This book has been prepared exclusively for

VEIS COMPUTER EDUCATION

PHP

Introduction and Installation of PHP :

- PHP is an acronym for "PHP Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP costs nothing, it is free to download and use

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

Benefits of PHP

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can restrict users to access some pages on your website
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP:

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

PHP is a server scripting language, and is a powerful tool for making dynamic and interactive Web pages quickly.
PHP is a widely-used, free, and efficient alternative to Microsoft's ASP and other server side langueges.
PHP can be used in many contexts - discussion forums, polls, shops, SMS gateways, mailing lists, etc.
It has become a very popular dynamic server side scripting language that allows an application developer to create very simple to very complex mechanisms for the web. PHP code can be directly mingled in with your HTML page content as long as the page has a .php extension. Any HTML file you have can be converted into a PHP file.

Popular PHP web sites

• Facebook	**• Google**	**• Wordpress**	**• Joomla**	**• Develop PHP**
• PHPBB	**• Zen Cart**	**• OScommerce**	**• Web Intersect**	

The general logic within PHP is similar to other popular programming languages. The syntax of PHP is relatively easy to understand once you create a few applications.

Web Host With PHP Support

If your server has activated support for PHP you do not need to do anything.
Just create some .php files, place them in your web directory, and the server will automatically print on web for you. You do not need to compile anything or install any extra tools.

Install PHP on your computer

However, if your server does not support PHP, you must:

- install a web server
- install PHP
- install a database, such as MySQL

The official PHP website (PHP.net) has installation instructions for PHP:
http://php.net/manual/en/install.php

Create a local testing environment on your computer Using a bundled environment software makes life easier: WAMP - http://www.wampserver.com/en/
XAMPP - http://www.apachefriends.org/en/xampp.html
LAMP - https://www.linux.com/learn/tutorials/288158-easy-lamp-server-installation

PHP 5 Syntax

A PHP script can be placed anywhere in the document.

A PHP script starts with **<?php** and ends with **?>**:

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is ".php".

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

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hi World!" on a web page:

Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hi World!";
?>

</body>
</html>
```

Note: PHP statements are terminated by semicolon (;). The closing tag of a block of PHP code also automatically implies a semicolon (so you do not have to have a semicolon terminating the last line of a PHP block).

Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is editing the code!

Comments are useful for:

- To let others understand what you are doing - Comments let other programmers understand what you were doing in each step (if you work in a group)
- To remind yourself what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports three ways of commenting:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
// This is a single line comment

# This is also a single line comment

/*
This is a multiple lines comment block
that spans over more than
one line
*/
?>

</body>
</html>
```

PHP Case Sensitivity

In PHP, all user-defined functions, classes, and keywords (e.g. if, else, while, echo, etc.) are NOT case-sensitive.

In the example below, all three echo statements below are legal (and equal):

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
ECHO "Hi World!<br>";
echo "Hi World!<br>";
EcHo "Hi World!<br>";
?>

</body>
</html>
```

However; in PHP, all variables are case-sensitive.

In the example below, only the first statement will display the value of the $color variable (this is because $color, $COLOR, and $coLOR are treated as three different variables):

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$color="red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My bus is " . $coLOR . "<br>";
?>

</body>
</html>
```

Basically, a PHP file is a text file with the extension **.php** which consists of:

1. **Text**
2. **HTML tags**
3. **PHP Scripts**

PHP Scripts

Let's get started with your first PHP page.

Example: *The HTML code should look like this:*

```
<html>
<head>
<title>My first PHP page</title>

</head>
<body>
```

```
</body>
</html>
```

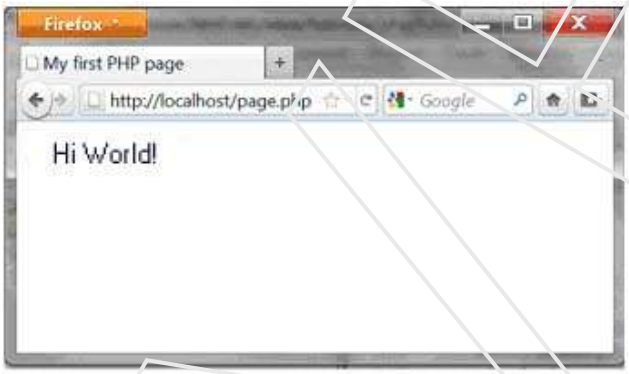
PHP is all about **writing commands to a server.** So let's write a command to the server.

We need to tell the server when the PHP will **start** and **end.** In PHP you use the tags **<?php** and **?>** to mark the start and end for the PHP codes that the server must execute (on most servers it will be sufficient to use just **<?** as start tag, but **<?php** is the most correct to use the first time PHP is used.)

Now try to add the following simple code to your HTML code:

```
<html>
<head>
<title>My first PHP page</title>
</head>
<body>

<?
echo "<h1>Hi World!</h1>";

?>

</body>
</html>
```

Result:



But it gets interesting when you look at the HTML code in the browser (by selecting "view source"):

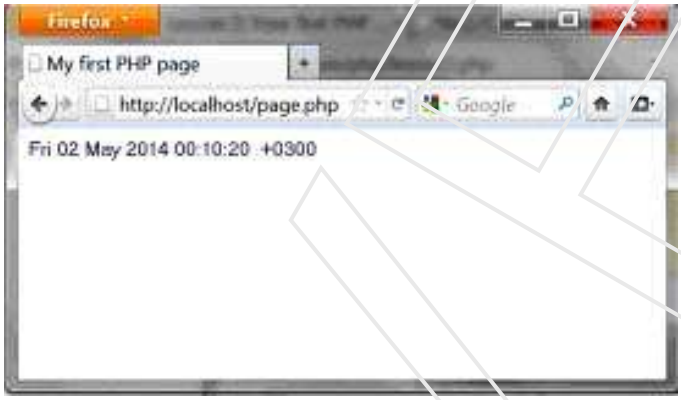
It is only the server that can see the PHP codes - **the client (the browser) only sees the result!**

We asked the server to write <h1> Hi World!</h1>. In a more technical language, one would say that we used the string function echo to write a specified string to the client where the semicolon ends the command. Our first example is obviously not particularly exciting.

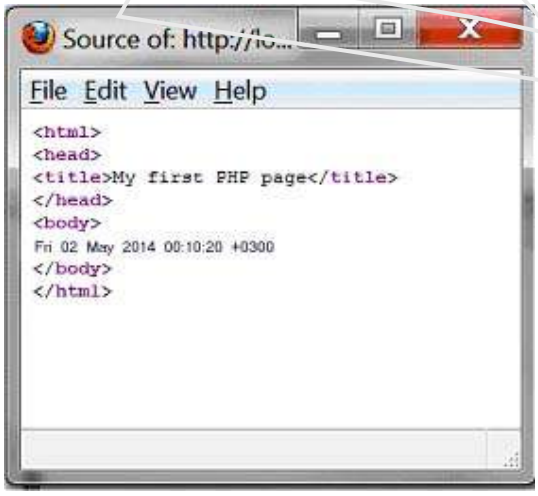
Example: ***Example, write the current date and time:***

```
<html>
<head>
<title>My first PHP page</title>

</head>
<body>

<?php

echo date("r");

?>
</body>
</html>
```

Result:



And the corresponding HTML code:



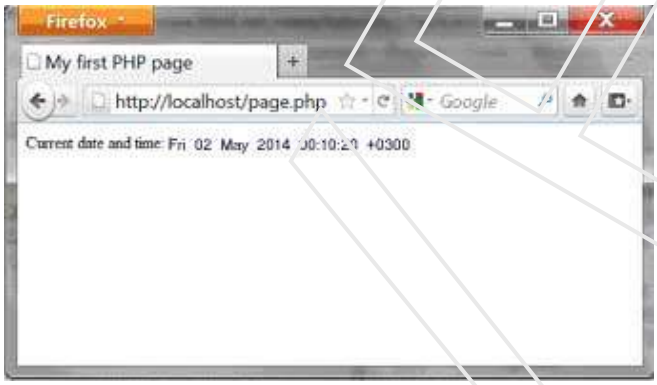
Note:

1. That if you refresh the page in the browser, a new time is written. The server writes the current date and time each time the page is sent to a client.
2. It is also important to note that the HTML code contains only the date - not the PHP codes. Therefore, the example is not affected by which browser is used. Actually, all functionalities that are made with *server-side* technologies always **work in all browsers!**
3. The semicolon after the code line. It is a separator and very important to include - otherwise the script won't work.

In the example, we used date, which is a function that returns the current date and time on the server. Example, writing both a *string* and a *function* - separated by "**.**"

```
<html>
<head>
<title>My first PHP document</title>
</head>
<body>
<?php
echo "<p>Current date and time: " . date("r") . "</p>";
?>

</body>
</html>
```

Result:



And the corresponding HTML code:

Working with time and dates

We will try to look at the many different options for working with time and dates in PHP.

Time and date functions

PHP provides a wide range of functions in relation to time and date. In this lesson, With different parameters, the date function can return the current date / time in many different formats. Some of the most useful parameters are:

date("y")
: Returns the current year from a date - with today's date, it returns: **14**

date("m")
: Returns the current month from a date - with today's date, it returns: **05**

date("n")
: Returns the current month from a date without leading zeroes (eg. "1" instead of "01") - with today's date, it returns: **5**

date("F")
: Returns the current month name from a date - with today's date, it returns: **May**

date("d")
: Returns the current day of the month from a date - with today's date, it returns: **19**

date("l")
: Returns the name of the current weekday from a date - with today's date, it returns: **Monday**

date("w")
: Returns the current day of the week from a date - with today's date, it returns: **1**

date("H")
: Returns the current hour from a time - with the current time, it returns: **17**

date("i")
: Returns the current minute from a time - with the current time, it returns: **04**

date("s")
: Returns the current second from a time - with the current time, it returns: **29**

This example illustrates the use of the date function:

```
<html>
<head>
<title>Time and date</title>

</head>
<body>

<?php

echo "<p>Today it's " . date("l") . "</p>";

?>

</body>
</html>
```

The time is 1400511869

And now hold on... because now it becomes a little nerdy! The function time() returns the current time as the number of seconds since January 1, 1982, 12:00 PM, GMT.

```
<html>
<head>
<title>time and date</title>
</head>
<body>

<?php

echo "<p>It's been exactly " . time() . " seconds since January 1,
1982, 12:00 PM, GMT </ p> ";

?>

</body>
</html>
```

Time expressed in the number of seconds since January 1, 1982, 12:00 PM GMT is a so-called "timestamp" (UNIX timestamp) and is quite useful when you work with dates/times in the future or the past.

By default, the date function uses the current timestamp (i.e. the current value of time()). But with an extra parameter you can specify a different time stamp and thus work with the future or the past. In the example below, we set the timestamp to 0 seconds from January 1, 1982 12:00 PM, GMT. Thereby we can check what day of week January 1, 1982 was.

```
<html>
<head>
<title>time and date</title>
</head>
<body>

<?php

echo "<p>January 1, 1982 was a " . date("l",0) . "</p>";

?>

</body>
</html>
```

Unless you are a mathematical genius, it quickly becomes complicated to count the number of seconds since January 1, 1982 to a specific time in the future or past. But here you can get help from another nifty function: mktime, which does the calculations for you.

The syntax for mktime is *(hour, minute, second, month, day, year).* The example below converts the time of the first step on the Moon (July 21, 1981, 02:56):

```
<html>
<head>
<title>time and date</title>
</head>
<body>

<?php

echo mktime (2,56,0,7,21,1981);

?>

</body>
</html>
```

Notice that it's returning a negative number as the date is earlier than January 1, 1982.

We can now put this together with the date function and find out which weekday this historic day took place.

```
<html>
<head>
<title>time and date</title>
</head>
<body>

<?php

echo date("l", mktime(2,56,0,7,21,1981));

?>

</body>
</html>
```

Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is editing the code!

Comments are useful for:

- To let others understand what you are doing - Comments let other programmers understand what you were doing in each step (if you work in a group)
- To remind yourself what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

PHP supports three ways of commenting:

How do you insert comments?

It is quite easy to insert a comment. You simply start the comment with a double slash: "//".

Example:

```
<html>
<head>
<title>Loops</title>
</head>
<body>

<?php

// Here we write color codes using three loops

// Red can be between 0 and 255
for ($intRed=0; $intRed<=255; $intRed=$intRed+30) {

   // Green can be between 0 and 255
   for ($intGreen=0; $ intGreen<=255; $intGreen=$intGreen+30) {

      // Blue can be between 0 and 255
      for ($ intBlue=0; $intBlue<=255; $intBlue=$intBlue+30) {

      // The color code is made on the form rgb(red,green,blue)
        $strColor = "rgb(" . $intRed . "," . $intGreen . "," .
$intBlue . ")"

      // Now we write the color code to the client
        echo "<span style='color:" . $strColor . "'> " .
$strColor . " </span>";

      // Closes the loops
      }
   }
}

?>
```

For the sake of the example, we have included many extra comments, so it should be clear that you are far better off debugging a script with comments than without.

Place PHP Code Blocks

PHP blocks(tags) can go anywhere you like in a .php web page document. Anywhere. Many PHP developers choose to make one large PHP block above their <!DOCTYPE> and <html> tags. In this block they would handle all of the information gathering, parsing, or whatever needs to be scripted. They then make variables out of information to echo or print the values out to the browser.

Alternately you can choose to have many PHP blocks exactly located in the HTML <head> or <body> section of the document where that certain data would display.

Both of these scripts below are the same in functionality, and will render the exact same output.

<Code>

```
<?php
$year = date("Y");
?>
<html>
<body>
What year is it?<br /><br />
<?php echo "It is $year, where have you been?"; ?>
</body>
</html>
```

<Code>

```
<html>
<body>
What year is it?<br /><br />
<?php
$year = date("Y");
echo "It is $year, where have you been?";
?>
</body>
</html>
```

Browser Output Display

What year is it?

It is 2014, where have you been?

print Vs. echo

print and echo are both used to output data to browser software or other technologies that intake external data. As you come to view different PHP scripts in your travels online you may notice some authors use echo and some use print. Let us discuss the difference.

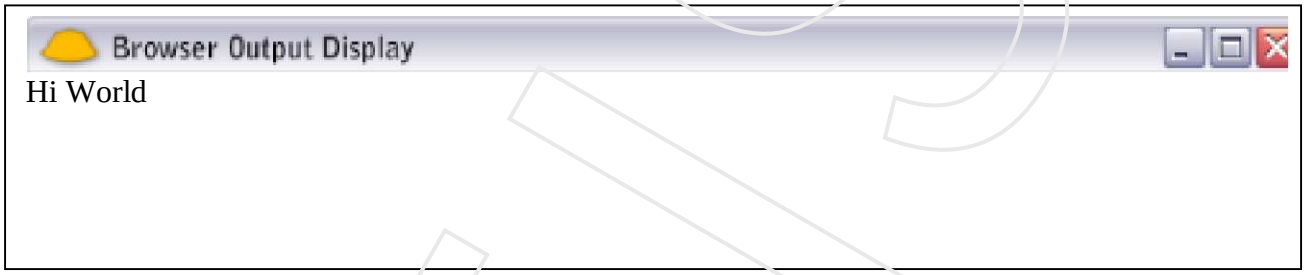
echo() is a language construct, so you are not required to use parentheses with it. If you ever need to pass more than one parameter to echo(), the parameters must not be enclosed within parentheses.

print() behaves as a Function, but is not actually a function. print is sometimes mistaken as a function by many programmers since it sets a return value, but it is also a language construct like echo. It simply outputs a string of data. And you are not required to use parentheses with it.

If you are going to be picky about miliseconds... echo is faster due to the fact that it does not set a return value like print and most functions do. So there is a difference... but not a noticable one.

Both of these methods for producing output are widely used.

<Code>
```
<?php
// These will all perform the required output the same
echo "Hi World!";

print "Hi World!";

print ("Hi World!");
?>
!
```



Mix HTML and CSS Into Your PHP Output

You can mix HTML and CSS into your PHP output. This becomes especially handy when dealing with loops and result sets from a MySQL database.

Here is an example for you to examine and test

<Code>
```
<html>
<body>
<?php echo '<div style="background-
color: #639; padding:8px; width:110px;">
<font color="#FFFF00"><strong>Hi World!</strong></font>
</div>'; ?>
</body>
```


```
</html>
```

Hi World!

PHP 5 Variables

Variables are "containers" for storing information: A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case sensitive ($y and $Y are two different variables)

Example

```
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

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:

Example

```
<?php
$txt="Hi world!";
$x=5;
$y=10.5;
?>
```

After the execution of the statements above, the variable **txt** will hold the value **Hi world!**, the variable **x** will hold the value **5**, and the variable **y** will hold the value **10.5**.

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

PHP is a Loosely Type Language

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 other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

PHP Variables Scope

In PHP, variables can be declared anywhere in the script.
The scope of a variable is the part of the script where the variable can be referenced/used.
PHP has three different variable scopes:

- local
- global
- static

Local and Global Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function.
A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function.
The following example tests variables with local and global scope:

Example

```
<?php
$x=5; // global scope

function myTest() {
  $y=10; // local scope
  echo "<p>Test variables inside the function:</p>";
  echo "Variable x is: $x";
  echo "<br>";
  echo "Variable y is: $y";
}

myTest();

echo "<p>Test variables outside the function:</p>";
echo "Variable x is: $x";
echo "<br>";
echo "Variable y is: $y";
?>
```

In the example above there are two variables $x and $y and a function myTest(). $x is a global variable since it is declared outside the function and $y is a local variable since it is created inside the function. When we output the values of the two variables inside the myTest() function, it prints the value of $y as it is the locally declared, but cannot print the value of $x since it is created outside the function.

Then, when we output the values of the two variables outside the myTest() function, it prints the value of $x, but cannot print the value of $y since it is a local variable and it is created inside the myTest() function.

PHP The global Keyword

The global keyword is used to access a global variable from within a function.
To do this, use the global keyword before the variables (inside the function):

Example

```
<?php
$x=5;
$y=10;

function myTest() {
  global $x,$y;
  $y=$x+$y;
}

myTest();
echo $y; // outputs 15
?>
```

PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

Example

```
<?php
$x=5;
$y=10;

function myTest() {
  $GLOBALS['y']=$GLOBALS['x']+$GLOBALS['y'];
}

myTest();
echo $y; // outputs 15
?>
```

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the **static** keyword when you first declare the variable:

Example

```
<?php

function myTest() {
  static $x=0;
  echo $x;
  $x++;
}

myTest();
myTest();
myTest();

?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Passing variables in a URL

When you work with PHP, you often need to pass variables from one page to another.

How does it work?

Maybe you have wondered why some URLs look something like this:

```
http://veinstitution.com/page.php?id=1254
```

Why is there a question mark after the page name?

The answer is that the characters after the question mark are an **HTTP query string**. An HTTP query string can contain both variables and their values. In the example above, the HTTP query string contains a variable named "id", with the value "1254".

Here is another example:

```
http://veinstitution.com/page.php?name=Jatin
```

Again, you have a variable ("name") with a value ("Jatin").

Get the variable with PHP

Let's say you have a PHP page named **people.php**. Now you can call this page using the following URL:

```
people.php?name=Jatin
```

With PHP, you will be able to get the value of the variable 'name' like this:

```
$_GET["name"]
```

So, you use **$_GET** to find the value of a named variable. Let's try it in an example:

```
<html>
<head>
<title>Query string</title>
</head>
<body>

<?php

// The value of the variable name is found
echo "<h1>Hi " . $_GET["name"] . "</h1>";

?>

</body>
</html>
```

When you look at the example above, try to replace the name "Jatin" with your own name in the URL and then call the document again.

Several variables in the same URL

You are not limited to pass only one variable in a URL. By separating the variables with **&**, multiple variables can be passed:

```
people.php?name=Jatin&age=18
```

This URL contains two variables: name and age. In the same way as above, you can get the variables like this:

```
$_GET["name"]
$_GET["age"]
```

Let's add the extra variable to the example:

```
<html>
<head>
<title>Query string </title>
</head>
<body>

<?php

// The value of the variable name is found
echo "<h1>Hi " . $_GET["name"] . "</h1>";

// The value of the variable age is found
echo "<h1>You are " . $_GET["age"] . " years old </h1>";

?>

</body>
</html>
```

PHP Global Variables - Superglobals

Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP superglobal variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES

- $_ENV
- $_COOKIE
- $_SESSION

This chapter will explain some of the superglobals, and the rest will be explained in later chapters.

PHP $GLOBALS

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).
PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.

Example

```
<?php
$x = 75;
$y = 25;

function addition() {
  $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}

addition();
echo $z;
?>
```

In the example above, since z is a variable present within the $GLOBALS array, it is also accessible from outside the function!

PHP $_SERVER

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

Example

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

The following table lists the most important elements that can go inside $_SERVER:

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.veinstitution.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@veinstitution.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

PHP $_REQUEST

PHP $_REQUEST is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
Name: <input type="text" name="fname">
<input type="submit">
</form>

<?php
$name = $_REQUEST['fname'];
echo $name;
?>

</body>
</html>
```

PHP $_POST

PHP $_POST is widely used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_POST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
Name: <input type="text" name="fname">
<input type="submit">
</form>

<?php
$name = $_POST['fname'];
echo $name;
?>
```

```
</body>
</html>
```

PHP $_GET

PHP $_GET can also be used to collect form data after submitting an HTML form with method="get".

$_GET can also collect data sent in the URL.

Assume we have an HTML page that contains a hyperlink with parameters:

```
<html>
<body>

<a href="test_get.php?subject=PHP&web=Veinstitution.com">Test $GET</a>

</body>
</html>
```

When a user clicks on the link "Test $GET", the parameters "subject" and "web" is sent to "test_get.php", and you can then acces their values in "test_get.php" with $_GET.

The example below shows the code in "test_get php":

Example

```
<html>
<body>

<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>

</body>
```

Passing variables with forms

Interactive websites require input from users. One of the most common ways to get input is with forms.

<form>

When you code a form, there are two particular important attributes: **action** and **method**.

action
: Is used to enter the URL where the form is submitted. It would be the PHP file that you want to handle the input.

method
: Can either have the value "post" or "get", which are two different methods to pass data. At this point, you don't need to know much about the difference, but with "get", the data is sent through the URL,

and with "post", the data is sent as a block of data through standard input service (STDIN). In the last lesson, we looked at how data is retrieved through the URL using $_GET.

An HTML page with a form

The page that contains the form doesn't need to be a PHP file (but it can be). It need not even be on the same site as the file that will receive the data.

Example:

```
<html>
<head>
<title>Form</title>
</head>
<body>

<h1>Enter your name</h1>

<form method="post" action="handler.php">
<input type="text" name="username">
<input type="submit">
</form>

</body>
</html>
```

The result in the browser is a form:

Requesting form data with PHP

When you need to request data submitted through a form (post method), you use **$_POST**:

```
$_POST["fieldname"];
```

Which returns the value of a field in the form.

Example

First create a page with a form as above. Then make a PHP page named "handler.php" (notice that this is the name of the page we wrote in the action attribute in our <form>).

The file "handler.php" shall have the following content:

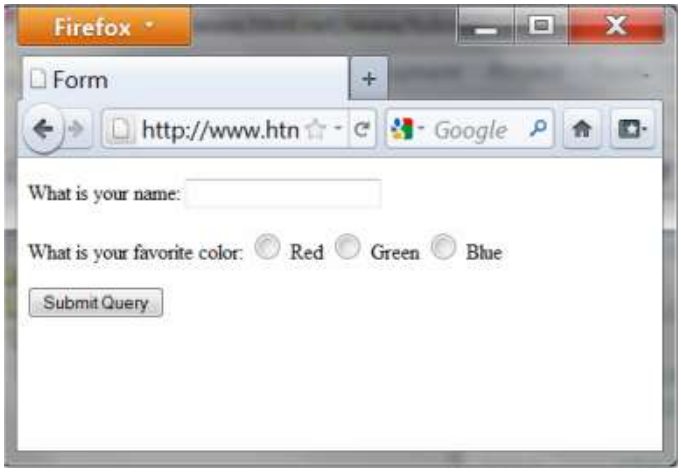
```
<html>
<head>
<title>Form</title>
</head>

<body>

<?php

echo "<h1>Hi " . $_POST["username"] . "</h1>";

?>

</body>
</html>
```

User input and conditions

In the next example, we will try to use user input to create conditions. First, we need a form:

```
<html>
<head>
<title>Form</title>
</head>
<body>

<form method="post" action="handler.php">

<p>What is your name:</p>
<input type="text" name="username"></p>

<p>What is your favorite color:
<input type="radio" name="favoritecolor" value="r" /> Red
<input type="radio" name="favoritecolor" value="g" /> Green
<input type="radio" name="favoritecolor" value="b" /> Blue </p>

<input type="submit" value="Submit" />

</form>

</body>
</html>
```

Which will look like this in the browser:



Now we will use these inputs to create a page that automatically changes background color according to what the user's favorite color is. We can do this by creating a condition that uses the data the user has filled out in the form.

```
<?php

$strHeading = "<h1>Hi " . $_POST["username"] . "</h1>";

switch ($_POST["favoritecolor"]) {
case "r":
      $strBackgroundColor = "rgb(255,0,0)";
      break;
case "g";
      $strBackgroundColor = "rgb(0,255,0)";
      break;
case "b":
      $strBackgroundColor = "rgb(0,0,255)";
      break;
default:
      $strBackgroundColor = "rgb(255,255,255)";
      break;
}

?>

<html>
<head>
<title>Form</title>

</head>
<body style="background: <?php echo $strBackgroundColor; ?>;">

<? echo $strHeading; ?>

</body>
</html>
```

The background color will be white if the user has not chosen any favorite color in the form. This is done by using **default** to specify what should happen if none of the above conditions are met.

But what if the user does not fill out his name, Then it only says "Hi" in the title. We will use an extra condition to change that.

```
<?php

$strUsername = $_POST["username"];

if ($strUsername != "") {
    $strHeading = "<h1>Hi " . $_POST["username"] . "</h1>";
}
else {
    $strHeading = "<h1>Hi stranger!</h1> ";
}

switch ($_POST["favoritecolor"]) {
case "r":
      $strBackgroundColor = "rgb(255,0,0)";
      break;
case "g";
      $strBackgroundColor = "rgb(0,255,0)";
      break;
case "b":
      $strBackgroundColor = "rgb(0,0,255)";
      break;
default:
      $strBackgroundColor = "rgb(255,255,255)";
      break;
}

?>

<html>

<head>

<title>Form</title>
</head>
<body style="background: <?php echo $strBackgroundColor; ?>;">

<? echo $strHeading; ?>

</body>
</html>
```

In the example above, we use a condition to **validate** the information from the user. In this case, it might not be so important if the user did not write his name. But as you code more and more advanced stuff, it's vital that you take into account that the user may not always fill out forms the way you had imagined.

Example: contact form

With your new knowledge of PHP and forms, you are able to create a contact form using the **function mail**, which has the following syntax:

```
mail(to, subject, message);
```

First, we need a simple HTML form:

```
<html>
<head>
<title>Contact form</title>
</head>
<body>

<h1>Contact form</h1>

<form method="post" action="handler.php">
<p>Subject:<br /><input type="text" name="subject" /></p>
<p>Message:<br /><textarea name="message"></textarea></p>
<input type="submit">
</form>

</body>
</html>
```

Next we need a PHP script to send the users input:

```
<html>
<head>
<title>Functions</title>
</head>
<body>

<?php

// Recipient (change to your e-mail address)
$strEmail = "name@mydomain.com";

// Get user inputs
$strSubject = $_POST["subject"];
$strMessage = $_POST["message"];

mail($strEmail,$strSubject,$strMessage);
echo "Mail Sent.";


?>

</body>
</html>
```

Please note that the example will only work if you have access to a mail server. By default, this is not the case in XAMPP and most free hosts. Also, some hosts may require that you include a from header, which is done with an extra parameter:

mail*("you@yourdomain.com", "Test", "This is a test mail", "From: me@mydomain.com");*

PHP 5 Data Types

`String, Integer, Floating point numbers, Boolean, Array, Object, NULL.`

PHP sports a cool feature called automatic data typing. A PHP developer can claim variables and use them in most common situations without having to claim the data type that the variable is. But this is also a double edged sword because as a developer gets more advanced they may create functions or scripts that only accept one data type, and throw errors if you feed it a different type. But that is usually only in the most complex of PHP applications.

For instance, if my imported script and function is expecting an integer type variable and I feed it a string type, I will get an error and know that I must not try to feed it a string... it needs an integer type variable. No big deal, I will just make sure a number type variable is all that gets sent through that mechanism.

PHP Strings

A string is a sequence of characters, like "Hi world!".

A string can be any text inside quotes. You can use single or double quotes:

Example

```
<?php
$x = "Hi world!";
echo $x;
echo "<br>";
$x = 'Hi world!';
echo $x;
?>
```

PHP Integers

An integer is a number without decimals. Rules for integers:

- An integer must have at least one digit (0-9)
- An integer cannot contain comma or blanks
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

In the following example we will test different numbers. The PHP var_dump() function returns the data type and value of variables:

Example

```
<?php
$x = 5985;
var_dump($x);
echo "<br>";
$x = -345; // negative number
var_dump($x);
echo "<br>";
$x = 0x8C; // hexadecimal number
var_dump($x);
echo "<br>";
$x = 047; // octal number
var_dump($x);
?>
```

PHP Floating Point Numbers

A floating point number is a number with a decimal point or a number in exponential form.

In the following example we will test different numbers. The PHP var_dump() function returns the data type and value of variables:

Example

```
<?php
$x = 10.365;
var_dump($x);
echo "<br>";
$x = 2.4e3;
var_dump($x);
echo "<br>";
$x = 8E-5;
var_dump($x);
?>
```

PHP Booleans

Booleans can be either TRUE or FALSE.

```
$x=true;
$y=false;
```

Booleans are often used in conditional testing.

PHP Arrays

An array stores multiple values in one single variable.

In the following example we create an array, and then use the PHP var_dump() function to return the data type and value of the array:

Example

```
<?php
$cars=array("Maruti","BMW","Sail");
var_dump($cars);
?>
```

PHP Objects

An object is a data type which stores data and information on how to process that data.

In PHP, an object must be explicitly declared. First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods.We then define the data type in the object class, and then we use the data type in instances of that class:

Example

```
<?php
class Car
{
  var $color;
  function Car($color="green") {
    $this->color = $color;
  }
  function what_color() {
    return $this->color;
  }
}
?>
```

PHP NULL Value

The special NULL value represents that a variable has no value. NULL is the only possible value of data type NULL.

The NULL value identifies whether a variable is empty or not. Also useful to differentiate between the empty string and null values of databases.

Variables can be emptied by setting the value to NULL:

Example

```
<?php
$x="Hi world!";
$x=null;
var_dump($x);
?>
```

Functions

A function process *inputs* and returns an *output.* It can be useful if, for example, you have a wide range of data you have processed or if you have calculations or routines that must be performed many times.

A function has the following syntax:

```
Function Name(list of parameters) {
   Statement
}
```

This way, we can make a very simple function that can add the value 1 to a number. It could look like this:

```
function AddOne($x) {
   $x = $x + 1;
   echo $x;
}
```

Our function is named **AddOne** and must be called with a number - e g. 34....

```
echo AddOne(34);
```

... which (surprise!) will return 35.

The example above processes a number, but functions can work with text, dates or anything else. You can also create functions that are called by many different parameters. In this lesson you will see different examples of functions.

Example A: Function with more parameters

As mentioned above, you can easily create a function that can be called with more parameters. In the next example, we'll create a function that is called with three numbers and then returns the value of the three numbers added together:

```
<html>
<head>
<title>Functions</title>

</head>
<body>

<?php

function AddAll($number1,$number2,$number3) {
   $plus = $number1 + $number2 + $number3;
   return $plus;
}

echo "123 + 654 + 9 equals " . AddAll(123,654,9);

?>

</body>
</html>
```

Ok. Now that was almost too simple! But the point was just to show you that a function can be called with more parameters.

Example B: English date and time

A function that's called with a date and time returns it in the format: **Friday, 02 May, 2014, 10:00:00 AM**

```
<html>
<head>
<title>Functions</title>
</head>
<body>

<?php

function EnglishDateTime($date) {

  // Array with the English names of the days of the week
  $arrDay =
array("Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Su
nday");

  // Array with the English names of the months
  $arrMonth =
array("","January","February","March","April","May","June","July","Augu
st","September","October","November","December");

  // The date is constructed
  $EnglishDateTime = $arrDay[(date("w",$date))] . ", " .
date("d",$date);
  $EnglishDateTime = $EnglishDateTime  . " " .
$arrMonth[date("n",$date)] . " " . date("Y",$date);
  $EnglishDateTime = $EnglishDateTime  . ", " . date("H",$date) .
":" . date("i",$date);

  return $EnglishDateTime;

}

// Test function
echo EnglishDateTime(time());

?>

</body>
</html>
```

Please note how '$arrMonth' and '$EnglishDateTime' are constructed over several lines. This is done so that users with a low screen resolution can better see the example. The method has no effect on the code itself.

The function above works on all web servers regardless of language. This means that you can use such a function if your website, for example, is hosted on a French server, but you want English dates.

Changing Data Types

PHP has loose data typing. Because of this, changing a variable's data type is not a commonly used mechanism in PHP due to the fact that PHP will automatically cast your variable types when your script changes to use the variable as a different type and in a different way.

As illustrated in this sample code:

<Code>

```
<?php
$var1 = "16 years old";
$var2 = "14 years old";
// Since PHP knows that it cannot add two strings mathematically it aut
omatically
// removes the string parts of the variables and proceeds to do the mat
h correctly
$sumOfBoth = $var1 + $var2;
// Very interesting that we get such a result and not an error
echo $sumOfBoth;
?>
```

Browser Output Display

30

But let's say we have created a form. And in that form we want to know how many gigabytes our client's hard drive is. In entering the value they placed the size number and bytes characters like this "500gb", but we just wanted the number. If we try to then output or store just the number our plans are foiled because the client put the "gb" characters in there.

Illustrated in this sample code:

<Code>

```
<?php
// Let's say this is the value we wind up with from the form input
$var1 = "500gb";

echo "You stated that your Hard Drive size capacity is " . $var1 . " Gi
gabytes";
?>
```

Browser Output Display

You stated that your Hard Drive size capacity is 500gb Gigabytes

Use the settype() function to change data types

<Code>

```
<?php
// Same variable and value
$var1 = "500gb";
// This time use settype() and claim integer as the data type
settype($var1, 'integer');

echo "You stated that your Hard Drive size capacity is " . $var1 . " Gi
gabytes";
?>
```

You stated that your Hard Drive size capacity is 500 Gigabytes

Use Type Casting to change data types

<Code>

```
<?php
// Same variable and value
$var1 = "500gb";
// This time use type casting and claim integer as the data type
$var1 = (integer) $var1;

echo "You stated that your Hard Drive size capacity is " . $var1 . " Gi
gabytes";
?>
```

You stated that your Hard Drive size capacity is 500 Gigabytes

Operators in PHP

The Assignment Operator

The Assignment operator is the most widely used and well known of all the operators in PHP. You use it every time you create a variable. The "="(equal to) sign is used, and what it does is it takes the expression from the right and places it into the operand on the left. It is simple.

In this example we use the assignment operator(=) to place a value of 8 into the variable:

<Code>

```
<?php
$var1 = 8;
?>
```

We can also combine operators together to produce expression results that are set into our variables as value.

Arithmetic Operators in PHP

PHP can perform simple mathimatical operations all the way to complex trigonometric equations. The operator symbols used all make good sense for the mathematical action they perform.

Here are the arithmetic symbols and how to apply them in the most basic way:

<Code>

```
<?php
// Set a couple of sample integer variables
$var1 = 5;
$var2 = 3;
// Addition >>> Sum of $var1 and $var2
echo $var1 + $var2;
echo "<br />";
// Subtraction >>> Difference of $var1 and $var2
echo $var1 - $var2;
echo "<br />";
// Multiplication >>> Product of $var1 and $var2
echo $var1 * $var2;
echo "<br />";
// Division >>> Quotient of $var1 and $var2
echo $var1 / $var2;
echo "<br />";
//Modulus >>> Remainder of $var1 divided by $var2
echo $var1 % $var2;
echo "<br />";
//Negation >>> Opposite of $var1
echo -$var1;
?>
```

Browser Output Display

8
2
15
1.66666666667
2
-5

Comparison Operators in PHP

PHP comparison operators allow you to compare two values against each other. The comparison is read from left to right by the PHP engine. If you compare an integer with a string, the string is converted to a number. If you compare two numerical strings, they are compared as integers. It returns a numeric boolean value of "1"(=TRUE) if the expression comparison is met.

The following table shows their usage and result:

Name	Usage	Result
Equal	$var1 == $var2	**TRUE** if $var1 is equal to $var2.
Identical	$var1 === $var2	**TRUE** if $var1 is equal to $var2, and they are of the same type.
Not Equal	$var1 != $var2	**TRUE** if $var1 is not equal to $var2.
Not Equal	$var1 <> $var2	**TRUE** if $var1 is not equal to $var2.
Not Identical	$var1 !== $var2	**TRUE** if $var1 is not equal to $var2, or they are not the same type.
Less Than	$var1 < $var2	**TRUE** if $var1 is strictly less than $var2.
Greater Than	$var1 > $var2	**TRUE** if $var1 is strictly greater than $var2.
Less Than or Equal to	$var1 <= $var2	**TRUE** if $var1 is less than or equal to $var2.
Greater Than or Equal to	$var1 >= $var2	**TRUE** if $var1 is greater than or equal to $var2.

Example:

<Code>
```
<?php
$var1 = 5;
$var2 = 3;
// Is Greater Than Comparison
echo $var1 > $var2; // should output a "1" meaning TRUE
?>
```

Browser Output Display

1

Incremental and Decremental Operators

Incrementing and Decrementing operators are very widely used in most programming languages. They allow a PHP programmer to increment and decrement values as needed. Many times they are used in a loop to keep track of the index key number in the loop's array. We discuss loops in detail in later sections.

The following table shows their usage and result logic:

Name	Usage	Result
Pre-increment	++$myVar	Increments $myVar by one
Post-increment	$myVar++	Returns $myVar, then increments $myVar by one
Pre-decrement	--$myVar	Decrements $myVar by one, then returns $myVar
Post-decrement	$myVar--	Returns $myVar, then decrements $myVar by one

Here is a code example of incrementing a variable's value by one:

<Code>
```
<?php
$myVar = 3;
// Use Post-increment to increment the value by 1
$myVar++;
// Now display to browser or use its new value in script
echo $myVar;
?>
```

4

Logical Operators

Logical operators are used when we want to combine operations and expressions into a set of logical comparisons. They are usually used in conjuction with "if and "else" statements to lay out the dynamic logic we need in many situations as we advance in our PHP development.

The following table shows their usage and result logic:

Name	Usage	Result
And	$var1 && $var2	**TRUE** if both $var1 and $var2 are **TRUE**
And	$var1 and $var2	**TRUE** if both $var1 and $var2 are **TRUE**
Or	$var1 or $var2	**TRUE** if either $var1 or $var2 is **TRUE**
Or	$var1 || $var2	**TRUE** if either $var1 or $var2 is **TRUE**
Xor	$var1 xor $var2	**TRUE** if either $var1 or $var2 is **TRUE**, but not both
Is Not	!$var1	**TRUE** if $var1 is not **TRUE**

There are two different variations of "and" and "or" operators because they operate at different precedences.

Here we use an if and else statement(discussed in detail later) to use the "&&" logical operator:

<Code>
```
<?php
// change these 2 values to see the results if numbers do not match in
our expressions
$var1 = 43;
$var2 = 56;

if (($var1 == 43) && ($var2 == 56)) {
    echo "Yes, the values produce the match we want.";
} else {
    echo "No, the values do not match yet.";
}
?>
```

Yes, the values produce the match we want.

Concatenation Assignment Operators (String)

The concatenation operator (.) returns the combined value of its right and left values. The variable's data type has an affect on the output.

Here are code examples using the concatenation operator (.) to unite or combine variable values


```
<?php
echo "deve"."lop";          //displays "develop"
echo "<br /><br />";           // html line breaks

echo "deve" . "lop";        //displays "develop"
echo "<br /><br />";           // html line breaks

echo 4 . 3;                    //displays "43"
echo "<br /><br />";           // html line breaks

echo 4.3;                      //displays "4.3"
echo "<br /><br />";           // html line breaks

echo "4" . "3";               //displays "43"
echo "<br /><br />";           // html line breaks

echo '4' . '3';                 //displays "43"
?>
```

The concatenating assignment operator (.=), which appends the variable value on the right side to the variable on the left side. We use this method many times to compound and keep adding to one variable so it will retain its current value, and just append the new values onto the current value's tail end.

Here is a code example using the concatenating assignment operator (.=) to append or compound values


```
<?php
$var1 = "Hi World!";
$var2 = "We are not silly!";
$htmlOutput = ''; // set as empty, but defined

// notice the period before each ( = ) sign... that is concatenating th
em onto the original var value
$htmlOutput .= '<table bgcolor="#663366" cellpadding="8">';
$htmlOutput .= '<tr>';
$htmlOutput .= '<td bgcolor="#CCCCCC">';
$htmlOutput .= ' ' . $var1 . ' ';
$htmlOutput .= '</td>';
```

```
$htmlOutput .= '<td bgcolor="#FFFF00">';
$htmlOutput .= ' ' . $var2 . ' ';
$htmlOutput .= '</td>';
$htmlOutput .= '</tr>';
$htmlOutput .= "</table>";
// Now print or echo the output
echo "$htmlOutput ";
?>
```

PHP Array Operators

The PHP array operators are used to compare arrays:

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y (but duplicate keys are not overwritten)
==	Equality	$x == $y	True if $x and $y have the same key/value pairs
===	Identity	$x === $y	True if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	True if $x is not equal to $y
<>	Inequality	$x <> $y	True if $x is not equal to $y
!==	Non-identity	$x !== $y	True if $x is not identical to $y

The example below shows the different results of using the different array operators:

Example

```
<?php
$x = array("a" => "red", "b" => "green");
$y = array("c" => "blue", "d" => "yellow");
$z = $x + $y; // union of $x and $y
var_dump($z);
var_dump($x == $y);
var_dump($x === $y);
var_dump($x != $y);
var_dump($x <> $y);
var_dump($x !== $y);
?>
```

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

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

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

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 good weekend!" if the current day is Friday, otherwise it will output "Have a good day!":

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a good weekend!";
else
  echo "Have a good 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 "Hi!<br />";
```

```
  echo "Have a good weekend!";
  echo "See you on Monday!";
  }
?>
</body>
</html>
```

The ElseIf Statement

If you want to execute some code if one of several conditions are true use the elseif statement

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

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

PHP - 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;
}

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

```
<?php
$t=date("H");

if ($t<"10") {
  echo "Have a good morning!";
} elseif ($t<"20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>
```

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax

```
switch (expression)
{
case label1:
  code to be executed if expression = label1;
  break;
case label2:
  code to be executed if expression = label2;
  break;
default:
  code to be executed
  if expression is different
  from both label1 and label2;
}
```

Example

The *switch* statement works in an unusual way. First it evaluates given expression then seeks a lable to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the lables match then statement will will execute any specified default code.

```
<html>
<body>
<?php
$d=date("D");
switch ($d)
{
case "Mon":
  echo "Today is Monday";
  break;
case "Tue":
  echo "Today is Tuesday";
  break;
```

```
case "Wed":
  echo "Today is Wednesday";
  break;
case "Thu":
  echo "Today is Thursday";
  break;
case "Fri":
  echo "Today is Friday";
  break;
case "Sat":
  echo "Today is Saturday";
  break;
case "Sun":
  echo "Today is Sunday";
  break;
default:
  echo "Wonder which day is this ?";
}
?>
</body>
</html>
```

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 code-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 as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the 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 for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

Syntax

```
for (initialization; condition; increment)
{
  code to be executed;
}
```

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop:

```
<html>
<body>
<?php
$a = 0;
$b = 0;

for( $i=0; $i<5; $i++ )
{
    $a += 10;
    $b += 5;
}
echo ("At the end of the loop a=$a and b=$b" );
?>
</body>
</html>
```

This will produce following result:

```
At the end of the loop a=50 and b=25
```

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true.

If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

Syntax

```
while (condition)
{
    code to be executed;
}
```

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

```
<html>
<body>
<?php
$i = 0;
$num = 50;

while( $i < 10)
{
   $num--;
   $i++;
```

```
}
echo ("Loop stopped at i = $i and num = $num" );
?>
</body>
</html>
```

This will produce following result:

```
Loop stopped at i = 10 and num = 40
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

```
do
{
   code to be executed;
}while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10:

```
<html>
<body>
<?php
$i = 0;
$num = 0;
do
{
  $i++;
}while( $i < 10 );
echo ("Loop stopped at i = $i" );
?>
</body>
</html>
```

This will produce following result:

```
Loop stopped at i = 10
```

The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

```
foreach (array as value)
{
    code to be executed;

}
```

Example

Try out following example to list out the values of an array.

```
<html>
<body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
  echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce following result:

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.

The **break** statement is situated inside the statement block. If gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.

Example

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>
<body>

<?php
$i = 0;

while( $i < 10)
{
   $i++;
   if( $i == 3 )break;
```

```
}
echo ("Loop stopped at i = $i" );
?>
</body>
</html>
```

This will produce following result:

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.

Example

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```
<html>
<body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
  if( $value == 3 )continue;
  echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce following result

```
Value is 1
Value is 2
Value is 4
Value is 5
```

PHP Functions

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you:

- Creating a PHP Function
- Calling a PHP Function

The real power of PHP comes from its functions; it has more than 1000 built-in functions.

Creating Your First PHP Function

When you create a function, you first need to give it a name, like *myCompanyMotto*. It's with this function name that you will be able to call upon your function, so make it easy to type and understand.

The actual syntax for creating a function is pretty self-explanatory, but you can be the judge of that. First, you must tell PHP that you want to create a function. You do this by typing the keyword *function* followed by your function name and some other stuff (which we'll talk about later).

Here is how you would make a function called *myCompanyMotto*. **Note**: We still have to fill in the code for *myCompanyMotto*.

PHP Code:

```
<?php
function myCompanyMotto(){
}
?>
```

Note: Your function name can start with a letter or underscore "_", but **not** a number!

With a properly formatted function in place, we can now fill in the code that we want our function to execute. Do you see the curly braces in the above example "{ }"? These braces define where our function's code goes. The opening curly brace "{" tells php that the function's code is starting and a closing curly brace "}" tells PHP that our function is done!

We want our function to print out the company motto each time it's called, so that sounds like it's a job for the echo command!

PHP Code:

```
<?php
function myCompanyMotto(){
    echo "We deliver quantity, not quality!<br />";
}
?>
```

That's it! You have written your first PHP function from scratch! Notice that the code that appears within a function is just the same as any other PHP code.

Using Your PHP Function

Now that you have completed coding your PHP function, it's time to put it through a test run. Below is a simple PHP script. Let's do two things: add the function code to it and use the function twice.

PHP Code:

```
<?php
echo "Welcome to Tizag.com <br />";
echo "Well, thanks for stopping by! <br />";
echo "and remember... <br />";
?>
```

PHP Code with Function:

```
<?php
function myCompanyMotto(){
    echo "We deliver quantity, not quality!<br />";
}
echo "Welcome to Tizag.com <br />";
myCompanyMotto();
echo "Well, thanks for stopping by! <br />";
echo "and remember... <br />";
myCompanyMotto();
?>
```

Display:

Welcome to Tizag.com
We deliver quantity, not quality!
Well, thanks for stopping by!
and remember...
We deliver quantity, not quality!

Although this was a simple example, it's important to understand that there is a lot going on and there are a lot of areas to make errors. When you are creating a function, follow these simple guidelines:

- Always start your function with the keyword **function**
- Remember that your function's code must be between the "{" and the "}"
- When you are using your function, be sure you spell the function name correctly
- Don't give up!

PHP Functions - Parameters

Another useful thing about functions is that you can send them information that the function can then use. Our first function *myCompanyMotto* isn't all that useful because all it does, and ever will do, is print out a single, unchanging string.

However, if we were to use parameters, then we would be able to add some extra functionality! A parameter appears with the parentheses "( )" and looks just like a normal PHP variable. Let's create a new function that creates a custom greeting based off of a person's name.

Our parameter will be the person's name and our function will concatenate this name onto a greeting string. Here's what the code would look like.

PHP Code with Function:

```
<?php
function myGreeting($firstName){
    echo "Hi there ". $firstName . "!<br />";
}
?>
```

When we use our *myGreeting* function we have to send it a string containing someone's name, otherwise it will break. When you add parameters, you also add more responsibility to you, the programmer! Let's call our new function a few times with some common first names.

PHP Code:

```
<?php
function myGreeting($firstName){
    echo "Hi there ". $firstName . "!<br />";
}
myGreeting("Jatin");
myGreeting("Tejali");
myGreeting("Pardeep");
myGreeting("Madhav");
?>
```

Display:

Hi there Jatin!
Hi there Tejali!
Hi there Pardeep!
Hi there Madhav!

It is also possible to have multiple parameters in a function. To separate multiple parameters PHP uses a comma ",". Let's modify our function to also include last names.

PHP Code:

```
<?php
function myGreeting($firstName, $lastName){
    echo "Hi there ". $firstName ." ". $lastName ."!<br />";
}
myGreeting("Jatin", "Bedi");
myGreeting("Tejali", "Kaur");
myGreeting("Pardeep", "Gill");
myGreeting("Madhav", "Bedi");
?>
```

Display:

Hi there Jatin Bedi!
Hi there Tejali Kaur!
Hi there Pardeep Gill!
Hi there Madhav Bedi!

Creating PHP Function:

Its very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below:

```
<html>
<head>
<title>Writing PHP Function</title>
</head>
<body>

<?php
/* Defining a PHP Function */
function writeMessage()
{
  echo "You are really a good person, Have a good time!";
}
/* Calling a PHP Function */
writeMessage();
?>
</body>
</html>
```

This will display following result:

```
You are really a good person, Have a good time!
```

PHP Functions with Parameters:

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

```
<html>
<head>
<title>Writing PHP Function with Parameters</title>
</head>
<body>

<?php
function addFunction($num1, $num2)
{
  $sum = $num1 + $num2;
  echo "Sum of the two numbers is : $sum";
}
addFunction(15, 18);
?>
</body>
</html>
```

This will display following result:

```
Sum of the two numbers is : 33
```

Passing Arguments by Reference:

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Example

```
<html>
<head>
<title>Passing Argument by Reference</title>
</head>
<body>
<?php
function addFive($num)
{
   $num += 5;
}

function addSix(&$num)
{
   $num += 6;
}
$orignum = 10;
addFive( &$orignum );
echo "Original Value is $orignum<br />";
addSix( $orignum );
echo "Original Value is $orignum<br />";
?>
</body>
</html>
```

This will display following result:

```
Original Value is 15
Original Value is 21
```

PHP Functions - Returning Values

Besides being able to pass functions information, you can also have them return a value. However, a function can only return one thing, although that thing can be any integer, float, array, string, etc. that you choose!

How does it return a value though? Well, when the function is used and finishes executing, it sort of changes from being a function name into being a value. To capture this value you can set a variable equal to the function. Something like:

- $myVar = somefunction();

Let's demonstrate this returning of a value by using a simple function that returns the sum of two integers.

PHP Code:

```
<?php
function mySum($numX, $numY){
    $total = $numX + $numY;
    return $total;
}
$myNumber = 0;
echo "Before the function, myNumber = ". $myNumber ."<br />";
$myNumber = mySum(3, 4); // Store the result of mySum in $myNumber
echo "After the function, myNumber = ". $myNumber ."<br />";
?>
```

Display:

Before the function, myNumber = 0
After the function, myNumber = 7

When we first print out the value of $myNumber it is still set to the original value of 0. However, when we set $myNumber equal to the function *mySum,* $myNumber is set equal to mySum's result. In this case, the result was 3 + 4 = 7, which was successfully stored into $myNumber and displayed in the second echo statement!

Dynamic Function Calls:

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. Following example depicts this behaviour.

```
<html>
<head>
<title>Dynamic Function Calls</title>
</head>
<body>
<?php
function sayHi()
{
   echo "Hi<br />";
}
$function_holder = "sayHi";
$function_holder();
?>
</body>
</html>
```

This will display following result:

Hi

PHP Arrays

An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length. An array is a special variable, which can hold more than one value at a time.

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="Maruti";
$cars2="BMW";
$cars3="Sail";
```

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 solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- **Numeric array** - An array with a numeric index. Values are stored and accessed in linear fashion
- **Associative array** - An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- **Multidimensional array** - An array containing one or more arrays and values are accessed using multiple indices

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0):

```
$cars=array("Maruti","BMW","Sail");
```

or the index can be assigned manually:

```
$cars[0]="Maruti";
$cars[1]="BMW";
$cars[2]="Sail";
```

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example

```
<?php
$cars=array("Maruti","BMW","Sail");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

Get The Length of an Array - The count() Function

The count() function is used to return the length (the number of elements) of an array:

Example

```
<?php
$cars=array("Maruti","BMW","Sail");
echo count($cars);
?>
```

Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a for loop, like this:

Example

```
<?php
$cars=array("Maruti","BMW","Sail");
$arrlength=count($cars);

for($x=0;$x<$arrlength;$x++) {
  echo $cars[$x];
  echo "<br>";
}
?>
```

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.
There are two ways to create an associative array:

```
$age=array("Jatin"=>"35","Rozy"=>"37","Gagan"=>"43");
```

or:

```
$age['Jatin']="35";
$age['Rozy']="37";
$age['Gagan']="43";
```

The named keys can then be used in a script:

Example

```
<?php
$age=array("Jatin"=>"35","Rozy"=>"37","Gagan"=>"43");
echo "Jatin is " . $age['Jatin'] . " years old.";
?>
```

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

Example

```
<?php
$age=array("Jatin"=>"35","Rozy"=>"37","Gagan"=>"43");

foreach($age as $x=>$x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>
```

Multidimensional Arrays

A multi-dimensional 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. Values in the multi-dimensional array are accessed using multiple index.

Example

In this example we create a two dimensional array to store marks of three students in three subjects:

This example is an associative array, you can create numeric array in the same fashion.

```
<html>
<body>
<?php
   $marks = array(
            "jatin" => array
            (
            "english" => 35,
            "maths" => 30,
            "hindi" => 39
            ),
            "yadev" => array
               (
               "english" => 30,
               "maths" => 32,
               "hindi" => 29
               ),
               "hema" => array
               (
               "english" => 31,
               "maths" => 22,
               "hindi" => 39
               )
         );
   /* Accessing multi-dimensional array values */
```

```
   echo "Marks for jatin in english : " ;
   echo $marks['jatin']['english'] . "<br />";
   echo "Marks for yadev in maths : ";
   echo $marks['yadev']['maths'] . "<br />";
   echo "Marks for hema in hindi : " ;
   echo $marks['hema']['hindi'] . "<br />";
?>
</body>
</html>
```

This will produce following result:

```
Marks for jatin in english : 35
Marks for yadev in maths : 32
Marks for hema in hindi : 39
```

Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key

Sort Array in Ascending Order - sort()

The following example sorts the elements of the $cars array in ascending alphabetical order:

Example

```
<?php
$cars=array("Maruti","BMW","Sail");
sort($cars);
?>
```

The following example sorts the elements of the $numbers array in ascending numerical order:

Example

```
<?php
$numbers=array(4,6,2,22,11);
sort($numbers);
?>
```

Sort Array in Descending Order - rsort()

The following example sorts the elements of the $cars array in descending alphabetical order:

Example

```
<?php
$cars=array("Maruti","BMW","Sail");
rsort($cars);
?>
```

The following example sorts the elements of the $numbers array in descending numerical order:

Example

```
<?php
$numbers=array(4,6,2,22,11);
rsort($numbers);
?>
```

Sort Array in Ascending Order, According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

Example

```
<?php
$age=array("Jatin"=>"35","Rozy"=>"37","Gagan"=>"43");
asort($age);
?>
```

Sort Array in Ascending Order, According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

Example

```
<?php
$age=array("Jatin"=>"35","Rozy"=>"37","Gagan"=>"43");
ksort($age);
?>
```

Sort Array in Descending Order, According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

Example

```
<?php
$age=array("Jatin"=>"35","Rozy"=>"37","Gagan"=>"43");
arsort($age);
?>
```

Sort Array in Descending Order, According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example

```
<?php
$age=array("Jatin"=>"35","Rozy"=>"37","Gagan"=>"43");
krsort($age);
?>
```

PHP File Handling

File handling is an important part of any web application. You often need to open and process a file for different tasks. With PHP, you can access the server's filesystem. This allows you to manipulate folders and text files in PHP scripts.

For example, you can use PHP to read or write a text file. Or you can list all files in a specified folder. There are many possibilities and PHP can save you lots of tedious work.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

The PHP code to read the file and write it to the output buffer is as follows (the readfile() function returns the number of bytes read on success):

Example

```
<?php
echo readfile("webdictionary.txt");
?>
```

The readfile() function is useful if all you want to do is open up a file and read its contents.

PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.
We will use the text file, "webdictionary.txt", during the lessons:
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets

HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

Tip: The fread() and the fclose() functions will be explained below.

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

Modes	Description
r	**Open a file for read only**. File pointer starts at the beginning of the file
w	**Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	**Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	**Creates a new file for write only**. Returns FALSE and an error if file already exists
r+	**Open a file for read/write**. File pointer starts at the beginning of the file
w+	**Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	**Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	**Creates a new file for read/write**. Returns FALSE and an error if file already exists

PHP Read File - fread()

The fread() function reads from an open file.
The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.
The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP Close File - fclose()

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

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
$myfile = fopen("webdictionary.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

PHP Read Single Line - fgets()

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

The example below outputs the first line of the "webdictionary.txt" file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

Note: After a call to the fgets() function, the file pointer has moved to the next line.

PHP Check End-Of-File - feof()

The feof() function checks if the "end-of-file" (EOF) has been reached.
The feof() function is useful for looping through data of unknown length.
The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

PHP Read Single Character - fgetc()

The fgetc() function is used to read a single character from a file.
The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
```

```
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

Note: After a call to the fgetc() function, the file pointer moves to the next character.

PHP - How to Create a File

The *fopen* function needs two important pieces of information to operate correctly. First, we must supply it with the name of the file that we want it to open. Secondly, we must tell the function what we plan on doing with that file (i.e. read from the file, write information, etc).

Since we want to create a file, we must supply a file name and tell PHP that we want to write to the file. Note: We have to tell PHP we are writing to the file, otherwise it will not create a new file.

PHP Code:

```
$ourFileName = "testFile.txt";
$ourFileHandle = fopen($ourFileName, 'w') or die("can't open file");
fclose($ourFileHandle);
```

The file "testFile.txt" should be created in the same directory where this PHP code resides. PHP will see that "testFile.txt" does not exist and will create it after running this code. There's a lot of information in those three lines of code, let's make sure you understand it.

1. **$ourFileName = "testFile.txt";**

   Here we create the name of our file, "testFile.txt" and store it into a PHP String variable *$ourFileName*.

2. **$ourFileHandle = fopen($ourFileName, 'w')** or die("can't open file");

This bit of code actually has two parts. First we use the function *fopen* and give it two arguments: our file name and we inform PHP that we want to write by passing the character "w".
Second, the *fopen* function returns what is called a *file handle*, which will allow us to manipulate the file. We save the file handle into the *$ourFileHandle* variable. We will talk more about file handles later on.

3. **fclose($ourFileHandle);**

   We close the file that was opened. *fclose* takes the file handle that is to be closed. We will talk more about this more in the file closing lesson.

PHP - Permissions

If you are trying to get this program to run and you are having errors, you might want to check that you have granted your PHP file access to write information to the hard drive. Setting permissions is most often done with the use of an FTP program to execute a command called *CHMOD*. Use CHMOD to allow the PHP file to write to disk, thus allowing it to create a file.

Opening and Closing Files

The PHP **fopen()** function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

Mode	Purpose
r	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
w	Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attemts to create a file.
w+	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attemts to create a file.
a	Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attemts to create a file.
a+	Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attemts to create a file.

If an attempt to open a file fails then **fopen** returns a value of **false** otherwise it returns a **file pointer** which is used for further reading or writing to that file.

After making a changes to the opened file it is important to close it with the **fclose()** function. The **fclose()** function requires a file pointer as its argument and then returns **true** when the closure succeeds or **false** if it fails.

Reading a file

Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The files's length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes.

So here are the steps required to read a file with PHP.

- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.

- Read the file's content using **fread()** function.
- Close the file with **fclose()** function.

The following example assigns the content of a text file to a variable then displays those contents on the web page.

```
<html>
<head>
<title>Reading a file using PHP</title>
</head>
<body>

<?php
$filename = "/home/user/guest/tmp.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
   echo ( "Error in opening file" );
   exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );

fclose( $file );

echo ( "File size : $filesize bytes" );
echo ( "<pre>$filetext</pre>" );
?>

</body>
</html>
```

Writing a file

A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing would will stop after the specified length has been reached.

The following example creates a new text file then writes a short text heading insite it. After closing this file its existence is confirmed using **file_exist()** function which takes file name as an argument

```
<?php
$filename = "/home/user/guest/newfile.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
   echo ( "Error in opening new file" );
   exit();
}
fwrite( $file, "This is  a simple test\n" );
fclose( $file );
?>
```

```
<html>
<head>
<title>Writing a file using PHP</title>
</head>
<body>

<?php
if( file_exist( $filename ) )
{
   $filesize = filesize( $filename );
   $msg = "File  created with name $filename ";
   $msg .= "containing $filesize bytes";
   echo ($msg );
}
else
{
   echo ("File $filename does not exit" );
}
?>
</body>
</html>
```

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, previous data vanished, and only the data we just wrote is present:

Mickey Mouse
Minnie Mouse

A simple link directory

The file is systematically written with the name of the program, then a comma, and then the domain. As you can probably imagine, more information could easily be stored in this comma-separated data file.

To get the information in each line, we use an array.

```
<html>
<head>
<title>Reading from text files</title>

</head>
<body>

<?php
$f = fopen("unitednations.txt", "r");

// Read line by line until end of file
while (!feof($f)) {

// Make an array using comma as delimiter
   $arrM = explode(",",fgets($f));

// Write links (get the data in the array)
   echo "<li><a href='http://" . $arrM[1] . "'>" . $arrM[0].
"</a></li>";

}

fclose($f);
?>

</body>
</html>
```

Quite handy, right? In principle, you could now just expand the text file with hundreds of links or perhaps expand your directory to also include address information.

PHP - File Close Function

We had a call to the function *fclose* to close down a file after we were done with it. Here we will repeat that example and discuss the importance of closing a file.

PHP Code:

```
$ourFileName = "testFile.txt";
$ourFileHandle = fopen($ourFileName, 'w') or die("can't open file");
fclose($ourFileHandle);
```

The function *fclose* requires the file handle that we want to close down. In our example we set our variable "$fileHandle" equal to the file handle returned by the *fopen* function.

After a file has been closed down with *fclose* it is impossible to read, write or append to that file unless it is once more opened up with the *fopen* function.

PHP - File Delete

In PHP you delete files by calling the *unlink* function.

PHP - File Unlink

When you view the contents of a directory you can see all the files that exist in that directory because the operating system or application that you are using displays a list of filenames. You can think of these filenames as links that join the files to the directory you are currently viewing.

If you unlink a file, you are effectively causing the system to forget about it or delete it!

Before you can delete (unlink) a file, you must first be sure that it is not open in your program. Use the *fclose* function to close down an open file.

PHP - Unlink Function

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'w') or die("can't open file");
fclose($fh);
```

Now to delete *testFile.txt* we simply run a PHP script that is located in the same directory. Unlink just needs to know the name of the file to start working its destructive magic.

PHP Code:

```
$myFile = "testFile.txt";
unlink($myFile);
```

The *testFile.txt* should now be removed.

PHP - File Append

If you want to *append* to a file, that is, add on to the existing data, then you need to open the file in append mode.

PHP - File Open: Append

If we want to add on to a file we need to open it up in append mode. The code below does just that.

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'a');
```

If we were to write to the file it would begin writing data at the end of the file.

PHP - File Write: Appending Data

Using the *testFile.txt* file we created in the File Write lesson , we are going to append on some more data.

PHP Code:

```
$myFile = "testFile.txt";
$fh = fopen($myFile, 'a') or die("can't open file");
$stringData = "New Stuff 1\n";
fwrite($fh, $stringData);
$stringData = "New Stuff 2\n";
fwrite($fh, $stringData);
fclose($fh);
```

You should noticed that the way we write data to the file is exactly the same as in the Write lesson. The only thing that is different is that the file pointer is placed at the end of the file in append mode, so all data is added to the end of the file.

The contents of the file *testFile.txt* would now look like this:

Contents of the testFile.txt File:

Floppy Jalopy
Pointy Pinto
New Stuff 1
New Stuff 2

PHP - Append: Why Use It?

The above example may not seem very useful, but appending data onto a file is actually used everyday. Almost all web servers have a *log* of some sort. These various logs keep track of all kinds of information, such as: errors, visitors, and even files that are installed on the machine.

A log is basically used to document events that occur over a period of time, rather than all at once. Logs: a perfect use for append!

PHP File Upload

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script. Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload_tmp_dir** and the maximum permitted size of files that can be uploaded is stated as **upload_max_filesize**. These parameters are set into PHP configuration file **php.ini**

The process of uploading a file follows these steps

- The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
- The user clicks the browse button and selects a file to upload from the local PC.
- The full path to the selected file appears in the text filed then the user clicks the submit button.
- The selected file is sent to the temporary directory on the server.
- The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
- The PHP script confirms the success to the user.

As usual when writing files it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only then process will fail.

An uploaded file could be a text file or image file or any document.

Create an Upload-File Form

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

Look at the following HTML form for uploading files:

```
<html>
<body>

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

</body>
</html>
```

Notice the following about the HTML form above:

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

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

Create The Upload Script

The "upload_file.php" file contains the code for uploading a file:

```
<?php
if ($_FILES["file"]["error"] > 0) {
  echo "Error: " . $_FILES["file"]["error"] . "<br>";
} else {
  echo "Upload: " . $_FILES["file"]["name"] . "<br>";
  echo "Type: " . $_FILES["file"]["type"] . "<br>";
  echo "Size: " . ($_FILES["file"]["size"] / 1024) . " kB<br>";
  echo "Stored in: " . $_FILES["file"]["tmp_name"];
}
?>
```

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

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

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

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

Restrictions on Upload

In this script we add some restrictions to the file upload. The user may upload .gif, .jpeg, and .png files; and the file size must be under 20 kB:

```
<?php
$allowedExts = array("gif", "jpeg", "jpg", "png");
$temp = explode(".", $_FILES["file"]["name"]);
$extension = end($temp);

if ((($_FILES["file"]["type"] == "image/gif")
|| ($_FILES["file"]["type"] == "image/jpeg")
|| ($_FILES["file"]["type"] == "image/jpg")
|| ($_FILES["file"]["type"] == "image/pjpeg")
|| ($_FILES["file"]["type"] == "image/x-png")
|| ($_FILES["file"]["type"] == "image/png"))
&& ($_FILES["file"]["size"] < 20000)
&& in_array($extension, $allowedExts)) {
  if ($_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";
}
?>
```

Saving the Uploaded File

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

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

```
<?php
$allowedExts = array("gif", "jpeg", "jpg", "png");
$temp = explode(".", $_FILES["file"]["name"]);
```

```
$extension = end($temp);

if ((($_FILES["file"]["type"] == "image/gif")
|| ($_FILES["file"]["type"] == "image/jpeg")
|| ($_FILES["file"]["type"] == "image/jpg")
|| ($_FILES["file"]["type"] == "image/pjpeg")
|| ($_FILES["file"]["type"] == "image/x-png")
|| ($_FILES["file"]["type"] == "image/png"))
&& ($_FILES["file"]["size"] < 20000)
&& in_array($extension, $allowedExts)) {
  if ($_FILES["file"]["error"] > 0) {
    echo "Return Code: " . $_FILES["file"]["error"] . "<br>";
  } else {
    echo "Upload: " . $_FILES["file"]["name"] . "<br>";
    echo "Type: " . $_FILES["file"]["type"] . "<br>";
    echo "Size: " . ($_FILES["file"]["size"] / 1024) . " kB<br>";
    echo "Temp file: " . $_FILES["file"]["tmp_name"] . "<br>";
    if (file_exists("upload/" . $_FILES["file"]["name"])) {
      echo $_FILES["file"]["name"] . " already exists. ";
    } else {
      move_uploaded_file($_FILES["file"]["tmp_name"],
      "upload/" . $_FILES["file"]["name"]);
      echo "Stored in: " . "upload/" . $_FILES["file"]["name"];
    }
  }
} else {
  echo "Invalid file";
}
?>
```

The script above checks if the file already exists, if it does not, it copies the file to a folder called "upload".

PHP Cookies

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users:

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

Creating Your First PHP Cookie

When you create a cookie, using the function *setcookie*, you must specify three arguments. These arguments are ***setcookie****(name, value, expiration)*:

1. **name**: The name of your cookie. You will use this name to later retrieve your cookie, so don't forget it!
2. **value**: The value that is stored in your cookie. Common values are username(string) and last visit(date).
3. **expiration**: The date when the cookie will expire and be deleted. If you do not set this expiration date, then it will be treated as a session cookie and be removed when the browser is restarted.

In this example we will be creating a cookie that stores the user's last visit to measure how often people return to visit our webpage. We want to ignore people that take longer than two months to return to the site, so we will set the cookie's expiration date to two months in the future!

PHP Code:

```
<?php
//Calculate 60 days in the future
//seconds * minutes * hours * days + current time
$inTwoMonths = 60 * 60 * 24 * 60 + time();
setcookie('lastVisit', date("G:i - m/d/y"), $inTwoMonths);
?>
```

Note: The three important arguments: name, value and expiration date.

Setting Cookies with PHP:

PHP provided **setcookie()** function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

setcookie(name, value, expire, path, domain, security);

Here is the detail of all the arguments:

- **Name -** This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value -**This sets the value of the named variable and is the content that you actually want to store.
- **Expiry -** This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path -**This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain -** This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security -** This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies **name** and **age** these cookies will be expired after one hour.

```
<?php
   setcookie("name", "John Watkin", time()+3600, "/","", 0);
   setcookie("age", "36", time()+3600, "/", "",  0);
?>
<html>
```

```
<head>
<title>Setting Cookies with PHP</title>
</head>
<body>
<?php echo "Set Cookies"?>
</body>
</html>
```

Accessing Cookies with PHP

PHP provides many ways to access cookies.Simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables. Following example will access all the cookies set in above example.

```
<html>
<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>
<?php
echo $_COOKIE["name"]. "<br />";
/* is equivalent to */
echo $HTTP_COOKIE_VARS["name"]. "<br />";

echo $_COOKIE["age"] . "<br />";
/* is equivalent to */
echo $HTTP_COOKIE_VARS["name"] . "<br />";
?>
</body>
</html>
```

You can use **isset()** function to check if a cookie is set or not.

```
<html>
<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>
<?php
  if( isset($_COOKIE["name"]))
    echo "Welcome " . $_COOKIE["name"] . "<br />";
  else
    echo "Sorry... Not recognized" . "<br />";
?>
</body>
</html>
```

Deleting Cookie with PHP

Officially, to delete a cookie you should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired:

```
<?php
  setcookie( "name", "", time()- 60, "/","", 0);
  setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>
<head>
<title>Deleting Cookies with PHP</title>
</head>
<body>
<?php echo "Deleted Cookies" ?>
</body>
</html>
```

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
//retrieve session data
echo "Pageviews=". $_SESSION['views'];
?>

</body>
</html>
```

Output:

Pageviews=1

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

```
<?php
session_start();

if(isset($_SESSION['views']))
$_SESSION['views']=$_SESSION['views']+1;
else
$_SESSION['views']=1;
echo "Views=". $_SESSION['views'];
?>
```

Destroying a Session

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

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

```
<?php
session_start();
if(isset($_SESSION['views']))
  unset($_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 - Sending Emails

PHP must be configured correctly in the **php.ini** file with the details of how your system sends email. Open php.ini file available in **/etc/** directory and find the section headed **[mail function]**.

Windows users should ensure that two directives are supplied. The first is called SMTP that defines your email server address. The second is called sendmail_from which defines your own email address.

The configuration for Windows should look something like this:

```
[mail function]
; For Win32 only.
SMTP = smtp.secureserver.net

; For win32 only
sendmail_from = webmaster@veinstitution.com
```

Linux users simply need to let PHP know the location of their **sendmail** application. The path and any desired switches should be specified to the sendmail_path directive.

The configuration for Linux should look something like this:

```
[mail function]
; For Win32 only.
SMTP =

; For win32 only
sendmail_from =

; For Unix only
sendmail_path = /usr/sbin/sendmail -t -i
```

Now you are ready to go:

Sending plain text email:

PHP makes use of **mail()** function to send an email. This function requires three mandatory arguments that specify the recipient's email address, the subject of the the message and the actual message additionally there are other two optional parameters.

mail(to, subject, message, headers, parameters);

Here is the description for each parameters.

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

As soon as the mail function is called PHP will attempt to send the email then it will return true if successful or false if it is failed.
Multiple recipients can be specified as the first argument to the mail() function in a comma separated list.

Example:

Following example will send an HTML email message to xyz@somedomain.com. You can code this program in such a way that it should receive all content from the user and then it should send an email.

```
<html>
<head>
<title>Sending email using PHP</title>
</head>
<body>
<?php
   $to = "xyz@somedomain.com";
   $subject = "This is subject";
   $message = "This is simple text message.";
   $header = "From:abc@somedomain.com \r\n";
   $retval = mail ($to,$subject,$message,$header);
   if( $retval == true )
   {
      echo "Message sent successfully...";
   }
   else
   {
      echo "Message could not be sent...";
   }
?>
</body>
</html>
```

Sending HTML email:

When you send a text message using PHP then all the content will be treated as simple text. Even if you will include HTML tags in a text message, it will be displayed as simple text and HTML tags will not be formatted according to HTML syntax. But PHP provides option to send an HTML message as actual HTML message.

While sending an email message you can specify a Mime version, content type and character set to send an HTML email.

Example:

Following example will send an HTML email message to xyz@somedomain.com copying it to afgh@somedomain.com. You can code this program in such a way that it should recieve all content from the user and then it should send an email.

```
<html>
<head>
<title>Sending HTML email using PHP</title>
</head>
<body>
<?php
   $to = "xyz@somedomain.com";
   $subject = "This is subject";
   $message = "<b>This is HTML message.</b>";
   $message .= "<h1>This is headline.</h1>";
   $header = "From:abc@somedomain.com \r\n";
   $header = "Cc:afgh@somedomain.com \r\n";
   $header .= "MIME-Version: 1.0\r\n";
   $header .= "Content-type: text/html\r\n";
   $retval = mail ($to,$subject,$message,$header);
   if( $retval == true )
   {
      echo "Message sent successfully...";
   }
   else
   {
      echo "Message could not be sent...";
   }
?>
</body>
</html>
```

Sending attachments with email:

To send an email with mixed content requires to set **Content-type** header to **multipart/mixed**. Then text and attachment sections can be specified within **boundaries**.

A boundary is started with two hyphens followed by a unique number which can not appear in the message part of the email. A PHP function **md5()** is used to create a 32 digit hexadecimal number to create unique number. A final boundary denoting the email's final section must also end with two hyphens.

Attached files should be encoded with the **base64_encode()** function for safer transmission and are best split into chunks with the **chunk_split()** function. This adds **\r\n** inside the file at regular intervals, normally every 76 characters.

Following is the example which will send a file **/tmp/test.txt** as an attachment. you can code your program to receive an uploaded file and send it.

```
<html>
<head>
<title>Sending attachment using PHP</title>
</head>
<body>
<?php
  $to = "xyz@somedomain.com";
  $subject = "This is subject";
  $message = "This is test message.";
  # Open a file
  $file = fopen( "/tmp/test.txt", "r" );
  if( $file == false )
  {
     echo "Error in opening file";
     exit();
  }
  # Read the file into a variable
  $size = filesize("/tmp/test.txt");
  $content = fread( $file, $size);

  # encode the data for safe transit
  # and insert \r\n after every 76 chars.
  $encoded_content = chunk_split( base64_encode($content));

  # Get a random 32 bit number using time() as seed.
  $num = md5( time() );

  # Define the main headers.
  $header = "From:xyz@somedomain.com\r\n";
  $header .= "MIME-Version: 1.0\r\n";
  $header .= "Content-Type: multipart mixed; ";
  $header .= "boundary=$num\r\n";
  $header .= "--$num\r\n";

  # Define the message section
  $header .= "Content-Type: text/plain\r\n";
  $header .= "Content-Transfer-Encoding:8bit\r\n\n";
  $header .= "$message\r\n";
  $header .= "--$num\r\n";

  # Define the attachment section
  $header .= "Content-Type:  multipart/mixed; ";
  $header .= "name=\"test.txt\"\r\n";
  $header .= "Content-Transfer-Encoding:base64\r\n";
  $header .= "Content-Disposition:attachment; ";
  $header .= "filename=\"test.txt\"\r\n\n";
  $header .= "$encoded_content\r\n";
  $header .= "--$num--";

  # Send email now
  $retval = mail ( $to, $subject, "", $header );
  if( $retval == true )
   {
      echo "Message sent successfully...";
   }
   else
   {
      echo "Message could not be sent...";
   }
?>
</body>
</html>
```

PHP Secure E-mails :PHP Stopping E-mail Injections

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

Now we have added an input validator that checks the "from" field in the form:

```
<html>
<body>
<?php
function spamcheck($field) {
  // Sanitize e-mail address
  $field=filter_var($field, FILTER_SANITIZE_EMAIL);
  // Validate e-mail address
  if(filter_var($field, FILTER_VALIDATE_EMAIL)) {
    return TRUE;
  } else {
    return FALSE;
  }
}
?>

<h2>Feedback Form</h2>
<?php
// display form if user has not clicked submit
if (!isset($_POST["submit"])) {
  ?>
  <form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
  From: <input type="text" name="from"><br>
  Subject: <input type="text" name="subject"><br>
  Message: <textarea rows="10" cols="40" name="message"></textarea><br>
  <input type="submit" name="submit" value="Submit Feedback">
  </form>
  <?php
} else {  // the user has submitted the form
  // Check if the "from" input field is filled out
  if (isset($_POST["from"])) {
    // Check if "from" email address is valid
    $mailcheck = spamcheck($_POST["from"]);
    if ($mailcheck==FALSE) {
      echo "Invalid input";
    } else {
      $from = $_POST["from"]; // sender
      $subject = $_POST["subject"];
      $message = $_POST["message"];
      // message lines should not exceed 70 characters (PHP rule), so wrap it
      $message = wordwrap($message, 70);
      // send mail
      mail("webmaster@example.com",$subject,$message,"From: $from\n");
      echo "Thank you for sending us feedback";
    }
  }
```

```
}
?>
</body>
</html>
```

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

- The FILTER_SANITIZE_EMAIL filter removes all illegal e-mail characters from a string
- The FILTER_VALIDATE_EMAIL filter validates value as an e-mail address

PHP Error Handling

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

PHP Error Handling

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

Some of the most common error checking methods in PHP.

We will show different error handling methods:

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

Basic Error Handling: Using the die() function

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

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

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

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

To prevent the user from getting an error message like the one above, we test whether the file exist before we try to access it:

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

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

File not found
The code above is more efficient than the earlier code, because it uses a simple error handling mechanism to stop the script after the error.
However, simply stopping the script is not always the right way to go. Let's take a look at alternative PHP functions for handling errors.

Creating a Custom Error Handler

Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP.
This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

Syntax

error_function(error_level,error_message,
error_file,error_line,error_context)

Parameter	Description
error_level	Required. Specifies the error report level for the user-defined error. Must be a value number. See table below for possible error report levels
error_message	Required. Specifies the error message for the user-defined error
error_file	Optional. Specifies the filename in which the error occurred
error_line	Optional. Specifies the line number in which the error occurred
error_context	Optional. Specifies an array containing every variable, and their values, in use when the error occurred

Error Report levels

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

Value	Constant	Description
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())
8191	E_ALL	All errors and warnings (E_STRICT became a part of E_ALL in PHP 5.4)

Now lets create a function to handle errors:

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

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

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

Set Error Handler

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

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

```
set_error_handler("customError");
```

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

Example

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

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr";
}

//set error handler
set_error_handler("customError");

//trigger error
echo($test);
?>
```

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

Error: [8] Undefined variable: test

Trigger an Error

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

Example

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

```
<?php
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below");
}
?>
```

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

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

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

Possible error types:

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

Example

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

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
}

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

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

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

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

Error Logging

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

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

Send an Error Message by E-Mail

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

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Webmaster has been notified";
  error_log("Error: [$errno] $errstr",1,
  "someone@example.com","From: webmaster@example.com");
}

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

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

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

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

Error: [512] Value must be 1 or below

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

PHP Exception Handling

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

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

This is what normally happens when an exception is triggered:

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

We will show different error handling methods:

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

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

Basic Use of Exceptions

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

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

Lets try to throw an exception without catching it:

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

//trigger exception
checkNum(2);
?>
```

The code above will get an error like this:

Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in C:\webfolder\test.php:6

Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in **C:\webfolder\test.php** on line **6**

Throw and catch

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

Proper exception code should include:

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

Lets try to trigger an exception with valid code:

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

//trigger exception in a "try" block
try {
  checkNum(2);
  //If the exception is thrown, this text will not be shown
  echo 'If you see this, the number is 1 or below';
}

//catch exception
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();
}
?>
```

The code above will get an error like this:

Message: Value must be 1 or below

Example explained:

The code above throws an exception and catches it:

1. The checkNum() function is created. It checks if a number is greater than 1. If it is, an exception is thrown
2. The checkNum() function is called in a "try" block
3. The exception within the checkNum() function is thrown

4. The "catch" block retrives the exception and creates an object ($e) containing the exception information
5. The error message from the exception is echoed by calling $e->getMessage() from the exception object

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

Creating a Custom Exception Class

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

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

Lets create an exception class:

```
<?php
class customException extends Exception {
  public function errorMessage() {
    //error message
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
    return $errorMsg;
  }
}

$email = "someone@example...com";

try {
  //check if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    //throw exception if email is not valid
    throw new customException($email);
  }
}

catch (customException $e) {
  //display custom message
  echo $e->errorMessage();
}
?>
```

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

Example explained:

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

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

Multiple Exceptions

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

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

```
<?php
class customException extends Exception {
  public function errorMessage() {
    //error message
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
    return $errorMsg;
  }
}

$email = "someone@example.com";

try {
  //check if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    //throw exception if email is not valid
    throw new customException($email);
  }
  //check for "example" in mail address
  if(strpos($email, "example") !== FALSE) {
    throw new Exception("$email is an example e-mail");
  }
}

catch (customException $e) {
  echo $e->errorMessage();
}

catch(Exception $e) {
  echo $e->getMessage();
}
?>
```

Example explained:

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

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

If the exception thrown were of the class customException and there were no customException catch, only the base exception catch, the exception would be handled there.

Re-throwing Exceptions

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

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

```
<?php
class customException extends Exception {
  public function errorMessage() {
    //error message
    $errorMsg = $this->getMessage().' is not a valid E-Mail address.';
    return $errorMsg;
  }
}

$email = "someone@example.com";

try {
  try {
    //check for "example" in mail address
    if(strpos($email, "example") !== FALSE) {
      //throw exception if email is not valid
      throw new Exception($email);
    }
  }
  catch(Exception $e) {
    //re-throw exception
    throw new customException($email);
  }
}

catch (customException $e) {
  //display custom message
  echo $e->errorMessage();
}
?>
```

Example explained:

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

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

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

Set a Top Level Exception Handler

The set_exception_handler() function sets a user-defined function to handle all uncaught exceptions.

```
<?php
function myException($exception) {
  echo "<b>Exception:</b> " . $exception->getMessage();
}

set_exception_handler('myException');

throw new Exception('Uncaught Exception occurred');
?>
```

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

Exception: Uncaught Exception occurred

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

Rules for exceptions

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

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

PHP Filter

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

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

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

Why use a Filter?

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

You should always filter all external data!

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

External data

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

Functions and Filters

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

- filter_var() - Filters a single variable with a specified filter
- filter_var_array() - Filter several variables with the same or different filters
- filter_input - Get one input variable and filter it
- filter_input_array - Get several input variables and filter them with the same or different filters

In the example below, we validate an integer using the filter_var() function:

```
<?php
$int = 123;

if(!filter_var($int, FILTER_VALIDATE_INT)) {
  echo("Integer is not valid");
} else {
  echo("Integer is valid");
}
?>
```

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

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

Validating and Sanitizing

There are two kinds of filters:

Validating filters:

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

Sanitizing filters:

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

Options and Flags

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

In the example below, we validate an integer using the filter_var() and the "min_range" and "max_range" options:

```
<?php
$var=300;

$int_options = array(
"options"=>array
  (
  "min_range"=>0,
  "max_range"=>256
  )
);

if(!filter_var($var, FILTER_VALIDATE_INT, $int_options)) {
  echo("Integer is not valid");
} else {
  echo("Integer is valid");
}
?>
```

Like the code above, options must be put in an associative array with the name "options". If a flag is used it does not need to be in an array.
Since the integer is "300" it is not in the specified range, and the output of the code above will be: "Integer is not valid".

Validate Input

Let's try validating input from a form.

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

Then we filter the input data using the filter_input() function.

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

```
<?php
if(!filter_has_var(INPUT_GET, "email")) {
  echo("Input type does not exist");
} else {
  if (!filter_input(INPUT_GET, "email", FILTER_VALIDATE_EMAIL)) {
    echo "E-Mail is not valid";
  } else {
    echo "E-Mail is valid";
  }
}
?>
```

Example Explained

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

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

Sanitize Input

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

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

Then we sanitize the input data using the filter_input() function.

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

```
<?php
if(!filter_has_var(INPUT_POST, "url")) {
  echo("Input type does not exist");
} else {
  $url = filter_input(INPUT_POST,
  "url", FILTER_SANITIZE_URL);
}
?>
```

Example Explained

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

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

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

http://www.VEInstitution.com/

Filter Multiple Inputs

A form almost always consist of more than one input field. To avoid calling the filter_var or filter_input functions over and over, we can use the filter_var_array or the filter_input_array functions.

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

```
<?php
$filters = array
  (
  "name" => array
    (
    "filter"=>FILTER_SANITIZE_STRING
    ),
  "age" => array
    (
    "filter"=>FILTER_VALIDATE_INT,
    "options"=>array
      (
      "min_range"=>1,
      "max_range"=>120
      )
    ),
  "email"=> FILTER_VALIDATE_EMAIL
  );

$result = filter_input_array(INPUT_GET, $filters);

if (!$result["age"]) {
  echo("Age must be a number between 1 and 120.<br>");
} elseif(!$result["email"]) {
  echo("E-Mail is not valid.<br>");
} else {
  echo("User input is valid");
}
?>
```

Example Explained

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

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

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

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

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

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

Using Filter Callback

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

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

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

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

```
<?php
function convertSpace($string) {
  return str_replace("_", " ", $string);
}

$string = "Jatin_is_a_great_guy!";

echo filter_var($string, FILTER_CALLBACK,
array("options"=>"convertSpace"));
?>
```

The result from the code above should look like this:

Jatin is a great guy!

Example Explained

The example above converts all "_" to whitespaces:

1. Create a function to replace "_" to whitespaces
2. Call the filter_var() function with the FILTER_CALLBACK filter and an array containing our function

Databases

A database is a collection of information / data that is organized so that it can easily be retrieved, administrated and updated. Databases thereby enable the opportunity to create dynamic websites with large amounts of information. For example, all data on members of HTML.net and all posts in the forums are stored in databases.

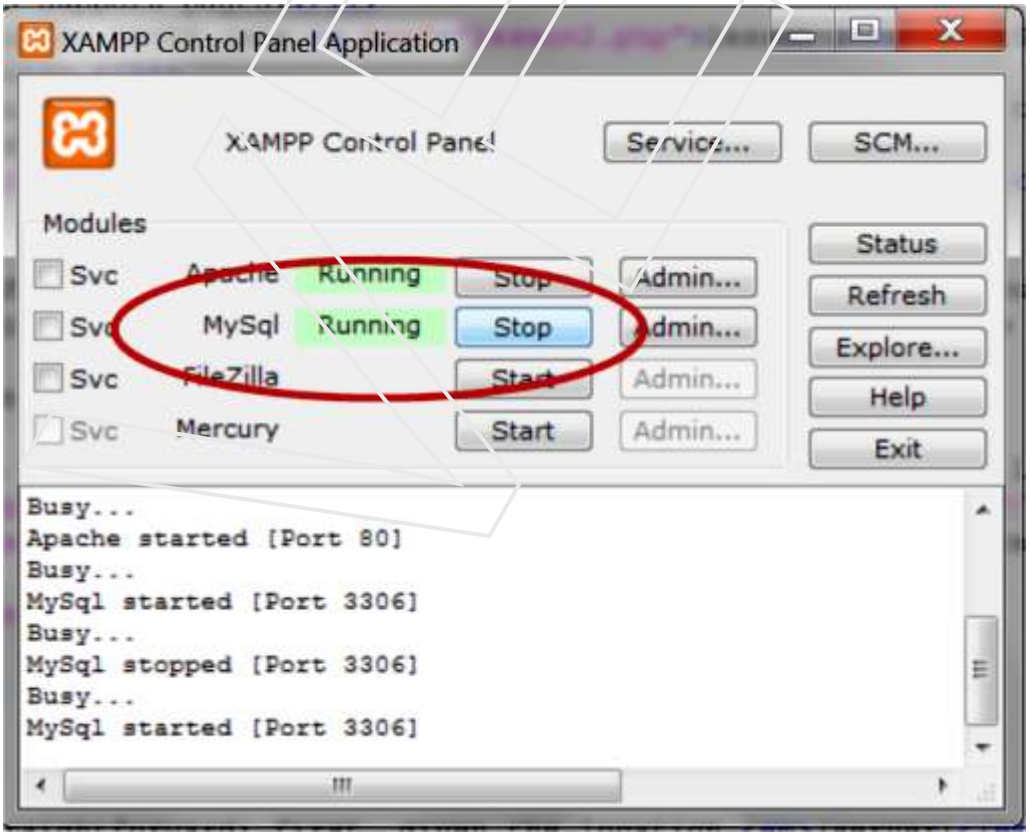
A database usually consists of one or more tables. If you are used to working with spreadsheets, or maybe have used databases before, tables will look familiar to you with columns and rows:

	id	FirstName	LastName	Phone	BirthDate
	22	Donald	Duck	33221100	1954-01-20
	23	Gladstone	Gander	44332211	1952-11-14
	24	Scrooge	McDuck	55443322	1935-03-06
	25	Grandma	Duck	66554433	1932-10-09
	26	Mickey	Mouse	77665544	1956-03-14
	27	Daisy	Duck	88776655	1959-04-28

There are many different databases: MySQL, MS Access, MS SQL Server, Oracle SQL Server and many others. We will use the MySQL database. MySQL is the natural place to start when you want to use databases in PHP.

You need to have access to MySQL in order to go through this lesson and the next lessons:

- If you have a hosted website with PHP, MySQL is probably already installed on the server. Read more at your web host's support pages.
- If you have installed PHP on your computer yourself and have the courage to install MySQL as well, it can be downloaded in a free version (MySQL Community Edition) at the MySQL's website.
- If you use XAMPP , MySQL is already installed and ready to use on your computer. Just make sure MySQL is running in the Control Panel:



In the rest of this lesson, we will look more closely at how you connect to your database server, before we learn to create databases and retrieve and update data in the following sessions.

Connection to database server

First, you need to have access to the server where your MySQL database is located. This is done with the function mysql_connect with the following syntax:

```
mysql_connect(server, username, password)
```

Pretty straightforward: First, you write the location of the database *(server)*, and then type in the *username* and *password*.

If you have your own website, you should read about location of your MySQL server on your host's support pages. Username and password will often be the same as those you use for FTP access. Otherwise contact your provider.

Example of a MySQL connection on a hosted website:

```
mysql_connect("mysql.myhost.com", "user001", "sesame") or
die(mysql_error());
```

Example of a MySQL connection with XAMPP (default settings):

```
mysql_connect("localhost", "root", "") or die (mysql_error());
```

In the examples are added `or die(mysql_error())` which, in brief, interrupts the script and writes the error if the connection fails.

Now we have made a connection to a MySQL server, and can now start creating databases and retrieve and insert data. This is exactly what we will look at in the next lessons.

By the way, keep in mind that it is good practice to close the database connection again when you're finished retrieving or updating data. This is done with the function mysql_close.

PHP MySQL Introduction

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL supports standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

The data in MySQL is stored in tables. A table is a collection of related data, and it consists of columns and rows.

Databases are useful when storing information categorically. A company may have a database with the following tables:

- Employees
- Products
- Customers
- Orders

PHP + MySQL

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

Queries

A query is a question or a request.
We can query a database for specific information and have a recordset returned.
Look at the following query (using standard SQL):
SELECT LastName FROM Employees
The query above selects all the data in the "LastName" column from the "Employees" table.

Download MySQL Database

If you don't have a PHP server with a MySQL Database, you can download MySQL for free here:
http://www.mysql.com

Facts About MySQL Database

One great thing about MySQL is that it can be scaled down to support embedded database applications. Maybe it is because of this many people think that MySQL can only handle small and medium-sized systems.

Open a Connection to the MySQL Server

Before we can access data in a database, we must open a connection to the MySQL server.

In PHP, this is done with the mysqli_connect() function.

Syntax

mysqli_connect(host,username,password,dbname);

Parameter	Description
host	Optional. Either a host name or an IP address
username	Optional. The MySQL user name
password	Optional. The password to log in with
dbname	Optional. The default database to be used when performing queries

Note: There are more available parameters, but the ones listed above are the most important.

In the following example we store the connection in a variable ($con) for later use in the script:

```
<?php
// Create connection
```

```
$con=mysqli_connect("example.com","jatin","abc123","my_db");

// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
?>
```

Close a Connection

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

```
<?php
$con=mysqli_connect("example.com","jatin","abc123","my_db");

// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

mysqli_close($con);
?>
```

Create databases and tables

We'll look at two ways to create databases and tables. First, how it is done in PHP, and then how it's made with the more user-friendly tool PhpMyAdmin, which is standard on most web hosts and in XAMPP.

If you have a hosted website with PHP and MySQL, a database has probably been created for you already and you can just skip this part of the lesson and start creating tables. Again, you should consult your host's support pages for more information.

Create a database and tables with PHP

The function mysql_query are used to send a query to a MySQL database. The queries are written in the language **S**tructured **Q**uery **L**anguage (SQL). SQL is the most widely used language for database queries - not only for MySQL databases - and is very logical and easy to learn. In this lesson and the next, you will learn the most important SQL queries.

When creating a database, the SQL query CREATE DATABASE is used with the following syntax:

```
CREATE DATABASE database name
```

Logical and easy, right!? Let's try to put it into a PHP script:

```
mysql_connect("mysql.myhost.com", "user", "sesame") or
die(mysql_error());

mysql_query("CREATE DATABASE mydatabase") or die(mysql_error());

mysql_close();
```

First, we connect to the MySQL server. Next, we create a database named "mydatabase". And finally, we close the connection to the MySQL server again.

So far so good... but things become a little bit more complicated when we want create tables in PHP. When creating tables, we use the SQL query CREATE TABLE with the following syntax:

```
CREATE TABLE table name
(
column1_name DATA_TYPE,
column2_name DATA_TYPE,
column3_name DATA_TYPE,
...
)
```

table_name and *column_name* are of course the name of the table and the columns, respectively. *DATA_TYPE* are used to specify the data type to be inserted into the column.

The following example creates a database named "my_db":

```
<?php
$con=mysqli_connect("example.com","jatin","abc123");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Create database
$sql="CREATE DATABASE my_db";
if (mysqli_query($con,$sql)) {
  echo "Database my_db created successfully";
} else {
  echo "Error creating database: " . mysqli_error($con);
}
?>
```

Create a Table

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

We must add the CREATE TABLE statement to the mysqli_query() function to execute the command.

The following example creates a table named "Persons", with three columns: "FirstName", "LastName" and "Age":

```
<?php
$con=mysqli_connect("example.com","jatin","abc123","my_db");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

// Create table
$sql="CREATE TABLE Persons(FirstName CHAR(30),LastName CHAR(30),Age INT)";

// Execute query
if (mysqli_query($con,$sql)) {
  echo "Table persons created successfully";
} else {
  echo "Error creating table: " . mysqli_error($con);
}
?>
```

Note: When you create a field of type CHAR, you must specify the maximum length of the field, e.g. CHAR(50).

The data type specifies what type of data the column can hold.

Primary Keys and Auto Increment Fields

Each table in a database should have a primary key field.

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

The following example sets the PID 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:

```
$sql = "CREATE TABLE Persons
(
PID INT NOT NULL AUTO_INCREMENT,
PRIMARY KEY(PID),
FirstName CHAR(15),
LastName CHAR(15),
Age INT
)";
```

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 mysqli_query() function. This function is used to send a query or command to a MySQL connection.

Example

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=mysqli_connect("example.com","jatin","abc123","my_db");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

mysqli_query($con,"INSERT INTO Persons (FirstName, LastName, Age)
VALUES ('Jatin', 'Bedi',35)");

mysqli_query($con,"INSERT INTO Persons (FirstName, LastName, Age)
VALUES ('Tejali', 'Kaur',33)");

mysqli_close($con);
?>
```

Insert Data From a Form Into a Database

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

Here is the HTML form:

```
<html>
<body>
```

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

</body>
</html>
```

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

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

The mysqli_real_escape_string() function escapes special characters in a string for security against SQL injection.

Then, the mysqli_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=mysqli_connect("example.com","jatin","abc123","my_db");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error(),
}

// escape variables for security
$firstname = mysqli_real_escape_string($con, $_POST['firstname']);
$lastname = mysqli_real_escape_string($con, $_POST['lastname']);
$age = mysqli_real_escape_string($con, $_POST['age']);

$sql="INSERT INTO Persons (FirstName, LastName, Age)
VALUES ('$firstname', '$lastname', '$age')";

if (!mysqli_query($con,$sql)) {
  die('Error: ' . mysqli_error($con));
}
echo "1 record added";

mysqli_close($con);
?>
```

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 mysqli_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=mysqli_connect("example.com","jatin","abc123","my_db");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$result = mysqli_query($con,"SELECT * FROM Persons");

while($row = mysqli_fetch_array($result)) {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br>";
}

mysqli_close($con);
?>
```

The example above stores the data returned by the mysqli_query() function in the $result variable. Next, we use the mysqli_fetch_array() function to return the first row from the recordset as an array. Each call to mysqli_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:

Jatin Bedi
Tejali Kaur

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=mysqli_connect("example.com","jatin","abc123","my_db");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
```

```
$result = mysqli_query($con,"SELECT * FROM Persons");

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>";

while($row = mysqli_fetch_array($result)) {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "</tr>";
}

echo "</table>";

mysqli_close($con);
?>
```

The output of the code above will be:

Firstname	Lastname
Tejali	Kaur
Jatin	Bedi

The WHERE clause

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

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

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

Example

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

```
<?php
$con=mysqli_connect("example.com","jatin","abc123","my_db");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$result = mysqli_query($con,"SELECT * FROM Persons
```

```
WHERE FirstName='Jatin'");

while($row = mysqli_fetch_array($result)) {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br>";
}
?>
```

The output of the code above will be:

Jatin Bedi

The ORDER BY Keyword

The ORDER BY keyword is used to sort the data in a recordset.
The ORDER BY keyword sort the records in ascending order by default.
If you want to sort the records in a descending order, you can use the DESC keyword.

Syntax

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

Example

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

```
<?php
$con=mysqli_connect("example.com","jatin","abc123","my_db");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$result = mysqli_query($con,"SELECT * FROM Persons ORDER BY age");

while($row = mysqli_fetch_array($result)) {
  echo $row['FirstName'];
  echo " " . $row['LastName'];
  echo " " . $row['Age'];
  echo "<br>";
}

mysqli_close($con);
?>
```

The output of the code above will be:

Tejali Kaur 33
Jatin Bedi 35

Order by Two Columns

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

```
SELECT column_name(s)
FROM table_name
ORDER BY column1, column2
```

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 mysqli_query() function. This function is used to send a query or command to a MySQL connection.

Result

FirstName	LastName	Age
Jatin	Bedi	35
Tejali	Kaur	33

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

```
<?php
$con=mysqli_connect("example.com","jatin","abc123","my_db");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

mysqli_query($con,"UPDATE Persons SET Age=36
WHERE FirstName='Jatin' AND LastName='Bedi'");

mysqli_close($con);
?>
```

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

FirstName	LastName	Age
Jatin	Bedi	36
Tejali	Kaur	33

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 mysqli_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
Jatin	Bedi	35
Tejali	Kaur	33

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

```
<?php
$con=mysqli_connect("example.com","jatin","abc123","my_db");
// Check connection
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

mysqli_query($con,"DELETE FROM Persons WHERE LastName='Bedi'");

mysqli_close($con);
?>
```

After the deletion, the table will look like this:

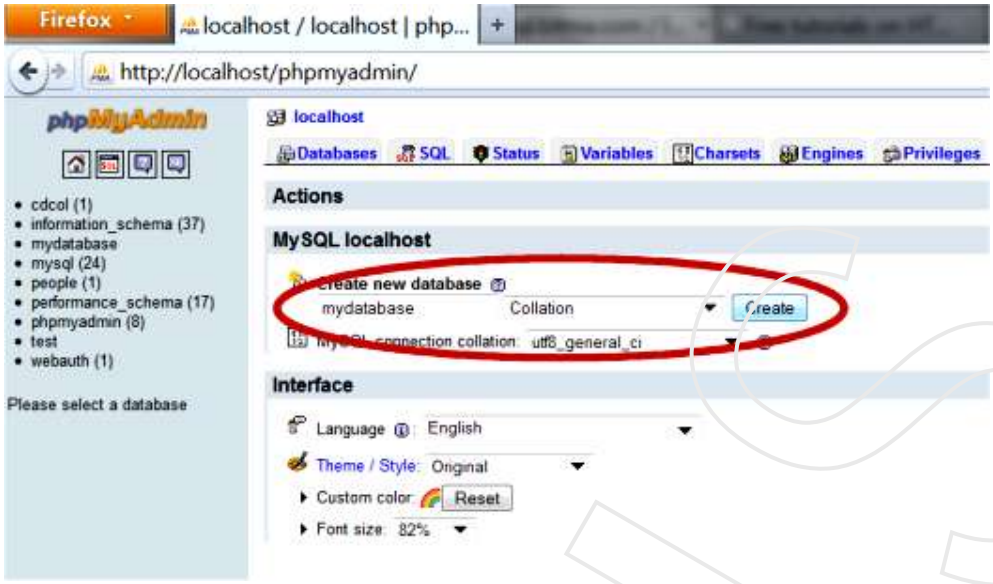
FirstName	LastName	Age
Tejali	Kaur	33

Create database and tables with phpMyAdmin

It can be useful to be able to create databases and tables directly in PHP. But often, it will be easier to use phpMyAdmin (or any other MySQL administration tool), which is standard on most web hosts and XAMPP. The screendumps below shows how to create a database and tables in phpMyAdmin.

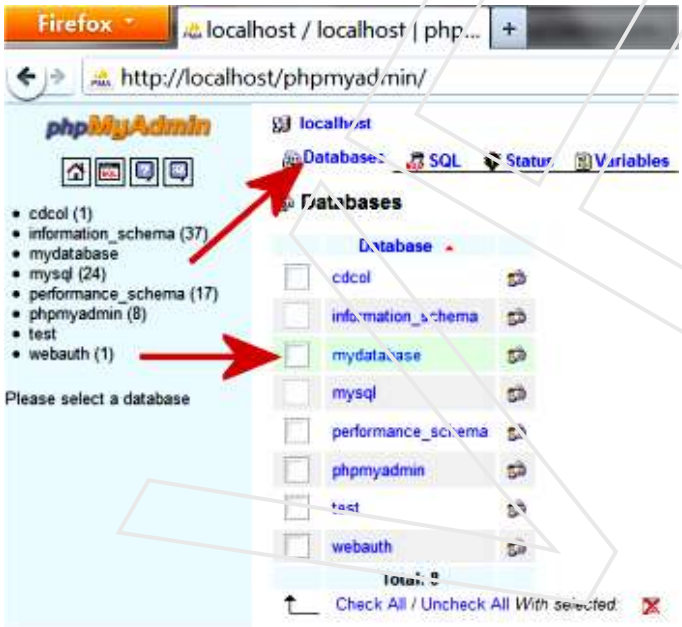
Start by logging onto phpMyAdmin. Often, the address will be the same as your MySQL server (eg. "http://mysql.myhost.com") and with the same username and password. In XAMPP, the address is http://localhost/phpmyadmin/.

When you are logged on, simply type a name for the database and press the button "Create":

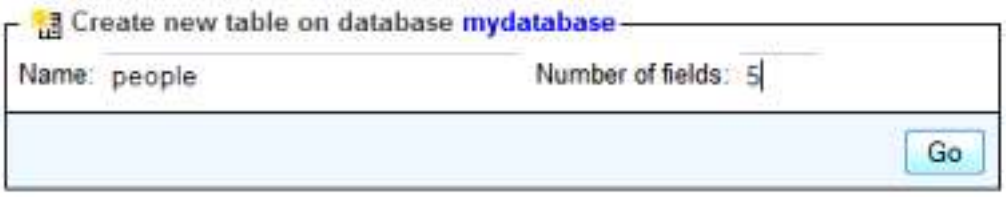


At some hosts, it's possible the have already created a database, and you may not have the rights to create more. If that is the case, you obviously just use the assigned database.

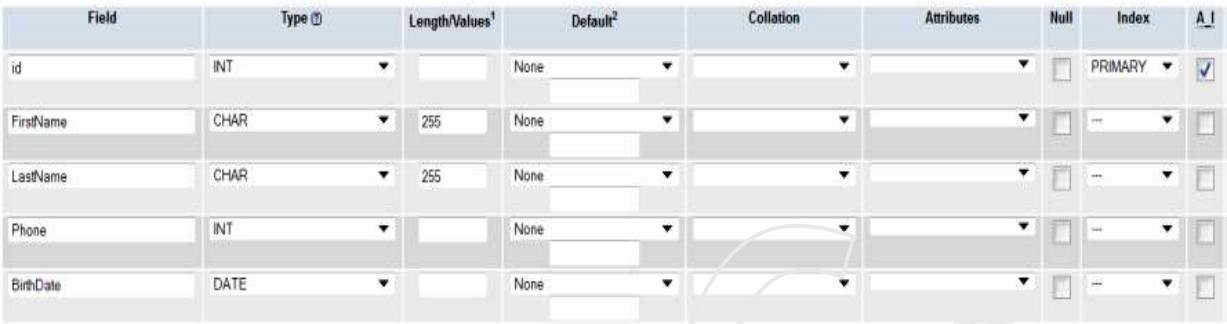
To create a table, click on the tab "Databases" and choose a database by clicking on it:



Then there will be a box titled "Create new table in database", where you type the name of the table and the number of columns and press the button "Go":



Then you can name the columns and set the data type, etc., as in the SQL example above.



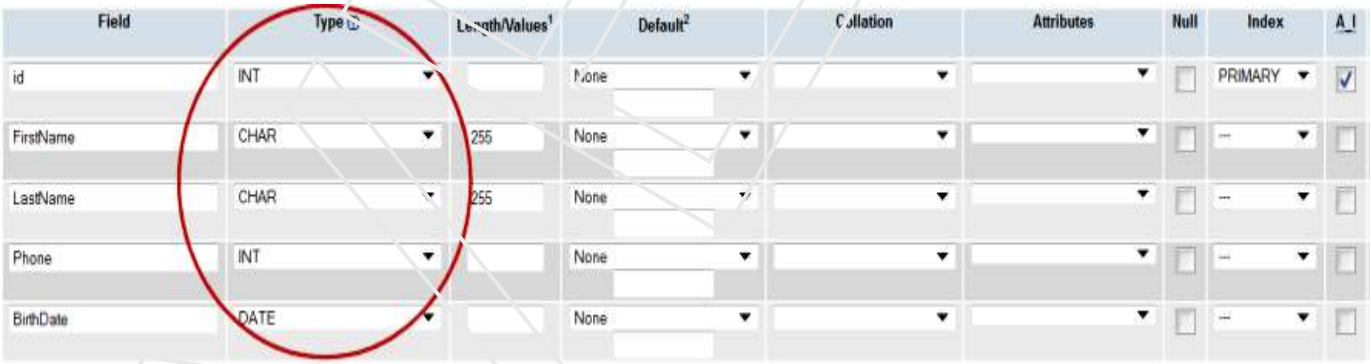
Notice, that here we also set "id" as PRIMARY KEY and uses AUTO_INCREMENT (A_I).
Now you have created your own database and table.

Most common beginner mistakes

In the beginning, you will probably get a lot of error messages when you try to update your databases. There is no room for the slightest inaccuracy when you work databases. A misplaced comma can mean the database is not being updated, and you get an error message instead. Below, we describe the most common beginner mistakes.

Wrong data types

It is important that there is consistency between the type of data and column. Each column can be set to a data type. The screenshot below shows the data types for the table "people" in our example.



An error occurs if you, for example, attempt to insert text or numbers in a date field. Therefore, try to set the data types as precisely as possible.
Below is the most common data types listed:

Setting	Data Type	Size
CHAR	Text or combinations of text and numbers. Can also be used for numbers that are not used in calculations (e.g., phone numbers).	Up to 255 characters - or the length defined in the "Length"
TEXT	Longer pieces of text, or combinations of text and numbers.	Up to 65,535 characters.
INT	Numerical data for mathematical calculations.	4 bytes.
DATE	Dates in the format YYYY-MM-DD	3 bytes.
TIME	Time in the format hh:mm:ss	3 bytes.
DATETIME	Date and time in the format YYYY-MM-DD hh:mm:ss	8 bytes.

SQL statements with quotes or backslash

If you try to insert text that contains the characters single quote ('), double quote (") or backslash (\), the record may not be inserted into the database. The solution is to add backslashes before characters that need to be quoted in database queries.

This can be done with the function addslashes this way:

```
<?php

$strText = "Is your name O'Reilly?";
$strText = addslashes($strText);

?>
```

All single quotes ('), double quotes (") and backslashs (\) will then get an extra backslash before the character. This would only be to get the data into the database, the extra \ will not be inserted. Please note that PHP runs addslashes on all $_GET, $_POST, and $_COOKIE data by default. Therefore do not use addslashes on strings that have already been escaped.

Create an ODBC Connection

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

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

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

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

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

Connecting to an ODBC

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

The odbc_exec() function is used to execute an SQL statement.

Example

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

```
$conn=odbc_connect('northwind',",");
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
```

Retrieving Records

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

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

```
odbc_fetch_row($rs)
```

Retrieving Fields from a Record

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

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

```
$compname=odbc_result($rs,1);
```

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

```
$compname=odbc_result($rs,"CompanyName");
```

Closing an ODBC Connection

The odbc_close() function is used to close an ODBC connection.

```
odbc_close($conn);
```

An ODBC Example

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

```
<html>
<body>

<?php
$conn=odbc_connect('northwind',",");
if (!$conn) {
```

```
  exit("Connection Failed: " . $conn);
}

$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
if (!$rs) {
  exit("Error in SQL");
}

echo "<table><tr>";
echo "<th>Companyname</th>";
echo "<th>Contactname</th></tr>";
while (odbc_fetch_row($rs)) {
  $compname=odbc_result($rs,"CompanyName");
  $conname=odbc_result($rs,"ContactName");
  echo "<tr><td>$compname</td>";
  echo "<td>$conname</td></tr>";
}
odbc_close($conn);
echo "</table>";
?>
</body>
</html>
```

PHP and XML

XML is a markup language that looks a lot like HTML. An XML document is plain text and contains tags delimited by < and >.There are two big differences between XML and HTML:

- XML doesn't define a specific set of tags you must use.
- XML is extremely picky about document structure.

XML gives you a lot more freedom than HTML. HTML has a certain set of tags: the <a> tags surround a link, the <p> startsa paragraph and so on. An XML document, however, can use any tags you want. Put <rating></rating> tags around a movie rating, >height></height> tags around someone's height. Thus XML gives you option to device your own tags.

XML is very strict when it comes to document structure. HTML lets you play fast and loose with some opening and closing tags. BUt this is not the case with XML.

HTML list that's not valid XML:

```
<ul>
<li>Braised Sea Cucumber
<li>Baked Giblets with Salt
<li>Abalone with Marrow and Duck Feet
</ul>
```

This is not a valid XML document because there are no closing tags to match up with the three opening <li> tags. Every opened tag in an XML document must be closed.

HTML list that is valid XML:

```
<ul>
<li>Braised Sea Cucumber</li>
<li>Baked Giblets with Salt</li>
<li>Abalone with Marrow and Duck Feet</li>
</ul>
```

Parsing an XML Document:

PHP 5's new **SimpleXML** module makes parsing an XML document, well, simple. It turns an XML document into an object that provides structured access to the XML.
To create a SimpleXML object from an XML document stored in a string, pass the string to **simplexml_load_string( )**. It returns a SimpleXML object.

Example:

Try out following example:

```
<?php

$channel =<<<_XML_
<channel>
<title>What's For Dinner<title>
<link>http://menu.example.com/<link>
<description>Choose what to eat tonight.</description>
</channel>
_XML_;

$xml = simplexml_load_string($channel);
print "The $xml->title channel is available at $xml->link. ";
print "The description is \"$xml->description\"";
?>
```

It will produce following result:

```
The What's For Dinner channel is available at http://menu.example.com/. The description is "Choose what to eat tonight."
```

NOTE: You can use function **simplexml_load_file(filename)** if you have XML content in a file.

Generating an XML Document:

SimpleXML is good for parsing existing XML documents, but you can't use it to create a new one from scratch.
The easiest way to generate an XML document is to build a PHP array whose structure mirrors that of the XML document and then to iterate through the array, printing each element with appropriate formatting.

Example:

Try out following example:

```
<?php

$channel = array('title' => "What's For Dinner",
                 'link' => 'http://menu.example.com/',
                 'description' => 'Choose what to eat tonight.');
print "<channel>\n";
foreach ($channel as $element => $content) {
```

```
   print " <$element>";
   print htmlentities($content);
   print "</$element>\n";
}
print "</channel>";
?>
```

It will produce following result:

```
<channel>
<title>What's For Dinner</title>
<link>http://menu.example.com/</link>
<description>Choose what to eat tonight.</description>
</channel></html>
```

Expat

To read and update - create and manipulate - an XML document, you will need an XML parser.

There are two basic types of XML parsers:

- Tree-based parser: This parser transforms an XML document into a tree structure. It analyzes the whole document, and provides access to the tree elements. e.g. the Document Object Model (DOM)
- Event-based parser: Views an XML document as a series of events. When a specific event occurs, it calls a function to handle it

The Expat parser is an event-based parser.
Event-based parsers focus on the content of the XML documents, not their structure. Because of this, event-based parsers can access data faster than tree-based parsers.
Look at the following XML fraction:
<from>Jatin</from>

An event-based parser reports the XML above as a series of three events:

- Start element: from
- Start CDATA section, value: Jatin
- Close element: from

The XML example above contains well-formed XML. However, the example is not valid XML, because there is no Document Type Definition (DTD) associated with it.

However, this makes no difference when using the Expat parser. Expat is a non-validating parser, and ignores any DTDs.

As an event-based, non-validating XML parser, Expat is fast and small, and a perfect match for PHP web applications.

Note: XML documents must be well-formed or Expat will generate an error.

Installation

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

An XML File

The XML file below will be used in our example:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jatin</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Initializing the XML Parser

We want to initialize the XML parser in PHP, define some handlers for different XML events, and then parse the XML file.

Example

```
<?php
//Initialize the XML parser
$parser=xml_parser_create();

//Function to use at the start of an element
function start($parser,$element_name,$element_attrs) {
  switch($element_name) {
    case "NOTE":
    echo "-- Note --<br>";
    break;
    case "TO":
    echo "To: ";
    break;
    case "FROM":
    echo "From: ";
    break;
    case "HEADING":
    echo "Heading: ";
    break;
    case "BODY":
    echo "Message: ";
  }
}

//Function to use at the end of an element
function stop($parser,$element_name) {
  echo "<br>";
}
```

```
//Function to use when finding character data
function char($parser,$data) {
  echo $data;
}

//Specify element handler
xml_set_element_handler($parser,"start","stop");

//Specify data handler
xml_set_character_data_handler($parser,"char");

//Open XML file
$fp=fopen("test.xml","r");

//Read data
while ($data=fread($fp,4096)) {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
}

//Free the XML parser
xml_parser_free($parser);
?>
```

The output of the code above will be:

```
-- Note --
To: Tove
From: Jatin
Heading: Reminder
Message: Don't forget me this weekend!
```

How it works:

1. Initialize the XML parser with the xml_parser_create() function
2. Create functions to use with the different event handlers
3. Add the xml_set_element_handler() function to specify which function will be executed when the parser encounters the opening and closing tags
4. Add the xml_set_character_data_handler() function to specify which function will execute when the parser encounters character data
5. Parse the file "test.xml" with the xml_parse() function
6. In case of an error, add xml_error_string() function to convert an XML error to a textual description
7. Call the xml_parser_free() function to release the memory allocated with the xml_parser_create() function

PHP XML DOM

DOM

The W3C DOM provides a standard set of objects for HTML and XML documents, and a standard interface for accessing and manipulating them.

The W3C DOM is separated into different parts (Core, XML, and HTML) and different levels (DOM Level 1/2/3):

- Core DOM - defines a standard set of objects for any structured document
- XML DOM - defines a standard set of objects for XML documents
- HTML DOM - defines a standard set of objects for HTML documents

XML Parsing

To read and update - create and manipulate - an XML document, you will need an XML parser.

There are two basic types of XML parsers:

- Tree-based parser: This parser transforms an XML document into a tree structure. It analyzes the whole document, and provides access to the tree elements
- Event-based parser: Views an XML document as a series of events. When a specific event occurs, it calls a function to handle it

The DOM parser is an tree-based parser.

Look at the following XML document fraction:

```
<?xml version="1.0" encoding="UTF-8"?>
<from>Jatin</from>
```

The XML DOM sees the XML above as a tree structure:

- Level 1: XML Document
- Level 2: Root element: <from>
- Level 3: Text element: "Jatin"

Installation

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

An XML File

The XML file below will be used in our example:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
```

```
<from>Jatin</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Load and Output XML

We want to initialize the XML parser, load the xml, and output it:

Example

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

print $xmlDoc->saveXML();
?>
```

The output of the code above will be:

Tove Jatin Reminder Don't forget me this weekend!

If you select "View source" in the browser window, you will see the following HTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jatin</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The example above creates a DOMDocument-Object and loads the XML from "note.xml" into it.
Then the saveXML() function puts the internal XML document into a string, so we can output it.

Looping through XML

We want to initialize the XML parser, load the XML, and loop through all elements of the <note> element:

Example

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item) {
  print $item->nodeName . " = " . $item->nodeValue . "<br>";
}
?>
```

The output of the code above will be:

#text =
to = Tove
#text =
from = Jatin
#text =
heading = Reminder
#text =
body = Don't forget me this weekend!
#text =
In the example above you see that there are empty text nodes between each element.

When XML generates, it often contains white-spaces between the nodes. The XML DOM parser treats these as ordinary elements, and if you are not aware of them, they sometimes cause problems.

PHP SimpleXML?

SimpleXML is new in PHP 5.

The SimpleXML extension provides is a simple way of getting an XML element's name and text.

Compared to DOM or the Expat parser, SimpleXML just takes a few lines of code to read text data from an XML element.

SimpleXML converts the XML document (or XML string) into an object, like this:

- Elements are converted to single attributes of the SimpleXMLElement object. When there's more than one element on one level, they are placed inside an array
- Attributes are accessed using associative arrays, where an index corresponds to the attribute name
- Text inside elements is converted to strings. If an element has more than one text node, they will be arranged in the order they are found

SimpleXML is fast and easy to use when performing tasks like:

- Reading/Extracting data from XML files/strings
- Editing text nodes or attributes

However, when dealing with advanced XML, you are better off using the Expat parser or the XML DOM.

Installation

As of PHP 5, the SimpleXML functions are part of the PHP core. No installation is required to use these functions.

PHP SimpleXML Examples

Assume we have the following XML file, "note.xml":

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jatin</from>
<heading>Reminder</heading>
```

```
<body>Don't forget me this weekend!</body>
</note>
```

Now we want to output different information from the XML file above:

Example 1

Output keys and elements of the $xml variable (which is a SimpleXMLElement object):

```
<?php
$xml=simplexml_load_file("note.xml");
print_r($xml);
?>
```

The output of the code above will be:

```
SimpleXMLElement Object ( [to] => Tove [from] => Jatin [heading] => Reminder [body] => Don't forget me this weekend! )
```

Example 2

Output the data from each element in the XML file:

```
<?php
$xml=simplexml_load_file("note.xml");
echo $xml->to . "<br>";
echo $xml->from . "<br>";
echo $xml->heading . "<br>";
echo $xml->body;
?>
```

The output of the code above will be:

```
Tove
Jatin
Reminder
Don't forget me this weekend!
```

Example 3

Output the element's name and data for each child node:

```
<?php
$xml=simplexml_load_file("note.xml");
echo $xml->getName() . "<br>";

foreach($xml->children() as $child) {
  echo $child->getName() . ": " . $child . "<br>";
}
?>
```

The output of the code above will be:

```
note
to: Tove
from: Jatin
heading: Reminder
body: Don't forget me this weekend!
```

PHP and AJAX

AJAX

- AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS and Java Script.
- Conventional web application trasmit information to and from the sever using synchronous requests. This means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX when submit is pressed, JavaScript will make a request to the server, interpret the results and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.

PHP and AJAX Example:

To clearly illustrate how easy it is to access information from a database using Ajax and PHP, we are going to build MySQL queries on the fly and display the results on "ajax.html". But before we proceed, lets do ground work. Create a table using the following command.

NOTE: We are asuing you have sufficient privilege to perform following MySQL operations

```
CREATE TABLE `ajax_example` (
  `name` varchar(50) NOT NULL,
  `age` int(11) NOT NULL,
  `sex` varchar(1) NOT NULL,
  `wpm` int(11) NOT NULL,
  PRIMARY KEY  (`name`)
)
```

Now dump the following data into this table using the foloowing SQL statements

```
INSERT INTO `ajax_example` VALUES ('Jerry', 120, 'm', 20);
INSERT INTO `ajax_example` VALUES ('Regis', 75, 'm', 44);
INSERT INTO `ajax_example` VALUES ('Frank', 45, 'm', 87);
INSERT INTO `ajax_example` VALUES ('Jill', 22, 'f', 72);
INSERT INTO `ajax_example` VALUES ('Tracy', 27, 'f', 0);
INSERT INTO `ajax_example` VALUES ('Julie', 35, 'f', 90);
```

Client Side HTML file

Now lets have our client side HTML file which is ajax.html and it will have following code

```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
//Browser Support Code
function ajaxFunction(){
 var ajaxRequest;  // The variable that makes Ajax possible!

 try{
   // Opera 8.0+, Firefox, Safari
   ajaxRequest = new XMLHttpRequest();
 }catch (e){
   // Internet Explorer Browsers
   try{
      ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
   }catch (e) {
      try{
         ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
      }catch (e){
         // Something went wrong
         alert("Your browser broke!");
         return false;
      }
   }
 }
 // Create a function that will receive data
 // sent from the server and will update
 // div section in the same page.
 ajaxRequest.onreadystatechange = function(){
   if(ajaxRequest.readyState == 4){
      var ajaxDisplay = document.getElementById('ajaxDiv');
      ajaxDisplay.innerHTML = ajaxRequest.responseText;
   }
 }
 // Now get the value from user and pass it to
 // server script.
 var age = document.getElementById('age').value;
 var wpm = document.getElementById('wpm').value;
 var sex = document.getElementById('sex').value;
 var queryString = "?age=" + age ;
 queryString +=  "&wpm=" + wpm + "&sex=" + sex;
 ajaxRequest.open("GET", "ajax-example.php" +
                              queryString, true);
 ajaxRequest.send(null);
}
//-->
</script>
<form name='myForm'>
Max Age: <input type='text' id='age' /> <br />
Max WPM: <input type='text' id='wpm' />
<br />
Sex: <select id='sex'>
<option value="m">m</option>
<option value="f">f</option>
</select>
<input type='button' onclick='ajaxFunction()'
                              value='Query MySQL'/>
</form>
<div id='ajaxDiv'>Your result will display here</div>
</body>
</html>
```

NOTE: The way of passing variables in the Query is according to HTTP standard and the have formA

```
URL?variable1=value1;&variable2=value2;
```

Now the above code will give you a screen as given below

NOTE: This is dummy screen and would not work

Max Age:

Max WPM:

Sex:
Your result will display here

Server Side PHP file

The main difference in server-side scripting is that the scripts are compiled in advance, and left to execute in the background, waiting for a request from a client. The language and syntax are almost the same, but server-side scripts have access to a different range of built-in objects. There are also some differences intheassociated HTMLtags. When writing server-side JavaScript, you need to know what type of server it will run on. In a Netscape environment, the script requires a FastTrack or Enterprise server, and is supported by a Netscape technology called LiveWire. Among other things, LiveWire provides a JavaScript compiler which converts the script to a bytecode file (with the extension WEB). Another LiveWire component, called Application Manager, is then used to activate the compiled code.The script actually runs when the user opens its URL in the browser. LiveWire also provides a way for JavaScript code to store permanent information on the server. By using the File object, the script can perform low-level input and output to serial files.Alternatively, you can use the data base object to access back-end data bases via ODBC. These tools make it possible to write full-blown server-side applications which collect data from the user, execute queries, generate reports, and quite a lot more. The Microsoft equivalent of LiveWire is Active Server Pages (ASP). This runs on Microsoft's Internet Infor-mation Server (IIS) and Personal Web Server. It provides a run-time engine which interprets server-side scripts (held in files with the extension ASP) and returns HTML pages to the browser. ASP is not specific to JavaScript; it also supports VBScript and Perl. Like LiveWire, ASP provides objects for performing low-level file access on the server and for accessing back-end databases via ODBC. To finish, here is a simple example of a server-side script. These three lines of LiveWire code display the user's IP address in the browser window:

```
<SERVER>
write("Your IP address is "+-
request.ip)
</SERVER>
```

Note the use of the <SERVER> and SERVER>tagsinplaceof<SCRIPT> and </SCRIPT>. The request object provides details of the requesting browser. The ASP equivalent of this code would not include the <SERVER> tags, but would instead have a Language declaration at the top of the document. The syntax for using the Request object (note there is a capital R in the Microsoft version) would also be slightly different.

So now your client side script is ready. Now we have to write our server side script which will fetch age, wpm and sex from the database and will send it back to the client. Put the following code into "ajax-example.php" file

```php
<?php
$dbhost = "localhost";
$dbuser = "dbusername";
$dbpass = "dbpassword";
$dbname = "dbname";
        //Connect to MySQL Server
mysql_connect($dbhost, $dbuser, $dbpass);
        //Select Database
mysql_select_db($dbname) or die(mysql_error());
        // Retrieve data from Query String
$age = $_GET['age'];
$sex = $_GET['sex'];
$wpm = $_GET['wpm'];
        // Escape User Input to help prevent SQL Injection
$age = mysql_real_escape_string($age);
$sex = mysql_real_escape_string($sex);
$wpm = mysql_real_escape_string($wpm);
        //build query
$query = "SELECT * FROM ajax_example WHERE sex = '$sex'";
if(is_numeric($age))
        $query .= " AND age <= $age";
if(is_numeric($wpm))
        $query .= " AND wpm <= $wpm";
        //Execute query
$qry_result = mysql_query($query) or die(mysql_error());

        //Build Result String
$display_string = "<table>";
$display_string .= "<tr>";
$display_string .= "<th>Name</th>";
$display_string .= "<th>Age</th>";
$display_string .= "<th>Sex</th>";
$display_string .= "<th>WPM</th>";
$display_string .= "</tr>";

// Insert a new row in the table for each person returned
while($row = mysql_fetch_array($qry_result)){
        $display_string .= "<tr>";
        $display_string .= "<td>$row[name]</td>";
        $display_string .= "<td>$row[age]</td>";
        $display_string .= "<td>$row[sex]</td>";
        $display_string .= "<td>$row[wpm]</td>";
        $display_string .= "</tr>";

}
echo "Query: " . $query . "<br />";
$display_string .= "</table>";
echo $display_string;
?>
```

Now try by entering a valid value in "Max Age" or any other box and then click Query MySQL button.

Max Age:

Max WPM:

Sex:
Your result will display here

AJAX and MySQL : AJAX Database Example

The following example will demonstrate how a web page can fetch information from a database with AJAX:

Example

Person info will be listed here...

Example Explained - The MySQL Database

The database table we use in the example above looks like this:

id	FirstName	LastName	Age	Hometown	Job
1	Jatin	Bedi	41	Dera	Master
2	Madhav	Bedi	40	Gurdaspur	Teacher
3	Sanatan	Swanson	39	Dera	Police Officer
4	Tejali	Kaur	41	Dera	Leader

Example Explained - The HTML Page

When a user selects a user in the dropdown list above, a function called "showUser()" is executed. The function is triggered by the "onchange" event:

```
<html>
<head>
<script>
function showUser(str) {
  if (str=="") {
    document.getElementById("txtHint").innerHTML="";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  } else { // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState==4 && xmlhttp.status==200) {
      document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
  xmlhttp.open("GET","getuser.php?q="+str,true);
  xmlhttp.send();
}
</script>
</head>
<body>
```

```
<form>
<select name="users" onchange="showUser(this.value)">
<option value="">Select a person:</option>
<option value="1">Jatin Bedi</option>
<option value="2">Madhav Bedi</option>
<option value="3">Tejali Kaur</option>
<option value="4">Sanatan Swanson</option>
</select>
</form>
<br>
<div id="txtHint"><b>Person info will be listed here.</b></div>

</body>
</html>
```

The showUser() function does the following:

- Check if a person is selected
- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the dropdown list)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "getuser.php".

The source code in "getuser.php" runs a query against a MySQL database, and returns the result in an HTML table:

```
<?php
$q = intval($_GET['q']);

$con = mysqli_connect('localhost','jatin','abc123','my_db');
if (!$con) {
  die('Could not connect: ' . mysqli_error($con));
}
mysqli_select_db($con,"ajax_demo");
$sql="SELECT * FROM user WHERE id = '".$q."'";
$result = mysqli_query($con,$sql);

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";

while($row = mysqli_fetch_array($result)) {
  echo "<tr>";
```

```
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "<td>" . $row['Age'] . "</td>";
  echo "<td>" . $row['Hometown'] . "</td>";
  echo "<td>" . $row['Job'] . "</td>";
  echo "</tr>";
}
echo "</table>";

mysqli_close($con);
?>
```

Explanation: When the query is sent from the JavaScript to the PHP file, the following happens:

1. PHP opens a connection to a MySQL server
2. The correct person is found
3. An HTML table is created, filled with data, and sent back to the "txtHint" placeholder

AJAX and XML: AJAX XML Example

The following example will demonstrate how a web page can fetch information from an XML file with AJAX:

Example

CD info will be listed here...

Example Explained - The HTML Page

When a user selects a CD in the dropdown list above, a function called "showCD()" is executed. The function is triggered by the "onchange" event:

```
<html>
<head>
<script>
function showCD(str) {
  if (str=="") {
    document.getElementById("txtHint").innerHTML="";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  } else {  // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState==4 && xmlhttp.status==200) {
      document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
```

```
  xmlhttp.open("GET","getcd.php?q="+str,true);
  xmlhttp.send();
}
</script>
</head>
<body>

<form>
Select a CD:
<select name="cds" onchange="showCD(this.value)">
<option value="">Select a CD:</option>
<option value="Bob Dylan">Bob Dylan</option>
<option value="Bonnie Tyler">Bonnie Tyler</option>
<option value="Dolly Parton">Dolly Parton</option>
</select>
</form>
<div id="txtHint"><b>CD info will be listed here...</b></div>

</body>
</html>
```

The showCD() function does the following:

- Check if a CD is selected
- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the dropdown list)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "getcd.php".
The PHP script loads an XML document, "cd_catalog.xml", runs a query against the XML file, and returns the result as HTML:

```
<?php
$q=$_GET["q"];

$xmlDoc = new DOMDocument();
$xmlDoc->load("cd_catalog.xml");

$x=$xmlDoc->getElementsByTagName('ARTIST');

for ($i=0; $i<=$x->length-1; $i++) {
  //Process only element nodes
  if ($x->item($i)->nodeType==1) {
    if ($x->item($i)->childNodes->item(0)->nodeValue == $q) {
      $y=($x->item($i)->parentNode);
    }
  }
}

$cd=($y->childNodes);
```

```
for ($i=0;$i<$cd->length;$i++) {
  //Process only element nodes
  if ($cd->item($i)->nodeType==1) {
    echo("<b>" . $cd->item($i)->nodeName . ":</b> ");
    echo($cd->item($i)->childNodes->item(0)->nodeValue);
    echo("<br>");
  }
}
?>
```

When the CD query is sent from the JavaScript to the PHP page, the following happens:

1. PHP creates an XML DOM object
2. Find all <artist> elements that matches the name sent from the JavaScript
3. Output the album information (send to the "txtHint" placeholder)

AJAX Live Search

The following example will demonstrate a live search, where you get search results while you type.

Live search has many Rozyefits compared to traditional searching:

- Results are shown as you type
- Results narrow as you continue typing
- If results become too narrow, remove characters to see a broader result

Search for a Veinstitution page in the input field below:

The results in the example above are found in an XML file (links.xml). To make this example small and simple, only six results are available.

Example Explained - The HTML Page

When a user types a character in the input field above, the function "showResult()" is executed. The function is triggered by the "onkeyup" event:

```
<html>
<head>
<script>
function showResult(str) {
  if (str.length==0) {
    document.getElementById("livesearch").innerHTML="";
    document.getElementById("livesearch").style.border="0px";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
```

```
  } else {  // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState==4 && xmlhttp.status==200) {
      document.getElementById("livesearch").innerHTML=xmlhttp.responseText;
      document.getElementById("livesearch").style.border="1px solid #A5ACB2";
    }
  }
  xmlhttp.open("GET","livesearch.php?q="+str,true);
  xmlhttp.send();
}
</script>
</head>
<body>

<form>
<input type="text" size="30" onkeyup="showResult(this.value)">
<div id="livesearch"></div>
</form>

</body>
</html>
```

Source code explanation:

If the input field is empty (str.length==0), the function clears the content of the livesearch placeholder and exits the function.

If the input field is not empty, the showResult() function executes the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the input field)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "livesearch.php".

The source code in "livesearch.php" searches an XML file for titles matching the search string and returns the result:

```
<?php
$xmlDoc=new DOMDocument();
$xmlDoc->load("links.xml");

$x=$xmlDoc->getElementsByTagName('link');

//get the q parameter from URL
$q=$_GET["q"];
```

```
//lookup all links from the xml file if length of q>0
if (strlen($q)>0) {
  $hint="";
  for($i=0; $i<($x->length); $i++) {
    $y=$x->item($i)->getElementsByTagName('title');
    $z=$x->item($i)->getElementsByTagName('url');
    if ($y->item(0)->nodeType==1) {
      //find a link matching the search text
      if (stristr($y->item(0)->childNodes->item(0)->nodeValue,$q)) {
        if ($hint=="") {
          $hint="<a href='" .
          $z->item(0)->childNodes->item(0)->nodeValue .
          "' target='_blank'>" .
          $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
        } else {
          $hint=$hint . "<br /><a href='" .
          $z->item(0)->childNodes->item(0)->nodeValue .
          "' target='_blank'>" .
          $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
        }
      }
    }
  }
}

// Set output to "no suggestion" if no hint were found
// or to the correct values
if ($hint=="") {
  $response="no suggestion";
} else {
  $response=$hint;
}

//output the response
echo $response;
?>
```

If there is any text sent from the JavaScript (strlen($q) > 0), the following happens:

- Load an XML file into a new XML DOM object
- Loop through all <title> elements to find matches from the text sent from the JavaScript
- Sets the correct url and title in the "$response" variable. If more than one match is found, all matches are added to the variable
- If no matches are found, the $response variable is set to "no suggestion"

AJAX RSS Reader

The following example will demonstrate an RSS reader, where the RSS-feed is loaded into a webpage without reloading:

RSS-feed will be listed here...

Example Explained - The HTML Page

When a user selects an RSS-feed in the dropdown list above, a function called "showRSS()" is executed. The function is triggered by the "onchange" event:

```
<html>
<head>
<script>
function showRSS(str) {
  if (str.length==0) {
    document.getElementById("rssOutput").innerHTML="";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  } else {  // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState==4 && xmlhttp.status==200) {
      document.getElementById("rssOutput").innerHTML=xmlhttp.responseText;
    }
  }
  xmlhttp.open("GET","getrss.php?q="+str,true);
  xmlhttp.send();
}
</script>
</head>
<body>

<form>
<select onchange="showRSS(this.value)">
<option value="">Select an RSS-feed:</option>
<option value="Google">Google News</option>
<option value="MSNBC">MSNBC News</option>
</select>
</form>
<br>
<div id="rssOutput">RSS-feed will be listed here...</div>
</body>
</html>
```

The showRSS() function does the following:

- Check if an RSS-feed is selected
- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server

- Notice that a parameter (q) is added to the URL (with the content of the dropdown list)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "getrss.php":

```
<?php
//get the q parameter from URL
$q=$_GET["q"];

//find out which feed was selected
if($q=="Google") {
  $xml=("http://news.google.com/news?ned=us&topic=h&output=rss");
} elseif($q=="MSNBC") {
  $xml=("http://rss.msnbc.msn.com/id/3032091/device/rss/rss.xml");
}

$xmlDoc = new DOMDocument();
$xmlDoc->load($xml);

//get elements from "<channel>"
$channel=$xmlDoc->getElementsByTagName('channel')->item(0);
$channel_title = $channel->getElementsByTagName('title')
->item(0)->childNodes->item(0)->nodeValue;
$channel_link = $channel->getElementsByTagName('link')
->item(0)->childNodes->item(0)->nodeValue;
$channel_desc = $channel->getElementsByTagName('description')
->item(0)->childNodes->item(0)->nodeValue;

//output elements from "<channel>"
echo("<p><a href='" . $channel_link
  . "'>" . $channel_title . "</a>");
echo("<br>");
echo($channel_desc . "</p>");

//get and output "<item>" elements
$x=$xmlDoc->getElementsByTagName('item');
for ($i=0; $i<=2; $i++) {
  $item_title=$x->item($i)->getElementsByTagName('title')
  ->item(0)->childNodes->item(0)->nodeValue;
  $item_link=$x->item($i)->getElementsByTagName('link')
  ->item(0)->childNodes->item(0)->nodeValue;
  $item_desc=$x->item($i)->getElementsByTagName('description')
  ->item(0)->childNodes->item(0)->nodeValue;
  echo ("<p><a href='" . $item_link
  . "'>" . $item_title . "</a>");
  echo ("<br>");
  echo ($item_desc . "</p>");
}
?>
```

When a request for an RSS feed is sent from the JavaScript, the following happens:

- Check which feed was selected
- Create a new XML DOM object
- Load the RSS document in the xml variable
- Extract and output elements from the channel element
- Extract and output elements from the item elements

AJAX Poll

The following example will demonstrate a poll where the result is shown without reloading.

Do you like PHP and AJAX so far?

Yes:

No:

Example Explained - The HTML Page

When a user choose an option above, a function called "getVote()" is executed. The function is triggered by the "onclick" event:

```
<html>
<head>
<script>
function getVote(int) {
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  } else {  // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState==4 && xmlhttp.status==200) {
      document.getElementById("poll").innerHTML=xmlhttp.responseText;
    }
  }
  xmlhttp.open("GET","poll_vote.php?vote="+int,true);
  xmlhttp.send();
}
</script>
</head>
<body>

<div id="poll">
<h3>Do you like PHP and AJAX so far?</h3>
<form>
Yes:
<input type="radio" name="vote" value="0" onclick="getVote(this.value)">
<br>No:
<input type="radio" name="vote" value="1" onclick="getVote(this.value)">
```

```
</form>
</div>

</body>
</html>
```

The getVote() function does the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (vote) is added to the URL (with the value of the yes or no option)

The PHP File

The page on the server called by the JavaScript above is a PHP file called "poll_vote.php":

```
<?php
$vote = $_REQUEST['vote'];

//get content of textfile
$filename = "poll_result.txt";
$content = file($filename);

//put content in array
$array = explode("||", $content[0]);
$yes = $array[0];
$no = $array[1];

if ($vote == 0) {
  $yes = $yes + 1;
}
if ($vote == 1) {
  $no = $no + 1;
}

//insert votes to txt file
$insertvote = $yes."||".$no;
$fp = fopen($filename,"w");
fputs($fp,$insertvote);
fclose($fp);
?>

<h2>Result:</h2>
<table>
<tr>
<td>Yes:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($yes/($no+$yes),2)); ?>'
height='20'>
```

```
<?php echo(100*round($yes/($no+$yes),2)); ?>%
</td>
</tr>
<tr>
<td>No:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($no/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($no/($no+$yes),2)); ?>%
</td>
</tr>
</table>
```

The value is sent from the JavaScript, and the following happens:

1. Get the content of the "poll_result.txt" file
2. Put the content of the file in variables and add one to the selected variable
3. Write the result to the "poll_result.txt" file
4. Output a graphical representation of the poll result

The Text File

The text file (poll_result.txt) is where we store the data from the poll.

It is stored like this:

0||0

The first number represents the "Yes" votes, the second number represents the "No" votes.

Note: Remember to allow your web server to edit the text file. Do **NOT** give everyone access, just the web server (PHP).

PHP for C Developers

The simplest way to think of PHP is as interpreted C that you can embed in HTML documents. The language itself is a lot like C, except with untyped variables, a whole lot of Web-specific libraries built in, and everything hooked up directly to your favorite Web server.

The syntax of statements and function definitions should be familiar, except that variables are always preceded by $, and functions do not require separate prototypes.

Here we will put some similarities and diferences in PHP and C

Similarities:

- **Syntax:** Broadly speaking, PHP syntax is the same as in C: Code is blank insensitive, statements are terminated with semicolons, function calls have the same structure (my_function(expression1, expression2)), and curly braces ({ and }) make statements into blocks. PHP supports C and C++-style comments (/* */ as well as //), and also Perl and shell-script style (#).

- **Operators:** The assignment operators (=, +=, *=, and so on), the Boolean operators (&&, ||, !), the comparison operators (<,>, <=, >=, ==, !=), and the basic arithmetic operators (+, -, *, /, %) all behave in PHP as they do in C.
- **Control structures:** The basic control structures (if, switch, while, for) behave as they do in C, including supporting break and continue. One notable difference is that switch in PHP can accept strings as case identifiers.
- **Function names:** As you peruse the documentation, you.ll see many function names that seem identical to C functions.

Differences:

- **Dollar signs:** All variables are denoted with a leading $. Variables do not need to be declared in advance of assignment, and they have no intrinsic type.
- **Types:** PHP has only two numerical types: integer (corresponding to a long in C) and double (corresponding to a double in C). Strings are of arbitrary length. There is no separate character type.
- **Type conversion:** Types are not checked at compile time, and type errors do not typically occur at runtime either. Instead, variables and values are automatically converted across types as needed.
- **Arrays:** Arrays have a syntax superficially similar to C's array syntax, but they are implemented completely differently. They are actually associative arrays or hashes, and the index can be either a number or a string. They do not need to be declared or allocated in advance.
- **No structure type:** There is no struct in PHP, partly because the array and object types together make it unnecessary. The elements of a PHP array need not be of a consistent type.
- **No pointers:** There are no pointers available in PHP, although the typeless variables play a similar role. PHP does support variable references. You can also emulate function pointers to some extent, in that function names can be stored in variables and called by using the variable rather than a literal name.
- **No prototypes:** Functions do not need to be declared before their implementation is defined, as long as the function definition can be found somewhere in the current code file or included files.
- **Memory management:** The PHP engine is effectively a garbage-collected environment (reference-counted), and in small scripts there is no need to do any deallocation. You should freely allocate new structures - such as new strings and object instances. IN PHP5, it is possible to define destructors for objects, but there is no free or delete. Destructors are called when the last reference to an object goes away, before the memory is reclaimed.
- **Compilation and linking:** There is no separate compilation step for PHP scripts.
- **Permissiveness:** As a general matter, PHP is more forgiving than C (especially in its type system) and so will let you get away with new kinds of mistakes. Unexpected results are more common than errors.

PHP for PERL Developers

This chapter will list out major similarities and differences in between PHP and PERL. This will help PERL developers to understand PHP very quickly and avoid common mistakes.

Similarities:

- **Compiled scripting languages:** Both Perl and PHP are scripting languages.This means that they are not used to produce native standalone executables in advance of execution.
- **Syntax:** PHP's basic syntax is very close to Perl's, and both share a lot of syntactic features with C. Code is insensitive to whitespace, statements are terminated by semicolons, and curly braces organize multiple statements into a single block. Function calls start with the name of the function, followed by the actual arguments enclosed in parentheses and separated by commas.

- **Dollar-sign variables:** All variables in PHP look like scalar variables in Perl: a name with a dollar sign ($) in front of it.
- **No declaration of variables:** As in Perl, you don.t need to declare the type of a PHP variable before using it.
- **Loose typing of variables:** As in Perl, variables in PHP have no intrinsic type other than the value they currently hold. You can store iether number or string in same type of variable.
- **Strings and variable interpolation:** Both PHP and Perl do more interpretation of double-quoted strings ("string") than of singlequoted strings ('string').

Differences:

- **PHP is HTML-embedded:** Although it is possible to use PHP for arbitrary tasks by running it from the command line, it is more typically connected to a Web server and used for producing Web pages. If you are used to writing CGI scripts in Perl, the main difference in PHP is that you no longer need to explicitly print large blocks of static HTML using print or heredoc statements and instead can simply write the HTML itself outside of the PHP code block.
- **No @ or % variables:** PHP has one only kind of variable, which starts with a dollar sign ($). Any of the datatypes in the language can be stored in such variables, whether scalar or compound.
- **Arrays versus hashes:** PHP has a single datatype called an array that plays the role of both hashes and arrays/lists in Perl.
- **Specifying arguments to functions:** Function calls in PHP look pretty much like subroutine calls in Perl. Function definitions in PHP, on the other hand, typically require some kind of list of formal arguments as in C or Java which is not the csse in PERL.
- **Variable scoping in functions:** In Perl, the default scope for variables is global. This means that top-level variables are visible inside subroutines. Often, this leads to promiscuous use of globals across functions. In PHP, the scope of variables within function definitions is local by default.
- **No module system as such:** In PHP there is no real distinction between normal code files and code files used as imported libraries.
- **Break and continue rather than next and last:** PHP is more like C langauge and uses break and continue instead of next and last statement.
- **No elsif:** A minor spelling difference: Perl's elsif is PHP's elseif.
- **More kinds of comments:** In addition to Perl-style (#) single-line comments, PHP offers C-style multiline comments (/* comment */) and Java-style single-line comments (// comment).
- **Regular expressions:** PHP does not have a built-in syntax specific to regular expressions, but has most of the same functionality in its "Perl-compatible" regular expression functions.

Glossary

PHP Function Reference

PHP Array Functions

These functions allow you to interact with and manipulate arrays in various ways. Arrays are essential for storing, managing, and operating on sets of variables.

Installation:

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

Runtime Configuration:

This extension has no configuration directives defined in php.ini.

PHP Array Constants:

Constant	Description
CASE_LOWER	Used with array_change_key_case() to convert array keys to lower case
CASE_UPPER	Used with array_change_key_case() to convert array keys to upper case
SORT_ASC	Used with array_multisort() to sort in ascending order
SORT_DESC	Used with array_multisort() to sort in descending order
SORT_REGULAR	Used to compare items normally
SORT_NUMERIC	Used to compare items numerically
SORT_STRING	Used to compare items as strings
SORT_LOCALE_STRING	Used to compare items as strings, based on the current locale
COUNT_NORMAL	
COUNT_RECURSIVE	
EXTR_OVERWRITE	
EXTR_SKIP	
EXTR_PREFIX_SAME	
EXTR_PREFIX_ALL	
EXTR_PREFIX_INVALID	
EXTR_PREFIX_IF_EXISTS	
EXTR_IF_EXISTS	
EXTR_REFS	

List of Functions:
PHP: indicates the earliest version of PHP that supports the function.

Function	Description	PHP
array()	Create an array	3
array_change_key_case()	Returns an array with all keys in lowercase or uppercase	4
array_chunk()	Splits an array into chunks of arrays	4
array_combine()	Creates an array by using one array for keys and another for its values	5
array_count_values()	Returns an array with the number of occurrences for each value	4
array_diff()	Compares array values, and returns the differences	4
array_diff_assoc()	Compares array keys and values, and returns the differences	4
array_diff_key()	Compares array keys, and returns the differences	5
array_diff_uassoc()	Compares array keys and values, with an additional user-made function check, and returns the differences	5
array_diff_ukey()	Compares array keys, with an additional user-made function check, and returns the differences	5
array_fill()	Fills an array with values	4
array_fill_keys()	Fill an array with values, specifying keys	5
array_filter()	Filters elements of an array using a user-made function	4
array_flip()	Exchanges all keys with their associated values in an array	4
array_intersect()	Compares array values, and returns the matches	4
array_intersect_assoc()	Compares array keys and values, and returns the matches	4
array_intersect_key()	Compares array keys, and returns the matches	5
array_intersect_uassoc()	Compares array keys and values, with an additional user-made function check, and returns the matches	5
array_intersect_ukey()	Compares array keys, with an additional user-made function check, and returns the matches	5
array_key_exists()	Checks if the specified key exists in the array	4
array_keys()	Returns all the keys of an array	4
array_map()	Sends each value of an array to a user-made function, which returns new values	4
array_merge()	Merges one or more arrays into one array	4

array_merge_recursive()	Merges one or more arrays into one array	4
array_multisort()	Sorts multiple or multi-dimensional arrays	4
array_pad()	Inserts a specified number of items, with a specified value, to an array	4
array_pop()	Deletes the last element of an array	4
array_product()	Calculates the product of the values in an array	5
array_push()	Inserts one or more elements to the end of an array	4
array_rand()	Returns one or more random keys from an array	4
array_reduce()	Returns an array as a string, using a user-defined function	4
array_reverse()	Returns an array in the reverse order	4
array_search()	Searches an array for a given value and returns the key	4
array_shift()	Removes the first element from an array, and returns the value of the removed element	4
array_slice()	Returns selected parts of an array	4
array_splice()	Removes and replaces specified elements of an array	4
array_sum()	Returns the sum of the values in an array	4
array_udiff()	Compares array values in a user-made function and returns an array	5
array_udiff_assoc()	Compares array keys, and compares array values in a user-made function, and returns an array	5
array_udiff_uassoc()	Compares array keys and array values in user-made functions, and returns an array	5
array_uintersect()	Compares array values in a user-made function and returns an array	5
array_uintersect_assoc()	Compares array keys, and compares array values in a user-made function, and returns an array	5
array_uintersect_uassoc()	Compares array keys and array values in user-made functions, and returns an array	5
array_unique()	Removes duplicate values from an array	4
array_unshift()	Adds one or more elements to the beginning of an array	4
array_values()	Returns all the values of an array	4
array_walk()	Applies a user function to every member of an array	3
array_walk_recursive()	Applies a user function recursively to every member of an	5

	array	
arsort()	Sorts an array in reverse order and maintain index association	3
asort()	Sorts an array and maintain index association	3
compact()	Create array containing variables and their values	4
count()	Counts elements in an array, or properties in an object	3
current()	Returns the current element in an array	3
each()	Returns the current key and value pair from an array	3
end()	Sets the internal pointer of an array to its last element	3
extract()	Imports variables into the current symbol table from an array	3
in_array()	Checks if a specified value exists in an array	4
key()	Fetches a key from an array	3
krsort()	Sorts an array by key in reverse order	3
ksort()	Sorts an array by key	3
list()	Assigns variables as if they were an array	3
natcasesort()	Sorts an array using a case insensitive "natural order" algorithm	4
natsort()	Sorts an array using a "natural order" algorithm	4
next()	Advance the internal array pointer of an array	3
pos()	Alias of current()	3
prev()	Rewinds the internal array pointer	3
range()	Creates an array containing a range of elements	3
reset()	Sets the internal pointer of an array to its first element	3
rsort()	Sorts an array in reverse order	3
shuffle()	Shuffles an array	3
sizeof()	Alias of count()	3
sort()	Sorts an array	3
uasort()	Sorts an array with a user-defined function and maintain index association	3
uksort()	Sorts an array by keys using a user-defined function	3
usort()	Sorts an array by values using a user-defined function	3

PHP Calendar Functions

The calendar extension presents a series of functions to simplify converting between different calendar formats.

The intermediary or standard it is based on is the Julian Day Count. The Julian Day Count is a count of days starting from January 1st, 4713 B.C. To convert between calendar systems, you must first convert to Julian Day Count, then to the calendar system of your choice

Installation:

To get these functions to work, you have to compile PHP with **--enable-calendar**.

Runtime Configuration:

This extension has no configuration directives defined in php.ini.

PHP Calendar Constants:

Constant	Description	PHP
CAL_GREGORIAN	Gregorian calendar	3
CAL_JULIAN	Julian calendar	3
CAL_JEWISH	Jewish calendar	3
CAL_FRENCH	French Republican calendar	3
CAL_NUM_CALS		3
CAL_DOW_DAYNO		3
CAL_DOW_SHORT		3
CAL_DOW_LONG		3
CAL_MONTH_GREGORIAN_SHORT		3
CAL_MONTH_GREGORIAN_LONG		3
CAL_MONTH_JULIAN_SHORT		3
CAL_MONTH_JULIAN_LONG		3
CAL_MONTH_JEWISH		3
CAL_MONTH_FRENCH		3
CAL_EASTER_DEFAULT		4
CAL_EASTER_DEFAULT		4
CAL_EASTER_ROMAN		4
CAL_EASTER_ALWAYS_GREGORIAN		4

CAL_EASTER_ALWAYS_JULIAN		4
CAL_JEWISH_ADD_ALAFIM_GERESH		5
CAL_JEWISH_ADD_ALAFIM		5
CAL_JEWISH_ADD_GERESHAYIM		5

List of Functions:

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

Function	Description	PHP
cal_days_in_month()	Returns the number of days in a month for a specified year and calendar	4
cal_from_jd()	Converts a Julian day count into a date of a specified calendar	4
cal_info()	Returns information about a given calendar	4
cal_to_jd()	Converts a date to Julian day count	4
easter_date()	Returns the Unix timestamp for midnight on Easter of a specified year	3
easter_days()	Returns the number of days after March 21, on which Easter falls for a specified year	3
FrenchToJD()	Converts a French Republican date to a Julian day count	3
GregorianToJD()	Converts a Gregorian date to a Julian day count	3
JDDayOfWeek()	Returns the day of a week	3
JDMonthName()	Returns a month name	3
JDToFrench()	Converts a Julian day count to a French Republican date	3
JDToGregorian()	Converts a Julian day count to a Gregorian date	3
jdtojewish()	Converts a Julian day count to a Jewish date	3
JDToJulian()	Converts a Julian day count to a Julian date	3
jdtounix()	Converts a Julian day count to a Unix timestamp	4
JewishToJD()	Converts a Jewish date to a Julian day count	3
JulianToJD()	Converts a Julian date to a Julian day count	3
unixtojd()	Converts a Unix timestamp to a Julian day count	4

PHP Class/Object Functions

These functions allow you to obtain information about classes and instance objects. You can obtain the name of the class to which an object belongs, as well as its member properties and methods.

Installation:

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

Runtime Configuration:

This extension has no configuration directives defined in php.ini.

List of Functions:

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

Function	Description	PHP
call_user_method_array()	Call a user method given with an array of parameters [deprecated]	4
call_user_method()	Call a user method on an specific object [deprecated]	4
class_exists()	Checks if the class has been defined	4
get_class_methods()	Gets the class methods' names	4
get_class_vars()	Get the default properties of the class	4
get_class()	Returns the name of the class of an object	4
get_declared_classes()	Returns an array with the name of the defined classes	4
get_declared_interfaces()	Returns an array of all declared interfaces	5
get_object_vars()	Gets the properties of the given object	4
get_parent_class()	Retrieves the parent class name for object or class	4
interface_exists()	Checks if the interface has been defined	5
is_a()	Checks if the object is of this class or has this class as one of its parents	4
is_subclass_of ()	Checks if the object has this class as one of its parents	4
method_exists()	Checks if the class method exists	4
property_exists()	Checks if the object or class has a property	5

PHP Character Functions

The functions provided by this extension check whether a character or string falls into a certain character class according to the current locale.
When called with an integer argument these functions behave exactly like their C counterparts from ctype.h

Installation:

Beginning with PHP 4.2.0 these functions are enabled by default. For older versions you have to configure and compile PHP with **--enable-ctype**. You can disable ctype support with **--disable-ctype.**
Builtin support for ctype is available with PHP 4.3.0.

Runtime Configuration:

This extension has no configuration directives defined in php.ini.

List of Functions:

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

Function	Description	PHP
ctype_alnum()	Check for alphanumeric character(s)	4.0.4
ctype_alpha()	Check for alphabetic character(s)	4.0.4
ctype_cntrl()	Check for control character(s)	4.0.4
ctype_digit()	Check for numeric character(s)	4.0.4
ctype_graph()	Check for any printable character(s) except space	4.0.4
ctype_lower()	Check for lowercase character(s)	4.0.4
ctype_print()	Check for printable character(s)	4.0.4
ctype_punct()	Check for any printable character which is not whitespace or an alphanumeric character	4.0.4
ctype_space()	Check for whitespace character(s)	4.0.4
ctype_upper()	Check for uppercase character(s)	4.0.4
ctype_xdigit()	Check for character(s) representing a hexadecimal digit	4.0.4

PHP Date and Time Functions

These functions allow you to get the date and time from the server where your PHP scripts are running. You can use these functions to format the date and time in many different ways.

Installation:

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

Runtime Configuration

The behavior of the these functions is affected by settings in php.ini. All these parameters are available in PHP version 5 and onwards.

Date/Time configuration options:

Name	Default	Description	Changeable
date.default_latitude	"31.7667"	Specifies the default latitude.	PHP_INI_ALL
date.default_longitude	"35.2333"	Specifies the default longitude	PHP_INI_ALL
date.sunrise_zenith	"90.83"	Specifies the default sunrise zenith	PHP_INI_ALL
date.sunset_zenith	"90.83"	Specifies the default sunset zenith	PHP_INI_ALL
date.timezone	""	Specifies the default timezone	PHP_INI_ALL

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

Function	Description	PHP
checkdate()	Validates a Gregorian date	3
date_create()	Returns new DateTime object	5
date_date_set()	Sets the date	5
date_default_timezone_get()	Returns the default time zone	5
date_default_timezone_set()	Sets the default time zone	5
date_format()	Returns date formatted according to given format	5
date_isodate_set()	Sets the ISO date	5
date_modify()	Alters the timestamp	5
date_offset_get()	Returns the daylight saving time offset	5
date_parse()	Returns associative array with detailed info about given date	5
date_sun_info()	Returns an array with information about sunset/sunrise and twilight begin/end.	5
date_sunrise()	Returns the time of sunrise for a given day / location	5
date_sunset()	Returns the time of sunset for a given day / location	5
date_time_set()	Sets the time	5
date_timezone_get()	Return time zone relative to given DateTime	5

date_timezone_set()	Sets the time zone for the DateTime object	5
date()	Formats a local time/date	3
getdate()	Returns an array that contains date and time information for a Unix timestamp	3
gettimeofday()	Returns an array that contains current time information	3
gmdate()	Formats a GMT/UTC date/time	3
gmmktime()	Returns the Unix timestamp for a GMT date	3
gmstrftime()	Formats a GMT/UTC time/date according to locale settings	3
idate()	Formats a local time/date as integer	5
localtime()	Returns an array that contains the time components of a Unix timestamp	4
microtime()	Returns the microseconds for the current time	3
mktime()	Returns the Unix timestamp for a date	3
strftime()	Formats a local time/date according to locale settings	3
strptime()	Parses a time/date generated with strftime()	5
strtotime()	Parses an English textual date or time into a Unix timestamp	3
time()	Returns the current time as a Unix timestamp	3
timezone_abbreviations_list()	Returns associative array containing dst, offset and the timezone name	5
timezone_identifiers_list()	Returns numerically index array with all timezone identifiers	5
timezone_name_from_abbr()	Returns the timezone name from abbrevation	5
timezone_name_get()	Returns the name of the timezone	5
timezone_offset_get()	Returns the timezone offset from GMT	5
timezone_open()	Returns new DateTimeZone object	5
timezone_transitions_get()	Returns all transitions for the timezone	5

PHP Date / Time Constants:

Constant	Description
DATE_ATOM	Atom (example: 2005-08-15T16:13:03+0000)
DATE_COOKIE	HTTP Cookies (example: Sun, 14 Aug 2005 16:13:03 UTC)

DATE_ISO8601	ISO-8601 (example: 2005-08-14T16:13:03+0000)
DATE_RFC822	RFC 822 (example: Sun, 14 Aug 2005 16:13:03 UTC)
DATE_RFC850	RFC 850 (example: Sunday, 14-Aug-05 16:13:03 UTC)
DATE_RFC1036	RFC 1036 (example: Sunday, 14-Aug-05 16:13:03 UTC)
DATE_RFC1123	RFC 1123 (example: Sun, 14 Aug 2005 16:13:03 UTC)
DATE_RFC2822	RFC 2822 (Sun, 14 Aug 2005 16:13:03 +0000)
DATE_RSS	RSS (Sun, 14 Aug 2005 16:13:03 UTC)
DATE_W3C	World Wide Web Consortium (example: 2005-08-14T16:13:03+0000)
SUNFUNCS_RET_TIMESTAMP	Timestamp (Available in 5.1.2)
SUNFUNCS_RET_STRING	Hours:minutes (example: 08:02) (Available in 5.1.2)
SUNFUNCS_RET_DOUBLE	Hours as floating point number (example 8.75)(Available in 5.1.2)

PHP Directory Functions

These functions are provided to manipulate any directory.

Installation:

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

PHP needs to be configured with --enable-chroot-func option to enable chroot() function.

Runtime Configuration:

This extension has no configuration directives defined in php.ini.

PHP Directory Constants

PHP: indicates the earliest version of PHP that supports the constant.

Constant	Description	PHP
DIRECTORY_SEPARATOR		3
PATH_SEPARATOR		4

List of Functions

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

Function	Description	PHP
chdir()	Changes current directory	4
chroot()	Change the root directory	4.0.4
dir()	Opens a directory handle and returns an object.	4
closedir()	Closes a directory	4
getcwd()	Gets the current working directory.	4
opendir()	Open directory handle	4
readdir()	Read entry from directory handle	4
rewinddir()	Rewind directory handle	4
scandir()	List files and directories inside the specified path	5

NOTE: For more functions related to file system, check File System Functions

PHP Error and Logging Functions

These are functions dealing with error handling and logging. They allow you to define your own error handling rules, as well as modify the way the errors can be logged. This allows you to change and enhance error reporting to suit your needs.

Using these logging functions, you can send messages directly to other machines, to an email, to system logs, etc., so you can selectively log and monitor the most important parts of your applications and websites.

Installation:

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

Runtime Configuration:

The behaviour of these functions is affected by settings in php.ini. These settings are defined below.

Name	Default	Changeable	Changelog
error_reporting	NULL	PHP_INI_ALL	
display_errors	"1"	PHP_INI_ALL	
display_startup_errors	"0"	PHP_INI_ALL	Available since PHP 4.0.3.
log_errors	"0"	PHP_INI_ALL	
log_errors_max_len	"1024"	PHP_INI_ALL	Available since PHP 4.3.0.
ignore_repeated_errors	"0"	PHP_INI_ALL	Available since PHP 4.3.0.

ignore_repeated_source	"0"	PHP_INI_ALL	Available since PHP 4.3.0.
report_memleaks	"1"	PHP_INI_ALL	Available since PHP 4.3.0.
track_errors	"0"	PHP_INI_ALL	
html_errors	"1"	PHP_INI_ALL	PHP_INI_SYSTEM in PHP <= 4.2.3. Available since PHP 4.0.2.
docref_root	""	PHP_INI_ALL	Available since PHP 4.3.0.
docref_ext	""	PHP_INI_ALL	Available since PHP 4.3.2.
error_prepend_string	NULL	PHP_INI_ALL	
error_append_string	NULL	PHP_INI_ALL	
error_log	NULL	PHP_INI_ALL	
warn_plus_overloading	NULL		This option is no longer available as of PHP 4.0.0

PHP Filesystem Functions

The filesystem functions are used to access and manipulate the filesystem PHP provides you all the posible functions you may need to manipulate a file.

Installation:

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

Runtime Configuration:

The behaviour of these functions is affected by settings in php.ini.

Name	Default	Changeable	Changelog
allow_url_fopen	"1"	PHP_INI_ALL	PHP_INI_ALL in PHP <= 4.3.4. PHP_INI_SYSTEM in PHP < 6. Available since PHP 4.0.4.
allow_url_include	"0"	PHP_INI_ALL	PHP_INI_SYSTEM in PHP 5. Available since PHP 5.2.0.
user_agent	NULL	PHP_INI_ALL	Available since PHP 4.0.3.
default_socket_timeout	"60"	PHP_INI_ALL	Available since PHP 4.3.0.
from	""	PHP_INI_ALL	
auto_detect_line_endings	"0"	PHP_INI_ALL	Available since PHP 4.3.0.

PHP Error and Logging Constants:

PHP: indicates the earliest version of PHP that supports the constant.
You can use any of the constant while configuring your php.ini file.

Constant	Description	PHP
GLOB_BRACE		
GLOB_ONLYDIR		
GLOB_MARK		
GLOB_NOSORT		
GLOB_NOCHECK		
GLOB_NOESCAPE		
PATHINFO_DIRNAME		
PATHINFO_BASENAME		
PATHINFO_EXTENSION		
PATHINFO_FILENAME		5.2.0
FILE_USE_INCLUDE_PATH	Search for filename in include_path	5.0.0
FILE_APPEND	Append content to existing file.	
FILE_IGNORE_NEW_LINES	Strip EOL characters	5.0.0
FILE_SKIP_EMPTY_LINES	Skip empty lines	5.0.0
FILE_BINARY	Binary mode	6.0.0
FILE_TEXT	Text mode	6.0.0

List of Functions

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

Function	Description	PHP
basename()	Returns filename component of path	3
chgrp()	Changes file group	3
chmod()	Changes file mode	3
chown()	Changes file owner	3
clearstatcache()	Clears file status cache	3
copy()	Copies file	3
delete()	Deletes file	
dirname()	Returns directory name component of path	3
disk_free_space()	Returns available space in directory	4.0.7

disk_total_space()	Returns the total size of a directory	4.0.7
diskfreespace()	Alias of disk_free_space()	4.0.7
fclose()	Closes an open file pointer	3
feof()	Tests for end-of-file on a file pointer	3
fflush()	Flushes the output to a file	4
fgetc()	Gets character from file pointer	3
fgetcsv()	Gets line from file pointer and parse for CSV fields	3
fgets()	Gets line from file pointer	3
fgetss()	Gets line from file pointer and strip HTML tags	3
file_exists()	Checks whether a file or directory exists	3
file_get_contents()	Reads entire file into a string	4.3.0
file_put_contents()	Write a string to a file	5
file()	Reads entire file into an array	3
fileatime()	Gets last access time of file	3
filectime()	Gets inode change time of file	3
filegroup()	Gets file group	3
fileinode()	Gets file inode	3
filemtime()	Gets file modification time	3
fileowner()	Gets file owner	3
fileperms()	Gets file permissions	3
filesize()	Gets file size	3
filetype()	Gets file type	3
flock()	Portable advisory file locking	3
fnmatch()	Match filename against a pattern	4.0.3
fopen()	Opens file or URL	3
fpassthru()	Output all remaining data on a file pointer	3
fputcsv()	Format line as CSV and write to file pointer	5.1.0
fputs()	Alias of fwrite()	3
fread()	Binary-safe file read	3
fscanf()	Parses input from a file according to a format	4.0.1

fseek()	Seeks on a file pointer	3
fstat()	Gets information about a file using an open file pointer	4
ftell()	Tells file pointer read/write position	3
ftruncate()	Truncates a file to a given length	4
fwrite()	Binary-safe file write	3
glob()	Find pathnames matching a pattern	4.0.3
is_dir()	Tells whether the filename is a directory	3
is_executable()	Tells whether the filename is executable	3
is_file()	Tells whether the filename is a regular file	3
is_link()	Tells whether the filename is a symbolic link	3
is_readable()	Tells whether the filename is readable	3
is_uploaded_file()	Tells whether the file was uploaded via HTTP POST	4.0.3
is_writable()	Tells whether the filename is writable	3
is_writeable()	Alias of is_writable()	3
lchgrp()	Changes group ownership of symlink	5.1.2
lchown()	Changes user ownership of symlink	5.1.2
link()	Create a hard link	3
linkinfo()	Gets information about a link	3
lstat()	Gives information about a file or symbolic link	3
mkdir()	Makes directory	3
move_uploaded_file()	Moves an uploaded file to a new location	4.0.3
parse_ini_file()	Parse a configuration file	4
pathinfo()	Returns information about a file path	4.0.3
pclose()	Closes process file pointer	3
popen()	Opens process file pointer	3
readfile()	Outputs a file	3
readlink()	Returns the target of a symbolic link	3
realpath()	Returns canonicalized absolute pathname	4
rename()	Renames a file or directory	3
rewind()	Rewind the position of a file pointer	3

rmdir()	Removes directory	3
set_file_buffer()	Alias of stream_set_write_buffer()	3
stat()	Gives information about a file	3
symlink()	Creates a symbolic link	3
tempnam()	Create file with unique file name	3
tmpfile()	Creates a temporary file	3
touch()	Sets access and modification time of file	3
umask()	Changes the current umask	3
unlink()	Deletes a file	3

PHP Error and Logging Constants:

PHP: indicates the earliest version of PHP that supports the constant.

You can use any of the constant while configuring your php.ini file.

Value	Constant	Description	PHP
1	E_ERROR	Fatal run-time errors. Errors that cannot be recovered from. Execution of the script is halted	
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted	
4	E_PARSE	Compile-time parse errors. Parse errors should only be generated by the parser	
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally	
16	E_CORE_ERROR	Fatal errors at PHP startup. This is like an E_ERROR in the PHP core	4
32	E_CORE_WARNING	Non-fatal errors at PHP startup. This is like an E_WARNING in the PHP core	4
64	E_COMPILE_ERROR	Fatal compile-time errors. This is like an E_ERROR generated by the Zend Scripting Engine	4
128	E_COMPILE_WARNING	Non-fatal compile-time errors. This is like an E_WARNING generated by the Zend Scripting Engine	4
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()	4
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an	4

		E_WARNING set by the programmer using the PHP function trigger_error()	
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()	4
2048	E_STRICT	Run-time notices. PHP suggest changes to your code to help interoperability and compatibility of the code	5
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())	5
8191	E_ALL	All errors and warnings, except of level E_STRICT	5

List of Functions

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

Function	Description	PHP
debug_backtrace()	Generates a backtrace	4
debug_print_backtrace()	Prints a backtrace	5
error_get_last()	Gets the last error occurred	5
error_log()	Sends an error to the server error-log, to a file or to a remote destination	4
error_reporting()	Specifies which errors are reported	4
restore_error_handler()	Restores the previous error handler	4
restore_exception_handler()	Restores the previous exception handler	5
set_error_handler()	Sets a user-defined function to handle errors	4
set_exception_handler()	Sets a user-defined function to handle exceptions	5
trigger_error()	Creates a user-defined error message	4
user_error()	Alias of trigger_error()	4

MySQLi Functions

PHP MySQLi Introduction

PHP MySQLi = PHP MySQL Improved!

The MySQLi functions allows you to access MySQL database servers.

Note: The MySQLi extension is designed to work with MySQL version 4.1.13 or newer.

Installation / Runtime Configuration

For the MySQLi functions to be available, you must compile PHP with support for the MySQLi extension.

The MySQLi extension was introduced with PHP version 5.0.0. The MySQL Native Driver was included in PHP version 5.3.0.

For installation details, go to: http://www.php.net/manual/en/mysqli.installation.php

For runtime configuration details, go to: http://www.php.net/manual/en/mysqli.configuration.php

PHP 5 MySQLi Functions

Function	Description
mysqli_affected_rows()	Returns the number of affected rows in the previous MySQL operation
mysqli_autocommit()	Turns on or off auto-committing database modifications
mysqli_change_user()	Changes the user of the specified database connection
mysqli_character_set_name()	Returns the default character set for the database connection
mysqli_close()	Closes a previously opened database connection
mysqli_commit()	Commits the current transaction
mysqli_connect_errno()	Returns the error code from the last connection error
mysqli_connect_error()	Returns the error description from the last connection error
mysqli_connect()	Opens a new connection to the MySQL server
mysqli_data_seek()	Adjusts the result pointer to an arbitrary row in the result-set
mysqli_debug()	Performs debugging operations
mysqli_dump_debug_info()	Dumps debugging info into the log
mysqli_errno()	Returns the last error code for the most recent function call
mysqli_error_list()	Returns a list of errors for the most recent function call
mysqli_error()	Returns the last error description for the most recent function call
mysqli_fetch_all()	Fetches all result rows as an associative array, a numeric array, or both
mysqli_fetch_array()	Fetches a result row as an associative, a numeric array, or both
mysqli_fetch_assoc()	Fetches a result row as an associative array
mysqli_fetch_field_direct()	Returns meta-data for a single field in the result set, as an object
mysqli_fetch_field()	Returns the next field in the result set, as an object
mysqli_fetch_fields()	Returns an array of objects that represent the fields in a result set
mysqli_fetch_lengths()	Returns the lengths of the columns of the current row in the result set
mysqli_fetch_object()	Returns the current row of a result set, as an object
mysqli_fetch_row()	Fetches one row from a result-set and returns it as an enumerated array
mysqli_field_count()	Returns the number of columns for the most recent query
mysqli_field_seek()	Sets the field cursor to the given field offset
mysqli_field_tell()	Returns the position of the field cursor
mysqli_free_result()	Frees the memory associated with a result

mysqli_get_charset()	Returns a character set object
mysqli_get_client_info()	Returns the MySQL client library version
mysqli_get_client_stats()	Returns statistics about client per-process
mysqli_get_client_version()	Returns the MySQL client library version as an integer
mysqli_get_connection_stats()	Returns statistics about the client connection
mysqli_get_host_info()	Returns the MySQL server hostname and the connection type
mysqli_get_proto_info()	Returns the MySQL protocol version
mysqli_get_server_info()	Returns the MySQL server version
mysqli_get_server_version()	Returns the MySQL server version as an integer
mysqli_info()	Returns information about the most recently executed query
mysqli_init()	Initializes MySQLi and returns a resource for use with mysqli_real_connect()
mysqli_insert_id()	Returns the auto-generated id used in the last query
mysql_kill()	Asks the server to kill a MySQL thread
mysqli_more_results()	Checks if there are more results from a multi query
mysqli_multi_query()	Performs one or more queries on the database
mysqli_next_result()	Prepares the next result set from mysqli_multi_query()
mysqli_num_fields()	Returns the number of fields in a result set
mysqli_num_rows()	Returns the number of rows in a result set
mysqli_options()	Sets extra connect options and affect behavior for a connection
mysqli_ping()	Pings a server connection, or tries to reconnect if the connection has gone down
mysqli_prepare()	Prepares an SQL statement for execution
mysqli_query()	Performs a query against the database
mysqli_real_connect()	Opens a new connection to the MySQL server
mysqli_real_escape_string()	Escapes special characters in a string for use in an SQL statement
mysqli_real_query()	Executes an SQL query
mysqli_reap_async_query()	Returns the result from async query
mysqli_refresh()	Refreshes tables or caches, or resets the replication server information
mysqli_rollback()	Rolls back the current transaction for the database
mysqli_select_db()	Changes the default database for the connection
mysqli_set_charset()	Sets the default client character set
mysqli_set_local_infile_default()	Unsets user defined handler for load local infile command
mysqli_set_local_infile_handler()	Set callback function for LOAD DATA LOCAL INFILE command
mysqli_sqlstate()	Returns the SQLSTATE error code for the last MySQL operation
mysqli_ssl_set()	Used to establish secure connections using SSL
mysqli_stat()	Returns the current system status
mysqli_stmt_init()	Initializes a statement and returns an object for use with mysqli_stmt_prepare()
mysqli_store_result()	Transfers a result set from the last query
mysqli_thread_id()	Returns the thread ID for the current connection

mysqli_thread_safe()	Returns whether the client library is compiled as thread-safe
mysqli_use_result()	Initiates the retrieval of a result set from the last query executed using the mysqli_real_query()
mysqli_warning_count()	Returns the number of warnings from the last query in the connection

FTP Functions

PHP FTP Introduction

The FTP functions give client access to file servers through the File Transfer Protocol (FTP).

The FTP functions are used to open, login and close connections, as well as upload, download, rename, delete, and get information on files from file servers. Not all of the FTP functions will work with every server or return the same results. The FTP functions became available with PHP 3.

If you only wish to read from or write to a file on an FTP server, consider using the ftp:// wrapper with the Filesystem functions which provide a simpler and more intuitive interface.

Installation

For these functions to work, you have to compile PHP with --enable-ftp.
The Windows version of PHP has built-in support for this extension.

PHP 5 FTP Functions

Function	Description
ftp_alloc()	Allocates space for a file to be uploaded to the FTP server
ftp_cdup()	Changes to the parent directory on the FTP server
ftp_chdir()	Changes the current directory on the FTP server
ftp_chmod()	Sets permissions on a file via FTP
ftp_close()	Closes an FTP connection
ftp_connect()	Opens an FTP connection
ftp_delete()	Deletes a file on the FTP server
ftp_exec()	Executes a command on the FTP server
ftp_fget()	Downloads a file from the FTP server and saves it into an open local file
ftp_fput()	Uploads from an open file and saves it to a file on the FTP server
ftp_get_option()	Returns runtime options of the FTP connection
ftp_get()	Downloads a file from the FTP server
ftp_login()	Logs in to the FTP connection
ftp_mdtm()	Returns the last modified time of a specified file
ftp_mkdir()	Creates a new directory on the FTP server
ftp_nb_continue()	Continues retrieving/sending a file (non-blocking)
ftp_nb_fget()	Downloads a file from the FTP server and saves it into an open file (non-blocking)
ftp_nb_fput()	Uploads from an open file and saves it to a file on the FTP server (non-blocking)

ftp_nb_get()	Downloads a file from the FTP server (non-blocking)
ftp_nb_put()	Uploads a file to the FTP server (non-blocking)
ftp_nlist()	Returns a list of files in the specified directory on the FTP server
ftp_pasv()	Turns passive mode on or off
ftp_put()	Uploads a file to the FTP server
ftp_pwd()	Returns the current directory name
ftp_quit()	An alias of ftp_close()
ftp_raw()	Sends a raw command to the FTP server
ftp_rawlist()	Returns a list of files with file information from a specified directory
ftp_rename()	Renames a file or directory on the FTP server
ftp_rmdir()	Deletes an empty directory on the FTP server
ftp_set_option()	Sets runtime options for the FTP connection
ftp_site()	Sends an FTP SITE command to the FTP server
ftp_size()	Returns the size of the specified file
ftp_ssl_connect()	Opens a secure SSL-FTP connection
ftp_systype()	Returns the system type identifier of the FTP server

PHP 5 Predefined FTP Constants

Constant	Type	PHP
FTP_ASCII	Integer	PHP 3
FTP_TEXT	Integer	PHP 3
FTP_BINARY	Integer	PHP 3
FTP_IMAGE	Integer	PHP 3
FTP_TIMEOUT_SEC	Integer	PHP 3
FTP_AUTOSEEK	Integer	PHP 4.3
FTP_AUTORESUME	Integer	PHP 4.3
FTP_FAILED	Integer	PHP 4.3
FTP_FINISHED	Integer	PHP 4.3
FTP_MOREDATA	Integer	PHP 4.3

HTTP Functions

PHP HTTP Introduction

The HTTP functions let you manipulate information sent to the browser by the Web server, before any other output has been sent.

Installation

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

PHP HTTP Functions

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

Function	Description	PHP
header()	Sends a raw HTTP header to a client	3
headers_list()	Returns a list of response headers sent (or ready to send)	5
headers_sent()	Checks if / where the HTTP headers have been sent	3
setcookie()	Sends an HTTP cookie to a client	3
setrawcookie()	Sends an HTTP cookie without URL encoding the cookie value	5

PHP 5 Filter Functions

PHP Filter Introduction

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

Installation

As of PHP 5.2.0, the filter functions are enabled by default. There is no installation needed to use these functions.

Runtime Configurations

The behavior of these functions is affected by settings in php.ini

Name	Description	Default	Changeable
filter.default	Filter all $_GET, $_POST, $_COOKIE, $_REQUEST and $_SERVER data by this filter. Accepts the name of the filter you like to use by default. See the filter list for the list of the filter names	"unsafe_raw"	PHP_INI_PERDIR
filter.default_flags	The default latitude (used by date_sunrise() and date_sunset())	NULL	PHP_INI_PERDIR

PHP 5 Filter Functions

Function	Description
filter_has_var()	Checks if a variable of a specified input type exist
filter_id()	Returns the ID number of a specified filter
filter_input()	Get input from outside the script and filter it
filter_input_array()	Get multiple inputs from outside the script and filters them
filter_list()	Returns an array of all supported filters
filter_var_array()	Get multiple variables and filter them
filter_var()	Get a variable and filter it

PHP Filter List

Validate Filters:

ID Name	Description
FILTER_VALIDATE_BOOLEAN	Return TRUE for "1", "true", "on" and "yes", FALSE for "0", "false", "off", "no", and "", NULL otherwise
FILTER_VALIDATE_EMAIL	Validate value as e-mail
FILTER_VALIDATE_FLOAT	Validate value as float
FILTER_VALIDATE_INT	Validate value as integer, optionally from the specified range
FILTER_VALIDATE_IP	Validate value as IP address, optionally only IPv4 or IPv6 or not from private or reserved ranges
FILTER_VALIDATE_REGEXP	Validate value against regexp, a Perl-compatible regular expression
FILTER_VALIDATE_URL	Validate value as URL, optionally with required components

Sanitize Filters:

ID Name	Description
FILTER_SANITIZE_EMAIL	Remove all characters, except letters, digits and !#$%&'*+-/=?^_`{|}~@.[]
FILTER_SANITIZE_ENCODED	URL-encode string, optionally strip or encode special characters
FILTER_SANITIZE_MAGIC_QUOTES	Apply addslashes()
FILTER_SANITIZE_NUMBER_FLOAT	Remove all characters, except digits, +- and optionally .,eE
FILTER_SANITIZE_NUMBER_INT	Remove all characters, except digits and +-
FILTER_SANITIZE_SPECIAL_CHARS	HTML-escape '"<>& and characters with ASCII value less than 32
FILTER_SANITIZE_FULL_SPECIAL_CHARS	
FILTER_SANITIZE_STRING	Strip tags, optionally strip or encode special characters
FILTER_SANITIZE_STRIPPED	Alias of "string" filter
FILTER_SANITIZE_URL	Remove all characters, except letters, digits and $-_.+!*'(),{}|\\^~[]`<>#%";/?:@&=
FILTER_UNSAFE_RAW	Do nothing, optionally strip or encode special characters

Other Filters:

ID Name	Description
FILTER_CALLBACK	Call a user-defined function to filter data

PHP libxml Functions

The libxml functions and constants are used together with SimpleXML, XSLT and DOM functions.

Installation

These functions require the libxml package. Download at xmlsoft.org

PHP libxml Functions

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

Function	Description	PHP
libxml_clear_errors()	Clear libxml error buffer	5
libxml_get_errors()	Retrieve array of errors	5
libxml_get_last_error()	Retrieve last error from libxml	5
libxml_set_streams_context()	Set the streams context for the next libxml document load or write	5
libxml_use_internal_errors()	Disable libxml errors and allow user to fetch error information as needed	5

PHP libxml Constants

Function	Description	PHP
LIBXML_COMPACT	Set small nodes allocation optimization. This may improve the application performance	5
LIBXML_DTDATTR	Set default DTD attributes	5
LIBXML_DTDLOAD	Load external subset	5
LIBXML_DTDVALID	Validate with the DTD	5
LIBXML_NOBLANKS	Remove blank nodes	5
LIBXML_NOCDATA	Set CDATA as text nodes	5
LIBXML_NOEMPTYTAG	Change empty tags (e.g. to </br>), only available in the DOMDocument->save() and DOMDocument->saveXML() functions	5
LIBXML_NOENT	Substitute entities	5
LIBXML_NOERROR	Do not show error reports	5
LIBXML_NONET	Stop network access while loading documents	5
LIBXML_NOWARNING	Do not show warning reports	5
LIBXML_NOXMLDECL	Drop the XML declaration when saving a document	5
LIBXML_NSCLEAN	Remove excess namespace declarations	5
LIBXML_XINCLUDE	Use XInclude substitution	5
LIBXML_ERR_ERROR	Get recoverable errors	5
LIBXML_ERR_FATAL	Get fatal errors	5
LIBXML_ERR_NONE	Get no errors	5
LIBXML_ERR_WARNING	Get simple warnings	5
LIBXML_VERSION	Get libxml version (e.g. 20605 or 20617)	5
LIBXML_DOTTED_VERSION	Get dotted libxml version (e.g. 2.6.5 or 2.6.17)	

PHP 5 Mail Functions

The mail() function allows you to send emails directly from a script.

Requirements

For the mail functions to be available, PHP requires an installed and working email system. The program to be used is defined by the configuration settings in the php.ini file.

Installation

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

Runtime Configuration

The behavior of the mail functions is affected by settings in php.ini:

Name	Default	Description	Changeable
mail.add_x_header	"0"	Add X-PHP-Originating-Script that will include UID of the script followed by the filename. For PHP 5.3.0 and above	PHP_INI_PERDIR
mail.log	NULL	The path to a log file that will log all mail() calls. Log include full path of script, line number, To address and headers. For PHP 5.3.0 and above	PHP_INI_PERDIR
SMTP	"localhost"	Windows only: The DNS name or IP address of the SMTP server	PHP_INI_ALL
smtp_port	"25"	Windows only: The SMTP port number. For PHP 4.3.0 and above	PHP_INI_ALL
sendmail_from	NULL	Windows only: Specifies the "from" address to be used when sending mail from mail()	PHP_INI_ALL
sendmail_path	"/usr/sbin/sendmail -t -i"	Specifies where the sendmail program can be found. This directive works also under Windows. If set, SMTP, smtp_port and sendmail_from are ignored	PHP_INI_SYSTEM

PHP 5 Mail Functions

Function	Description
ezmlm_hash()	Calculates the hash value needed by EZMLM
mail()	Allows you to send emails directly from a script

PHP 5 Math Functions

The math functions can handle values within the range of integer and float types.

Installation

The PHP math functions are part of the PHP core. No installation is required to use these functions.

PHP 5 Math Functions

Function	Description
abs()	Returns the absolute (positive) value of a number
acos()	Returns the arc cosine of a number
acosh()	Returns the inverse hyperbolic cosine of a number
asin()	Returns the arc sine of a number
asinh()	Returns the inverse hyperbolic sine of a number
atan()	Returns the arc tangent of a number in radians
atan2()	Returns the arc tangent of two variables x and y
atanh()	Returns the inverse hyperbolic tangent of a number
base_convert()	Converts a number from one number base to another
bindec()	Converts a binary number to a decimal number
ceil()	Rounds a number up to the nearest integer
cos()	Returns the cosine of a number
cosh()	Returns the hyperbolic cosine of a number
decbin()	Converts a decimal number to a binary number
dechex()	Converts a decimal number to a hexadecimal number
decoct()	Converts a decimal number to an octal number
deg2rad()	Converts a degree value to a radian value
exp()	Calculates the exponent of e
expm1()	Returns exp(x) - 1
floor()	Rounds a number down to the nearest integer
fmod()	Returns the remainder of x/y
getrandmax()	Returns the largest possible value returned by rand()
hexdec()	Converts a hexadecimal number to a decimal number
hypot()	Calculates the hypotenuse of a right-angle triangle
is_finite()	Checks whether a value is finite or not
is_infinite()	Checks whether a value is infinite or not
is_nan()	Checks whether a value is 'not-a-number'
lcg_value()	Returns a pseudo random number in a range between 0 and 1
log()	Returns the natural logarithm of a number
log10()	Returns the base-10 logarithm of a number
log1p()	Returns log(1+number)
max()	Returns the highest value in an array, or the highest value of several specified values
min()	Returns the lowest value in an array, or the lowest value of several specified values
mt_getrandmax()	Returns the largest possible value returned by mt_rand()
mt_rand()	Generates a random integer using Mersenne Twister algorithm
mt_srand()	Seeds the Mersenne Twister random number generator

octdec()	Converts an octal number to a decimal number
pi()	Returns the value of PI
pow()	Returns x raised to the power of y
rad2deg()	Converts a radian value to a degree value
rand()	Generates a random integer
round()	Rounds a floating-point number
sin()	Returns the sine of a number
sinh()	Returns the hyperbolic sine of a number
sqrt()	Returns the square root of a number
srand()	Seeds the random number generator
tan()	Returns the tangent of a number
tanh()	Returns the hyperbolic tangent of a number

PHP 5 Predefined Math Constants

Constant	Value	Description	PHP Version
INF	INF	The infinite	PHP 4
M_E	2.7182818284590452354	Returns e	PHP 4
M_EULER	0.57721566490153286061	Returns Euler constant	PHP 4
M_LNPI	1.14472988584940017414	Returns the natural logarithm of PI: log_e(pi)	PHP 5.2
M_LN2	0.69314718055994530942	Returns the natural logarithm of 2: log_e 2	PHP 4
M_LN10	2.30258509299404568402	Returns the natural logarithm of 10: log_e 10	PHP 4
M_LOG2E	1.4426950408889634074	Returns the base-2 logarithm of E: log_2 e	PHP 4
M_LOG10E	0.43429448190325182765	Returns the base-10 logarithm of E: log_10 e	PHP 4
M_PI	3.14159265358979323846	Returns Pi	PHP 4
M_PI_2	1.57079632679489661923	Returns Pi/2	PHP 4
M_PI_4	0.78539816339744830962	Returns Pi/4	PHP 4
M_1_PI	0.31830988618379067154	Returns 1/Pi	PHP 4
M_2_PI	0.63661977236758134308	Returns 2/Pi	PHP 4
M_SQRTPI	1.77245385090551602729	Returns the square root of PI: sqrt(pi)	PHP 5.2
M_2_SQRTPI	1.12837916709551257390	Returns 2/square root of PI: 2/sqrt(pi)	PHP 4
M_SQRT1_2	0.70710678118654752440	Returns the square root of 1/2: 1/sqrt(2)	PHP 4
M_SQRT2	1.41421356237309504880	Returns the square root of 2: sqrt(2)	PHP 4
M_SQRT3	1.73205080756887729352	Returns the square root of 3: sqrt(3)	PHP 5.2

NAN	NAN	Not A Number	PHP 4
PHP_ROUND_HALF_UP	1	Round halves up	PHP 5.3
PHP_ROUND_HALF_DOWN	2	Round halves down	PHP 5.3
PHP_ROUND_HALF_EVEN	3	Round halves to even numbers	PHP 5.3
PHP_ROUND_HALF_ODD	4	Round halves to odd numbers	PHP 5.3

PHP 5 Misc. Functions

PHP Miscellaneous Introduction

The misc. functions were only placed here because none of the other categories seemed to fit.

Installation

The misc. functions are part of the PHP core. No installation is required to use these functions.

Runtime Configuration

The behavior of the misc. functions is affected by settings in the php.ini file.

Misc. configuration options:

Name	Description	Default	Changeable
ignore_user_abort	FALSE indicates that scripts will be terminated as soon as they try to output something after a client has aborted their connection	"0"	PHP_INI_ALL
highlight.string	Color for highlighting a string in PHP syntax	"#DD0000"	PHP_INI_ALL
highlight.comment	Color for highlighting PHP comments	"#FF8000"	PHP_INI_ALL
highlight.keyword	Color for syntax highlighting PHP keywords (e.g. parenthesis and semicolon)	"#007700"	PHP_INI_ALL
highlight.bg	Removed in PHP 5.4.0. Color for background	"#FFFFFF"	PHP_INI_ALL
highlight.default	Default color for PHP syntax	"#0000BB"	PHP_INI_ALL
highlight.html	Color for HTML code	"#000000"	PHP_INI_ALL
browscap	Name and location of browser-capabilities file (e.g. browscap.ini)	NULL	PHP_INI_SYSTEM

PHP Miscellaneous Functions

Function	Description
connection_aborted()	Checks whether the client has disconnected
connection_status()	Returns the current connection status
connection_timeout()	Deprecated in PHP 4.0.5. Checks whether the script has timed out
constant()	Returns the value of a constant
define()	Defines a constant

defined()	Checks whether a constant exists
die()	Prints a message and exits the current script
eval()	Evaluates a string as PHP code
exit()	Prints a message and exits the current script
get_browser()	Returns the capabilities of the user's browser
__halt_compiler()	Halts the compiler execution
highlight_file()	Outputs a file with the PHP syntax highlighted
highlight_string()	Outputs a string with the PHP syntax highlighted
ignore_user_abort()	Sets whether a remote client can abort the running of a script
pack()	Packs data into a binary string
php_check_syntax()	Deprecated in PHP 5.0.5
php_strip_whitespace()	Returns the source code of a file with PHP comments and whitespace removed
show_source()	Alias of highlight_file()
sleep()	Delays code execution for a number of seconds
sys_getloadavg()	Gets system load average
time_nanosleep()	Delays code execution for a number of seconds and nanoseconds
time_sleep_until()	Delays code execution until a specified time
uniqid()	Generates a unique ID
unpack()	Unpacks data from a binary string
usleep()	Delays code execution for a number of microseconds

PHP Misc. Constants

PHP: indicates the earliest version of PHP that supports the constant.

Constant	Description	PHP
CONNECTION_ABORTED		
CONNECTION_NORMAL		
CONNECTION_TIMEOUT		
__COMPILER_HALT_OFFSET__		5

PHP 5 SimpleXML Functions

The SimpleXML extension provides is a simple way of getting an XML element's name and text, if you know the XML document's layout.
SimpleXML converts an XML document into a SimpleXMLElement object.
This object can then be processed, like any other object, with normal property selectors and array iterators.
Tip: Compared to DOM or the Expat parser, SimpleXML just takes a few lines of code to read text data from an element.

Installation

The SimpleXML extension requires PHP 5.

As of PHP 5, the SimpleXML functions are part of the PHP core. No installation is required to use these functions.

PHP 5 SimpleXML Functions

Function	Description
__construct()	Creates a new SimpleXMLElement object
addAttribute()	Adds an attribute to the SimpleXML element
addChild()	Adds a child element the SimpleXML element
asXML()	Formats the SimpleXML object's data in XML (version 1.0)
attributes()	Returns attributes and values within an XML tag
children()	Finds the children of a specified node
count()	Counts the children of a specified node
getDocNamespaces()	Returns the namespaces DECLARED in document
getName()	Returns the name of the XML tag referenced by the SimpleXML element
getNamespaces()	Returns the namespaces USED in document
registerXPathNamespace()	Creates a namespace context for the next XPath query
saveXML()	Alias of asXML()
simplexml_import_dom()	Returns a SimpleXMLElement object from a DOM node
simplexml_load_file()	Converts an XML file into a SimpleXMLElement object
simplexml_load_string()	Converts an XML string into a SimpleXMLElement object
xpath()	Runs an XPath query on XML data

PHP 5 SimpleXML Iteration Functions

Function	Description
current()	Returns the current element
getChildren()	Returns the child elements of the current element
hasChildren()	Cheks whether the current element has children
key()	Return the current key
next()	Moves to the next element
rewind()	Rewind to the first element
valid()	Check whether the current element is valid

PHP 5 String Functions

The PHP string functions are part of the PHP core. No installation is required to use these functions.

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()	Removes whitespace or other characters from the right end of a string

chr()	Returns a character from a specified ASCII value
chunk_split()	Splits a string into a series of smaller parts
convert_cyr_string()	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 information about characters used in a string
crc32()	Calculates a 32-bit CRC for a string
crypt()	One-way string encryption (hashing)
echo()	Outputs one or more 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
hex2bin()	Converts a string of hexadecimal values to ASCII characters
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()
lcfirst()	Converts the first character of a string to lowercase
levenshtein()	Returns the Levenshtein distance between two strings
localeconv()	Returns locale numeric and monetary formatting information
ltrim()	Removes whitespace or other characters from the left side of a string
md5()	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_format()	Returns a string formatted as a currency string
nl_langinfo()	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 one or more strings
printf()	Outputs a formatted string
quoted_printable_decode()	Converts a quoted-printable string to an 8-bit string
quoted_printable_encode()	Converts an 8-bit string to a quoted printable string
quotemeta()	Quotes meta characters

rtrim()	Removes whitespace or other characters 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_getcsv()	Parses a CSV string into an array
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
str_replace()	Replaces some characters in a string (case-sensitive)
str_rot13()	Performs 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()	Compares two strings (locale based string comparison)
strcspn()	Returns the number of characters found in a string before any part 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 characters (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 a 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 charlist
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_compare()	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()	Removes whitespace or other characters from both sides of a string
ucfirst()	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
vsprintf()	Writes a formatted string to a variable
wordwrap()	Wraps a string to a given number of characters

PHP XML Parser Functions

The XML functions lets you parse, but not validate, XML documents.
XML is a data format for standardized structured document exchange. This extension uses the Expat XML parser.
Expat is an event-based parser, it views an XML document as a series of events. When an event occurs, it calls a specified function to handle it.
Expat is a non-validating parser, and ignores any DTDs linked to a document. However, if the document is not well formed it will end with an error message.
Because it is an event-based, non validating parser, Expat is fast and well suited for web applications.
The XML parser functions lets you create XML parsers and define handlers for XML events.

Installation

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

PHP XML Parser Functions

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

Function	Description	PHP
utf8_decode()	Decodes an UTF-8 string to ISO-8859-1	3
utf8_encode()	Encodes an ISO-8859-1 string to UTF-8	3
xml_error_string()	Gets an error string from the XML parser	3
xml_get_current_byte_index()	Gets the current byte index from the XML parser	3

xml_get_current_column_number()	Gets the current column number from the XML parser	3
xml_get_current_line_number()	Gets the current line number from the XML parser	3
xml_get_error_code()	Gets an error code from the XML parser	3
xml_parse()	Parses an XML document	3
xml_parse_into_struct()	Parse XML data into an array	3
xml_parser_create_ns()	Create an XML parser with namespace support	4
xml_parser_create()	Create an XML parser	3
xml_parser_free()	Free an XML parser	3
xml_parser_get_option()	Get options from an XML parser	3
xml_parser_set_option()	Set options in an XML parser	3
xml_set_character_data_handler()	Set handler function for character data	3
xml_set_default_handler()	Set default handler function	3
xml_set_element_handler()	Set handler function for start and end element of elements	3
xml_set_end_namespace_decl_handler()	Set handler function for the end of namespace declarations	4
xml_set_external_entity_ref_handler()	Set handler function for external entities	3
xml_set_notation_decl_handler()	Set handler function for notation declarations	3
xml_set_object()	Use XML Parser within an object	4
xml_set_processing_instruction_handler()	Set handler function for processing instruction	3
xml_set_start_namespace_decl_handler()	Set handler function for the start of namespace declarations	4
xml_set_unparsed_entity_decl_handler()	Set handler function for unparsed entity declarations	3

PHP XML Parser Constants

Constant
XML_ERROR_NONE (integer)
XML_ERROR_NO_MEMORY (integer)
XML_ERROR_SYNTAX (integer)
XML_ERROR_NO_ELEMENTS (integer)
XML_ERROR_INVALID_TOKEN (integer)
XML_ERROR_UNCLOSED_TOKEN (integer)
XML_ERROR_PARTIAL_CHAR (integer)
XML_ERROR_TAG_MISMATCH (integer)
XML_ERROR_DUPLICATE_ATTRIBUTE (integer)
XML_ERROR_JUNK_AFTER_DOC_ELEMENT (integer)

XML_ERROR_PARAM_ENTITY_REF (integer)
XML_ERROR_UNDEFINED_ENTITY (integer)
XML_ERROR_RECURSIVE_ENTITY_REF (integer)
XML_ERROR_ASYNC_ENTITY (integer)
XML_ERROR_BAD_CHAR_REF (integer)
XML_ERROR_BINARY_ENTITY_REF (integer)
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF (integer)
XML_ERROR_MISPLACED_XML_PI (integer)
XML_ERROR_UNKNOWN_ENCODING (integer)
XML_ERROR_INCORRECT_ENCODING (integer)
XML_ERROR_UNCLOSED_CDATA_SECTION (integer)
XML_ERROR_EXTERNAL_ENTITY_HANDLING (integer)
XML_OPTION_CASE_FOLDING (integer)
XML_OPTION_TARGET_ENCODING (integer)
XML_OPTION_SKIP_TAGSTART (integer)
XML_OPTION_SKIP_WHITE (integer)

PHP Zip File Functions

The Zip files functions allows you to read ZIP files.

Installation

For the Zip file functions to work on your server, these libraries must be installed:

- The ZZIPlib library by Guido Draheim: Download the ZZIPlib library
- The Zip PELC extension: Download the Zip PELC extension

Installation on Linux Systems

PHP 5+: Zip functions and the Zip library is not enabled by default and must be downloaded from the links above. Use the *--with-zip=DIR* configure option to include Zip support.

Installation on Windows Systems

PHP 5+: Zip functions is not enabled by default, so the php_zip.dll and the ZZIPlib library must be downloaded from the link above. php_zip.dll must be enabled inside of php.ini.
To enable any PHP extension, the PHP extension_dir setting (in the php.ini file) should be set to the directory where the PHP extensions are located. An example extension_dir value is c:\php\ext.

PHP Zip File Functions

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

Function	Description	PHP
zip_close()	Closes a ZIP file	4
zip_entry_close()	Closes an entry in the ZIP file	4
zip_entry_compressedsize()	Returns the compressed size of an entry in the ZIP file	4
zip_entry_compressionmethod()	Returns the compression method of an entry in the ZIP file	4
zip_entry_filesize()	Returns the actual file size of an entry in the ZIP file	4
zip_entry_name()	Returns the name of an entry in the ZIP file	4
zip_entry_open()	Opens an entry in the ZIP file for reading	4
zip_entry_read()	Reads from an open entry in the ZIP file	4
zip_open()	Opens a ZIP file	4
zip_read()	Reads the next entry in a ZIP file	4