# INTRODUCTION TO PHP

PHP (recursive acronym for "Hypertext Preprocessor") is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

PHP is one of the most popular server-side scripting languages now; we will take a brief look at the general overview of PHP.

### Features and characteristics of PHP

* **Open source**: Available on the Internet for free.
* **Popularity**: PHP is at least the most popular open source server-side scripting language.
* **Performance**: PHP is typically the fastest server-side script language
* **Database connectivity**: PHP supports wide range of popular databases including MySQL, Oracle, DB2, and ODBC. By using PEAR, you can even write the same code to support all databases supported by PHP.
* **Platform independent**: PHP runs on various OS including Linux, UNIX, Windows and Mac OSX. It also runs with major Web servers including Apache and IIS.
* **Ease of programming**: PHP is very easy for novices and beginners to write practical script from the start. Yet, you can also develop large scale Web applications. Notice how a script written in PHP is different from ones written in other languages like Perl or C; instead of writing a program with lots of commands to output HTML, you write an HTML script with some embedded code to do something (e.g. to output some text). The PHP code is enclosed in special [start and end tags](http://www.php.net/manual/en/language.basic-syntax.php#language.basic-syntax.phpmode) that allow you to jump into and out of "PHP mode".
* **International support:** PHP supports multiple languages in the world, together with the Unicode standard.

## 

## How PHP works in LAMP/WAMP environment

PHP typically runs as a module of the Web server. It starts executing the script when it receives the call from Web server, and returns the resulted data back to the Web server. The overall steps of PHP execution is as follows:

* The web browser requests a document with a **.**php extension(or any extension set to be treated as a PHP file in the Web server);
* The Web server sends the request on to the PHP parser;
* The PHP parser scans the requested file for PHP code;
* When the PHP parser finds PHP code, it executes that code and places the resulting output(if any) into the place in the file formerly occupied by the code;
* This new output file is sent back to the Web server;
* The Web server sends the output file along to the Web browser;
* The Web browser displays the output.

## Typical workflow of PHP development

Even if PHP can generate any kind of binary data, the PHP script itself is always a text. So you can basically use any text editor to write the script. Typical workflow of PHP development is as follows:

* Prepare Testing Server for your PHP development;
* Create and edit the PHP script by your favorite text editor or dedicated editor software;
* Test your script in the testing server. In order to make this procedure the most efficient, you should setup your testing server within your development PC that your edited script will be immediately run by the testing server without the needs to copy or upload those files;
* Once you have fully tested your script in the testing server, then you can upload your script onto the real LAMP/WAMP server.

# BASIC PROGRAMMING IN PHP

## Comparison of PHP with other programming languages

As for language structure resembles C, and most of the text and array handling features resemble Perl. Therefore, PHP is rather easy to learn for those who have experience in these languages.

**Compiler vs interpreter ………………**

## Basic syntax

### The PHP tag

A PHP script must begin with special tag, **<?php**, which is called PHP tag. The PHP script must also end with the closing tag **?>**. The region in your text file that has been enclosed with this pair of tags will be parsed and executed by PHP.

<?php

…..PHP code……  
?>

You can place any number of this pair of PHP tags in your PHP file.

**N.B.:** The file must be saved with the extension of “.**php**”, not. “**.html**”

### Basic structure of PHP script

Within PHP tags, you write PHP script with as many statements as you like. A PHP statement must ends with semi-colon(;). If you want to create a group of statement, you use curly braces { } to enclose multiple statements.

### Comments

Comments in PHP are just like comments in C++. There are two ways to add comments in PHP like shown below:

// one-line comment

/\*

….. Multiple-lines comment

\*/

### Output something to page

You can use **echo** or **print** command for output anything on the page. Both commands have almost the same meaning and you can use either one of them.

*echo “Hello world”;*

*print “Hello again”;*

## Types, constants and variables

### Variables in PHP

The biggest difference from other structural languages is that you cannot declare a variable in PHP. You can just start using variables anywhere in the script.

**N.B**.: Variable names in PHP must begin with “$”. You must start the name with alphabets or underscore, but you can use numbers in subsequent characters.

This also means that you cannot declare the type of variables either. The type of a variable is determined by the content of execution, and it is possible to dynamically change the type of variables in your code.

e.g.

$a= “0”; // $a is a character string of “0”

$a= $a + 3; // $a is an integer(3)

$a=$a + 2.4; // $a is a float(5.4)

$a=5 + “10 characters”; // $a is integer(15)

In PHP, plus(+) always means arithmetic addition, so PHP will try to find the numeric part in the string and do arithmetic. In order to concatenate strings in PHP, you must use dot “.” operator.

Notes on string

String values can be enclosed by either double quotation or single quotation.

$a= “ABC”; or $a= ‘ABC’;

In order to access to individual characters within a string, use curly brace { } with index number starting from 0.

**e.g.**

$a= “ABCDEFGHIJKLMNOPQRSTUVWXYZ”;

$b= $a{5}; // $b contains ‘F’

Type casting

Though there is no explicit type declaration, it is possible to cast your variable to a certain type like shown below:

$a= (int) 3.14; // casting floating number to an integer; $a is 3

$a= (int) “5 cats”; // casting a string to an integer; $a is 5

$a= (string) 432; // casting a number to a string; $a is “432”

Scope of the variables

The scope of variables is limited to within the same page only, and that variable cannot be accessible across multiple pages. In order to get values from other pages, the only way is to use HTML forms or sessions.

### Output variables to page

In PHP, you can directly embed a variable name into double-quoted strings. However, this does not work in single-quoted strings.

e.g.

**$a=2/3;**

**echo “The answer to 2/3 is” .$a. “.”;**

**echo “The answer to 2/3 is $a.”;**

**echo ‘The answer to 2/3 is $a.’;**

### Constants

You can declare constants in PHP script by using define statement.

define (“constant\_name”, “constant\_value”);

**Operators**

### Common operators

PHP supports many operators. Most of them are similar to those in C.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| + - \* / | Arithmetic | $a= $b + $c; |
| . | String concatenation | $a= $b . $c; |
| ++ -- | Increment/decrement | $a= $b++ / --$c; |
| % | Modulus | $a= 17/5; //$a is 2 |
| & | ^ ~ | Bitwise AND, OR, XOR and NOT | $a= $b & $c | ~$d; |

### Assignment operators

The normal assignment operator in PHP is “=”. You can also use combined assignment with common operators like +=, -=, \*=, /=, .=, etc.

## Control structures

A program is usually not limited to a linear sequence of instructions. During its process it may bifurcate, repeat code or take decisions. For that purpose, PHP, like other structured programming languages, provides control structures that serve to specify what has to be done by our program, when and under which circumstances.

### Conditions

Conditions in PHP can be expressed by using the following conditional operators:

a. Conditional operators

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| == != | Equal, not equal | $a==$b |
| < > <= >= | Less than, greater than, less or equal, greater or equal | $a <= $b |
| === !== | Equal both in values and types, not equal either in values or types | $a === $b |

You can combine multiple conditions by &&, || or xor.

b. Ternary operator

You can assign different values based on a condition by using ternary operator. It is a combination of ? and :

e.g.: $a = ($b==$c)? $d : $e;

### Branching(If, if…else, if…elseif…else)

The **if** keyword is used to execute a statement or block only if a condition is fulfilled. Its syntax is:

**if (condition) statement**

Where **condition** is the expression that is being evaluated. If this condition is true, **statement** is executed. If it is false, statement is ignored (not executed) and the program continues right after this conditional structure.

**e.g.:**

<?php  
if ($a > $b)  
    echo "a is greater than b";  
?>

We can additionally specify what we want to happen if the condition is not fulfilled by using the keyword **else**. Its form used in conjunction with *if* is:

**if (condition) statement1 else statement2**

**e.g:** <?php  
if ($a > $b)

{  
    echo "a is greater than b";  
}

else

{  
    echo "a is less than b";  
}  
?>

The if + else structures can be concatenated with the intention of verifying a range of values.

**e.g.:**

<?php  
if ($a > $b) {  
    echo "a is greater than b";  
} else if ($a < $b) {  
    echo "a is less than b";  
}

else {  
    echo "a is equal to b";  
}  
  
?>

### Multiple-value branching

In PHP, you use **switch** for this purpose.

**The switch Statement**

The switch statement acts like a bunch of if….elseif….else statements. Take the following if…elseif…else statement for example:

<?php

if($a == 1)

{

echo(“The value of $a is 1”);

}

elseif($a == 2)

{

echo(“The value of $a is 2”);

}

elseif($a == 3)

{

echo(“The value of $a is 3”);

}

else

{

echo(“Unknown value”);

}

?>

In this example you are repeatedly checking the variable $a for a certain value,

even though the value of $a does not change from line to line. To make this

more efficient you would want to check the value of $a only once. You can do

that with the switch statement. Here is how you can convert the example above into a switch statement:

<?php

switch($a)

{

case 1:

{

echo(“The value of $a is 1”);

break;

}

case 2:

{

echo(“The value of $a is 2”);

break;

}

case 3:

{

echo(“The value of $a is 3”);

break;

}

default:

{

echo(“Unknown value”);

}

}

?>

### Looping

You have standard set of looping structure in PHP as follows:

#### “for” loop

*for* loops are the most complex loops in PHP. They behave like their C counterparts. The syntax of a *for* loop is:

for (expr1; expr2; expr3)

statement

The first expression (*expr1*) is evaluated (executed) once unconditionally at the beginning of the loop.

In the beginning of each iteration, *expr2* is evaluated. If it evaluates to **TRUE**, the loop continues and the nested statement(s) are executed. If it evaluates to **FALSE**, the execution of the loop ends.

At the end of each iteration, *expr3* is evaluated (executed).

It works in the following way:

1. *expr1(***initialization)** is executed. Generally it is an initial value setting for a counter variable. This is executed only once.
2. *expr2(***condition)** is evaluated at the beginning of each loop iteration. If it is true the loop continues, otherwise the loop ends and statement is skipped (not executed).
3. **statement** is executed. As usual, it can be either a single statement or a block enclosed in braces { }.
4. *expr3:* finally, whatever is specified in the **increase** field is executed at the end of each loop iteration and the loop gets back to step 2.

Each of the expressions can be empty or contain multiple expressions separated by commas. In *expr2*, all expressions separated by a comma are evaluated but the result is taken from the last part. *expr2* being empty means the loop should be run indefinitely (PHP implicitly considers it as **TRUE**, like C). This may not be as useless as you might think, since often you'd want to end the loop using a conditional [break](http://www.php.net/manual/en/control-structures.break.php) statement instead of using the *for* truth expression.

e.g.:

Consider the following examples. All of them display the numbers 1 through 10:

<?php  
/\* example 1 \*/  
  
for ($i = 1; $i <= 10; $i++) {  
    echo $i;  
}  
  
/\* example 2 \*/  
  
for ($i = 1; ; $i++) {  
    if ($i > 10) {  
        break;  
    }  
    echo $i;  
}  
  
/\* example 3 \*/  
  
$i = 1;  
for (; ; ) {  
    if ($i > 10) {  
        break;  
    }  
    echo $i;  
    $i++;  
}  
  
/\* example 4 \*/  
  
for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);  
?>

1. **Pre-evaluation loop (while)**

*while* loops are the simplest type of loop in PHP. They behave just like their C counterparts. The basic form of a *while* statement is:

while (expr)

statement

The meaning of a *while* statement is simple. It tells PHP to execute the nested statement(s) repeatedly, as long as the *while* expression evaluates to **TRUE**. The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration (each time PHP runs the statements in the loop is one iteration). Sometimes, if the *while* expression evaluates to **FALSE** from the very beginning, the nested statement(s) won't even be run once.

Like with the *if* statement, you can group multiple statements within the same *while* loop by surrounding a group of statements with curly braces, or by using the alternate syntax:

while (expr):

statement

...

endwhile;

**e.g.:**

<?php  
/\* example 1 \*/  
  
$i = 1;  
while ($i <= 10) {  
    echo $i++;  /\* the printed value would be  
                    $i before the increment  
                    (post-increment) \*/  
}  
  
/\* example 2 \*/  
  
$i = 1;  
while ($i <= 10):  
    echo $i;  
    $i++;  
endwhile;  
?>

1. **Post-evaluation loop (do while)**

Its format is:

**do statement while (condition);**

Its functionality is exactly the same as the *while* loop, except that condition in the *do-while* loop is evaluated after the execution of statement instead of before, granting at least one execution of statement even if condition is never fulfilled.

e.g.:

<?php  
$i = 0;  
do {  
    echo $i;  
} while ($i > 0);  
?>

1. **Use of “continue”**

The continue statement causes the program to skip the rest of the loop in the current iteration as if the end of the statement block had been reached, causing it to jump to the start of the following iteration.

e.g.:

<?php  
  for ($i = 0; $i < 5; $i++) {  
      if ($i == 2)  
          continue;  
      print "$i\n";  
  }  
?>

## Arrays

Arrays in PHP are quite different from normal indexed arrays in typical programming languages. They are called “Associative array”. Arrays in PHP are so powerful and convenient that you should be familiar with the usage of them.

### The structure of array in PHP

An array in PHP is implemented not as a continuous memory space divided into sequential indexed elements like in C, but as a collection of independent, arbitrary size and type of elements with assigned “keys”.

### Creating arrays

$a= array(); // Create empty array

$b= array(2,4,6,8,10); // Create an array for the specified values with default keys

$c= array(‘id’=> 32, ‘name’=> “Apple”, ‘price’=> 25.00); /\* create array with explicit assignment of keys\*/

The last structure is used often for storing the data for a database, since the keys represent field names, and values represent data stored in database.

You can also create multi-dimensional array in PHP.

### Basic array operations

* You can simply merge two or more arrays into an array by using ‘+’ operator

e.g.:

$a= array(0=> “Mon”, 1=> “Tue”, 2=> “Wed”);

$b= array(3=> “Thu”, 4=> “Fri”, 5=> “Sat”);

$c= $a + $b; // This will be the combined array of $a and $b

* You can also compare two arrays by using ‘= =’ or ‘= = =’ and ‘!= =’ or ‘!= = =’
* You can get the number of elements in an array by **count()** function.(**echo** count($a);)
* In order to access to every element in an array, use **foreach** loop. It is specifically designed to work with arrays in PHP, and you can individually retrieve both the key and value of each element stored order in the array.

### Array functions

Here are some array functions:

* [is\_array()](http://www.phpfreaks.com/phpmanual/page/function.is-array.html):Finds whether the given variable is an array.

e.g.:

|  |
| --- |
| <?php $yes = [array](http://www.phpfreaks.com/phpmanual/page/ref.array.html)('this', 'is', 'an array');  echo is\_array($yes) ? 'Array' : 'not an Array'; echo "\n";  $no = 'this is a string';  echo is\_array($no) ? 'Array' : 'not an Array'; ?> |

The above example will output:

|  |
| --- |
| Array  not an Array |

* [explode()](http://www.phpfreaks.com/phpmanual/page/function.explode.html):Returns an [array](http://www.phpfreaks.com/phpmanual/page/ref.array.html) of strings, [each](http://www.phpfreaks.com/phpmanual/page/function.each.html) of which is a substring of string formed by splitting it on boundaries formed by the string separator.

e.g.: <?php  
  
$pizza  = "piece1 piece2 piece3 piece4 piece5 piece6";  
$pieces = explode(" ", $pizza);  
echo $pieces[0]; // piece1  
echo $pieces[1]; // piece2

?>

* [implode()](http://www.phpfreaks.com/phpmanual/page/function.implode.html):Returns a string containing a string representation of all the [array](http://www.phpfreaks.com/phpmanual/page/ref.array.html) elements in the same order, with the glue string between [each](http://www.phpfreaks.com/phpmanual/page/function.each.html) element.

e.g.: <?php  
  
$array = [array](http://www.phpfreaks.com/phpmanual/page/ref.array.html)('lastname', 'email', 'phone');  
$comma\_separated = implode(",", $array);  
  
echo $comma\_separated; // lastname,email,phone  
  
?>

* [split()](http://www.phpfreaks.com/phpmanual/page/function.split.html):Split string into [array](http://www.phpfreaks.com/phpmanual/page/ref.array.html) by regular expression

e.g.: <?php  
// Delimiters may be slash, dot, or hyphen  
$date = "04/30/1973";  
list($month, $day, $year) = split('[/.-]', $date);  
echo "Month: $month; Day: $day; Year: $year<br />\n";  
?>

* Etc.

## Functions

You can define your own functions in PHP. The function can return the value, or you can make the function without returning any value. Use **function** keyword for defining a function in PHP.

<?php  
function foo($arg\_1, $arg\_2, /\* ..., \*/ $arg\_n)  
{  
    echo "Example function.\n";  
    return $retval; // this is optional  
}  
?>

One important caution of creating functions is that the variables used in the function are all **local** to that function only. You cannot access to the variables in functions from outside of the function. There is, however, a way to access external variables from inside of the function definition. Use **global** keyword with the name of the external variables like shown below:

There are 2 different types of php function

* User-defined function
* Built-in function

# Getting data from the client

The fundamental advantage of dynamic Web content by server-side technologies is the ability to receive the information from the clients. You can use HTML forms and other technologies to get data from the client in LAMP/WAMP application.

## HTML forms

The most common and important way to retrieve data from the client is by HTML forms.

### <form> element

A form is an area that can contain form elements.

Form elements are elements that allow the user to enter information (like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.) in a form.

A form is defined with the <form> tag.

|  |
| --- |
| <form>  <input>  <input>  </form> |

You can specify the following attributes to <form> elements:

* **Method**= “post | get” : this attribute determines the method of transmission of form data to the web server.
  + POST:

e.g.: <form action="process.php" method="post">

<select name="item">

...

<input name="quantity" type="text" />

The form data will be submitted to the "process.php" web page using the POST method. The way that PHP does this is to store all the "posted" values into an *associative array* called "$\_POST". Be sure to take notice the names of the form data names, as they represent the *keys* in the "$\_POST" associative array.

Now that you know about associative arrays, the PHP code from "process.php" should make a litte more sense.

$quantity = $\_POST['quantity'];

$item = $\_POST['item'];

The form data names are used as the *keys* in the associative array, so be sure that you never have two input items in your HTML form that have the same name. If you do, then you might see some problems arise.

* + GET:

e.g.:

<form action="process.php" method="get">

<select name="item">

...

<input name="quantity" type="text" />

The *get* method is different in that it passes the variables along to the "process.php" web page by appending them onto the end of the URI. The URI, after clicking submit, would have this added on to the end of it:

"?item=##&quantity=##"

The question mark "?" tells the browser that the following items are variables. Now that we changed the method of sending information on "order.html", we must change the "process.php" code to use the "$\_GET" associative array.

$quantity = $\_GET['quantity'];

$item = $\_GET['item'];

After changing the array name the script will function properly. Using the *get* method displays the variable information to your visitor, so be sure you are not sending password information or other sensitive items with the *get* method. You would not want your visitors seeing something they are not supposed to.

* **Action**= “destination\_URI”: this attribute determines the destination URI that the form will be submitted. This target URI must be a page that uses server-side technology(.php, .asp,…). If this attribute is omitted, the same page as the form page will be the destination.

### Single-line text box/password box

A single-line text box is created by an <input> tag. The name and value are the pair of information that will be sent to the web server.

Text fields are used when you want the user to type letters, numbers, etc. in a form.

|  |
| --- |
| <form>  First name:  <input type="text" name="firstname">  <br>  Last name:  <input type="text" name="lastname">  </form> |

The <input> has a few attributes that you should be aware of:

* *type* - Determines what kind of input field it will be. Possible choices are text, submit, and password.
* *name* - Assigns a name to the given field so that you may reference it later.
* *size* - Sets the horizontal width of the field. The unit of measurement is in blank spaces.
* *maxlength* - Dictates the maximum number of characters that can be entered.

There are some optional attributes:

value= “**value**”

type= “password”

type= “hidden”

size= “n”

maxlength= “n”

### Multi-line text area

Multi-line text area is used for entering long text.

Any text between <textarea> and </textarea> will be shown as the text data of multi-line text box.

<textarea rows="2" cols="20">  
The cat was playing in the garden. Suddenly a dog showed up.....  
</textarea>

### List box and drop-down list

Both the list box and drop-down list can be created by <select> tag and <option> tag.

<select>  
  <option value ="volvo">Volvo</option>  
  <option value ="saab">Saab</option>  
  <option value ="opel">Opel</option>  
  <option value ="audi">Audi</option>  
</select>

The size= “n” attribute determines the number of rows displayed on the screen. If n=1(or omitted), then it will be a drop down list and if n>1, it will be a list box.

### Check box

Check box can be created by <input> tag with “checkbox” as the type attribute.

Checkboxes are used when you want the user to select one or more options of a limited number of choices.

|  |
| --- |
| <form>  I have a bike:  <input type="checkbox" name="vehicle" value="Bike" />  <br />  I have a car:  <input type="checkbox" name="vehicle" value="Car" />  <br />  I have an airplane:  <input type="checkbox" name="vehicle" value="Airplane" />  </form> |

If you specify “**checked”,** then the checkbox will be initially checked.

Use multiple checkboxes as a group in PHP

There is a way to use multiple checkboxes as a group. You use the same “**name**” that ends with a pair of square bracket “[]” for multiple checkboxes, and the returned value to the web server can be retrieved as an array (only the checked elements will be returned as elements of array).

### Radio button

Radio button can be created by <input> tag with “**radio**” as the type attribute.

One <input> tag will create one radio button itself. In order to make a group (when you need two or more radio buttons), all radio buttons must have the same “**name**”, and each radio button mast have different “**value**”. Then the form will return appropriate “**name=value**” pair back to the web server.

Radio Buttons are used when you want the user to select one of a limited number of choices.

|  |
| --- |
| <form>  <input type="radio" name="sex" value="male"> Male  <br>  <input type="radio" name="sex" value="female"> Female  </form> |

Bottom of Form

If you specify “**checked”,** then the radio button will be initially checked.

Only one radio button is allowed to be selected within a group of radio buttons.

### Buttons

Push buttons can be created by <input> tag with the following types:

<input type= “submit” name= “name” value= “title”>

<input type= “reset” name= “name” value= “title”>

<input type= “image” src= “image\_URI” name= “name” alt= “alternate\_text”>

<input type= “button” name= “name” value= “title”>

All these three <input> tags show buttons on screen. The title determines the caption displayed on top of the button.

### Labels

Basically, <label> will not add any display effect on the screen. However, by enclosing any form element into a <label> tag together with the label text, the label land element will behave as a single element for user’s operation. For example, clicking on the label text by mouse will make the associated form element automatically active.

## Processing data with PHP

### Retrieving form data

You can retrieve submitted form data basically by the following “super global variables” predefined in PHP.

* $\_GET: This super global variable contains all data submitted by “**get**” method.
* $\_POST: This super global variable contains all data submitted by “**post**” method.

### Handling of string data

In practice, however, simply using raw values returned from $\_GET and $\_POST can lead to many problems if they are string data. There are basically two major problems in using raw string data:

* The returned value in PHP is not identical to the original data: In order to retrieve original string entered from the user, you must use **stripslashes()** function ($original\_string= stripslashes($\_GET[“element\_name”]);)
* The returned value cannot be directly displayed on the page: Since HTML has special characters (such as < and >, and any more), you must convert your string to HTML encoded string by **htmlspecialchars()** function( echo htmlspecialchars($raw\_string);).

### Validation of user input

Basically, you should not trust any data returned from the user. Users may enter value that is not in the preferred format. You should always check and validate data returned from them, regardless that you add value-checking mechanism using JavaScript or any other client-side technologies.

* Validation for existing of data: using **empty()** or **isset()** functions.
* Validation of numbers
* Validation of strings

### Showing default values in HTML form

In many cases, the data returned from HTML form must be checked for validity, and if there are some problems, you should let users correct them. Or in case you want to create a Web page for modifying existing data in database, you must first show the existing data within the HTML form. For those cases, you must assign the data to HTML form element’s default value.

e.g.:

<input type="text" name="Name" value = <?php echo "$Name";?> size = "30">

## Getting data from Web server

Other than HTML forms, you can also get information from Web server. $\_SERVER “super global variable” contains many server information as well as information retrieved from HTTP header that client sent to the server.

## Cookie

A cookie is a very basic data file. It has a name and a value and also stores the address of websites which are allowed to access it and an expiry time. Basically, a website will set a cookie and give it a name and value. This name is used by the website to refer to it, and no other website can access the cookie, even if they know it's name. The name should be unique to the website, but it doesn't matter if it clashes with the name of a cookie from another website.

* Many portals and search engines use them to provide customized pages and results to their users, allowing such features as 'My Yahoo' etc.
* Many websites use cookies to log their users in automatically. By storing a few pieces of user information they can automatically authenticate the user's details and use them to save the user time when they log in>/li>
* Visitor tracking and statistics systems often use them to track visitors. By assigning the visitor a cookie, they will not be counted more than once, so accurate unique visitor statistics can be obtained. Also, if a user has a unique cookie the system can 'follow' them through a website, showing the webmaster exactly where the visitor has been, and in what order.

Setting a Basic Cookie  
  
The PHP function for setting cookies is called:  
  
*setcookie()*  
  
It is a PHP function which can be used without returning a value (for example you can simply execute a setcookie()) command, or you can take the return value and use it. The setcookie() function returns a boolean (true or false) value depending on whether it is successful. So you could execute:  
  
*if*(setcookie())  
{  
*echo* "Cookie set";  
}  
*else*  
{  
*echo* "Cookie not set";  
}  
  
For the purposes of this tutorial, though, we will not be using the return value, instead simply setting the cookie.  
  
The most basic information for a cookie is its name and its value. The name of the cookie must be something which you can refer to it later as. You don't need to worry about it clashing with other sites as cookie names are site specific but you should try and use a descriptive and unique name for your cookies.  
  
For this first example, assume that you have used PHP to load the user's name into the variable $name and want to greet the user in the future by their name. You would need to create a cookie which stores their name as follows:  
  
*setcookie*("UsersName",$name);  
  
This creates the most basic of cookies, storing the user's name in a cookie called 'UsersName'. By setting cookies like this, you don't set any specific options, so by default the cookie will be available to the domain in which it was set (e.g. yoursite.com) and will be deleted when the user closes their browser.  
  
Reading Cookie Values  
  
PHP makes it extremely simple to read the value of a cookie. In PHP, reading form values are achieved using $\_GET and $\_POST. PHP has a similar global variable for cookies:  
  
*$\_COOKIE*['CookieName'];  
  
This variable contains the value of the cookie with name 'CookieName'. So on your website, if you wanted to display the name of the user, you could simply use the following:  
  
*echo* "Hello, ".$\_COOKIE['UsersName']."! Welcome back!";  
  
Of course, the user may not already have the cookie, so you should use the PHP function isset. This returns true if a variable has been set and false if not. Using this, your site could do the following:  
  
if(isset($\_COOKIE['UsersName'])  
{  
echo "Hello, ".$\_COOKIE['UsersName']."! Welcome back!";  
}  
else  
{  
setcookie("UsersName",$name);  
}  
  
  
Cookie Settings  
  
Although the code we have seen allows us to set a simple cookie on the user's computer, it isn't very powerful because, for example, it is lost when the browser closes. One of the most powerful features of cookies is the ability to set and expiry date for the cookie. The cookie will remain on the users’ computer until the expiry date, and then will automatically delete itself.  
  
To set a cookie with an expiry date, use:  
  
setcookie("UsersName", $name, time()+3600);  
  
This code takes the current time (using time()) and then adds 3600 seconds to it, and uses this value to set as the expiry time for the cookie. Basically this means that the cookie will remain on the user's computer for an hour (it expires 3600 seconds (1 hour) from the current time). For one week (for example) you would set the cookie as:  
  
setcookie("UsersName", $name, time()+604800);  
  
There are three other options which can be used when setting cookies. Firstly the path. This refers to where in the domain you are able to access the cookie in future. By default this is the current directory (so if you set the cookie at the page: www.mysite.com/scripts/setcookie.php, it would only be available to scripts in the scripts directory and below). You can set this to any part of your site, though, which can be useful in some situations.  
  
A second setting you can change is the domain. By default, a cookie is only available in the domain you set it in, for example if you set the cookie on www.mysite.com you can only ever access it from www.mysite.com (and not mail.mysite.com etc.). The most common need to change this setting is to allow the cookie to be viewed across all subdomains of a site.   
  
Finally, a cookie has the option to be set as a secure cookie. If this is turned on, the cookie will only ever be surrendered to the site over a secure connection, not an insecure one.  
  
The following code shows the imiplementation of a cookie with all settings specified:  
  
setcookie("UsersName", $name, time()+3600, "/", ".mysite.com", 1);  
  
The cookie set here, is called 'UsersName' and again stores the value $name. It will expire an hour from the current time. It is available in all directories of the site (/ is the root directory). It is available across any subdomain of the site mysite.com as '.mysite.com' has been given as the domain. The final 1 means that this is a secure cookie, and can only be transmitted over a secure connection. This would be 0 for a standard (non-secure) cookie.  
  
Deleting Cookies  
  
There are occasions on which you may wish to delete a cookie from a user's computer. This could be if, for example, you want to log the user out of a system (perhaps they are on a public computer). Deleting a cookie is quite simple to do because all you have to do is to set the expiry time in the past. By doing this, the cookie will be automatically deleted as soon as it is created, and will remove any data that already exists there. The simplest way is using:  
  
setcookie("UsersName", "", time()-3600);  
  
This sets the expiry time in the past so it should be deleted immediately. There is also no information stored in the cookie.  
  
There is a known problem with this, though. Although it works in most cases, there can be problems if a user's timezone is set wrongly. The safest way to completely delete a cookie is to use the following:  
  
setcookie("UsersName", "", mktime(12,0,0,1, 1, 1990));  
  
The **mktime()** function is a PHP function for setting up a time specified. The time specified here is in the year 1990, so even a badly configured computer should still delete the cookie immediately.

Handle errors and exception

|  |
| --- |
| Description of program error types  Syntax errors  Logical errors   Run time errors   Introduction to error functions  die  custom error functions   Application of exception block  Try and catch keywords |

# ACCESSING MySQL from PHP

One of the most important features of LAMP/WAMP application is the utilization of MySQL database through PHP script.

## MySQL access basics

### Basic PHP

There are basically 5 functions you should know in order to access MySQL from your PHP code. The most fundamental PHP code to show the query result as a table is like this:

<?php

*// Connect to database server*

    $hd = mysql\_connect("myhost", "username", "password")

          or die ("Unable to connect");

*// Select database*

    mysql\_select\_db ("database", $hd)

          or die ("Unable to select database");

*// Execute sample query*

    $res = mysql\_query("SELECT \* FROM customer", $hd)

          or die ("Unable to run query");

    // Iteration loop, for each row in rowset

    while ($row = mysql\_fetch\_array($res, MYSQL\_ASSOC))

    {

        // Assigning variables from cell values

        $data1 = $row["title"];

        $data2 = $row["fname"];

        $data3 = $row["lname"];

        $data4 = $row["phone"];

        // Outputting data to browser

        echo "ROW# $nr : $data1 $data2 $data3 $data4\n";

    }

    mysql\_close($hd);

?>

**N.B**: There are many other MySQL functions in PHP.

# PRACTICAL TECHNIQUES OF PHP FOR WEB APPLICATION

## Popular PHP functions

* String functions: e.g.: substr()
* Array functions: e.g.: count()
* Other popular function: e.g. header()

## Session handling

Creating sessions allows you to keep track of the actions of a particular user over the period of time he is viewing your web site.

To associate session data with a user, you need a session identity number, that is, a key that ties the user to his data.

Session management is a mechanism to maintain state about a series of requests from the same user across some period of time. That is, the term "session" refers to the time that a user is at a particular web site. The problem is, that HTTP has no mechanism to maintain state. Individual requests aren't related to each other. The Web server can't easily distinguish between single users and doesn't know about user sessions. Session management refers to the way that associate data with a user during a visit to a Web page. This tutorial uses the term session for a single visit of a user. For example, a typical online shopping session might include logging in, putting an item into the shopping cart, going to the checkout page, entering address and credit card data, submitting the order, and closing the browser window. PHP 4.0 includes native session management functions to ease the task of managing user sessions.

The basic usage of session is very easy. You will first call **session\_start()** at the top of the page, and then you can store any information into the $\_SESSION super global variables.

e.g.:

<?php  
  
session\_start();  
  
   $\_SESSION[“My\_data”] =$my\_variable; // store data to session

?>

The stored data can be retrieved in any other page as far as you call session\_start() function in every page.

e.g.:

<?php  
session\_start();  
  
$my\_variable= $\_SESSION[“My\_data”]; // retrieve data from session

?>

**Starting a Session**

Before you can begin storing user information in your PHP session, you must first start the session. When you start a session, it must be at the beginning of your code, before any HTML or text is sent.

Below is a simple script that you should place at the beginning of your PHP code to start up a PHP session.

e.g.:

<?php

session\_start(); // start up your PHP session!

?>

This piece of code will register the user's session with the server, allow you to start saving user information and assign a UID (unique identification number) for that user's session.

**Storing a Session Variable**

When you want to store user data in a session, use the $\_SESSION [associative array](http://www.tizag.com/phpT/arrays.php). This is where you both store and retrieve session data. In previous versions of PHP there were other ways to perform this store operation, but it has been updated and this is the correct way to do it.

<?php

session\_start();

$\_SESSION['views'] = 1; // store session data

echo "Pageviews = ". $\_SESSION['views']; //retrieve data

?>

**Display:**

Pageviews = 1

In this example we learned how to store a variable to the session associative array $\_SESSION and also how to retrieve data from that same array.

**Using *isset* Function**

Now, you can easily store and retrieve data from the $\_SESSION array; we can now explore some of the real functionality of sessions. When you create a variable and store it in a session, you probably want to use it in the future. However, before you use a session variable it is necessary that you check to see if it exists already. *isset* is a function that takes any variable you want to use and checks to see if it has been **set**. That is, it has already been assigned a value.

With our previous example, we can create a very simple pageview counter by using *isset* to check if the pageview variable has already been created. If it has, we can increment our counter. If it doesn't exist we can create a pageview counter and set it to one. Here is the code to get this job done:

<?php

session\_start();

if(isset($\_SESSION['views']))

$\_SESSION['views'] = $\_SESSION['views']+ 1;

else

$\_SESSION['views'] = 1;

echo "views = ". $\_SESSION['views'];

?>

**Cleaning and Destroying a Session**

Although a session's data is temporary and does not require that you explicitly clean after yourself, you may wish to delete some data for your various tasks.

Imagine that you were running an online business and a user used your website to buy your goods. The user has just completed a transaction on your website and you now want to remove everything from their shopping cart.

<?php

session\_start();

if(isset($\_SESSION['cart']))

unset($\_SESSION['cart']);

?>

You can also completely destroy the session entirely by calling the *session\_destroy* function.

<?php

session\_start();

session\_destroy();

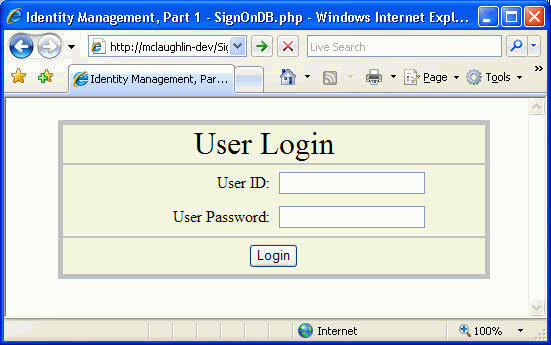
?>

Destroy will reset your session, so don't call that function unless you are entirely comfortable losing all your stored session data.

## PHP-based authentication

Any login based system requires user authentication mechanism.

Your PHP authentication script takes a name and value pair for the user name and password. The script then compares the pair against data stored in your users table.



While it is customary to store server-side passwords in encrypted forms, you should recognize that any compromise of the clear text user and password values compromises your system security. Therefore, you should not force users to create complex passwords because they aren't easily remembered.

Even if we use basic authentication, there are many small techniques to make it more secure. Below are some of the good security tips:

* Store user’s password in database as MD5 hash;
* You should limit the number of login attempt per session. For example, if the user failed to login more than 5 times, completely refuse the login.

## File uploading

Before you can use PHP to manage your uploads, you must first build an HTML form that lets users select a file to upload.

<form enctype="multipart/form-data" action="uploader.php" method="POST">

<input type="hidden" name="MAX\_FILE\_SIZE" value="100000" />

Choose a file to upload: <input name="uploadedfile" type="file" /><br />

<input type="submit" value="Upload File" />

</form>

Here is a brief description of the important parts of the above code:

* **enctype="multipart/form-data"** - This attribute specifies the [content type](http://www.w3.org/TR/html4/interact/forms.html#form-content-type#form-content-type) used to submit the form to the server
* **action="uploader.php"** - The name of our PHP page that will be created, shortly.
* **method="POST"** - Informs the browser that we want to send information to the server using POST.
* **input type="hidden" name="MA...** - Sets the maximum allowable file size, in bytes, that can be uploaded. This safety mechanism is easily bypassed and we will show a solid backup solution in PHP. We have set the max file size to 100KB in this example.
* **input name="uploadedfile"** - *uploadedfile* is how we will access the file in our PHP script.

Save that form code into a file and call it *upload.html*.

Bottom of Form

After the user clicks submit, the data will be posted to the server and the user will be redirected to *uploader.php*. This PHP file is going to process the form data and do all the work.

Now that we have the right HTML form we can begin to code the PHP script that is going to handle our uploads. Typically, the PHP file should make a key decision with **all** uploads: keep the file or throw it away. A file might be thrown away from many reasons, including:

* The file is too large and you do not want to have it on your server.
* You wanted the person to upload a picture and they uploaded something else, like an executable file (.exe).
* There were problems uploading the file and so you can't keep it.

**uploader.php**

When the *uploader.php* file is executed, the uploaded file exists in a temporary storage area on the server. If the file is not moved to a different location it will be **destroyed**. To save our precious file we are going to need to make use of the *$\_FILES* [associative array](http://www.tizag.com/phpT/arrays.php).

The *$\_FILES* array is where PHP stores all the information about files. There are two elements of this array that we will need to understand for this example.

* **uploadedfile** - *uploadedfile* is the reference we assigned in our HTML form. We will need this to tell the $\_FILES array which file we want to play around with.
* **$\_FILES['uploadedfile']['name']** - *name* contains the original path of the user uploaded file.
* **$\_FILES['uploadedfile']['tmp\_name']** - *tmp\_name* contains the path to the temporary file that resides on the server. The file should exist on the server in a temporary directory with a temporary name.

Now we can finally start to write a basic PHP upload manager script! Here is how we would get the temporary file name, choose a permanent name, and choose a place to store the file.

// Where the file is going to be placed

$target\_path = "uploads/";

/\* Add the original filename to our target path.

Result is "uploads/filename.extension" \*/

$target\_path = $target\_path . basename( $\_FILES['uploadedfile']['name']);

$\_FILES['uploadedfile']['tmp\_name'];

**NOTE:** You will need to create a new directory in the directory where *uploader.php* resides, called "uploads", as we are going save files there.

We now have all we need to successfully save our file to the server. *$target\_path* contains the path where we want to save our file to.

**move\_uploaded\_file Function**

Now all we have to do is call the *move\_uploaded\_file* function and let PHP do its magic. The *move\_uploaded\_file* function needs to know

1) The path of the temporary file

2) The path where it is to be moved to.

$target\_path = "uploads/";

$target\_path = $target\_path . basename( $\_FILES['uploadedfile']['name']);

if(move\_uploaded\_file($\_FILES['uploadedfile']['tmp\_name'], $target\_path)) {

echo "The file ". basename( $\_FILES['uploadedfile']['name']).

" has been uploaded";

} else{

echo "There was an error uploading the file, please try again!";

}

If the upload is successful, then you will see the text "The file *filename* has been uploaded". This is because *$move\_uploaded\_file* returns *true* if the file was moved, and *false* if it had a problem.

If there was a problem then the error message "There was an error uploading the file, please try again!" would be displayed.

## Generating image by GD(Graphics Drawing image functions)

GD library is the graphics generation library for PHP, and now is the integrated part of PHP as “PHP image functions”. You can use this library without installing any additional libraries to the system.

**N.B.:** If you want to display a PHP page that contains dynamically generated image by GD, actually you must prepare 2 PHP pages, one for HTML page and another for the image itself. In the HTML page, you should specify the PHP file name to generate image at the “**src**” attribute of the <img> tag.

## Combined usage with client-side technologies

PHP is a server-side technology. It is sometimes convenient to use client-side technologies such as JavaScript together with the PHP.

### Form control

Controlling form elements during the user’s data entry is the technique of client technologies used without submitting the form to the server (request of the page).

With JavaScript, you could do form validation, quickly popup messages, make cool web tools and more.

### Ajax(Asynchronous JavaScript and XML)

As we have mentioned it above, you could do form validation, quickly popup messages, make cool web tools and more using JavaScript. However, JavaScript had no way of sending information between the web browser and the web server.

If you wanted to get any information from a database on the server, or send user information to a server-side script like PHP, you had to make an HTML form to GET or POST data to the server. The user would then have to click "Submit", wait for the server to respond, then a new page would load with the results.

Ajax attempts to remedy this problem by letting your Javascript communicate directly with the server, using a special Javascript object **XMLHttpRequest**. With this object, your Javascript can get information from the server without having to load a new page.

To clearly illustrate how easy it is to access information from a database using Ajax, we are going to build MySQL queries on the fly and display the results on "order.html".

[Create a new database](http://www.tizag.com/mysqlTutorial/mysqldatabase.php) or use an existing one and then import the table [ajax\_example.sql](http://www.tizag.com/ajaxTutorial/ajax_example.sql) to that database. This sql file will create the table *ajax\_example* and insert all the data rows. The table has four columns:

* **ae\_name** - The name of the person
* **ae\_age** - Person's age
* **ae\_sex** - The gender of the person
* **ae\_wpm** - The words per minute that person can type

**Update order.html**

We want to be able to build queries from our HTML file, so there are a few form elements that will need to be added. The three inputs we are going to implement are:

* Maximum Age (Text Input) - Let the user select the maximum age to be returned.
* Maximum WPM (Text Input) - Let the user select the maximum wpm to returned.
* Gender (Select Input) - Let the user select the gender of a valid person.

**order.html**

<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

ajaxRequest.onreadystatechange = function(){

if(ajaxRequest.readyState == 4){

document.myForm.time.value = ajaxRequest.responseText;

}

}

var age = document.getElementById('age').value;

var wpm = document.getElementById('wpm').value;

var sex = document.getElementById('sex').value;

var queryString = "?age=" + age + "&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>m</option>

<option>f</option>

</select>

<input type='button' onclick='ajaxFunction()' value='Query MySQL' />

</form>

</body>

</html>

### 

### Where to insert PHP tag in HTML

You can insert PHP tags basically anywhere in HTML page. There is no limitation of the location.