|  |
| --- |
|  |
| **Workflow Softwares**  **2nd floor, Panchratna plaza, Sankeshwar**  **Mob: +91-7026123356, Office: +91-7026046613**  [**http://www.workflowsoftwares.in**](http://www.workflowsoftwares.in) |

PHP

**Hypertext Preprocessor**

A Guide to Learn PHP

V.1

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Contents** | **Page No** |
| 1. | INTRODUCTION | 3 |
| 2. | PHP Syntax | 4 |
| 3. | PHP Operators | 6 |
| 4. | PHP Switch Statement | 10 |
| 5. | PHP Arrays | 12 |
| 6. | PHP String Variables | 16 |
| 7. | PHP Looping-While Loops | 17 |
| 8. | PHP Looping-ForLoops | 19 |
| 9. | PHP Functions | 22 |
| 10. | PHP Forms and User Input | 26 |
| 11. | PHP$\_GETFunction | 27 |
| 12. | PHP$\_POSTFunction | 28 |
| 13. | PHP Include File | 31 |
| 14. | PHP File Upload | 35 |
| 15. | PHP Sessions | 39 |
| 16. | PHP MySQL Connection to Database | 42 |
| 17. | PHP MySQL Create Database and Tables | 43 |
| 18. | PHP MySQL InsertInto | 46 |
| 19. | PHP MySQL Select | 48 |
| 20. | PHP MySQL Update | 50 |
| 21. | PHP MySQL Delete | 52 |
| 22. | PHP Date/Time Functions | 54 |
| 23. | PHP String Functions | 60 |

**INTRODUCTION**

**PHP** is a scripting language originally designed for producing dynamic web pages. It has evolved to include a command line interface capability and can be used in standalone graphical applications. While PHP was originally created by Rasmus Lerdorf in 1995, the main implementation of PHP is now produced by The PHP Group and serves as the de facto standard for PHP as there is no formal specification.

PHP is free software released under the PHP License; however it incompatible with the GNU General Public License (GPL), due to restrictions on the usage of the term PHP. It is a widely-used general-purpose scripting language that is especially suited for web development and can be embedded into HTML. It generally runs on a web server, taking PHP code as its input and creating web pages as output. It can be deployed on most web servers and on almost every operating system and platform free of charge.PHP is installed on more than 20 million websites and 1 million web servers. PHP originally stood for Personal Home Page.

**PHP Syntax**

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 **?>**. And it 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 and /\* … \*/ to make a large comment block.

**<html>**

**<body>**

**<?php**

**//This is a comment**

**/\***

**This is**

**a comment**

**block**

**\*/**

**?>**

**</body>**

**</html>**

**PHP Operators**

Operators are used to operate on values.

**PHP Operators**

This section lists the different operator s 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%2 | 1  0 |
| ++ Increment | x=5 | x++ | 6 |
| -- Decrement | x=5 | x-- | 4 |

**Assignment Operators**

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>***

Note that there is no..else.. in this syntax. You tell the browser to execute some code 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>***

**PHP Switch Statement**

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 pr event 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>***

**PHP Arrays**

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 kinds 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");

1. 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 refer ring 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.";***

***?>***

**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.";

?>

**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***

***)***

***[Quagmir e] => Array***

***(***

***[0] => Glenn***

***)***

***[Brown] => Array***

***(***

***[0] => Cleveland***

***[1] => Loretta***

***[2] => Junior***

***)***

***)***

**Example 2**

Let’s try displaying a single value from the array above:

echo "I s " . $families[ 'Gr iffin'][2] . " a part of the Griffin family?";

**Output :**

Is Megan a part of the Griffin family?

**PHP String Variables**

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 Wor ld";***

***echo $txt;***

***?>***

The output of the code above will be: **Hello World**

Now, let’s 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 Wor ld!" ;***

***$txt2="What a nice day!";***

***echo $txt1 . " " . $txt2;***

***?>***

The output of the code above will be: Hello World! What a nice day!

**PHP Looping-While Loops**

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:

***<?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

**PHP Looping-ForLoops**

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 parameter s 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

**PHP Functions**

The real power of PHP comes from its functions. In PHP, there are more than 700 built- in functions.

**PHP Functions**

In this chapter we will show you how to create your own functions. To keep the browser from executing a script when the page loads, you can put your script 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 *function Name* ()

{

*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

**PHP Forms and User Input**

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 for m 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"]; ?> year s old.***

***</body></html>***

**Output could be something like this:** Welcome John!

You are 28 years old.

**PHP$\_GETFunction**

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 (max. 100 characters) .

**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.vertexcity.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 large variable values; the value cannot exceed 100 characters.

**PHP$\_POSTFunction**

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"]; ?> year s 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.***

**Advanced PHP**

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

**The PHP Date() Function**

The PHP date() function for mats a timestamp to a more readable date and time. A 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)

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 specif y 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 (Januar y 1 197000:00:00 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

**PHP Include File**

**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.

**PHP File Upload**

With PHP, it is possible to upload files to the ser ver.

**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 enc type attribute of the <form> tag specifies which content-type to use when submitting the form. "multipart/for m-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", "tmpname" 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")***

***|| ($\_FI LES["file"][ "type"] == "image/jpeg")***

***|| ($\_FI LES["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 disappear 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")***

***|| ($\_FI LES["file"][ "type"] == "image/jpeg")***

***|| ($\_FI LES["file"][ "type"] == "image/pjpeg"))***

***&& ($\_FILES["file"]["size"] < 20000) )***

***{***

***if ($\_FILES["file"]["error"] > 0)***

***{***

***echo "Return Code: " . $\_FILES["file"]["er ror"] . "<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"

**PHP Sessions**

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***

***//retr ieve 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 MySQL Connection to Database**

The free MySQL database is very often used with PHP.

**Create a Connection to a MySQL Database**

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

**Syntax**

mysql\_connect(servername,username,password);

**Parameter Description**

Server name 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 ""

**Example**

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**

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);***

***?>***

**PHP MySQL Create Database and Tables**

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 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\_err or();***

***}***

***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,*....*)

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\_err or();***

***}***

***// Create table mysql\_select\_db("my\_db", $con);***

***$sql = "CREATE TABLE Persons(FirstName varchar( 15),LastName varchar(15), Age int***

***)";***

***// Execute query mysql\_quer y($sql,$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).

**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. further more, 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\_quer y($sql,$con);***

**PHP MySQL InsertInto**

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\_name

VALUES (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 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\_quer y("INSERT INTO Persons ( FirstName, LastName, Age) VALUES ('Peter', 'Griffin', '35')");***

***mysql\_quer y("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=" fir stname" /> 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="I NSERT 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)***

***?>***

**PHP MySQL Select**

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 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($r ow = mysql\_fetch\_arr ay($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 Griff in

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($r ow = mysql\_fetch\_arr ay($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

**PHP MySQL Update**

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 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\_quer y("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

**PHP MySQL Delete**

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 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\_quer y("DELETE FROM Persons WHERE LastName='Gr iffin'");***

***mysql\_close($con);***

***?>***

After the deletion, the table will look like this:

**FirstName LastName Age**

Glenn Quagmire 33

**PHP Date/Time Functions**

**PHP Date / Time Introduction**

The date/time functions allow you to extract and format the date and time on the server.

**Note:** These functions depend on the locale settings of the server !

**Installation**

The date/time functions are part of the PHP core. There is no installation needed to use these functions.

**Runtime Configuration**

The behavior of the date/time functions is affected by settings in php.ini.Date/Time configuration options:

**Name Default Description Changeable**

date.default\_latitude "31.7667" Specifies the default latitude (available PHP\_INI\_ALL

since PHP 5). This option is used by

date\_sunrise() and date\_sunset()

date.default\_longitude "35.2333" Specifies the default longitude PHP\_INI\_ALL (available since PHP 5) . This option is

used by date\_sunrise() and date\_sunset( )

date.sunrise\_zenith "90.83" Specifies the default sunrise zenith PHP\_INI\_ALL (available since PHP 5). This option is

used by date\_sunrise() and

date\_sunset( )

date.sunset\_zenith "90.83" Specifies the default sunset zenith PHP\_INI\_ALL (available since PHP 5) . This option is

used by date\_sunrise() and date\_sunset( )

date.timezone "" Specifies the default timezone PHP\_INI\_ALL (available since PHP 5.1)

**PHP Date / Time Functions**

**PHP**: indicates the earliest version of PHP that supports the function.

**Function Description PHP**

checkdate() Validates a Gregorian date date\_default\_timezone\_get() Returns the default time zone

date\_default\_timezone\_set() Sets the default time zone

date\_sunr ise() Returns the time of sunrise for a given day / location

date\_sunset() Returns the time of sunset for a given day / location date() Formats a local time/date

getdate() Returns an array that contains date and time information for a Unix timestamp

gettimeofday() Returns an array that contains current time information

gmdate() Formats a GMT/UTC date/time

gmmktime() Returns the Unix timestamp for a GMT date

gmstr ftime() Formats a GMT/UTC time/date according to locale settings

idate() Formats a local time/date as integer

localtime() Returns an array that contains the time components of a Unix timestamp

microtime() Returns the microseconds for the current time

mktime( ) Returns the Unix timestamp for a date

strftime() Formats a local time/date according to locale settings

strptime() Parses a time/date generated with strftime()

strtotime( ) Parses an English textual date or time into a Unix timestamp

time() Returns the current time as a Unix timestamp

**PHP String Functions**

**PHP String Introduction**

The string functions allow you to manipulate strings.

**Installation**

The string functions are part of the PHP core. There is no installation needed to use these functions.

**PHP String Functions**

**PHP**: indicates the earliest version of PHP that supports the function.

**Function Description**

addcslashes() Returns a string with backslashes in front of the specified characters

addslashes() Returns a string with backslashes in front of predefined characters

bin2hex() Converts a string of ASCII characters to hexadecimal values

chop() Alias of rtrim()

chr() Returns a character from a specified ASCII value chunk\_split() Splits a string into a series of smaller parts convert\_cyr\_str ing() Converts a string from one Cyrillic character -set to another

convert\_uudecode() Decodes a uuencoded string convert\_uuencode() Encodes a string using the uuencode algorithm

count\_chars() Returns how many times an ASCII character occurs within a string and returns the information

crc32() Calculates a 32-bit CRC for a string crypt() One-way string encryption (hashing) echo() Outputs strings

explode() Breaks a string into an array

fprintf( ) Writes a formatted string to a specified output stream get\_html\_translation\_table() Returns the translation table used by

htmlspecialchars() and htmlentities()

hebrev() Converts Hebrew text to visual text

hebrevc() Converts Hebrew text to visual text and new lines ( \n) into <br />

html\_entity\_decode() Converts HTML entities to characters htmlentities() Converts characters to HTML entities htmlspecialchars\_decode( ) Converts some predefined HTML entities to characters

htmlspecialchars() Converts some predefined characters to HTML entities

implode() Returns a string from the elements of an array

join() Alias of implode()

levenshtein( ) Returns the Levenshtein distance between two strings localeconv() Returns locale numeric and monetary formatting information

ltrim() Strips whitespace from the left side of a string

md() Calculates the MD5 hash of a string

md5\_file() Calculates the MD5 hash of a file

metaphone() Calculates the metaphone key of a string money\_for mat() Returns a string formatted as a currency string

nllanginfo() Returns specific local information

nl2br() Inserts HTML line breaks in front of each newline in a string

number\_format() Formats a number with grouped thousands

ord() Returns the ASCII value of the first character of a string

parse\_str() Parses a query string into variables print() Outputs a string

printf() Outputs a formatted string quoted\_printable\_decode() Decodes a quoted-printable string quotemeta() Quotes meta characters

rtrim() Strips whitespace from the right side of a string

setlocale() Sets locale information

sha1() Calculates the SHA-1 hash of a string sha1\_file() Calculates the SHA-1 hash of a file similar\_text() Calculates the similarity between two strings soundex() Calculates the soundex key of a string sprintf() Writes a formatted string to a variable

sscanf() Parses input from a string according to a format

str\_ireplace() Replaces some characters in a string (case-insensitive) str\_pad() Pads a string to a new length

str\_repeat() Repeats a string a specified number of times 4str\_replace() Replaces some characters in a string (case-sensitive) str\_rot13() Per forms the ROT13 encoding on a string str\_shuffle() Randomly shuffles all characters in a string

str\_split() Splits a string into an array

str\_word\_count() Count the number of words in a string strcasecmp() Compares two strings (case-insensitive)

strchr() Finds the first occurrence of a string inside another string (alias of strstr())

strcmp() Compares two strings (case-sensitive) strcoll() Locale based string comparison

strcspn() Returns the number of characters found in a string before any par t of some specified characters are found

strip\_tags() Strips HTML and PHP tags from a string stripcslashes( ) Unquotes a string quoted with addcslashes() stripslashes() Unquotes a string quoted with addslashes()

stripos() Returns the position of the first occurrence of a string inside another string (case-insensitive)

stristr() Finds the first occurrence of a string inside another string (case- insensitive)

strlen() Returns the length of a string

strnatcasecmp() Compares two strings using a "natural order" algorithm (case-insensitive)

strnatcmp() Compares two strings using a "natural order" algorithm (case-sensitive)

strncasecmp() String comparison of the first n character s (case- insensitive)

strncmp() String comparison of the first n characters (case- sensitive)

strpbrk() Searches a string for any of a set of characters

strpos() Returns the position of the first occurrence of string inside another string (case-sensitive)

strrchr() Finds the last occurrence of a string inside another string

strrev() Reverses a string

strripos() Finds the position of the last occurrence of a string inside another string (case-insensitive)

strrpos() Finds the position of the last occurrence of a string inside another string (case-sensitive)

strspn() Returns the number of characters found in a string that contains only characters from a specified char list

strstr() Finds the first occurrence of a string inside another string (case-sensitive)

strtok() Splits a string into smaller strings

strtolower() Converts a string to lowercase letters

strtoupper() Converts a string to uppercase letters

strtr( ) Translates certain characters in a string

substr() Returns a part of a string

substr\_compar e() Compares two strings from a specified start position ( binary safe and optionally case-sensitive)

substr\_count() Counts the number of times a substring occurs in a string

substr\_replace() Replaces a part of a string with another string

trim() Strips whitespace from both sides of a string

ucfir st() Converts the first character of a string to uppercase

ucwords() Converts the first character of each word in a string to uppercase

vfprintf( ) Writes a formatted string to a specified output stream

vprintf() Outputs a formatted string

vspr intf() Writes a formatted string to a variable

wordwrap() Wraps a string to a given number of characters