PHP is a powerful tool for making dynamic and interactive Web pages.

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"

## 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

The PHP script is executed on the server, and the plain HTML result is sent back to the browser.

The PHP instructions are placed inside special HTML tags. All PHP instructions have to be placed inside such tags, although you can place large blocks of code inside the same set. You do not have to use a new tag for each line of code. The inclusion of PHP within HTML files is called *escaping* in the PHP documentation. Four different methods are provided for escaping from HTML:

1. <? echo("<hl>Hello</hl>"); ?>
2. <?php echo("<hl>Hello</hl>"); ?>
3. <script language="php">

echo("<hl>Hello</hl>");

</script >

1. <% echo("<hl>Hello<:/hl>") ; %>

## Basic PHP Syntax

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

<?php  
?>

A PHP file must have a .php extension.

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

Below, we have an example of a simple PHP script that sends the text "Hello World" back 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".

## Comments in PHP

In PHP, we use **//** to make a one-line comment or **/\*** and **\*/** to make a comment block:

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

## PHP Variables

As with algebra, PHP variables are used to hold values or expressions.

A variable can have a short name, like x, or a more descriptive name, like carName.

Rules for PHP variable names:

* Variables in PHP starts with a $ sign, followed by the name of the variable
* The variable name must begin with a letter or the underscore character
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* A variable name should not contain spaces
* Variable names are case sensitive (y and Y are two different variables)

## Creating (Declaring) PHP Variables

PHP has no command for declaring a variable.

A variable is created the moment you first assign a value to it:

$myCar="Volvo";

After the execution of the statement above, the variable **myCar** will hold the value **Volvo**.

**Tip:** If you want to create a variable without assigning it a value, then you assign it the value of*null*.

Let's create a variable containing a string, and a variable containing a number:

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

**Note:** When you assign a text value to a variable, put quotes around the value.

## 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, notice that we did 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.

# PHP Operators

## Arithmetic Operators

The table below lists the arithmetic operators in PHP:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** | **Result** |
| x + y | Addition | Sum of x and y | 2 + 2 | 4 |
| x - y | Subtraction | Difference of x and y | 5 - 2 | 3 |
| x \* y | Multiplication | Product of x and y | 5 \* 2 | 10 |
| x / y | Division | Quotient of x and y | 15 / 5 | 3 |
| x % y | Modulus | Remainder of x divided by y | 5 % 2 10 % 8 10 % 2 | 1 2 0 |
| - x | Negation | Opposite of x | - 2 |  |
| a . b | Concatenation | Concatenate two strings | "Hi" . "Ha" | HiHa |

## Assignment Operators

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the expression on the right. That is, the value of "$x = 5" is 5.

|  |  |  |
| --- | --- | --- |
| **Assignment** | **Same as...** | **Description** |
| x = y | x = y | The left operand gets set to the value of the expression on the right |
| x += y | x = x + y | Addition |
| x -= y | x = x - y | Subtraction |
| x \*= y | x = x \* y | Multiplication |
| x /= y | x = x / y | Division |
| x %= y | x = x % y | Modulus |
| a .= b | a = a . b | Concatenate two strings |

## Incrementing/Decrementing Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| ++ x | Pre-increment | Increments x by one, then returns x |
| x ++ | Post-increment | Returns x, then increments x by one |
| -- x | Pre-decrement | Decrements x by one, then returns x |
| x -- | Post-decrement | Returns x, then decrements x by one |

## Comparison Operators

Comparison operators allows you to compare two values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** |
| x == y | Equal | True if x is equal to y | 5==8 returns false |
| x === y | Identical | True if x is equal to y, and they are of same type | 5==="5" returns false |
| x != y | Not equal | True if x is not equal to y | 5!=8 returns true |
| x <> y | Not equal | True if x is not equal to y | 5<>8 returns true |
| x !== y | Not identical | True if x is not equal to y, or they are not of same type | 5!=="5" returns true |
| x > y | Greater than | True if x is greater than y | 5>8 returns false |
| x < y | Less than | True if x is less than y | 5<8 returns true |
| x >= y | Greater than or equal to | True if x is greater than or equal to y | 5>=8 returns false |
| x <= y | Less than or equal to | True if x is less than or equal to y | 5<=8 returns true |

## Logical Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** |
| x and y | And | True if both x and y are true | x=6 y=3  (x < 10 and y > 1) returns true |
| x or y | Or | True if either or both x and y are true | x=6 y=3  (x==6 or y==5) returns true |
| x xor y | Xor | True if either x or y is true, but not both | x=6 y=3  (x==6 xor y==3) returns false |
| x && y | And | True if both x and y are true | x=6 y=3 (x < 10 && y > 1) returns true |
| x || y | Or | True if either or both x and y are true | x=6 y=3 (x==5 || y==5) returns false |
| ! x | Not | True if x is not true | x=6 y=3 !(x==y) returns true |

## Array Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| x + y | Union | Union of x and y |
| x == y | Equality | True if x and y have the same key/value pairs |
| x === y | Identity | True if x and y have the same key/value pairs in the same order and of the same types |
| x != y | Inequality | True if x is not equal to y |
| x <> y | Inequality | True if x is not equal to y |
| x !== y | Non-identity | True if x is not identical to y |

PHP data types

***Numbers*** PHP supports two different numerical types. Most often you'll use ***integers.*** The range of integers in PHP is defined by the operating system and generally runs from approximately -2,000,000,000 up to 2,000,000,000. In computing terms that is ±232. PHP integers can be declared in base 10, base 8 (octel) or base 16 (hexadecimal):

$positiveInt = 14;

$negativeInt = -78;

$octelInt = 0421;

$hexadecinialInt = Oxic;

The second numerical data type in PHP is the **floating-point number**. Floating point is a computer representation of positive and negative decimal numbers. They can be expressed as simple decimals or as exponential values. In PHP, floating point values are system de­pendent in the same way that integers are; however, the range is far larger. Floating points are 64 bit numbers with a range of approximately ±1.8e38. That's more than big enough for most Web scripts you'll ever write. Declaring floating point values is straightforward. The second example here uses the letter e to indicate that it's holding an exponential value:

$floater = 23.567;

$bigger = 4.6e7 ;

***Strings*** *Most* of the data that gets manipulated on the Web is text. In PHP, text is stored in strings. These must be surrounded with quotes **so** that the interpreter doesn't mistake them for commands. PHP lets you use either single or double quotes around a string. If you use double quotes, any PHP variables inside the string are replaced by their value. Here are a few strings:

$str = "A Simple String" ;

$str2 = 'Another String' ;

$str3 = "This is $str2";

$str4 = $str;

Like most programming languages, certain character sequences carry special meaning in PHP. For instance you need some way of telling the system to insert a newline inside the current piece of text. Such control characters must be replaced with **escape sequences** in your scripts. When the interpreter finds an escape sequence, those characters are replaced with the appropriate control code.

\n Insert a newline character.

\r Carriage return.

\t Horizontal tab.

\\ Backslash.

\$ Dollar.

\” Double quote.

Joining strings together is done by the process of ***concatenation****.* PHP uses the dot opera­tor to concatenate strings.

$strl = "Have A Nice";

$str2 = "Day\n" ;

$str3 = $strl . " ";

$str3 .= $str2;

Line three appends a string consisting of a single space onto the end of $strl and stores the result in $str3. The operation uses $strl but does not alter its value. In line four the value in $str2 is appended onto the value in $str3 and the result stored in $str3. PHP has a lot of operations which store their result in one of the original values, although most of these actually work with numbers not string

**Type Casting**

An example of **PHP's automatic type conversion** is the addition operator '+'. If either operand is a [float](http://www.php.net/manual/en/language.types.float.php), then both operands are evaluated as [float](http://www.php.net/manual/en/language.types.float.php)s, and the result will be a [float](http://www.php.net/manual/en/language.types.float.php). Otherwise, the operands will be interpreted as [integer](http://www.php.net/manual/en/language.types.integer.php)s, and the result will also be an [integer](http://www.php.net/manual/en/language.types.integer.php). Note that this does ***not*** change the types of the operands themselves; the only change is in how the operands are evaluated and what the type of the expression itself is.

<?php  
$foo = "0";  // $foo is string (ASCII 48)  
$foo += 2;   // $foo is now an integer (2)  
$foo = $foo + 1.3;  // $foo is now a float (3.3)

?>

Type casting in PHP works much as it does in C: the name of the desired type is written in parentheses before the variable which is to be cast.

The casts allowed are:

* (int), (integer) - cast to [integer](http://www.php.net/manual/en/language.types.integer.php)
* (bool), (boolean) - cast to [boolean](http://www.php.net/manual/en/language.types.boolean.php)
* (float), (double), (real) - cast to [float](http://www.php.net/manual/en/language.types.float.php)
* (string) - cast to [string](http://www.php.net/manual/en/language.types.string.php)
* (array) - cast to [array](http://www.php.net/manual/en/language.types.array.php)
* (object) - cast to [object](http://www.php.net/manual/en/language.types.object.php)
* (unset) - cast to [NULL](http://www.php.net/manual/en/language.types.null.php) (PHP 5)

Eg:

<?php  
$a = 10;   // $a is an integer  
$b = 10.5;   // $a is float

$res=$b+ (float)$a

Echo $res  
?>

PHP program control

## Conditional Statements

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

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>

## 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  
$x=1;  
switch ($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 Loops

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

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.

## 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 iteration)

**Note:**The *init* and *increment* 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

# PHP Arrays

An array stores multiple values in one single variable.

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 (basic array)

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?

## PHP Array Functions

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

|  |  |
| --- | --- |
| **Function** | **Description** |
| [array()](http://www.w3schools.com/php/func_array.asp) | Creates an array |
| [array\_intersect()](http://www.w3schools.com/php/func_array_intersect.asp) | Compares array values, and returns the matches |
| [array\_keys()](http://www.w3schools.com/php/func_array_keys.asp) | Returns all the keys of an array |
| [array\_map()](http://www.w3schools.com/php/func_array_map.asp) | Sends each value of an array to a user-made function, which returns new values |
| [array\_merge()](http://www.w3schools.com/php/func_array_merge.asp) | Merges one or more arrays into one array |
| [array\_merge\_recursive()](http://www.w3schools.com/php/func_array_merge_recursive.asp) | Merges one or more arrays into one array |
| [array\_multisort()](http://www.w3schools.com/php/func_array_multisort.asp) | Sorts multiple or multi-dimensional arrays |
| [array\_pad()](http://www.w3schools.com/php/func_array_pad.asp) | Inserts a specified number of items, with a specified value, to an array |
| [array\_pop()](http://www.w3schools.com/php/func_array_pop.asp) | Deletes the last element of an array |
| [array\_product()](http://www.w3schools.com/php/func_array_product.asp) | Calculates the product of the values in an array |
| [array\_push()](http://www.w3schools.com/php/func_array_push.asp) | Inserts one or more elements to the end of an array |
| [array\_rand()](http://www.w3schools.com/php/func_array_rand.asp) | Returns one or more random keys from an array |
| [array\_reduce()](http://www.w3schools.com/php/func_array_reduce.asp) | Returns an array as a string, using a user-defined function |
| [array\_reverse()](http://www.w3schools.com/php/func_array_reverse.asp) | Returns an array in the reverse order |
| [array\_search()](http://www.w3schools.com/php/func_array_search.asp) | Searches an array for a given value and returns the key |
| [array\_shift()](http://www.w3schools.com/php/func_array_shift.asp) | Removes the first element from an array, and returns the value of the removed element |
| [array\_slice()](http://www.w3schools.com/php/func_array_slice.asp) | Returns selected parts of an array |
| [array\_splice()](http://www.w3schools.com/php/func_array_splice.asp) | Removes and replaces specified elements of an array |
| [array\_sum()](http://www.w3schools.com/php/func_array_sum.asp) | Returns the sum of the values in an array |
| [array\_unshift()](http://www.w3schools.com/php/func_array_unshift.asp) | Adds one or more elements to the beginning of an array |
| [array\_values()](http://www.w3schools.com/php/func_array_values.asp) | Returns all the values of an array |
| [array\_walk()](http://www.w3schools.com/php/func_array_walk.asp) | Applies a user function to every member of an array |
| [array\_walk\_recursive()](http://www.w3schools.com/php/func_array_walk_recursive.asp) | Applies a user function recursively to every member of an array |
| [arsort()](http://www.w3schools.com/php/func_array_arsort.asp) | Sorts an array in reverse order and maintain index association |
| [asort()](http://www.w3schools.com/php/func_array_asort.asp) | Sorts an array and maintain index association |
| [compact()](http://www.w3schools.com/php/func_array_compact.asp) | Create array containing variables and their values |
| [count()](http://www.w3schools.com/php/func_array_count.asp) | Counts elements in an array, or properties in an object |
| [current()](http://www.w3schools.com/php/func_array_current.asp) | Returns the current element in an array |
| [each()](http://www.w3schools.com/php/func_array_each.asp) | Returns the current key and value pair from an array |
| [end()](http://www.w3schools.com/php/func_array_end.asp) | Sets the internal pointer of an array to its last element |
| [extract()](http://www.w3schools.com/php/func_array_extract.asp) | Imports variables into the current symbol table from an array |
| [in\_array()](http://www.w3schools.com/php/func_array_in_array.asp) | Checks if a specified value exists in an array |
| [key()](http://www.w3schools.com/php/func_array_key.asp) | Fetches a key from an array |
| [krsort()](http://www.w3schools.com/php/func_array_krsort.asp) | Sorts an array by key in reverse order |
| [ksort()](http://www.w3schools.com/php/func_array_ksort.asp) | Sorts an array by key |
| [list()](http://www.w3schools.com/php/func_array_list.asp) | Assigns variables as if they were an array |
| [natcasesort()](http://www.w3schools.com/php/func_array_natcasesort.asp) | Sorts an array using a case insensitive "natural order" algorithm |
| [natsort()](http://www.w3schools.com/php/func_array_natsort.asp) | Sorts an array using a "natural order" algorithm |
| [next()](http://www.w3schools.com/php/func_array_next.asp) | Advance the internal array pointer of an array |
| [pos()](http://www.w3schools.com/php/func_array_pos.asp) | Alias of current() |
| [prev()](http://www.w3schools.com/php/func_array_prev.asp) | Rewinds the internal array pointer |
| [range()](http://www.w3schools.com/php/func_array_range.asp) | Creates an array containing a range of elements |
| [reset()](http://www.w3schools.com/php/func_array_reset.asp) | Sets the internal pointer of an array to its first element |
| [rsort()](http://www.w3schools.com/php/func_array_rsort.asp) | Sorts an array in reverse order |
| [shuffle()](http://www.w3schools.com/php/func_array_shuffle.asp) | Shuffles an array |
| [sizeof()](http://www.w3schools.com/php/func_array_sizeof.asp) | Alias of count() |
| [sort()](http://www.w3schools.com/php/func_array_sort.asp) | Sorts an array |
| [uasort()](http://www.w3schools.com/php/func_array_uasort.asp) | Sorts an array with a user-defined function and maintain index association |
| [uksort()](http://www.w3schools.com/php/func_array_uksort.asp) | Sorts an array by keys using a user-defined function |
| [usort()](http://www.w3schools.com/php/func_array_usort.asp) | Sorts an array by values using a user-defined function |

# PHP Functions

The real power of PHP comes from its functions.

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

## 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

**Php buit in functions**

**Regular Expression**

A regular expression, which may also be called a regex or regexp, is a string which denotes a search pattern. The search pattern is applied to text strings inside your script. Regular expressions have been used for many, many years in computing but are still, in some ways, considered a bit of a black art. Having created the pattern, you'll need to use some PHP-specific functions if you want to apply it.

**preg\_match(pattern, string [, matches])**

Hunts for strings which match the supplied pattern in the string subject. This match will stop once it has found a target. Optionally an array can be supplied as the third parameter. This will be used to store matches. The first item in the array, item 0, contains the text which matches.

**preg\_match\_all(pattern, string)**

Works like preg\_match () but matches *all* occurrences of the pattern in the string.

**preg\_replace(pattern, replacement, string)**

If the regex pattern is found in the string, it is replaced by the string supplied as the second parameter.

**preg\_split(pattern, string)**

Splits the string at all points which match the pattern. Returns an array of strings con­taining all of the substrings it creates.

Here's a match which stores its results:

<?php

$pattern = "/( [a-z]+\?)/";

$string = "Can you find the target?";

if(preg\_match($pattern, $string, $matches)) {

echo "Found $matches[0]" ;

} else {

echo "Didn't find it";

} ?>

The pattern matches repeated lower-case characters followed by a question mark. Clearly this will only match target? in the string. The pattern is placed inside paren­theses and gets stored in the first element of the array $matches. This example shows a simple replacement:

<?php

$pattern *=* "/[a-z]+\?/";

$string = "Can you find the target?";

if($new = preg\_replace($pattern, "replacement", ?string)) {

echo "$new";

*} else {*

echo "Didn't find it";

}?>

**Math Functions**

**abs(number)**

Returns the absolute value of a number. If the argument is a floating point number,

the return value will also be a float. Otherwise it will be an integer.

**acos(float)** Returns the arc cosine in radians.

**asin(float)** Returns the arc sine in radians.

**atan(float)** Returns the arc tangent in radians.

**ceil (float)**

Returns the integer which is directly higher than the argument, ceil (34 .56) would return 35.

**cos(float)** Returns the cosine in radians.

**exp(number)** Returns e raised to the power of the argument.

**floor(float)** Returns the integer directly lower than the argument

**log(float)** Returns the natural logarithm of the argument.

**Logl0(float)** Returns the base 10 logarithm of the argument.

**max(argl, arg2[, argn])** Returns the largest of its arguments.

**min(argl, arg2[, argn])** Returns the smallest of its arguments.

**pi()** Return the value of tt.

**pow(argl, arg2)** Returns argi raised to the power of arg2.

**rand([min[, max]])**

Returns a pseudo-random number. If max or min are set, they place limits upon the range in which the number will be generated. Before calling rand () you *must* call srand () to seed the random number generator.

**round(float[, precision])**

Rounds the floating point number. If no precision is specified, the number is rounded to the nearest whole number. For instance round (3.6) will return 4 . 0 and round (3.2) will return 3.0. If the precision is provided, it sets the number of decimal places in the result.

**sin(float)** Returns the sine in radians.

**sqrt(number)** Returns the square root of its argument.

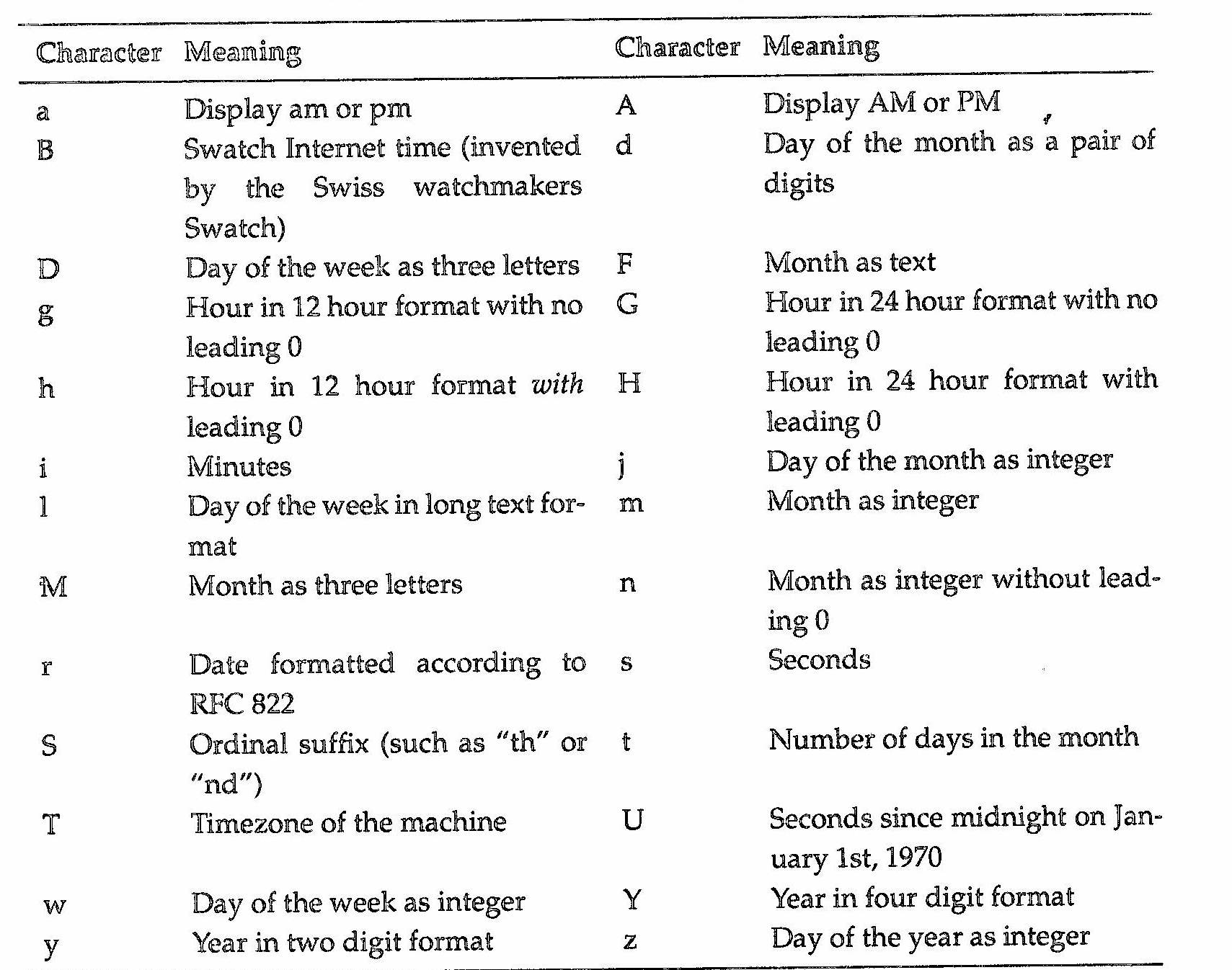
**srand(integer)**

Seed the random number generator using the supplied value. Random number gen­erators are not really random. If left to their own devices and called repeatedly, they actually produce a sequence of values. At some point the sequence will repeat, the trick in writing a random number generator is to make the sequence before repetition very long. The sequence, and it's length, are determined by the *seed* value which is supplied. Therefore to get the appearance of a random sequence you must re-seed the generator each time that you use it.

**tan(float)** Returns the tangent in radians.

**Dates and Times**

PHP has a decent set of functions for manipulating times and dates. Most often you'll want to get the current time and date to use as a timestamp either on a page or in a database. These functions are provided:



**date(format[, timestamp])**

Formats a timestamp as a date. If no timestamp is supplied, the current time is used. The format is defined by a string which uses the set of characters shown in Table 12.3. The following example displays a neatly formatted date:

<?php

echo(date("I ds F, Y"));

?>

The result of that code is shown in Figure 12.5. If you want formatting that is suitable for your locale, you should investigate the strftime () function.

**getdate([timestamp] )**

Returns an associative array containing the elements of the date. If no timestamp is given, the current time is used. The keys for the array are:

• seconds,

• minutes,

• hours,

• mday (day of the month),

• wday (day of the week),

• mon (numeric month),

• year (numeric year),

• yday (day of the year as an integer),

• weekday (as text, using the full name of the day),

• month (as text using the full name).

**localtime([timestamp[, associative]])**

Returns an array containing the elements of the timestamp. If no timestamp is given, the current time is used instead. If the Boolean value associative is TRUE the array will be returned as an associative array. The keys will be:

• tm\_sec

• tm\_min

• tm\_hour

• tm\_mday

• tm\_mon

• tm\_year

• tm\_wday

• tm\_yday

**mktime(hour, min, sec, month, day, year)**

Returns a UNIX timestamp in seconds for the given date and time. All parameters are supplied as integers. The last day of a month can be expressed as day 0 of the *next* month.

**time()**

Returns the a UNIX timestamp which represents the current time in seconds since midnight on January

**Using Files**

Most of your scripts will need to store data at some point. You may only need to write data into a temporary file which gets deleted once the script has completed, or you may want to store many megabytes of data gathered from all the visitors to your site.

Much of the literature and many of the tutorials which discuss PHP talk about storing all of this data in a relational database. If you surf the Web looking for PHP tutorials you'll see the same combination of PHP, Linux and MySQL described over and over again. For many applications a database is an irrelevance: they are designed to store and manipulate large volumes of data and many simultaneous connections. If you've got a low volume site with few connections and relatively little data you won't get any benefits from'a database. Instead if you're creating a guestbook or an online diary, you're far better off saving your data in simple files.

PHP has many functions which handle files and directories. A file is a collection of data stored on a disk and accessed via a unique name. Generally when we talk about files we mean stored data on the local machine or on a machine which appears to be the local machine.

PHP is designed for the Web, it is supposed to work in a connected. Web world. Can it be used to manipulate files on remote servers? The answer to that is an emphatic *yes,* and what's more, the process is almost transparent. The fopen () function which is used to open files is described and demonstrated below. At this stage I'd just like to outline its Internet facilities

Opening a file is done with fopen (filename) which accepts a filename as the first of its parameters. If the filename is a standard directory/file combination such as these:

$data = c:\\MyFiles\\data\\guests.dat);

$data = fopen(/home/chris/data/guests.dat);

the file is assumed to be on the same machine that the script is on. Actually, and to be more rigorous, it's assumed to be *mounted* by the operating system and available as if it is on that machine.

Note:

On Windows machines, backslashes in the path to a file must be escaped or re­placed with forward slashes.

If the filename starts with http: //, the file is assumed to be on a remote Web server. A standard HTTP connection is opened to the server and the file is retrieved. When you access data like this you're not really working with the original file. The *open* file is really the data sent back by the HTTP response. You're actually going to be working on a copy of the data. This has important implications since the file might change whilst you're working on it. You can never assume that remote data is bang up to date; generally, though, this doesn't matter. Web data will give you a snapshot of the state of a system which is accurate enough 99 per cent of the time.

Filenames which start with ftp: // are assumed to be on FTP servers. They can be accessed for reading and writing, although not at the same time. If the remote machine isn't running an FTP server you'll get an error if you try to use the FTP protocol to read or write a file on it. When you're using FTP you'll need to make *two* connections if you want to update a file: the first connection is made to download a *copy* of the file, the second to upload the amended version. During the period that your script is creating the amended file, the original is available to other users. As with HTTP access, you cannot assume that you've got the latest version. "Ah," you may say, "but I'm the only person who can access those files directly." That's fine, but how many copies of your script are executing concurrently? You've really no way of knowing. If concurrent access, especially to update the file, is likely to be important then you ought to consider using a database which enforces transactions and data integrity.

The files on FTP servers are protected by usernames and passwords. You may have downloaded files from FTP servers in the past via your Web browser without entering either a username or a password. Your Web browser will have done this for you automati­cally. Many file repositories around the Internet support *anonymous FTP* Which lets anyone log on to the system as user *anonymous* provided they give their email address as a pass­word. Files can then be downloaded from the server. If you're using remote files as part of a Web application, they won't be accessible via anonymous FTP, you'll need an account on the FTP server. The filename becomes a combination of username, password and file:

$data = fopen("ftp://chris:password@ftp.shu.ac.uk/guests.dat")

with the username and password separated by a colon.

The full set of PHP file functions is described in the supplied documentation. I'm only going to describe some of the more important functions and show you a simple example.

**copy($source, $dest)**

Copies the contents of $ source into $dest. The two parameters are the names of the files. If the copy fails, the function returns FALSE, otherwise it returns TRUE.

**fclose($fp)**

Closes the file which is pointed to by $ fp. The parameter $ fp is a file handle, created using fopen () *not* the name of the file.

**feof($fp)**

Checks if the end of file has been reached. Returns TRUE if it has, otherwise it returns FALSE;

**fflush($fp)**

Writes all buffered output to the file which $fp points to. You should always call f flush () before you close a file because the operating system may buffer data with­out your knowledge. By calling this function you ensure that all data gets written out.

**fgets($fp, length)**

Reads up to length-1 bytes of data from $fp. The call to fgets () ends when a new-line character or the end of file marker is reached. Newline characters are returned as part of the string.

**fgetss($fp, length)**

Works like fgets () but strips any HTML tags out of the data as it is read.

**file(filename)**

Reads the complete contents from the file and stores them in an array. Each line of data makes up a separate element in the array.

**file\_exists(filename)**

Returns TRUE if the file exists and FALSE if it doesn't. This function will not work on remote files such as those you might try to access with FTP or HTTP.

**flock($fp, operation)**

Locks a file so that access to it can be controlled. This function works under most UNIX systems and Microsoft Windows. Many similar functions in other languages are unavailable on Windows. Operation takes one of the following values:

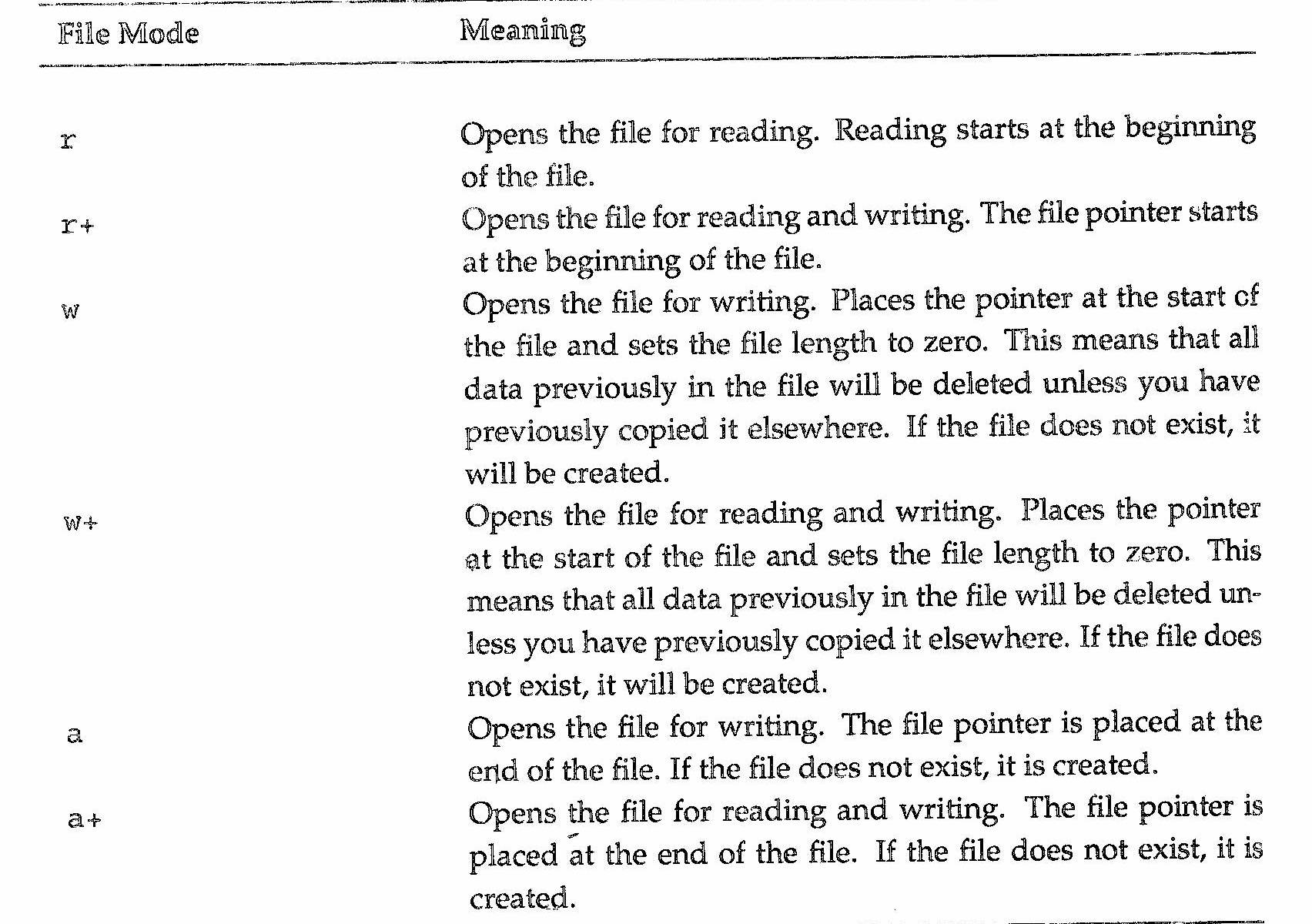
• LOCK\_SH denotes a shared lock - used for reading a file,

• LOCK\_EX denotes an exclusive lock - for writing to a file,

• LOCK\_UN releases a file lock.

**fopen(filename, mode)**

Opens the file. Works on local and remote files as described above.The modes are listed in Table



**fputs($fp, string[, length])**

Writes the string to the file pointed to by $ fp. If the optional length is given, that many bytes are written. Otherwise the entire string is written.

**fread($fp, length)**

Reads up to length bytes from $fp. This operation is suitable for reading either text or ASCII data on systems such as Windows which distinguish between the two.

**Tmpfile()**

Creates a temporary file which is ready to be written to. The function returns a file-handle to the new file. The file is deleted when f close () is called on it.

**unlink(f i1ename)**

Deletes the file whose name is given as parameter. Return FALSE if there is an error.

The following code operates on the simple cookie database, that code reads through the database and echoes the contents out to the browser:

<html> <head>

<title>Biscuit Database</title> </head> <body>

<hl>The Biscuit Database</hl> <?php

makeContent() ;

?> </body> </html>

<?php

function makeContent() {

$fp = fopen("cookies.dat", "r") ;

if($fp) {

while(!feof($fp)){

$row = fgets($fp, 1024);

$bicks = preg\_split("/::/", $row) ;

echo "<h3>Next Row</h3xul>" ;

for($i = 0; $i < count($bicks); $i++) { echo "<li>" ;

echo $bicks[$i] ;

echo "</li>" ;

}

echo "</ul>" ;

} fclose($fp) ;

}?>

**Cookies**

A cookie is a text string which is stored in your visitor's PC by *your* script. Cookies are both useful and controversial. If you're running a large site or doing e-commerce, then cookies are an excellent way of tracking users or of managing transactions. Most shopping cart applications are based around cookies, using them to track the items that a shopper has ordered. Users, especially those concerned with civil liberties in cyberspace, are worried about the wide-spread use of cookies. They're often used by advertisers, especially those selling banner advertising, to track users through sites. Whilst the advertiser won't know who is viewing their banners, they will be able to build a picture of the type of viewer they're attracting. I guess that whether this offends you or not depends upon your philo­sophical outlook. The important consideration for a developer is that if you use cookies too widely, you may drive potential customers away - or worse yet, make it impossible for them to do business with you.

Creating cookies in PHP is easy. Simply place a call to the **setcookie ()** function before the HTML tag at the top of your page. It needs to go there since cookies are sent as part of the HTTP header message which is sent *before* the HTML page.

**setcookie(name[, value[, expiry]])**

Each cookie has to have a name. Generally all cookies from each area of your site will have the same name so that you can retrieve them and track your visitors. If you don't pass a name to the function, you'll get an error. If the only parameter that you give to the function is the name, the cookie will be deleted. Cookies are designed to expire after a preset length of time. That expiry time is set using the third parameter which is an integer value. This should be created using the PHP mktime () or time () functions. If you use a time that has already passed, the cookie will be deleted.

Cookies are automatically returned and can be accessed using a variable which is the same as their name. This code reads in and displays a cookie, and sets it with a new value which was passed to the script as a parameter. The cookie will expire after 20 minutes:

<?php

setcookie("CookieTest", $val, time()+1200);

?> <html>

<head>

<tit Ie>Functions</title>

</head>

<body>

<hl>Using Cookies</hl> <?php

echo "<h2>The cookie is: $CookieTest</h2>" ?>

</body> </html>

**Other Useful Functions**

Here are a few more functions that you need to use the language in productions code:

**is\_array(var)** Returns TRUE if the variable is an array,

**is\_double(var)** Returns TRUE if the variable is a double,

**is\_float(var)** Returns TRUE if the variable is a floating point number,

**is\_int(var)** Returns TRUE if the variable is an integer,

**is\_string(var)** Returns TRUE if the variable is a string, '

**is\_obj ect(var)** Returns TRUE if the variable is an object.