **FirstName** | **LastName** | **Age** |
| Peter | Griffin | 35 |
| Glenn | Quagmire | 33 |

The following example updates some data in the "Persons" table:

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  mysql\_select\_db("my\_db", $con);  mysql\_query("UPDATE Persons SET Age = '36' WHERE FirstName = 'Peter' AND LastName = 'Griffin'");  mysql\_close($con); ?> |

After the update, the "Persons" table will look like this:

|  |  |  |
| --- | --- | --- |
| **FirstName** | **LastName** | **Age** |
| Peter | Griffin | 36 |
| Glenn | Quagmire | 33 |

The DELETE statement is used to delete records in a table.

## Delete Data In a Database

The DELETE FROM statement is used to delete records from a database table.

### Syntax

|  |
| --- |
| DELETE FROM table\_name WHERE some\_column = some\_value |

**Note:** Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

To learn more about SQL, please visit our [SQL tutorial](http://www.w3schools.com/sql/default.asp).

To get PHP to execute the statement above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

### Example

Look at the following "Persons" table:

|  |  |  |
| --- | --- | --- |
| **FirstName** | **LastName** | **Age** |
| Peter | Griffin | 35 |
| Glenn | Quagmire | 33 |

The following example deletes all the records in the "Persons" table where LastName='Griffin':

|  |
| --- |
| <?php $con = mysql\_connect("localhost","peter","abc123"); if (!$con)   {   die('Could not connect: ' . mysql\_error());   }  mysql\_select\_db("my\_db", $con);  mysql\_query("DELETE FROM Persons WHERE LastName='Griffin'");  mysql\_close($con); ?> |

After the deletion, the table will look like this:

|  |  |  |
| --- | --- | --- |
| **FirstName** | **LastName** | **Age** |
| Glenn | Quagmire | 33 |

ODBC is an Application Programming Interface (API) that allows you to connect to a data source (e.g. an MS Access database).

## Create an ODBC Connection

With an ODBC connection, you can connect to any database, on any computer in your network, as long as an ODBC connection is available.

Here is how to create an ODBC connection to a MS Access Database:

1. Open the **Administrative Tools** icon in your Control Panel.
2. Double-click on the **Data Sources (ODBC)** icon inside.
3. Choose the **System DSN** tab.
4. Click on **Add** in the System DSN tab.
5. Select the **Microsoft Access Driver**. Click **Finish.**
6. In the next screen, click **Select** to locate the database.
7. Give the database a **Data Source Name (DSN)**.
8. Click **OK**.

Note that this configuration has to be done on the computer where your web site is located. If you are running Internet Information Server (IIS) on your own computer, the instructions above will work, but if your web site is located on a remote server, you have to have physical access to that server, or ask your web host to to set up a DSN for you to use.

## Connecting to an ODBC

The odbc\_connect() function is used to connect to an ODBC data source. The function takes four parameters: the data source name, username, password, and an optional cursor type.

The odbc\_exec() function is used to execute an SQL statement.

### Example

The following example creates a connection to a DSN called northwind, with no username and no password. It then creates an SQL and executes it:

|  |
| --- |
| $conn=odbc\_connect('northwind','',''); $sql="SELECT \* FROM customers"; $rs=odbc\_exec($conn,$sql); |

## Retrieving Records

The odbc\_fetch\_row() function is used to return records from the result-set. This function returns true if it is able to return rows, otherwise false.

The function takes two parameters: the ODBC result identifier and an optional row number:

|  |
| --- |
| odbc\_fetch\_row($rs) |

## Retrieving Fields from a Record

The odbc\_result() function is used to read fields from a record. This function takes two parameters: the ODBC result identifier and a field number or name.

The code line below returns the value of the first field from the record:

|  |
| --- |
| $compname=odbc\_result($rs,1); |

The code line below returns the value of a field called "CompanyName":

|  |
| --- |
| $compname=odbc\_result($rs,"CompanyName"); |

## Closing an ODBC Connection

The odbc\_close() function is used to close an ODBC connection.

|  |
| --- |
| odbc\_close($conn); |

## An ODBC Example

The following example shows how to first create a database connection, then a result-set, and then display the data in an HTML table.

|  |
| --- |
| <html> <body>  <?php $conn=odbc\_connect('northwind','',''); if (!$conn)   {exit("Connection Failed: " . $conn);} $sql="SELECT \* FROM customers"; $rs=odbc\_exec($conn,$sql); if (!$rs)   {exit("Error in SQL");} echo "<table><tr>"; echo "<th>Companyname</th>"; echo "<th>Contactname</th></tr>"; while (odbc\_fetch\_row($rs))   {   $compname=odbc\_result($rs,"CompanyName");   $conname=odbc\_result($rs,"ContactName");   echo "<tr><td>$compname</td>";   echo "<td>$conname</td></tr>";   } odbc\_close($conn); echo "</table>"; ?>  </body> </html> |