Web Development using Server Side scripting language

About the Tutorial

The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web-based software applications. This tutorial will help you understand the basics of PHP and how to put it in practice.

Audience

This tutorial is designed for beginners who are completely unaware of the concepts of PHP but they have a basic understanding of computer programming.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of computer programming, HTML, Internet, Database, and MySQL etc is very helpful.

1. Overview of PHP

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire ecommerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL,
- Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the UNIX side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

Common uses of PHP:

PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them. The other uses of PHP are:

- PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.
- You add, delete, and modify elements within your database thru PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Five important characteristics make PHP's practical nature possible:

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

Syntax Overview

This chapter will give you an idea of very basic syntax of PHP and very important to make your PHP foundation strong.

Escaping to PHP

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP.' There are four ways to do this:

Canonical PHP tags

The most universally effective PHP tag style is:

```
<?php...?>
```

If you use this style, you can be positive that your tags will always be correctly interpreted.

Short-open (SGML-style) tags

Short or short-open tags look like this:

```
<?...?>
```

Short tags are, as one might expect, the shortest option you must do one of two things to enable PHP to recognize the tags:

- Choose the --enable-short-tags configuration option when you're building PHP.
- Set the short_open_tag setting in your php.ini file to on. This option must be disabled to parse XML with PHP because the same syntax is used for XML tags.

HTML script tags

HTML script tags look like this:

```
<script language="PHP">...</script>
```

Commenting PHP Code:

A comment is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

Single-line comments: They are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments.

```
<?
# This is a comment, and
# This is the second line of the comment
// This is a comment too. Each style comments only print "An example with single line comments";
?>
```

Multi-lines comments: They are generally used to provide pseudo code algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C. Here is the example of multi lines comments.

```
<?
/* This is a comment with multiline
        Author : Saqib Nazir
        Purpose: Multiline Comments Demo Subject: PHP
*/
print "An example with multi line comments";
?>
```

PHP is whitespace insensitive:

Whitespace is the stuff you type that is typically invisible on the screen, including spaces, tabs, and carriage returns (end-of-line characters).

PHP whitespace insensitive means that it almost never matters how many whitespace characters you have in a row.one whitespace character is the same as many such characters

For example, each of the following PHP statements that assigns the sum of 2 + 2 to the variable $four is equivalent:

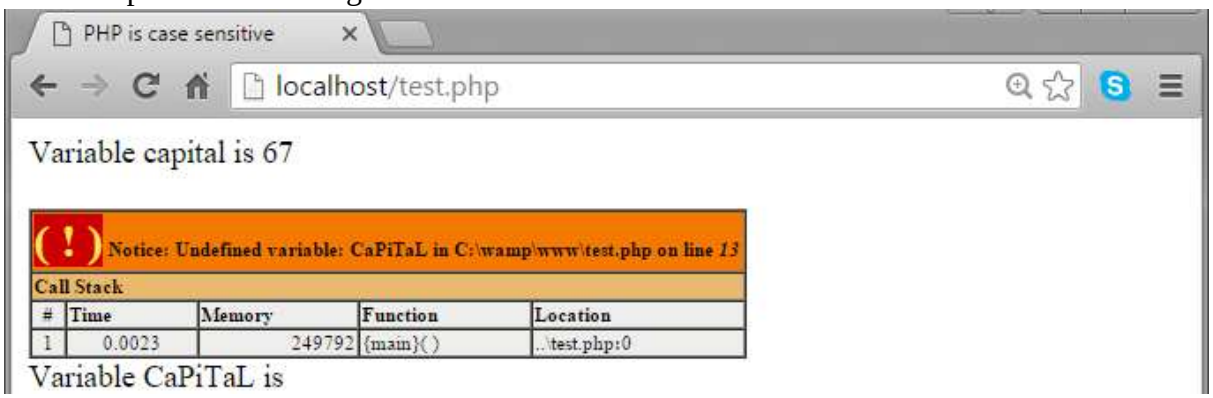
```
$four = 2 + 2; // single spaces
$four <tab>=<tab2<tab>+<tab>2 ; // spaces and tabs
$four = 2+
2; // multiple lines
```

PHP is case sensitive:

Yeah it is true that PHP is a case sensitive language. Try out following example:

```
<html>
<body>
<?php
$capital = 67;
print("Variable capital is $capital<br>");
print("Variable CaPiTaL is $CaPiTaL<br>");
?>
</body>
</html>
```

This will produce following result:



Statements are expressions terminated by semicolons:

A statement in PHP is any expression that is followed by a semicolon (;). Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program. Here is a typical statement in PHP, which in this case assigns a string of characters to a variable called

```
$greeting = "Welcome to PHP!";
```

Expressions are combinations of tokens:

The smallest building blocks of PHP are the indivisible tokens, such as numbers (3.14159), strings (.two.), variables ($two), constants (TRUE), and the special words that make up the syntax of PHP itself like if, else, while, for and so forth

Braces make blocks:

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces.

Here both statements are equivalent:

```
if (3 == 2 + 1)
print("Good - I haven't totally lost my mind.<br>");

if (3 == 2 + 1)
{
print("Good - I haven't totally"); print("lost my mind.<br>");
}
```

"Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello world</title>
</head>
<body>
     <?php echo "Hello, World!";?>
</body>
</html>
```

This will produce the following result:

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.

All PHP code must be included inside one of the three special markup tags ate are recognized by the PHP Parser.

```
    <? php PHP code goes here ?>
    <? PHP code goes here ?>
    <script language="php"> PHP code goes here </script>
```

Most common tag is the <?php...?> and we will also use the same tag in our tutorial.

2. Variables

The main way to store information in the middle of a PHP program is by using a variable. Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign ($).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

PHP has a total of eight data types which we use to construct our variables:

- Integers: are whole numbers, without a decimal point, like 4195.
- Doubles: are floating-point numbers, like 3.14159 or 49.1.
- Booleans: have only two possible values either true or false.
- NULL: is a special type that only has one value: NULL.
- Strings: are sequences of characters, like 'PHP supports string operations.'
- Arrays: are named and indexed collections of other values.
- Objects: are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- Resources: are special variables that hold references to resources external to PHP (such as database connections).

The first five are simple types, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

We will explain only simile data type in this chapter. Array and Objects will be explained separately.

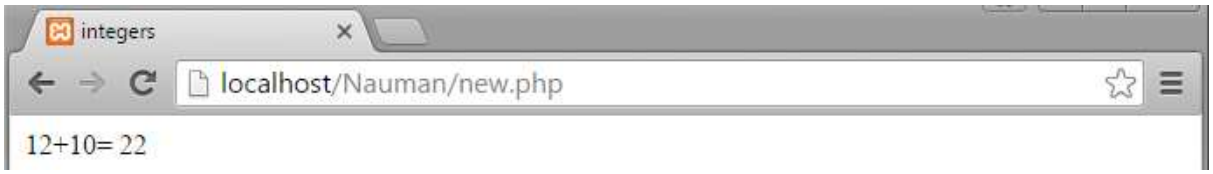
Integers

They are whole numbers, without a decimal point, like 4195. They are the simplest type .they correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like so:

```
<html>
<head>
   <title>integers</title>
</head>
<body>
<?php $int1 = 12;
$int2 = 10;
$int3=$int1+$int2;
echo "$int1+$int2= $int3";
?>
</body>
```

```
</html>
```

Result:



Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimals have a leading 0x.

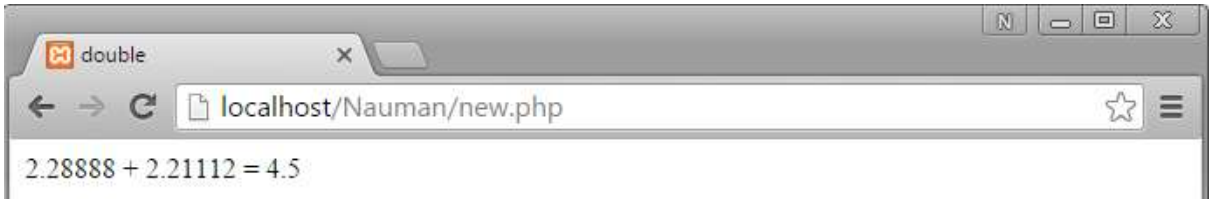
For most common platforms, the largest integer is (2**31 . 1) (or 2,147,483,647), and the smallest (most negative) integer is. (2**31 . 1) (or .2,147,483,647).

Doubles

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code:

```
<html>
<head>
   <title>double</title>
</head>
<body>
<?php $many = 2.2888800;
$many_2 = 2.2111200;
$few = $many + $many_2;
print("$many + $many_2 = $few");
?>
</body>
</html>
```

It produces the following browser output:



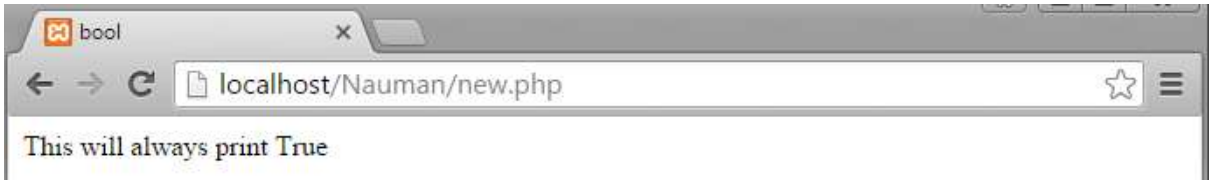
Boolean

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so:

```
<html>
<head>
   <title>bool</title>
</head>
<body>
<?php if (1)
print("This will always print True<br>");
else
print("This will never print False<br>");
```

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

Result:



Interpreting other types as Booleans

Here are the rules for determine the "truth" of any value not already of the Boolean type:

- If the value is a number, it is false if exactly equal to zero and true otherwise.
- If the value is a string, it is false if the string is empty (has zero characters) or is the string "0", and is true otherwise.
- Values of type NULL are always false.
- If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value.
- Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).
- Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;

$true_str = "Tried and true"

$true_array[49] = "An array element";

$false_array = array();

$false_null = NULL;

$false_num = 999 - 999;

$false_str = "";
```

NULL

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this:

$my_var = NULL;

The special constant NULL is capitalized by convention, but actually it is case insensitive;

you could just as well have typed:

$my_var = null;

A variable that has been assigned NULL has the following properties:

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with IsSet() function.

Strings

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

$string_1 = "This is a string in double quotes";

$string_2 = "This is a somewhat longer, singly quoted string";

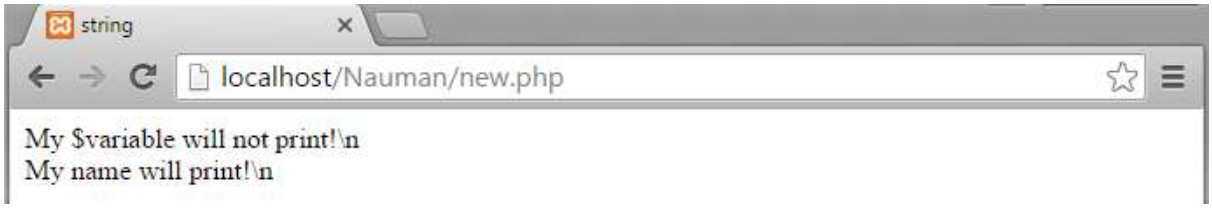
$string_39 = "This string has thirty-nine characters";

$string_0 = ""; // a string with zero characters

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<html>
<head>
   <title>string</title>
</head>
<body>
<?php
$variable = "name";
$literally = 'My $variable will not print!\\n';
print($literally);
$literally = "<br/>My $variable will print!\\n";
print($literally);
?>
</body>
</html>
```

This will produce following result:



There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with $) are replaced with string representations of their values.

The escape-sequence replacements are:

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \$ is replaced by the dollar sign itself ($)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

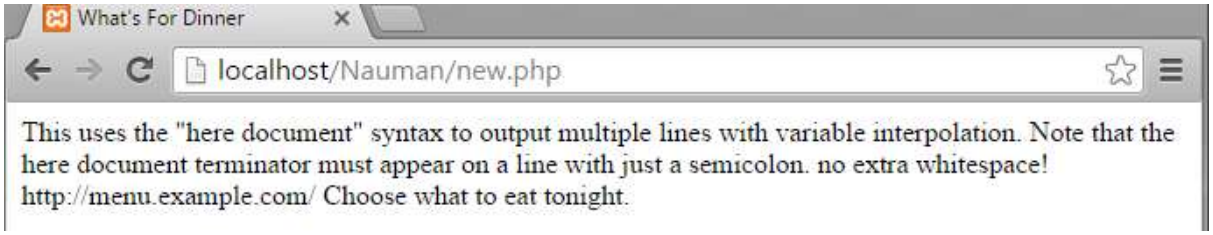
Here Document

You can assign multiple lines to a single string variable using here document:

```
<?php

$channel =<<<_XML_
```

```
<?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_;
echo <<<END
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!
<br />
END;
print $channel;
?>
```

This will produce the following result:



Variable Naming

Rules for naming a variable is:

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like + , - , % , (,) . & , etc
- There is no size limit for variables.

PHP – Variables

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types:

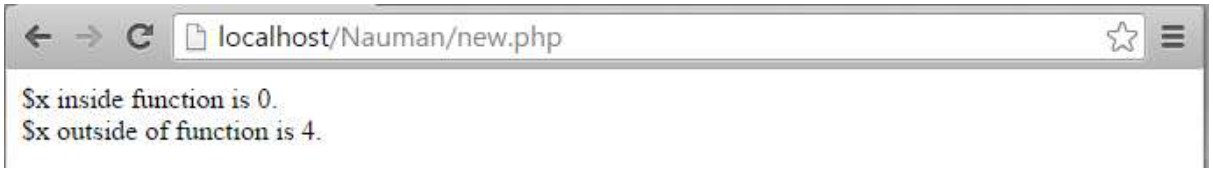
- Local variables
- Function parameters
- Global variables
- Static variables

PHP Local Variables

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function:

```
<?php
$x = 4;
function assignx () {
$x = 0;
print "\$x inside function is $x. ";
}
assignx();
print "</br>\$x outside of function is $x. ";
?>
```

This will produce the following result.



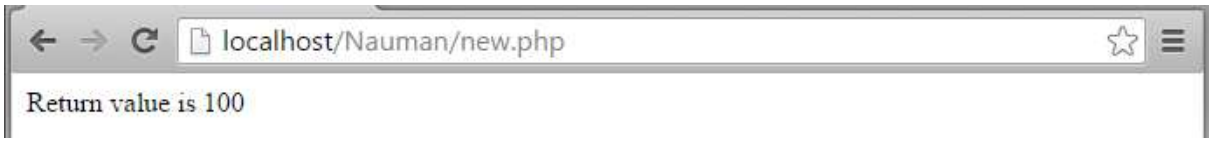
PHP Function Parameters

PHP Functions are covered in detail in PHP Function Chapter. In short, a function is a small unit of program which can take some input in the form of parameters and does some processing and may return a some value.

Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be:

```
<?php
// multiply a value by 10 and return it to the caller
function multiply ($value) {
$value = $value * 10;
return $value;
}
$retval = multiply (10);
Print "Return value is $retval\n";
?>
```

This will produce the following result.

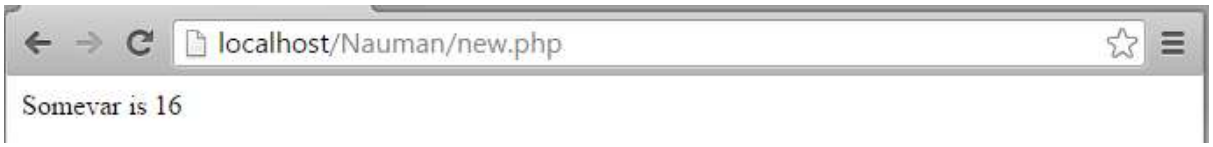


PHP Global Variables

In contrast to local variables, a global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword GLOBAL in front of the variable that should be recognized as global. Placing this keyword in front of an already existing variable tells PHP to use the variable having that name. Consider an example:

```
<?php
$somevar = 15; function addit() { GLOBAL $somevar;
$somevar++;
print "Somevar is $somevar";
}
addit();
?>
```

This will produce the following result.



PHP Static Variables

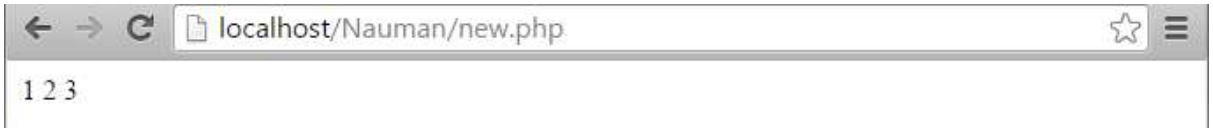
The final type of variable scoping that I discuss is known as static. In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

You can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

```
<?php
function keep_track() { STATIC $count = 0;
$count++;
print $count;
print "
";
} keep_track(); keep_track(); keep_track();
```

```
?>
```

This will produce the following result.



3. Constants

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

To define a constant you have to use define() function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a $. You can also use the function constant() to read a constant's value if you wish to obtain the constant's name dynamically.

constant() function

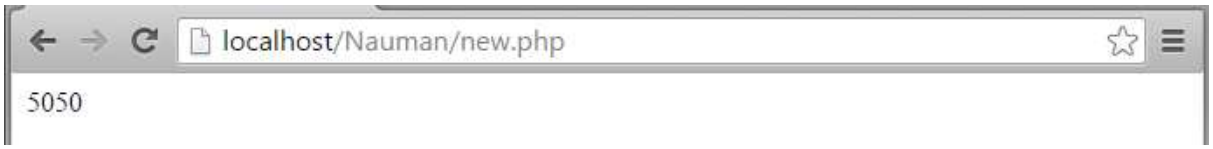
As indicated by the name, this function will return the value of the constant.

This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. It is stored in a variable or returned by a function.

constant() example

```
<?php
define("MINSIZE", 50);
echo MINSIZE;
echo constant("MINSIZE"); // same thing as the previous line
?>
```

Result:



Only scalar data (boolean, integer, float and string) can be contained in constants.

Differences between constants and variables are

- There is no need to write a dollar sign ($) before a constant, where as in Variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the define() function.
- Constants may be defined and accessed anywhere without regard to variable scoping rules.
- Once the Constants have been set, may not be redefined or undefined.

Valid and invalid constant names

```
// Valid constant names define
("ONE",    "first thing");
define("TWO2",   "second thing");
define("THREE_3", "third thing")
// Invalid constant names
define("2TWO",   "second thing");
define("  THREE  ", "third value");
```

PHP Magic constants

PHP provides a large number of predefined constants to any script which it runs.

There are five magical constants that change depending on where they are used. For example, the value of __LINE depends on the line that it's used on in your script. These special constants are case-insensitive and are as follows:

A few "magical" PHP constants ate given below:

Name	Description
LINE	The current line number of the file.
__FILE__	The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, ___FILE___ always contains an absolute path whereas in older versions it contained relative path under some circumstances.
FUNCTION ___	The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__CLASS__	The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__METHOD_	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

4. Operators

What is Operator? Simple answer can be given using expression 4 + 5 is equal to 9. Here 4 and 5 are called operands and + is called operator. PHP language supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators
- String Operators

Operators Categories

All the operators we have discussed above can be categorized into following categories:

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

Precedence of PHP Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right

Conditional	?:	Right to left
Assignment	= += -= *= /= %=	Right to left

Let's have a look on all operators one by one.

Arithmetic Operators

There are following arithmetic operators supported by PHP language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide the numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Example

Try the following example to understand all the arithmetic operators. Copy and paste following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>
<head><title>Arithmetical Operators</title><head>
<body>
<?php
```

```
$a = 42;
$b = 20;
$c = $a + $b;
echo "Addition Operation Result: $c <br/>";
$c = $a - $b;
echo "Subtraction Operation Result: $c <br/>";
$c = $a * $b;
echo "Multiplication Operation Result: $c <br/>";
$c = $a / $b;
echo "Division Operation Result: $c <br/>";
$c = $a % $b;
echo "Modulus Operation Result: $c <br/>";
$c = $a++;
echo "Increment Operation Result: $c <br/>";
$c = $a--;
echo "Decrement Operation Result: $c <br/>";
?>
</body>
</html>
```

This will produce the following result:

Comparison Operators

There are following comparison operators supported by PHP language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

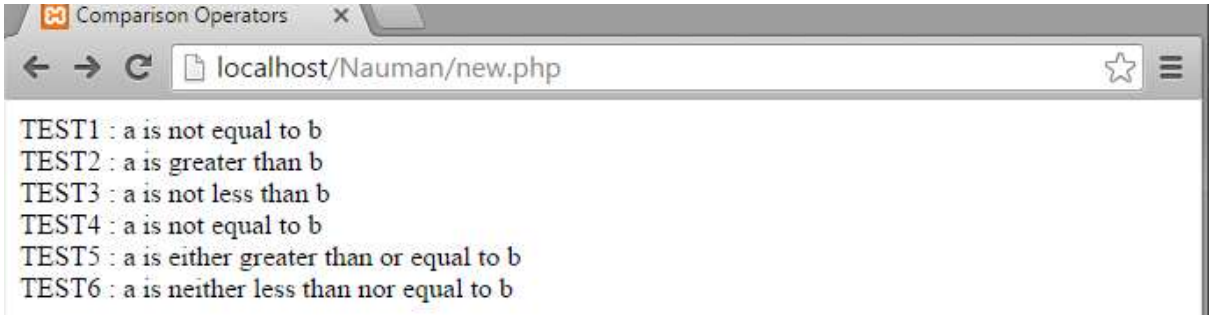
Example

Try the following example to understand all the comparison operators. Copy and paste following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>
<head><title>Comparison Operators</title><head>
<body>
<?php
$a = 42;
$b = 20;
if( $a == $b ){
echo "TEST1 : a is equal to b<br/>";
}else{
echo "TEST1 : a is not equal to b<br/>";
}
```

```
if( $a > $b ){
echo "TEST2 : a is greater than b<br/>";
}else{
echo "TEST2 : a is not greater than b<br/>";
}
if( $a < $b ){
echo "TEST3 : a is less than b<br/>";
}else{
echo "TEST3 : a is not less than b<br/>";
}
if( $a != $b ){
echo "TEST4 : a is not equal to b<br/>";
}else{
echo "TEST4 : a is equal to b<br/>";
}
if( $a >= $b ){
echo "TEST5 : a is either greater than or equal to b<br/>";
}else{
echo "TEST5 : a is neither greater than nor equal to b<br/>";
}
if( $a <= $b ){
echo "TEST6 : a is either less than or equal to b<br/>";
}else{
echo "TEST6 : a is neither less than nor equal to b<br/>";
}
?>
</body>
</html>
```

This will produce the following result:



Logical Operators

There are following logical operators supported by PHP language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

Example

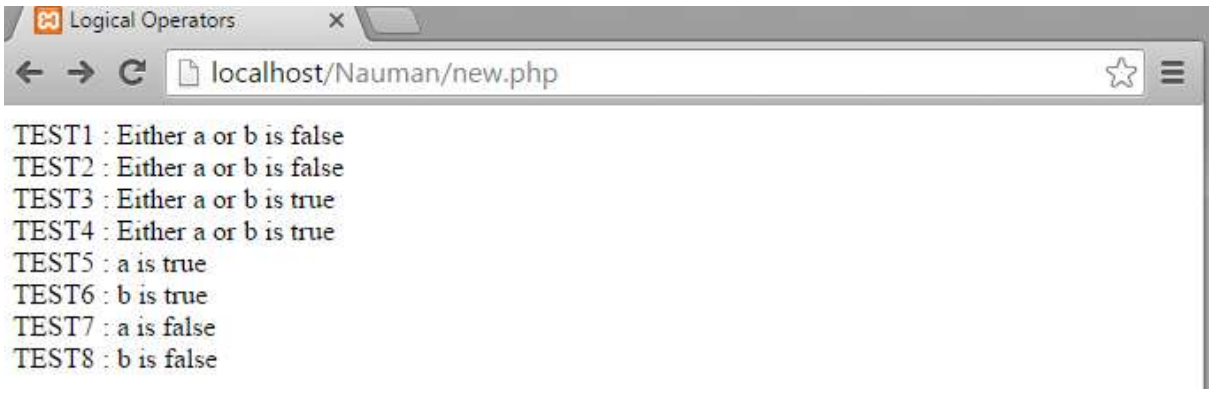
Try the following example to understand all the logical operators. Copy and paste the following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>
<head><title>Logical Operators</title><head>
<body>
<?php
$a = 42;
$b = 0;
if( $a && $b ){
echo "TEST1 : Both a and b are true<br/>";
```

```
}else{
echo "TEST1 : Either a or b is false<br/>";
}
if( $a and $b ){
echo "TEST2 : Both a and b are true<br/>";
}else{
echo "TEST2 : Either a or b is false<br/>";
}
if( $a || $b ){
echo "TEST3 : Either a or b is true<br/>";
}else{
echo "TEST3 : Both a and b are false<br/>";
}
if( $a or $b ){
echo "TEST4 : Either a or b is true<br/>";
}else{
echo "TEST4 : Both a and b are false<br/>";
}
$a = 10;
$b = 20;
if( $a ){
echo "TEST5 : a is true <br/>";
}else{
echo "TEST5 : a is false<br/>";
}
if( $b ){
echo "TEST6 : b is true <br/>";
}else{
echo "TEST6 : b is false<br/>";
}
if( !$a ){
echo "TEST7 : a is true <br/>";
```

```
}else{
echo "TEST7 : a is false<br/>";
}
if( !$b ){
echo "TEST8 : b is true <br/>";
}else{
echo "TEST8 : b is false<br/>";
}
?>
</body>
</html>
```

This will produce the following result:



Assignment Operators

PHP supports the following assignment operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A

/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Example

Try the following example to understand all the assignment operators. Copy and paste the following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>
<head><title>Assignment Operators</title><head>
<body>
<?php
$a = 42;
$b = 20;
$c = $a + $b;  /* Assignment operator */
echo "Addition Operation Result: $c <br/>";
$c += $a; /* c value was 42 + 20 = 62 */
echo "Add AND Assignment Operation Result: $c <br/>";
$c -= $a; /* c value was 42 + 20 + 42 = 104 */
echo "Subtract AND Assignment Operation Result: $c <br/>";
$c *= $a; /* c value was 104 - 42 = 62 */
echo "Multiply AND Assignment Operation Result: $c <br/>";
$c /= $a; /* c value was 62 * 42 = 2604 */
echo "Division AND Assignment Operation Result: $c <br/>";
$c %= $a; /* c value was 2604/42 = 62*/
echo "Modulus AND Assignment Operation Result: $c <br/>";
?>
</body>
</html>
```

This will produce the following result:

Conditional Operator

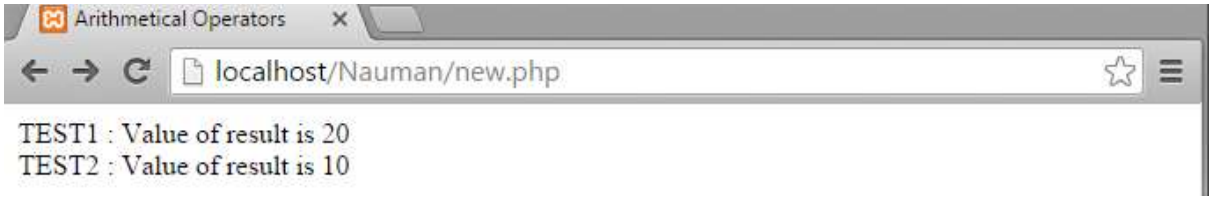
There is one more operator called the conditional operator. It first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Try the following example to understand the conditional operator. Copy and paste the following PHP program in test.php file and keep it in your PHP Server's document root and browse it using any browser.

```
<html>
<head><title>Arithmetical Operators</title><head>
<body>
<?php
$a = 10;
$b = 20;
/* If condition is true then assign a to result otherwise b */
$result = ($a > $b ) ? $a :$b;
echo "TEST1 : Value of result is $result<br/>";
/* If condition is true then assign a to result otherwise b */
$result = ($a < $b ) ? $a :$b;
echo "TEST2 : Value of result is $result<br/>";
?>
</body>
</html>
```

This will produce the following result:



String Operator (do your self)

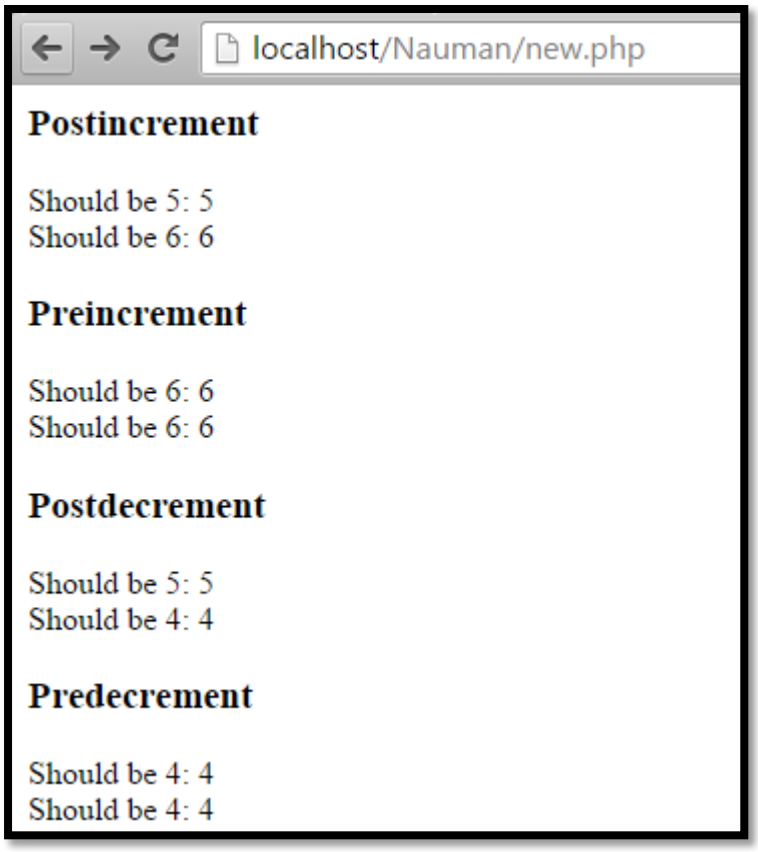
Incrementing/Decrementing Operators

Example	Name	Effect
++$a	Pre-increment	Increments $a by one, then returns $a.
$a++	Post-increment	Returns $a, then increments $a by one.
--$a	Pre-decrement	Decrements $a by one, then returns $a.
$a--	Post-decrement	Returns $a, then decrements $a by one.

Example

```
<?php
echo "<h3>Postincrement</h3>";
$a = 5;
echo "Should be 5: " . $a++ . "<br />\n";
echo "Should be 6: " . $a . "<br />\n";

echo "<h3>Preincrement</h3>";
$a = 5;
echo "Should be 6: " . ++$a . "<br />\n";
echo "Should be 6: " . $a . "<br />\n";

echo "<h3>Postdecrement</h3>";
$a = 5;
echo "Should be 5: " . $a-- . "<br />\n";
echo "Should be 4: " . $a . "<br />\n";

echo "<h3>Predecrement</h3>";
$a = 5;
echo "Should be 4: " . --$a . "<br />\n";
echo "Should be 4: " . $a . "<br />\n";
?>
```

Result:


localhost/Nauman/new.php
Postincrement
Should be 5: 5
Should be 6: 6
Preincrement
Should be 6: 6
Should be 6: 6
Postdecrement
Should be 5: 5
Should be 4: 4
Predecrement
Should be 4: 4
Should be 4: 4

5. Statement Blocks

Braces make blocks:

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces.

Here both statements are equivalent:

```
if (3 == 2 + 1)
print("Good - I haven't totally lost my mind.<br>");
if (3 == 2 + 1)
{
print("Good - I haven't totally"); print("lost my mind.<br>");
}
```

6. DECISION MAKING

The if, elseif ...else and switch statements are used to take decision based on the different condition.

You can use conditional statements in your code to make your decisions. PHP supports following three decision making statements:

- if...else statement - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- elseif statement - is used with the if...else statement to execute a set of code if one of several condition are true
- switch statement - is used 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.

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 (if Constructer)

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

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

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

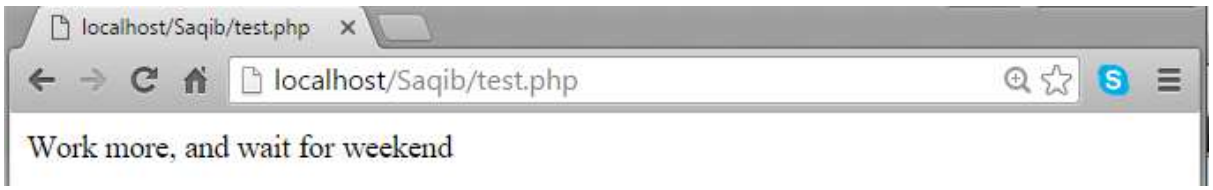
This will produce if you run this code on Friday otherwise it show blank.

Example (if and Else)

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
{
```

```
        echo "Hello!<br />";
        echo "Have a nice weekend!";
        echo "See you on Monday!";
}
else{
        echo "Hello!<br />";
        echo "Work more, and wait for weekend";
}
?>
</body>
</html>
```

This will produce the following result(if not Friday then):



This will produce the following result(if Friday then):

The ElseIf Statement

If you want to execute some code if one of the several conditions is true, then 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;
```

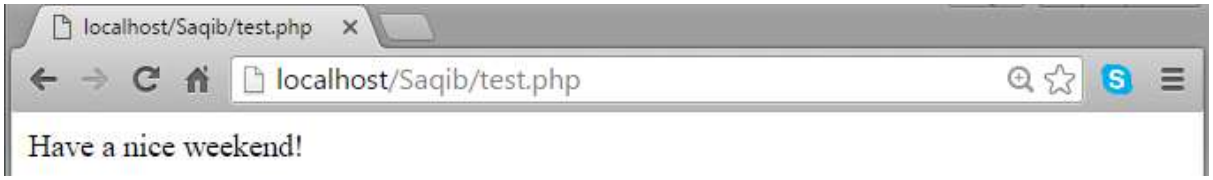
Example

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

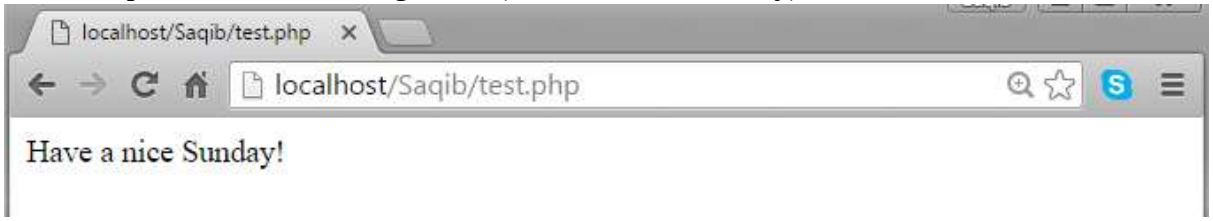
```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
echo "Have a nice weekend!";
elseif ($d=="Sun")
echo "Have a nice Sunday!";
else
echo "Have a nice day!";
```

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

This will produce the following result(if code run on Friday):



This will produce the following result (if code run on Sunday):



This will produce the following result (if code not run on Weekend or Sunday):

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;
}
```

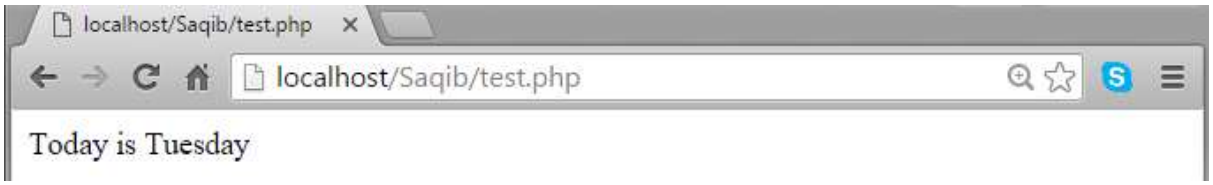
Example

The switch statement works in an unusual way. First it evaluates given expression then seeks a label to match the resulting value. If a matching value is found, then the code associated with the matching label will be executed. If none of the labels match, then the statement 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>
```

This will produce the following result:



7. LOOP TYPES

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

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

We will discuss about continue and break keywords used to control the loops execution.

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;
}
```

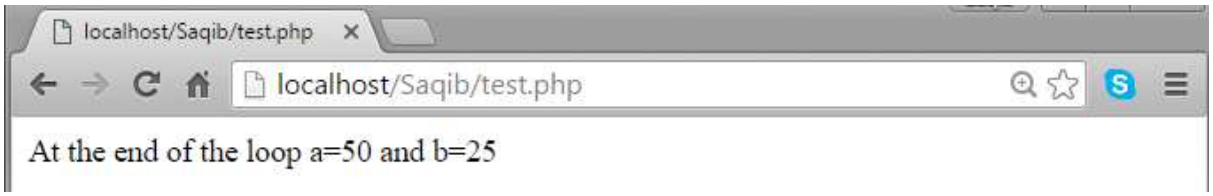
The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

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 the following result:



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

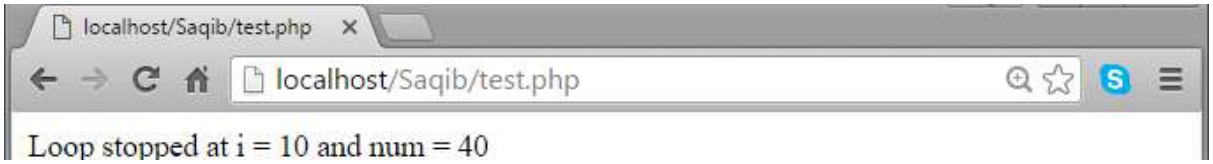
```
while (condition)
{
code to be executed;
}
```

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation becomes 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 the following result:



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);
```

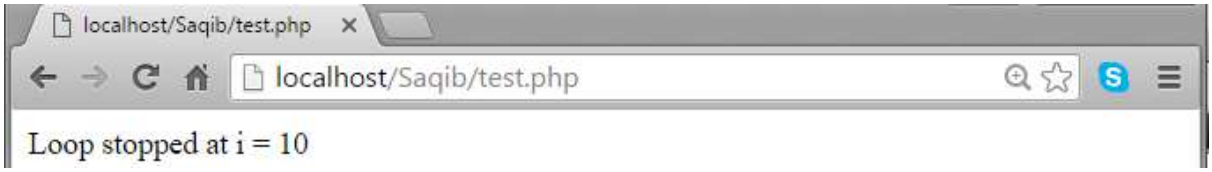
Example

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 the following result:



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

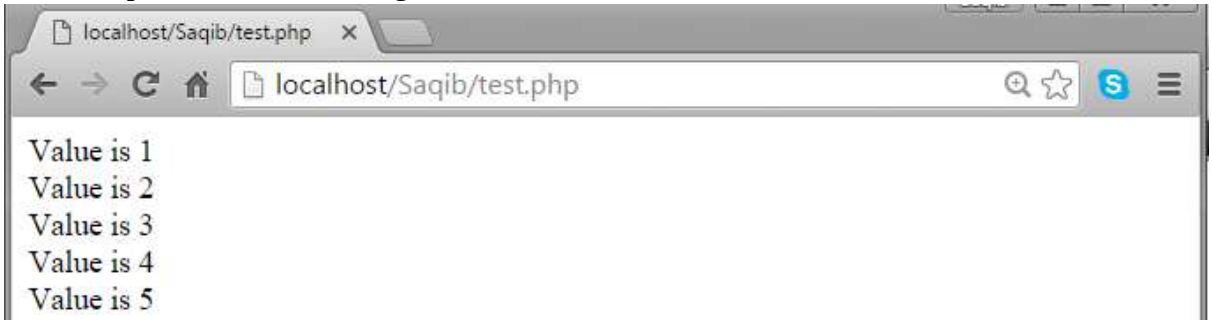
```
foreach (array as value)
{
code to be executed;
}
```

Example

Try out the 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 the following result:



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, the 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 the following result:

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.

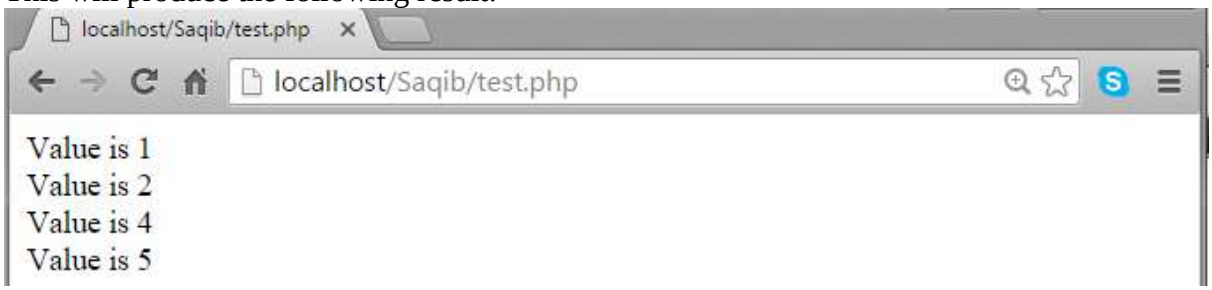
Example

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 the following result:



8. String

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

```
$string_1 = "This is a string in double quotes";
$string_2 = "This is a somewhat longer, singly quoted string";
$string_39 = "This string has thirty-nine characters";
$string_0 = ""; // a string with zero characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?php
  $variable = "name";
  $literally = 'My $variable will not print!\\n';

  print($literally);
  print "<br />";

  $literally = "My $variable will print!\\n";

  print($literally);
?>
```

This will produce the following result −

```
My $variable will not print!\n
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP −

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with $) are replaced with string representations of their values.

The escape-sequence replacements are −

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \$ is replaced by the dollar sign itself ($)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator −

```
<?php
  $string1="Hello World";
  $string2="1234";

  echo $string1 . " " . $string2;
?>
```

This will produce the following result −

```
Hello World 1234
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

Using the strlen() function

The strlen() function is used to find the length of a string.

Let's find the length of our string "Hello world!":

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

This will produce the following result −

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

Using the strpos() function

The strpos() function is used to search for a string or character within a string.

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string −

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

This will produce the following result −

```
6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

9. Array

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.

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

NOTE: Built-in array functions is given in function reference PHP Array Functions

Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default, the array index starts from zero.

Example

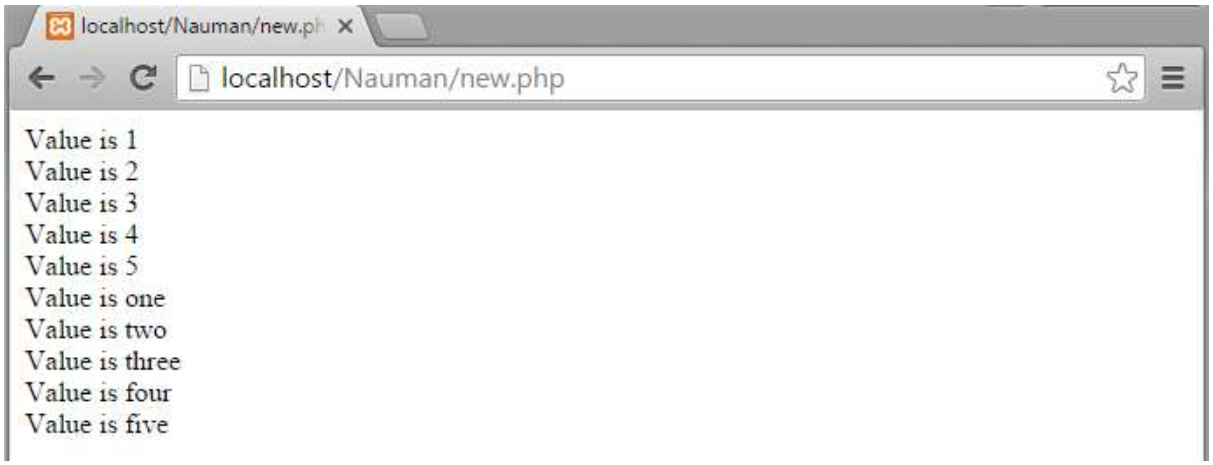
The following example demonstrates how to create and access numeric arrays.

Here we have used array() function to create array. This function is explained in function reference.

```
<html>
<body>
<?php
/* First method to create array. */
$numbers = array( 1, 2, 3, 4, 5);
foreach( $numbers as $value )
{
echo "Value is $value <br />";
}
/* Second method to create array. */
$numbers[0] = "one";
$numbers[1] = "two";
$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";
```

```
foreach( $numbers as $value )
{
echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce the following result:



Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

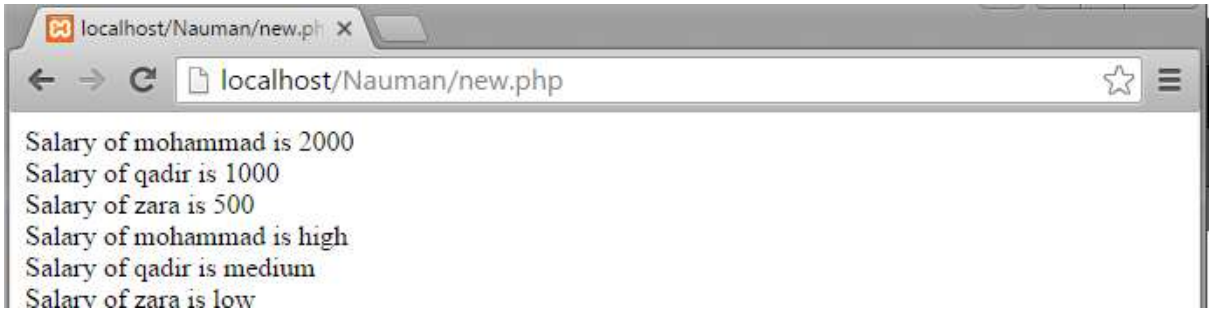
NOTE: Don't keep associative array inside double quote while printing, otherwise it would not return any value.

Example

```
<html>
<body>
<?php
/* First method to associate create array. */
$salaries = array(
"mohammad" => 2000, "qadir" => 1000, "zara" => 500
);
```

```
echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ". $salaries['qadir']. "<br />";
echo "Salary of zara is ". $salaries['zara']. "<br />";
/* Second method to create array. */
$salaries['mohammad'] = "high";
$salaries['qadir'] = "medium";
$salaries['zara'] = "low";
echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ". $salaries['qadir']. "<br />";
echo "Salary of zara is ". $salaries['zara']. "<br />";
?>
</body>
</html>
```

This will produce following result:



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

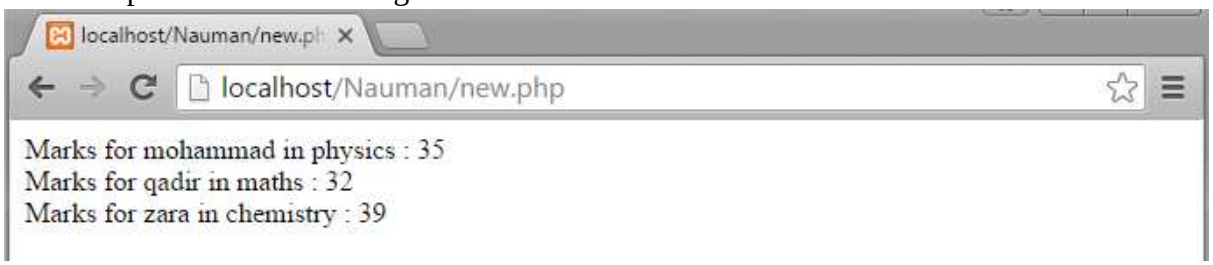
In this example, we create a two dimensional array to store marks of three students in three subjects:

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

```
<html>
<body>
<?php
$marks = array(
"mohammad" => array
(
```

```
"physics" => 35, "maths" => 30, "chemistry" => 39
),
"qadir" => array
(
"physics" => 30, "maths" => 32, "chemistry" => 29
),
"zara" => array
(
"physics" => 31, "maths" => 22, "chemistry" => 39
)
);
/* Accessing multi-dimensional array values */
echo "Marks for mohammad in physics : " ;
echo $marks['mohammad']['physics'] . "<br />";
echo "Marks for qadir in maths : ";
echo $marks['qadir']['maths'] . "<br />";
echo "Marks for zara in chemistry : " ;
echo $marks['zara']['chemistry'] . "<br />";
?>
</body>
</html>
```

This will produce the following result:



Array Functions

The array functions allow you to access and manipulate arrays.(do your self)

Sort Function For Array

Here, 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

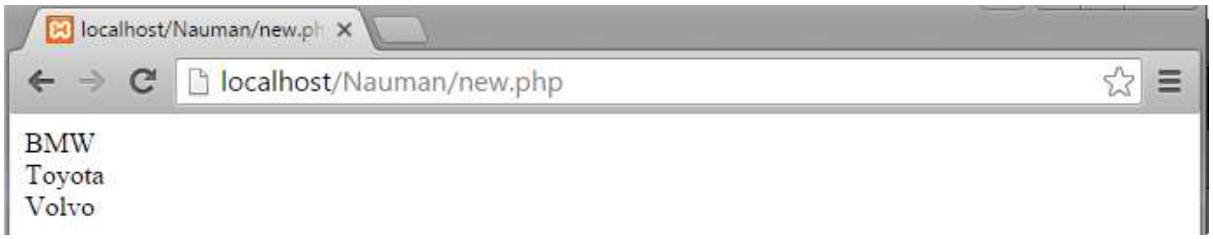
Sort Array in Ascending Order - sort()

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

Example

```
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
for($i=0;$i<3;$i++){
echo "$cars[$i]<br/>";
}
?>
</body>
</html>
```

Result;



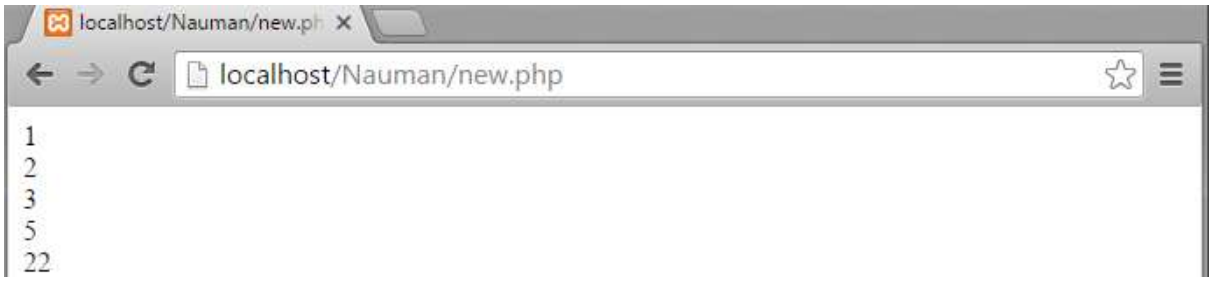
The following example sorts the elements of the $numbers array in ascending numerical order:

Example

```
<html>
<body>
<?php
$num = array(5,3,1,22,2);
sort($num);
for($i=0;$i<5;$i++){
echo "$num[$i]<br/>";
}
?>
</body>
```

```
</html>
```

Result:



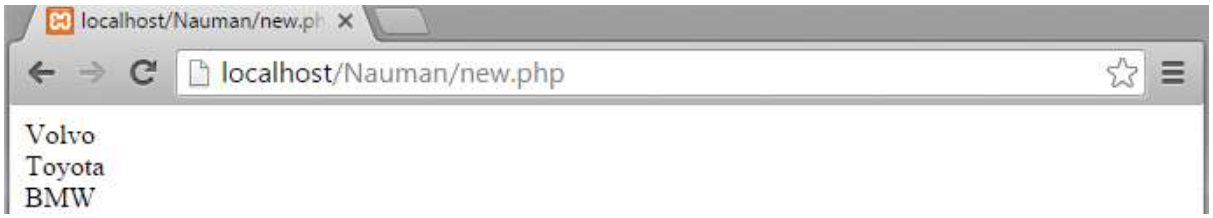
Sort Array in Descending Order - rsort()

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

Example

```
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);
for($i=0;$i<3;$i++){
echo "$cars[$i]<br/>";
}
?>
</body>
</html>
```

Result:

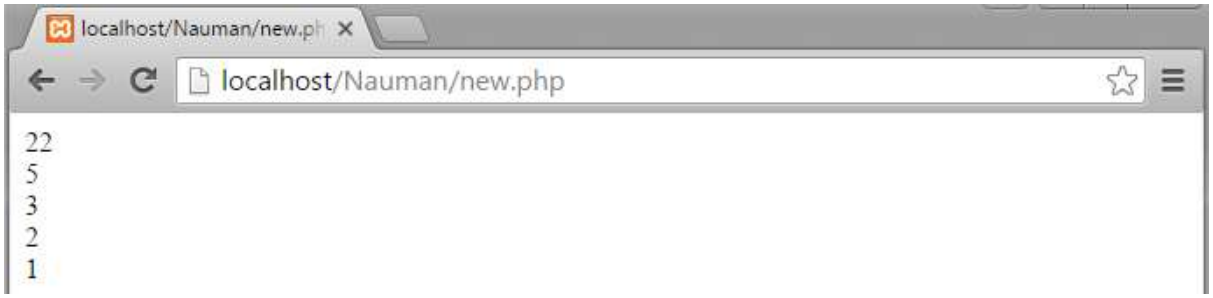


The following example sorts the elements of the $numbers array in descending numerical order:

Example

```
<html>
<body>
<?php
$num = array(5,3,1,22,2);
rsort($num);
for($i=0;$i<5;$i++){
echo "$num[$i]<br/>";
}
?>
</body>
</html>
```

Result:



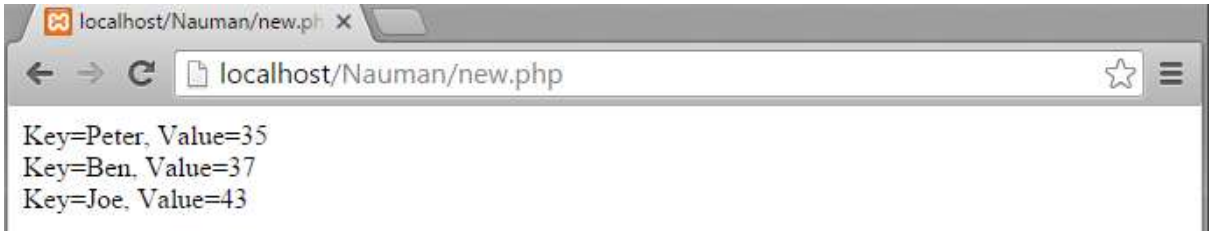
Sort Array (Ascending Order), According to Value - asort()

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

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);
foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
</body>
</html>
```

Result:



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

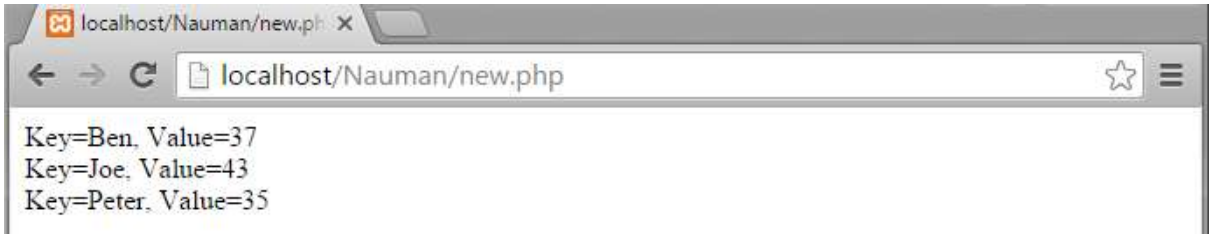
The following example sorts an associative array in ascending order, according to the key:

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
</body>
```

```
</html>
```

Result:



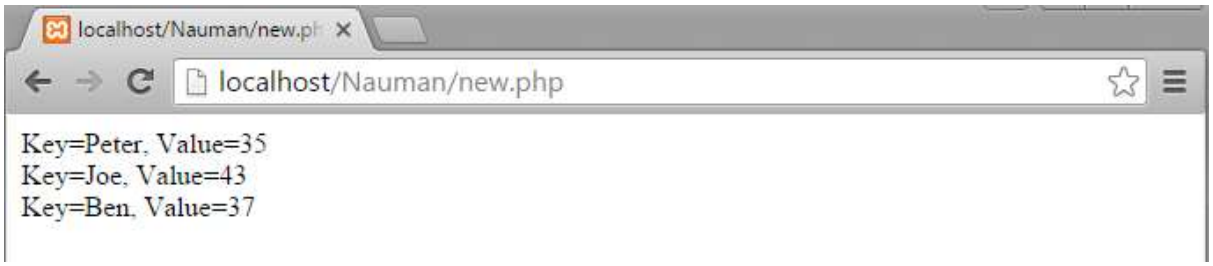
Sort Array (Descending Order), According to Key - krsort()

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

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);
foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
</body>
</html>
```

Result:



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

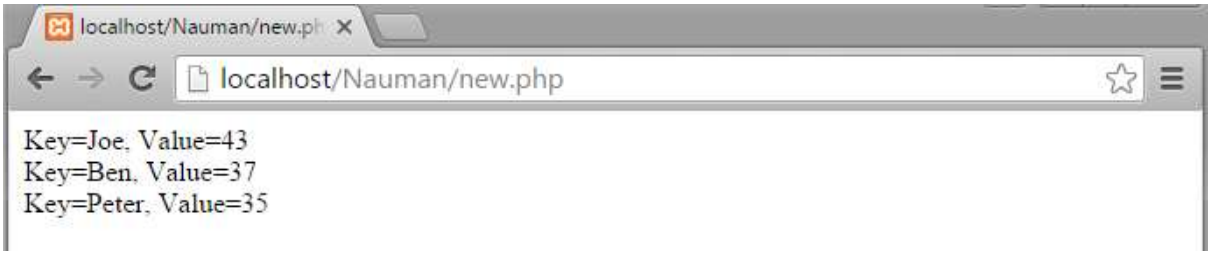
The following example sorts an associative array in descending order, according to the key:

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
```

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

Result:



10. 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

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Creating And Calling PHP Function

It is 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 nice person, Have a nice time!";
}
/* Calling a PHP Function */
writeMessage();
?>
</body>
</html>
```

This will display the following result:

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(10, 20);
?>
</body>
</html>
```

This will display following result:

Passing Arguments by Reference

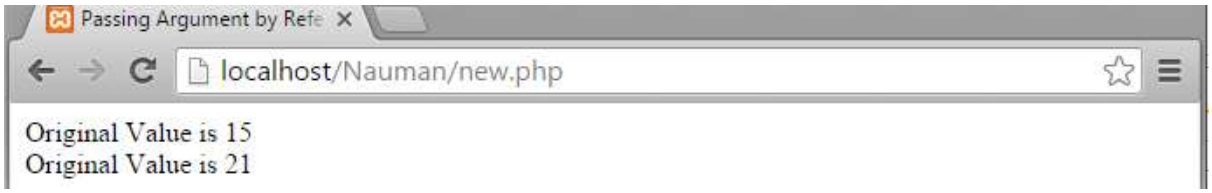
It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

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.

Following example depicts both the cases.

```
<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 the following result:



Passing Arguments by value

Call by value

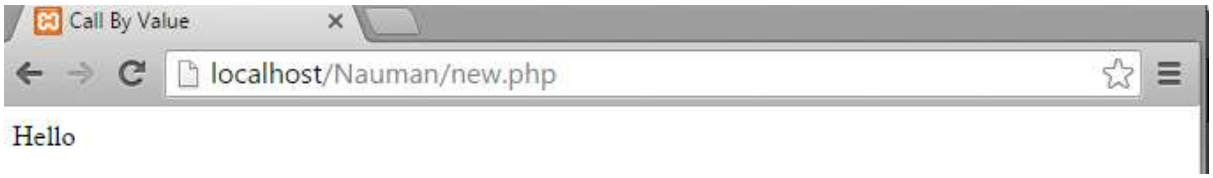
In call by value mechanism, the called function creates a new set of variables in stack and copies the values of the arguments into them.

Example: Program showing the Call by Value mechanism.

```
<html>
```

```
<head>
<title>Call By Value</title>
</head>
<body>
<?php
function adder($str2)
{
   $str2 = 'Call By Value';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
</body>
</html>
```

Result:



PHP Functions returning value

A function can return a value using the return statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using return array(1,2,3,4). Following example takes two integer parameters and add them together and then returns

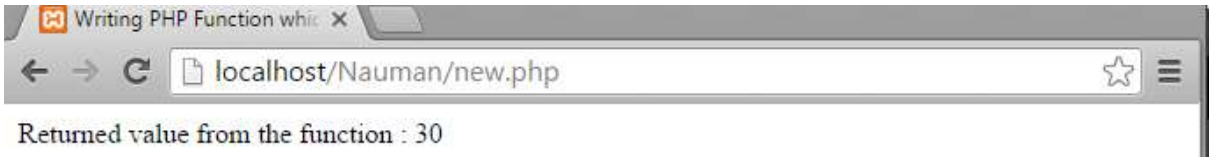
their sum to the calling program. Note that return keyword is used to return a value from

a function.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>
<?php
function addFunction($num1, $num2)
{
$sum = $num1 + $num2;
return $sum;
}
$return_value = addFunction(10, 20);
```

```
echo "Returned value from the function : $return_value";
?>
</body>
</html>
```

This will display the following result:

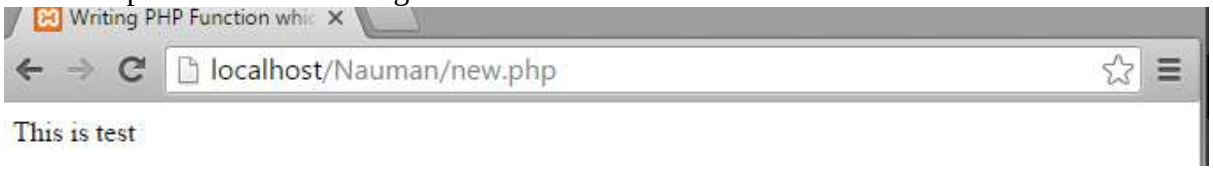


Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it. Following function prints NULL in case use does not pass any value to this function.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>
<?php
function printMe($param = NULL)
{
print $param;
}
printMe("This is test");
printMe();
?>
</body>
</html>
```

This will produce the following result:



Nested Functions

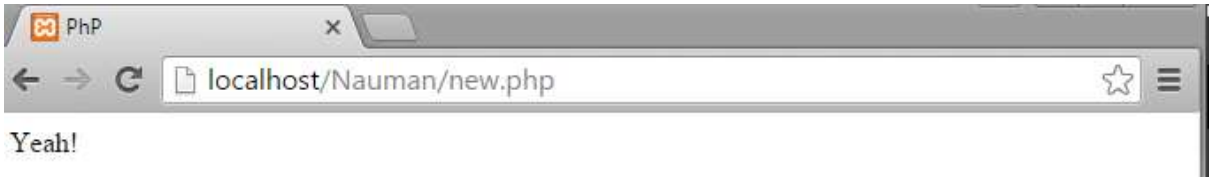
When you define a function within another function it does not exist until the parent function is executed. Once the parent function has been executed, the nested function is defined and as with any function, accessible from anywhere within the current document. If you have nested

functions in your code, you can only execute the outer function once. Repeated calls will try to redeclare the inner functions, which will generate an error.

Example:

```
<html>
<head>
<title>Passing Argument by Reference</title>
</head>
<body>
<?php
function MyFunc($content) {

 // The Nested Functions
 function DoThis() {
  return 'Yeah!';
 }
 function DoThat() {
  return 'Nah!';
 }

 // The Main Function Script
 if ($content == 'Yes yes') {
   return DoThis();
 } else {
   return DoThat();
 }

}

 // The Main Script
 $string = 'Yes yes';

 echo MyFunc($string);?>
</body>
</html>
```

Result:



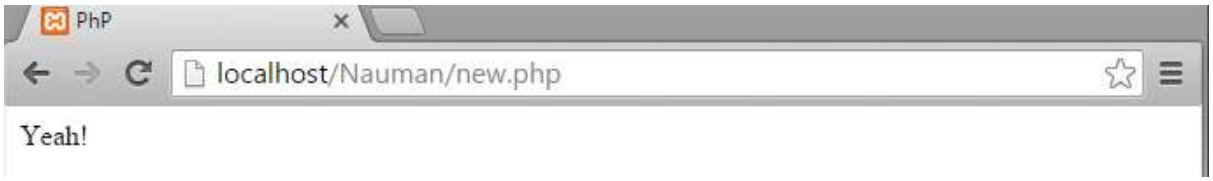
Conditionally Created Functions

Example:

```
<html>
<head>
<title>Passing Argument by Reference</title>
</head>
<body>
<?php
function MyFunc($content) {

 // The Nested Functions
```

```
 function DoThis() {
  return 'Yeah!';
 }
 function DoThat() {
  return 'Nah!';
 }

 // The Main Function Script
 if ($content == 'Yes yes') {
   return DoThis();
 } else {
   return DoThat();
 }

}

 // The Main Script
 $string = 'Yes yes';

 echo MyFunc($string);?>
</body>
</html>
```

Result:



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 behavior.

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

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

This will display the following result:

11. OBJECT ORIENTED PROGRAMMING

We can imagine our universe made of different objects like sun, earth, moon etc. Similarly, we can imagine our car made of different objects like wheel, steering, gear etc. In the same way, there are object oriented programming concepts which assume everything as an object and implement a software using different objects.

Object Oriented Concepts

Before we go in detail, let's define important terms related to Object Oriented Programming.

- **Class**: This is a programmer-defined datatype, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.
- **Object**: An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.
- **Member Variable**: These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.
- **Member function**: These are the function defined inside a class and are used to access object data.
- **Inheritance**: When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.
- **Parent class**: A class that is inherited from by another class. This is also called a base class or super class.
- **Child Class**: A class that inherits from another class. This is also called a subclass or derived class.
- **Polymorphism**: This is an object oriented concept where the same function can be used for different purposes. For example, function name will remain same but it make take different number of arguments and can do different task.
- **Overloading**: a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly, functions can also be overloaded with different implementation.
- **Data Abstraction**: Any representation of data in which the implementation details are hidden (abstracted).
- **Encapsulation**: refers to a concept where we encapsulate all the data and member functions together to form an object.
- **Constructor**: refers to a special type of function which will be called automatically whenever there is an object formation from a class.
- **Destructors**: refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

Defining PHP Classes

The general form for defining a new class in PHP is as follows:

```
<?php
class phpClass{
```

```
var $var1;
var $var2 = "constant string";
function myfunc ($arg1, $arg2) {
        [..]
}
       [..]
}
?>
```

Here is the description of each line:

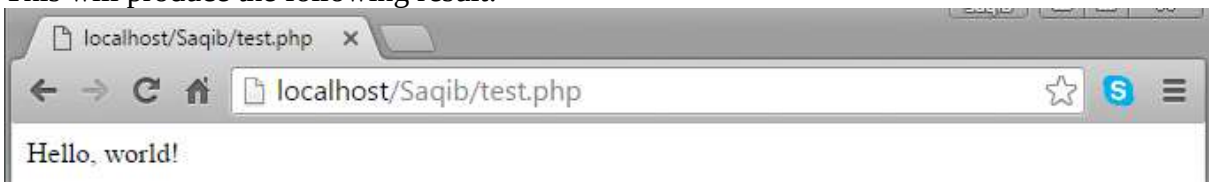
- The special form **class**, followed by the name of the class that you want to define.
- A set of braces enclosing any number of variable declarations and function definitions.
- Variable declarations start with the special form **var**, which is followed by a conventional $ variable name; they may also have an initial assignment to a constant value.
- Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.

Hello world with Class

```
<?php
class HelloWorld {
        function sayHello($language) {
                // Put the greeting within P tags:
                echo '<p>';
                // Print a message specific to alanguage:
                switch ($language) {
                        case 'Dutch':
                        echo 'Hallo, wereld!';
                        break;
                        case 'French':
                        echo 'Bonjour, monde!';
                        break;
                        case 'German':
                        echo 'Hallo, Welt!';
                        break;
                        case 'Italian':
                        echo 'Ciao, mondo!';
                        break;
                        case 'Spanish':
                        echo '¡Hola, mundo!';
                        break;
                        case 'English':
                        default:
                        echo 'Hello, world!';
                        break;
                } // End of switch.
                // Close the HTML paragraph:
                echo '</p>';
                } // End of sayHello() method.
```

```
}       //      End     of      HelloWorld      class.
?>
<?php
$newObj= new HelloWorld();
$newObj->sayHello('English');
?>
```

This will produce the following result:



Example

Here is an example which defines a class of Books type:

```
<?php
class Books{
        /* Member variables */
        var $price;
        var $title;
        /* Member functions */
        function setPrice($par){
                $this->price = $par;
        }
        function getPrice(){
                echo $this->price ."<br/>";
        }
        function setTitle($par){
                $this->title = $par;
        }
        function getTitle(){
                echo $this->title ." <br/>";
        }
}
?>
```

The variable **$this** is a special variable and it refers to the same object, i.e., itself.

Creating Objects in PHP

Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using new operator.

```
$physics = new Books;
$maths = new Books;
$chemistry = new Books;
```

Here we have created three objects and these objects are independent of each other and they will have their existence separately. Next, we will see how to access member function and process member variables.

Calling Member Functions

After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only.

Following example shows how to set title and prices for the three books by calling member functions.

```
$physics->setTitle( "Physics for High School" );
$chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );
$physics->setPrice( 10 );
$chemistry->setPrice( 15 );
$maths->setPrice( 7 );
```

Now you call another member functions to get the values set by in above example:

```
$physics->getTitle();
$chemistry->getTitle();
$maths->getTitle();
$physics->getPrice();
$chemistry->getPrice();
$maths->getPrice();
```

This will produce the following result:

```
Physics for High School
Advanced Chemistry
Algebra
10
15
7
```

Constructor Functions

Constructor Functions are special type of functions which are called automatically whenever an object is created. So we take full advantage of this behavior, by initializing many things through constructor functions.

PHP provides a special function called __construct() to define a constructor. You can pass as many as arguments you like into the constructor function.

Following example will create one constructor for Books class and it will initialize price and title for the book at the time of object creation.

```
function   construct( $par1, $par2 ){
        $this->price = $par1;
        $this->title = $par2;
}
```

Now we don't need to call set function separately to set price and title. We can initialize these two member variables at the time of object creation only. Check following example below:

```
$physics = new Books( "Physics for High School", 10 );

$maths = new Books ( "Advanced Chemistry", 15 );

$chemistry = new Books ("Algebra", 7 );

/* Get those set values */

$physics->getTitle();

$chemistry->getTitle();

$maths->getTitle();

$physics->getPrice();

$chemistry->getPrice();

$maths->getPrice();
```

This will produce the following result:

```
Physics for High School
Advanced Chemistry
Algebra
10
15
7
```

Destructor

Like a constructor function you can define a destructor function using function destruct(). You can release all the resources with-in a destructor.

Inheritance

PHP class definitions can optionally inherit from a parent class definition by using the extends clause. The syntax is as follows:

```
class Child extends Parent {

        <definition body>

}
```

The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics:

- Automatically has all the member variable declarations of the parent class.
- Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.

Following example inherit Books class and adds more functionality based on the requirement.

```
class Novel extends Books{
        var publisher;
        function setPublisher($par){
                $this->publisher = $par;
        }
        function getPublisher(){
                echo $this->publisher. "<br />";
        }
}
```

Now apart from inherited functions, class Novel keeps two additional member functions.

Function Overriding

Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.

In the following example, getPrice and getTitle functions are overriden to retrun some values.

```
function getPrice(){
        echo $this->price . "<br/>";
        return $this->price;
}
        function getTitle(){
        echo $this->title . "<br/>";
        return $this->title;
}
```

Public Members

Unless you specify otherwise, properties and methods of a class are public. That is to say, they may be accessed in three possible situations:

- From outside the class in which it is declared
- From within the class in which it is declared
- From within another class that implements the class in which it is declared

Till now we have seen all members as public members. If you wish to limit the accessibility of the members of a class then you define class members as private or protected.

Private members

By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.

A class member can be made private by using private keyword in front of the member.

```
class MyClass {
        private $car = "skoda";
        $driver = "SRK";
        function  construct($par) {
                // Statements here run every time
                // an instance of the class
                // is created.
        }
```

```
        function myPublicFunction() {
                return("I'm visible!");
        }
        private function myPrivateFunction() {
                return("I'm not visible outside!");
        }
}
```

When MyClass class is inherited by another class using extends, myPublicFunction() will be visible, as will $driver. The extending class will not have any awareness of or access to myPrivateFunction and $car, because they are declared private.

Protected members

A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class. Protected members are not available outside of those two kinds of classes. A class member can be made protected by using protected keyword in front of the member.

Here is different version of MyClass:

```
class MyClass {
        protected $car = "skoda";
        $driver = "SRK";
        function  construct($par) {
                // Statements here run every time
                // an instance of the class
                // is created.
        }
        function myPublicFunction() {
                return("I'm visible!");
        }
        protected function myPrivateFunction() {
                return("I'm visible in child class!");
        }
}
```

Interfaces

Interfaces are defined to provide a common function names to the implementors. Different implementors can implement those interfaces according to their requirements. You can say, interfaces are skeletons which are implemented by developers.

As of PHP5, it is possible to define an interface, like this:

```
interface Mail {
        public function sendMail();
}
```

Then, if another class implemented that interface, like this:

```
class Report implements Mail {
        // sendMail() Definition goes here
}
```

Constants

A constant is somewhat like a variable, in that it holds a value, but is really more like a function because a constant is immutable. Once you declare a constant, it does not change.

Declaring one constant is easy, as is done in this version of MyClass:

```
class MyClass {
        const requiredMargin = 1.7;
        function  construct($incomingValue) {
                // Statements here run every time
                // an instance of the class
                // is created.
        }
}
```

In this class, requiredMargin is a constant. It is declared with the keyword const, and under no circumstances can it be changed to anything other than 1.7. Note that the constant's name does not have a leading $, as variable names do.

Abstract Classes

An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword abstract, like this:

When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.

```
abstract class MyAbstractClass {
        abstract function myAbstractFunction() {
        }
}
```

Note that the function definitions inside an abstract class must also be preceded by the keyword abstract. It is not legal to have abstract function definitions inside a non-abstract class.

Static Keyword

Declaring class members or methods as static makes them accessible without needing an instantiation of the class. A member declared as static cannot be accessed with an instantiated class object (though a static method can).

Try out the following example:

```
<?php class Foo
{
        public static $my_static = 'foo';
        public function staticValue() {
        return self::$my_static;
}
}
        print Foo::$my_static . "\n";
        $foo = new Foo();
        print $foo->staticValue() . "\n";
?>
```

Final Keyword

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final then it cannot be extended.

Following example results in Fatal error: Cannot override final method BaseClass::moreTesting()

```
<?php
class BaseClass {
	public function test() {
		echo "BaseClass::test() called<br>";
	}
	final public function moreTesting() {
		echo "BaseClass::moreTesting() called<br>";
	}
}
class ChildClass extends BaseClass {
	public function moreTesting() {
		echo "ChildClass::moreTesting() called<br>";
	}
}
?>
```

Calling parent constructors

Instead of writing an entirely new constructor for the subclass, let's write it by calling the parent's constructor explicitly and then doing whatever is necessary in addition for instantiation of the subclass. Here's a simple example:

```
class Name
{
	var $_firstName;
	var $_lastName;
	function Name($first_name, $last_name)
	{
		$this->_firstName = $first_name;
		$this->_lastName = $last_name;
	}
	function toString() {
		return($this->_lastName .", " .$this->_firstName);
	}
}
class NameSub1 extends Name
{
	var $_middleInitial;
	function NameSub1($first_name, $middle_initial, $last_name) {
		Name::Name($first_name, $last_name);
		$this->_middleInitial = $middle_initial;
	}
	function toString() {
		return(Name::toString() . " " . $this->_middleInitial);
	}
```

```
}
```

In this example, we have a parent class (Name), which has a two-argument constructor, and a subclass (NameSub1), which has a three-argument constructor. The constructor of NameSub1 functions by calling its parent constructor explicitly using the :: syntax (passing two of its arguments along) and then setting an additional field. Similarly, NameSub1 defines its nonconstructor toString() function in terms of the parent function that it overrides.

NOTE: A constructor can be defined with the same name as the name of a class. It is defined in above example.

12. File Inclusion

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to included one PHP file into another PHP file.

- The include() Function
- The require() Function

This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required then instead of changing thousand of files just change included file.

The include() Function

The include() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **include()** function generates a warning but the script will continue execution.

Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

```
<a href="http://www.tutorialspoint.com/index.htm">Home</a> -
<a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -
<a href="http://www.tutorialspoint.com/ajax">AJAX</a> -
<a href="http://www.tutorialspoint.com/perl">PERL</a> <br />
```

Now create as many pages as you like and include this file to create header. For example now your test.php file can have following content.

```
<html>
  <body>
    <?php include("menu.php"); ?>
    <p>This is an example to show how to include PHP file!</p>
  </body>
</html>
```

It will produce the following result –

Home -
ebXML -
AJAX -
PERL

This is an example to show how to include PHP file!

The require() Function

The require() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **require()** function generates a fatal error and halt the execution of the script.

So there is no difference in require() and include() except they handle error conditions. It is recommended to use the require() function instead of include(), because scripts should not continue executing if files are missing or misnamed.

You can try using above example with require() function and it will generate same result. But if you will try following two examples where file does not exist then you will get different results.

```
<html>
  <body>
    <?php include("xxmenu.php"); ?>
    <p>This is an example to show how to include wrong PHP file!</p>
  </body>
</html>
```

This will produce the following result −

```
This is an example to show how to include wrong PHP file!
```

Now lets try same example with require() function.

```
<html>
  <body>
    <?php require("xxmenu.php"); ?>
    <p>This is an example to show how to include wrong PHP file!</p>
  </body>
</html>
```

This time file execution halts and nothing is displayed.

13. PHP FORMS

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

```
name1=value1&name2=value2&name3=value3
```

Spaces are removed and replaced with the + character and any other nonalphanumeric characters are replaced with a hexadecimal values. After the information is encoded it is sent to the server.

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?**character.

```
http://www.test.com/index.htm?name1=value1&name2=value2
```

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send upto 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **$_GET** associative array to access all the sent information using GET method.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_GET["name"] || $_GET["age"] ) {
     echo "Welcome ". $_GET['name']. "<br />";
     echo "You are ". $_GET['age']. " years old.";
```

```
    exit();
  }
?>
<html>
  <body>

    <form action = "<?php $_PHP_SELF ?>" method = "GET">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" />
    </form>

  </body>
</html>
```

It will produce the following result −

The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **$_POST** associative array to access all the sent information using POST method.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_POST["name"] || $_POST["age"] ) {
    if (preg_match("/[^A-Za-z'-]/",$_POST['name'] )) {
```

```
      die ("invalid name and name should be alpha");
    }
    echo "Welcome ". $_POST['name']. "<br />";
    echo "You are ". $_POST['age']. " years old.";

    exit();
  }
?>
<html>
  <body>

    <form action = "<?php $_PHP_SELF ?>" method = "POST">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" />
    </form>

  </body>
</html>
```

It will produce the following result −

The $_REQUEST variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE. We will discuss $_COOKIE variable when we will explain about cookies.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Try out following example by putting the source code in test.php script.

```
<?php
  if( $_REQUEST["name"] || $_REQUEST["age"] ) {
    echo "Welcome ". $_REQUEST['name']. "<br />";
```

```
    echo "You are ". $_REQUEST['age']. " years old.";
    exit();
  }
?>
<html>
  <body>

    <form action = "<?php $_PHP_SELF ?>" method = "POST">
      Name: <input type = "text" name = "name" />
      Age: <input type = "text" name = "age" />
      <input type = "submit" />
    </form>

  </body>
</html>
```

Here $_PHP_SELF variable contains the name of self script in which it is being called.

It will produce the following result −

PHP forms Handing:

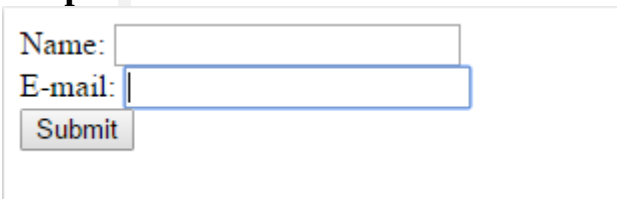
One of the most powerful features of PHP is the way it handles HTML forms. The basic concept that is important to understand is that any form element will automatically be available to your PHP scripts.

The example below displays a simple HTML form with two input fields and a submit button:

Code:

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

Output:



PHP forms Validation:

I tell you how construct and validate a simple form using HTML and PHP. The form is created using HTML and validation and processing of the form's contents is done with PHP. The goal is to teach you some basic HTML form elements and how their data is accessible to you in your PHP scripts.

First, let's look at the form. The purpose of the form is to capture user details (name, address, and email) to obtain Comment.

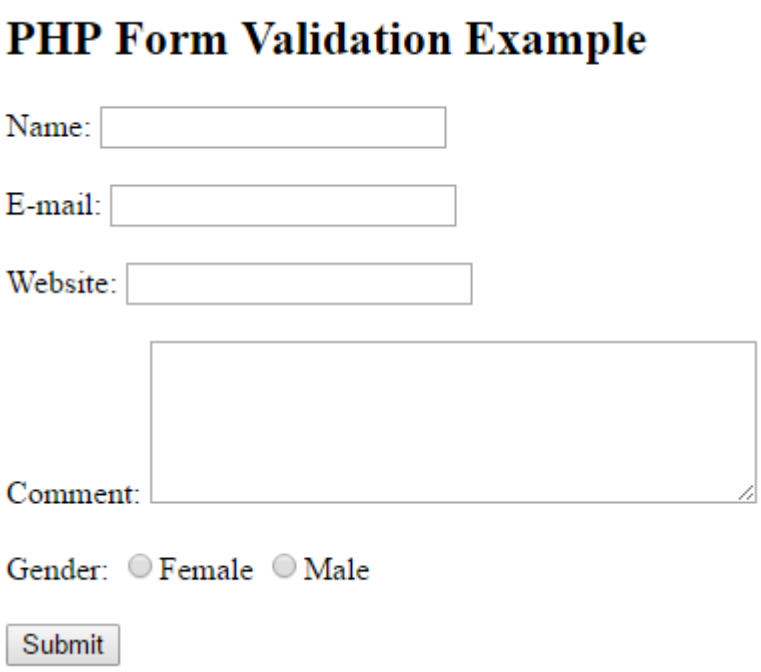
The example below displays a simple HTML form with input fields and a submit button code:

Code:

```
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>

<?php
// define variables and set to empty values
$name=$email=$gender=$comment=$website= "";

if ($_SERVER["REQUEST_METHOD"]== "POST")
{
  $name=test_input($_POST["name"]);
  $email=test_input($_POST["email"]);
  $website=test_input($_POST["website"]);
  $comment=test_input($_POST["comment"]);
  $gender=test_input($_POST["gender"]);
}

function test_input($data)
{
  $data=trim($data);
  $data=stripslashes($data);
  $data=htmlspecialchars($data);
  return $data;
}
?>

<h2>PHPFormValidationExample</h2>
<form method="post" action="<?php echohtmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name">
  <br><br>
  E-mail: <input type="email">
```

```
  <br><br>
  Website: <input type="text" name="website">
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>YourInput:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

Output:



Enter Name, E-mail, Website, Comment and Gender:

PHP Form Validation Example

Name: muzammil

E-mail: muzammil.hassan@kics.edu

Website: kics.edu.pk

Comment: good work

Gender: Female Male

Submit

After click on submit button:

PHP Form Validation Example

Name:

E-mail:

Website:

Comment:

Gender: Female Male

Submit

Your Input:

muzammil
muzammil.hassan@kics.edu.pk
kics.edu.pk
good work
male

PHP Required Fields:

From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

In the following code we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each $_POST variable. This checks if the $_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function:

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

Code:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr=$emailErr=$genderErr=$websiteErr= "";
$name=$email=$gender=$comment=$website= "";
```

```
if ($_SERVER["REQUEST_METHOD"] == "POST")
{
  if (empty($_POST["name"]))
{
 $nameErr= "Nameisrequired";
  } else {
  $name=test_input($_POST["name"]);
  }

  if (empty($_POST["email"]))
{
  $emailErr= "Emailisrequired";
  } else {
  $email=test_input($_POST["email"]);
  }

  if (empty($_POST["website"]))
{
  $website= "";
  } else {
  $website=test_input($_POST["website"]);
  }

  if (empty($_POST["comment"]))
{
   $comment= "";
  } else
{

$comment=test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"]))
{
  $genderErr= "Genderisrequired";
  } else {
  $gender=test_input($_POST["gender"]);
  }
}

function test_input($data)
{
  $data=trim($data);
  $data=stripslashes($data);
  $data=htmlspecialchars($data);
  return $data;
}
?>

<h2>PHPFormValidationExample</h2>
<p><span class="error">*requiredfield.</span></p>
<form method="post" action="<?php echohtmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name">
  <span class="error">* <?php echo $nameErr;?></span>
```

```
  <br><br>
  E-mail: <input type="text" name="email">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>YourInput:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

Output:

PHP Form Validation Example

* required field.

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male *

Submit

Your Input:

Error generate if fields is empty:

PHP Form Validation Example

* required field.

Name: * Name is required

E-mail: * Email is required

Website:

Comment:

Gender: Female Male * Gender is required

Submit

Your Input:

Enter Name, E-mail, Website, Comment and Gender:

PHP Form Validation Example

Name: muzammil

E-mail: muzammil.hassan@kics.edu

Website: kics.edu.pk

Comment: good work

Gender: Female Male

Submit

PHP Form Validation Example

Name:

E-mail:

Website:

Comment:

Gender: Female Male

Submit

Your Input:

muzammil
muzammil.hassan@kics.edu.pk
kics.edu.pk
good work
male

Validation Name, E-mail and URL:

Validate Name:

The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

Validate E-mail:

The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function.

In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

Validate URL:

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

Code for Validation Name, E-mail and URL:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr=$emailErr=$genderErr=$websiteErr= "";
$name=$email=$gender=$comment=$website= "";

if ($_SERVER["REQUEST_METHOD"] == "POST")
{
  if (empty($_POST["name"]))
{
```

```
  $nameErr= "Name_is_required";
  } else {
  $name=test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name))
{
  $nameErr= "Only_letters_and_white_space_allowed";
   }
  }

  if (empty($_POST["email"]))
{
  $emailErr= "Email_is_required";
  } else {
  $email=test_input($_POST["email"]);
   // check if e-mail address is well-formed
   if (!filter_var($email, FILTER_VALIDATE_EMAIL))
{
  $emailErr= "Invalidemailformat";
    }
  }

  if (empty($_POST["website"]))
{
  $website= "";
  } else
{
  $website=test_input($_POST["website"]);
   // check if URL address syntax is valid
  if                    (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website))
{
 $websiteErr= "InvalidURL";
   }
  }

  if (empty($_POST["comment"]))
{
  $comment= "";
  } else
 {
  $comment=test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"]))
{
  $genderErr= "Genderisrequired";
  } else
 {
  $gender=test_input($_POST["gender"]);
  }
}

function test_input($data)
```

```
{
  $data=trim($data);
  $data=stripslashes($data);
  $data=htmlspecialchars($data);
  return $data;
}
?>

<h2>PHPFormValidationExample</h2>
<p><span class="error">*requiredfield.</span></p>
<form method="post" action="<?php echohtmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>YourInput:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

Output:

PHP Form Validation Example

* required field.

Name: * Name is required

E-mail: * Email is required

Website:

Comment:

Gender: Female Male * Gender is required

Submit

Your Input:

If you enter invalid formats:

PHP Form Validation Example

* required field.

Name: * Only letters and white space allowed

E-mail: * Invalid email format

Website: Invalid URL

Comment:

Gender: Female Male * Gender is required

Submit

Your Input:

PHP Form Complete:

Here is the complete code for the PHP Form Validation Example:

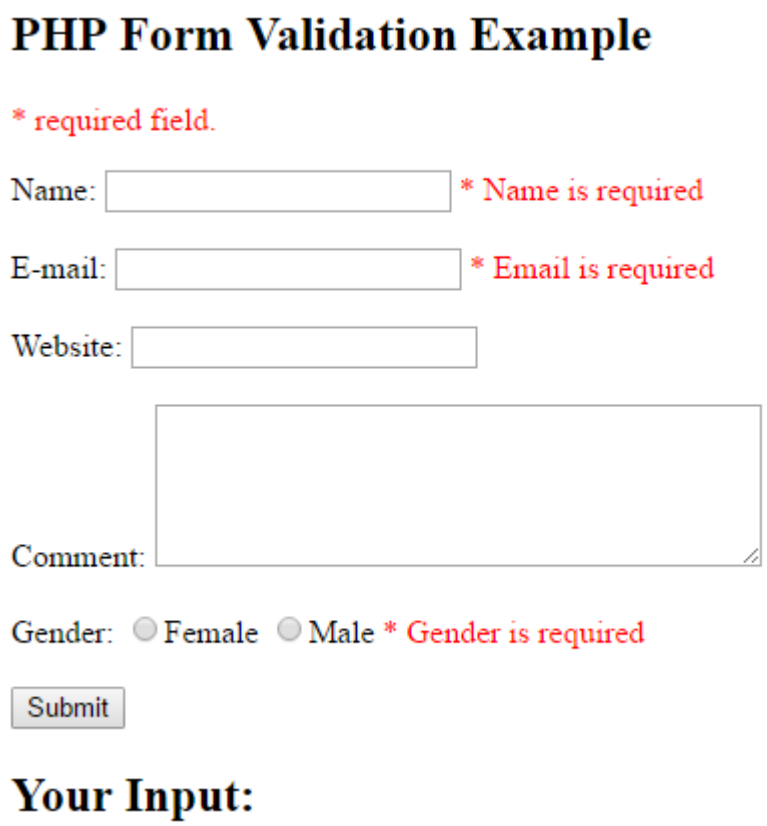
Code:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr=$emailErr=$genderErr=$websiteErr= "";
$name=$email=$gender=$comment=$website= "";

if ($_SERVER["REQUEST_METHOD"] == "POST")
{
  if (empty($_POST["name"]))
{
  $nameErr= "Name_is_required";
  } else {
  $name=test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name))
{
  $nameErr= "Only_letters_and_white_space_allowed";
   }
  }

  if (empty($_POST["email"]))
{
  $emailErr= "Email_is_required";
  } else {
  $email=test_input($_POST["email"]);
   // check if e-mail address is well-formed
   if (!filter_var($email, FILTER_VALIDATE_EMAIL))
{
  $emailErr= "Invalidemailformat";
    }
  }

  if (empty($_POST["website"]))
{
  $website= "";
  } else
{
  $website=test_input($_POST["website"]);
   // check if URL address syntax is valid
  if                (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website))
{
```

```
 $websiteErr= "InvalidURL";
   }
  }

  if (empty($_POST["comment"]))
{
  $comment= "";
  } else
 {
  $comment=test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"]))
{
  $genderErr= "Genderisrequired";
  } else
 {
  $gender=test_input($_POST["gender"]);
  }
}

function test_input($data)
{
  $data=trim($data);
  $data=stripslashes($data);
  $data=htmlspecialchars($data);
  return $data;
}
?>

<h2>PHPFormValidationExample</h2>
<p><span class="error">*requiredfield.</span></p>
<form method="post" action="<?php echohtmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>YourInput:</h2>";
```

```
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

Output:



If you enter invalid formats:

PHP Form Validation Example

* required field.

Name: * Only letters and white space allowed

E-mail: * Invalid email format

Website: Invalid URL

Comment:

Gender: Female Male * Gender is required

Submit

Your Input:

14. File Uploading

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.

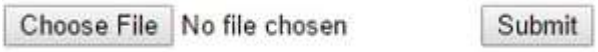
Creating an upload form

The following HTM code below creates an uploader form. This form is having method attribute set to **post** and enctype attribute is set to **multipart/form-data**

```
<?php
  if(isset($_FILES['image'])){
     $errors= array();
     $file_name = $_FILES['image']['name'];
     $file_size =$_FILES['image']['size'];
     $file_tmp =$_FILES['image']['tmp_name'];
     $file_type=$_FILES['image']['type'];
```

```
    $file_ext=strtolower(end(explode('.',$_FILES['image']['name'])));

    $expensions= array("jpeg","jpg","png");

    if(in_array($file_ext,$expensions)=== false){
      $errors[]="extension not allowed, please choose a JPEG or PNG file.";
    }

    if($file_size > 2097152){
      $errors[]='File size must be excately 2 MB';
    }

    if(empty($errors)==true){
      move_uploaded_file($file_tmp,"images/".$file_name);
      echo "Success";
    }else{
      print_r($errors);
    }
  }
?>
<html>
  <body>

    <form action="" method="POST" enctype="multipart/form-data">
      <input type="file" name="image" />
      <input type="submit"/>
    </form>

  </body>
</html>
```

It will produce the following result −



Creating an upload script

There is one global PHP variable called **$_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create following five variables −

- **$_FILES['file']['tmp_name']** − the uploaded file in the temporary directory on the web server.
- **$_FILES['file']['name']** − the actual name of the uploaded file.
- **$_FILES['file']['size']** − the size in bytes of the uploaded file.
- **$_FILES['file']['type']** − the MIME type of the uploaded file.
- **$_FILES['file']['error']** − the error code associated with this file upload.

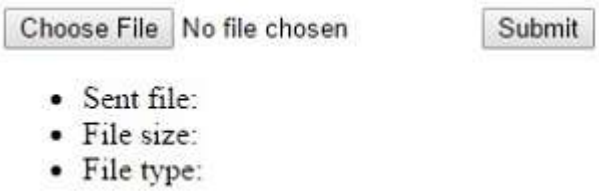
Example

Below example should allow upload images and gives back result as uploaded file information.

```
<?php
  if(isset($_FILES['image'])){
     $errors= array();
     $file_name = $_FILES['image']['name'];
     $file_size = $_FILES['image']['size'];
     $file_tmp = $_FILES['image']['tmp_name'];
     $file_type = $_FILES['image']['type'];
     $file_ext=strtolower(end(explode('.',$_FILES['image']['name'])));

     $expensions= array("jpeg","jpg","png");

     if(in_array($file_ext,$expensions)=== false){
       $errors[]="extension not allowed, please choose a JPEG or PNG file.";
     }

     if($file_size > 2097152) {
```

```
      $errors[]='File size must be excately 2 MB';
    }

    if(empty($errors)==true) {
      move_uploaded_file($file_tmp,"images/".$file_name);
      echo "Success";
    }else{
      print_r($errors);
    }
  }
?>
<html>
  <body>
    <form action = "" method = "POST" enctype = "multipart/form-data">
      <input type = "file" name = "image" />
      <input type = "submit"/>
      <ul>
        <li>Sent file: <?php echo $_FILES['image']['name'];  ?>
        <li>File size: <?php echo $_FILES['image']['size'];  ?>
        <li>File type: <?php echo $_FILES['image']['type'] ?>
      </ul>
    </form>
  </body>
</html>
```

It will produce the following result −



15. PHP & MySQL

PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.

What you should already have ?

- You have gone through MySQL tutorial to understand MySQL Basics.
- Downloaded and installed a latest version of MySQL.
- Created database user **guest** with password **guest123**.
- If you have not created a database then you would need root user and its password to create a database.

Opening Database Connection

PHP provides **mysql_connect** function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.

Syntax

```
connection mysql_connect(server,user,passwd,new_link,client_flag);
```

Sr.No	Parameter & Description
1	**server** Optional − The host name running database server. If not specified then default value is **localhost:3306**.
2	**user** Optional − The username accessing the database. If not specified then default is the name of the user that owns the server process.
3	**passwd** Optional − The password of the user accessing the database. If not specified then default is an empty password.
4	**new_link** Optional − If a second call is made to mysql_connect() with the same arguments, no new connection will be established; instead, the identifier of the already

	opened connection will be returned.
5	**client_flags** Optional − A combination of the following constants − • **MYSQL_CLIENT_SSL** − Use SSL encryption • **MYSQL_CLIENT_COMPRESS** − Use compression protocol • **MYSQL_CLIENT_IGNORE_SPACE** − Allow space after function names • **MYSQL_CLIENT_INTERACTIVE** − Allow interactive timeout seconds of inactivity before closing the connection

NOTE − You can specify server, user, passwd in **php.ini** file instead of using them again and again in your every PHP scripts. Check php.ini fileconfiguration.

Closing Database Connection

Its simplest function **mysql_close** PHP provides to close a database connection. This function takes connection resource returned by mysql_connect function. It returns TRUE on success or FALSE on failure.

Syntax

```
bool mysql_close ( resource $link_identifier );
```

If a resource is not specified then last opend database is closed.

Example

Try out following example to open and close a database connection −

```
<?php

  $dbhost = 'localhost:3036';
  $dbuser = 'guest';
  $dbpass = 'guest123';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }
```

```
  echo 'Connected successfully';
  mysql_close($conn);
?>
```

Creating a Database

To create and delete a database you should have admin privilege. Its very easy to create a new MySQL database. PHP uses **mysql_query** function to create a MySQL database. This function takes two parameters and returns TRUE on success or FALSE on failure.

Syntax

```
bool mysql_query( sql, connection );
```

Sr.No	Parameter & Description
1	**sql** Required - SQL query to create a database
2	**connection** Optional - if not specified then last opend connection by mysql_connect will be used.

Example

Try out following example to create a database −

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }

  echo 'Connected successfully';
```

```
  $sql = 'CREATE Database test_db';
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not create database: ' . mysql_error());
  }

  echo "Database test_db created successfully\n";
  mysql_close($conn);
?>
```

Selecting a Database

Once you establish a connection with a database server then it is required to select a particular database where your all the tables are associated.

This is required because there may be multiple databases residing on a single server and you can do work with a single database at a time.

PHP provides function **mysql_select_db** to select a database.It returns TRUE on success or FALSE on failure.

Syntax

```
bool mysql_select_db( db_name, connection );
```

Sr.No	Parameter & Description
1	**db_name** Required - Database name to be selected
2	**connection** Optional - if not specified then last opend connection by mysql_connect will be used.

Example

Here is the example showing you how to select a database.

```
<?php
```

```
  $dbhost = 'localhost:3036';
  $dbuser = 'guest';
  $dbpass = 'guest123';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }

  echo 'Connected successfully';

  mysql_select_db( 'test_db' );
  mysql_close($conn);

?>
```

Creating Database Tables

To create tables in the new database you need to do the same thing as creating the database. First create the SQL query to create the tables then execute the query using mysql_query() function.

Example

Try out following example to create a table −

```
<?php

  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }
```

```
   echo 'Connected successfully';

   $sql = 'CREATE TABLE employee( '.
      'emp_id INT NOT NULL AUTO_INCREMENT, '.
      'emp_name VARCHAR(20) NOT NULL, '.
      'emp_address  VARCHAR(20) NOT NULL, '.
      'emp_salary   INT NOT NULL, '.
      'join_date    timestamp(14) NOT NULL, '.
      'primary key ( emp_id ))';
   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not create table: ' . mysql_error());
   }

   echo "Table employee created successfully\n";

   mysql_close($conn);
?>
```

In case you need to create many tables then its better to create a text file first and put all the SQL commands in that text file and then load that file into $sql variable and excute those commands.

Consider the following content in sql_query.txt file

```
CREATE TABLE employee(
   emp_id INT NOT NULL AUTO_INCREMENT,
   emp_name VARCHAR(20) NOT NULL,
   emp_address  VARCHAR(20) NOT NULL,
   emp_salary   INT NOT NULL,
   join_date    timestamp(14) NOT NULL,
   primary key ( emp_id ));
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
```

```
   $dbpass = 'rootpassword';
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }

   $query_file = 'sql_query.txt';

   $fp = fopen($query_file, 'r');
   $sql = fread($fp, filesize($query_file));
   fclose($fp);

   mysql_select_db('test_db');
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not create table: ' . mysql_error());
   }

   echo "Table employee created successfully\n";
   mysql_close($conn);
?>
```

Deleting a Database

If a database is no longer required then it can be deleted forever. You can use pass an SQL command to **mysql_query** to delete a database.

Example

Try out following example to drop a database.

```
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';
```

```
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }

  $sql = 'DROP DATABASE test_db';
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not delete database db_test: ' . mysql_error());
  }

  echo "Database deleted successfully\n";

  mysql_close($conn);
?>
```

WARNING − its very dangerous to delete a database and any table. So before deleting any table or database you should make sure you are doing everything intentionally.

Deleting a Table

Its again a matter of issuing one SQL command through **mysql_query**function to delete any database table. But be very careful while using this command because by doing so you can delete some important information you have in your table.

Example

Try out following example to drop a table −

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
```

```
   die('Could not connect: ' . mysql_error());
  }

  $sql = 'DROP TABLE employee';
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not delete table employee: ' . mysql_error());
  }

  echo "Table deleted successfully\n";

  mysql_close($conn);
?>
```

Inserting Data Into Database

Data can be entered into MySQL tables by executing SQL INSERT statement through PHP function **mysql_query**. Below a simple example to insert a record into **employee** table.

Example

Try out following example to insert record into employee table.

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';
  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }

  $sql = 'INSERT INTO employee '.
    '(emp_name,emp_address, emp_salary, join_date) '.
```

```
    'VALUES ( "guest", "XYZ", 2000, NOW() )';

  mysql_select_db('test_db');
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not enter data: ' . mysql_error());
  }

  echo "Entered data successfully\n";

  mysql_close($conn);
?>
```

In real application, all the values will be taken using HTML form and then those values will be captured using PHP script and finally they will be inserted into MySQL tables.

While doing data insert its best practice to use function **get_magic_quotes_gpc()** to check if current configuration for magic quote is set or not. If this function returns false then use function **addslashes()** to add slashes before quotes.

Example

Try out this example by putting this code into add_employee.php, this will take input using HTML Form and then it will create records into database.

```
<html>

  <head>
    <title>Add New Record in MySQL Database</title>
  </head>

  <body>
    <?php
      if(isset($_POST['add'])) {
        $dbhost = 'localhost:3036';
        $dbuser = 'root';
```

```
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
  die('Could not connect: ' . mysql_error());
}

if(! get_magic_quotes_gpc() ) {
  $emp_name = addslashes ($_POST['emp_name']);
  $emp_address = addslashes ($_POST['emp_address']);
}else {
  $emp_name = $_POST['emp_name'];
  $emp_address = $_POST['emp_address'];
}

$emp_salary = $_POST['emp_salary'];

$sql = "INSERT INTO employee ". "(emp_name,emp_address, emp_salary,
  join_date) ". "VALUES('$emp_name','$emp_address',$emp_salary, NOW())";

mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );

if(! $retval ) {
  die('Could not enter data: ' . mysql_error());
}

echo "Entered data successfully\n";

mysql_close($conn);
}else {
  ?>
```

```
<form method = "post" action = "<?php $_PHP_SELF ?>">
  <table width = "400" border = "0" cellspacing = "1"
    cellpadding = "2">

    <tr>
      <td width = "100">Employee Name</td>
      <td><input name = "emp_name" type = "text"
        id = "emp_name"></td>
    </tr>

    <tr>
      <td width = "100">Employee Address</td>
      <td><input name = "emp_address" type = "text"
        id = "emp_address"></td>
    </tr>

    <tr>
      <td width = "100">Employee Salary</td>
      <td><input name = "emp_salary" type = "text"
        id = "emp_salary"></td>
    </tr>

    <tr>
      <td width = "100"> </td>
      <td> </td>
    </tr>

    <tr>
      <td width = "100"> </td>
      <td>
        <input name = "add" type = "submit" id = "add"
```

```
                    value = "Add Employee">
                </td>
              </tr>

            </table>
          </form>

        <?php
      }
    ?>

  </body>
</html>
```

Getting Data From Database

Data can be fetched from MySQL tables by executing SQL SELECT statement through PHP function mysql_query. You have several options to fetch data from MySQL.

The most frequently used option is to use function **mysql_fetch_array()**. This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.

Below is a simple example to fetch records from **employee** table.

Example

Try out following example to display all the records from employee table.

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';

  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
```

```
  }

  $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
  mysql_select_db('test_db');
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not get data: ' . mysql_error());
  }

  while($row = mysql_fetch_array($retval, MYSQL_ASSOC)) {
    echo "EMP ID :{$row['emp_id']}  <br> ".
      "EMP NAME : {$row['emp_name']} <br> ".
      "EMP SALARY : {$row['emp_salary']} <br> ".
      "--------------------------------<br>";
  }

  echo "Fetched data successfully\n";

  mysql_close($conn);
?>
```

The content of the rows are assigned to the variable $row and the values in row are then printed.

NOTE − Always remember to put curly brackets when you want to insert an array value directly into a string.

In above example the constant **MYSQL_ASSOC** is used as the second argument to mysql_fetch_array(), so that it returns the row as an associative array. With an associative array you can access the field by using their name instead of using the index.

PHP provides another function called **mysql_fetch_assoc()** which also returns the row as an associative array.

Example

Try out following example to display all the records from employee table using mysql_fetch_assoc() function.

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';

  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }

  $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
  mysql_select_db('test_db');
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not get data: ' . mysql_error());
  }

  while($row = mysql_fetch_assoc($retval)) {
    echo "EMP ID :{$row['emp_id']}  <br> ".
      "EMP NAME : {$row['emp_name']} <br> ".
      "EMP SALARY : {$row['emp_salary']} <br> ".
      "--------------------------------<br>";
  }

  echo "Fetched data successfully\n";

  mysql_close($conn);
```

```
?>
```

You can also use the constant **MYSQL_NUM**, as the second argument to mysql_fetch_array(). This will cause the function to return an array with numeric index.

Example

Try out following example to display all the records from employee table using MYSQL_NUM argument.

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';

  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }

  $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
  mysql_select_db('test_db');
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not get data: ' . mysql_error());
  }

  while($row = mysql_fetch_array($retval, MYSQL_NUM)) {
    echo "EMP ID :{$row[0]}  <br> ".
      "EMP NAME : {$row[1]} <br> ".
      "EMP SALARY : {$row[2]} <br> ".
      "--------------------------------<br>";
  }
```

```
  echo "Fetched data successfully\n";

  mysql_close($conn);
?>
```

All the above three examples will produce same result.

Releasing Memory

Its a good practice to release cursor memory at the end of each SELECT statement. This can be done by using PHP function **mysql_free_result()**. Below is the example to show how it has to be used.

Example

Try out following example

```
<?php
  $dbhost = 'localhost:3036';
  $dbuser = 'root';
  $dbpass = 'rootpassword';

  $conn = mysql_connect($dbhost, $dbuser, $dbpass);

  if(! $conn ) {
    die('Could not connect: ' . mysql_error());
  }

  $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
  mysql_select_db('test_db');
  $retval = mysql_query( $sql, $conn );

  if(! $retval ) {
    die('Could not get data: ' . mysql_error());
  }

  while($row = mysql_fetch_array($retval, MYSQL_NUM)) {
    echo "EMP ID :{$row[0]}  <br> ".
```

```
      "EMP NAME : {$row[1]} <br> ".
      "EMP SALARY : {$row[2]} <br> ".
      "--------------------------------<br>";
  }

  mysql_free_result($retval);
  echo "Fetched data successfully\n";

  mysql_close($conn);
?>
```

While fetching data you can write as complex SQL as you like. Procedure will remain same as mentioned above.

Using Paging Through PHP

Its always possible that your SQL SELECT statement query may result into thousand of records. But its is not good idea to display all the results on one page. So we can divide this result into many pages as per requirement.

Paging means showing your query result in multiple pages instead of just put them all in one long page.

MySQL helps to generate paging by using **LIMIT** clause which will take two arguments. First argument as OFFSET and second argument how many records should be returned from the database.

Below is a simple example to fetch records using **LIMIT** clause to generate paging.

Example

Try out following example to display 10 records per page.

```
<html>

  <head>
    <title>Paging Using PHP</title>
  </head>

  <body>
```

```php
<?php
   $dbhost = 'localhost:3036';
   $dbuser = 'root';
   $dbpass = 'rootpassword';

   $rec_limit = 10;
   $conn = mysql_connect($dbhost, $dbuser, $dbpass);

   if(! $conn ) {
      die('Could not connect: ' . mysql_error());
   }
   mysql_select_db('test_db');

   /* Get total number of records */
   $sql = "SELECT count(emp_id) FROM employee ";
   $retval = mysql_query( $sql, $conn );

   if(! $retval ) {
      die('Could not get data: ' . mysql_error());
   }
   $row = mysql_fetch_array($retval, MYSQL_NUM );
   $rec_count = $row[0];

   if( isset($_GET{'page'} ) ) {
      $page = $_GET{'page'} + 1;
      $offset = $rec_limit * $page ;
   }else {
      $page = 0;
      $offset = 0;
   }

   $left_rec = $rec_count - ($page * $rec_limit);
```

```
$sql = "SELECT emp_id, emp_name, emp_salary ".
   "FROM employee ".
   "LIMIT $offset, $rec_limit";

$retval = mysql_query( $sql, $conn );

if(! $retval ) {
   die('Could not get data: ' . mysql_error());
}

while($row = mysql_fetch_array($retval, MYSQL_ASSOC)) {
   echo "EMP ID :{$row['emp_id']}  <br> ".
      "EMP NAME : {$row['emp_name']} <br> ".
      EMP SALARY : {$row['emp_salary']} <br> ".
      "--------------------------------<br>";
}

if( $page > 0 ) {
   $last = $page - 2;
   echo "<a href = \"$_PHP_SELF?page = $last\">Last 10 Records</a> |";
   echo "<a href = \"$_PHP_SELF?page = $page\">Next 10 Records</a>";
}else if( $page == 0 ) {
   echo "<a href = \"$_PHP_SELF?page = $page\">Next 10 Records</a>";
}else if( $left_rec < $rec_limit ) {
   $last = $page - 2;
   echo "<a href = \"$_PHP_SELF?page = $last\">Last 10 Records</a>";
}

mysql_close($conn);
?>

</body>
```

```
</html>
```

Updating Data Into Database

Data can be updated into MySQL tables by executing SQL UPDATE statement through PHP function **mysql_query**.

Below is a simple example to update records into **employee** table. To update a record in any table it is required to locate that record by using a conditional clause. Below example uses primary key to match a record in employee table.

Example

Try out following example to understand update operation. You need to provide an employee ID to update an employee salary.

```
<html>

  <head>
    <title>Update a Record in MySQL Database</title>
  </head>

  <body>
    <?php
      if(isset($_POST['update'])) {
        $dbhost = 'localhost:3036';
        $dbuser = 'root';
        $dbpass = 'rootpassword';

        $conn = mysql_connect($dbhost, $dbuser, $dbpass);

        if(! $conn ) {
          die('Could not connect: ' . mysql_error());
        }

        $emp_id = $_POST['emp_id'];
        $emp_salary = $_POST['emp_salary'];
```

```
$sql = "UPDATE employee ". "SET emp_salary = $emp_salary ".
  "WHERE emp_id = $emp_id" ;
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );

if(! $retval ) {
  die('Could not update data: ' . mysql_error());
}
echo "Updated data successfully\n";

mysql_close($conn);
}else {
?>
  <form method = "post" action = "<?php $_PHP_SELF ?>">
    <table width = "400" border =" 0" cellspacing = "1"
      cellpadding = "2">

      <tr>
        <td width = "100">Employee ID</td>
        <td><input name = "emp_id" type = "text"
          id = "emp_id"></td>
      </tr>

      <tr>
        <td width = "100">Employee Salary</td>
        <td><input name = "emp_salary" type = "text"
          id = "emp_salary"></td>
      </tr>

      <tr>
        <td width = "100"> </td>
        <td> </td>
```

```
            </tr>

            <tr>
              <td width = "100"> </td>
              <td>
                <input name = "update" type = "submit"
                  id = "update" value = "Update">
              </td>
            </tr>

          </table>
        </form>
      <?php
    }
  ?>

  </body>
</html>
```

Deleting Data From Database

Data can be deleted from MySQL tables by executing SQL DELETE statement through PHP function **mysql_query**.

Below is a simple example to delete records into **employee** table. To delete a record in any table it is required to locate that record by using a conditional clause. Below example uses primary key to match a record in employee table.

Example

Try out following example to understand delete operation. You need to provide an employee ID to delete an employee record from employee table.

```
<html>

  <head>
    <title>Delete a Record from MySQL Database</title>
  </head>
```

```
<body>
  <?php
    if(isset($_POST['delete'])) {
      $dbhost = 'localhost:3036';
      $dbuser = 'root';
      $dbpass = 'rootpassword';
      $conn = mysql_connect($dbhost, $dbuser, $dbpass);

      if(! $conn ) {
        die('Could not connect: ' . mysql_error());
      }

      $emp_id = $_POST['emp_id'];

      $sql = "DELETE FROM employee WHERE emp_id = $emp_id" ;
      mysql_select_db('test_db');
      $retval = mysql_query( $sql, $conn );

      if(! $retval ) {
        die('Could not delete data: ' . mysql_error());
      }

      echo "Deleted data successfully\n";

      mysql_close($conn);
    }else {
      ?>
        <form method = "post" action = "<?php $_PHP_SELF ?>">
          <table width = "400" border = "0" cellspacing = "1"
            cellpadding = "2">
```

```
            <tr>
              <td width = "100">Employee ID</td>
              <td><input name = "emp_id" type = "text"
                id = "emp_id"></td>
            </tr>

            <tr>
              <td width = "100"> </td>
              <td> </td>
            </tr>

            <tr>
              <td width = "100"> </td>
              <td>
                <input name = "delete" type = "submit"
                  id = "delete" value = "Delete">
              </td>
            </tr>

          </table>
        </form>
      <?php
    }
  ?>

  </body>
</html>
```

16. Sessions

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

The location of the temporary file is determined by a setting in the **php.ini** file called **session.save_path**. Before using any session variable make sure you have setup this path.

When a session is started following things happen −

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
- A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie sess_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the **session_start()** function.This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to **session_start()** at the beginning of the page.

Session variables are stored in associative array called **$_SESSION[]**. These variables can be accessed during lifetime of a session.

The following example starts a session then registers a variable called **counter** that is incremented each time the page is visited during the session.

Make use of **isset()** function to check if session variable is already set or not.

Put this code in a test.php file and load this file many times to see the result −

```
<?php
  session_start();

  if( isset( $_SESSION['counter'] ) ) {
    $_SESSION['counter'] += 1;
  }else {
    $_SESSION['counter'] = 1;
  }

  $msg = "You have visited this page ".  $_SESSION['counter'];
  $msg .= "in this session.";
?>

<html>

  <head>
    <title>Setting up a PHP session</title>
  </head>

  <body>
    <?php  echo ( $msg ); ?>
  </body>

</html>
```

It will produce the following result −

```
You have visited this page 1in this session.
```

Destroying a PHP Session

A PHP session can be destroyed by **session_destroy()** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use **unset()** function to unset a session variable.

Here is the example to unset a single variable −

```
<?php
  unset($_SESSION['counter']);
?>
```

Here is the call which will destroy all the session variables −

```
<?php
  session_destroy();
?>
```

Turning on Auto Session

You don't need to call start_session() function to start a session when a user visits your site if you can set **session.auto_start** variable to 1 in **php.ini** file.

Sessions without cookies

There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

```
<?php
  session_start();

  if (isset($_SESSION['counter'])) {
    $_SESSION['counter'] = 1;
  }else {
    $_SESSION['counter']++;
  }

  $msg = "You have visited this page ".  $_SESSION['counter'];
  $msg .= "in this session.";
```

```
   echo ( $msg );
?>

<p>
   To continue  click following link <br />

   <a  href = "nextpage.php?<?php echo htmlspecialchars(SID); ?>">
</p>
```

It will produce the following result −

```
You have visited this page 1in this session.
To continue click following link
```

The **htmlspecialchars()** may be used when printing the SID in order to prevent XSS related attacks.

17. PHP ADVANCED

PHP Date and Time:

The PHP Date() Function:

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

Syntax
date(*format*,*timestamp*)

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

Note:

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

Code;

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

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

</body>
</html>
```

Output:

Today is 2016/10/07
Today is 2016.10.07
Today is 2016-10-07
Today is Friday

Another Code:

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

<?php
echo "The time is" .date("h:i:sa");
?>

</body>
</html>
```

Output;

The time is 03:54:47am

The PHP Time Function:

Here are some characters that are commonly used for times:

h - 12-hour format of an hour with leading zeros (01 to 12)

i - Minutes with leading zeros (00 to 59)

s - Seconds with leading zeros (00 to 59)

a - Lowercase Ante meridiem and Post meridiem (am or pm)

Code:

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

<?php
echo "The time is " . date("h:i:sa");
?>

</body>
</html>
```

Output:

The time is 05:21:43am

If the time you got back from the code is not the right time, it's probably because your server is in another country or set up for a different timezone.

So, if you need the time to be correct according to a specific location, you can set a timezone to use.

The example below sets the timezone to "America/New_York", then outputs the current time in the specified format:

Code:

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

<?php
date_default_timezone_set("America/New_York");
echo "The time is " . date("h:i:sa");
?>

</body>
</html>
```

Output:

The time is 05:23:20am

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.

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):

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

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

</body>
</html>
```

Output:

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 Language236

PHP File Open/Read/Close:

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:

Code:

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

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

</body>
</html>
```

Output:

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 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.

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 File Create/Write:

PHP Create File - fopen():

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

Code:

```
$myfile = fopen("testfile.txt", "w")
```

PHP Write to File - fwrite():

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "newfile.txt":

Code:

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

PHP Cookies:

A cookie is often used to identify a user.

Create Cookies With PHP:

A cookie is created with the setcookie() function.

Syntax
setcookie(*name, value, expire, path, domain, secure, httponly*);

Only the *name* parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie:

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

Code:

```
<!DOCTYPE html>
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
```

```
<p><strong>Note:</strong> You might have to reload the page to see the value of the cookie.</p>

</body>
</html>
```

Output:

Cookie named 'user' is not set!

Note: You might have to reload the page to see the value of the cookie.

PHP Filters:

Validating data = Determine if the data is in proper form.

Sanitizing data = Remove any illegal character from the data.

The PHP Filter Extension:

PHP filters are used to validate and sanitize external input.

The PHP filter extension has many of the functions needed for checking user input, and is designed to make data validation easier and quicker.

The filter_list() function can be used to list what the PHP filter extension offers:

Code:

```
<!DOCTYPE html>
<html>
<head>
<style>
table, th, td {
   border: 1px solid black;
   border-collapse: collapse;
}
th, td {
   padding: 5px;
}
</style>
</head>
<body>

<table>
  <tr>
    <td>Filter Name</td>
    <td>Filter ID</td>
  </tr>
  <?php
  foreach (filter_list() as $id =>$filter) {
     echo '<tr><td>' . $filter . '</td><td>' . filter_id($filter) . '</td></tr>';
  }
  ?>
</table>
```

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

Output:

Filter Name	Filter ID
int	257
boolean	258
float	259
validate_regexp	272
validate_url	273
validate_email	274
validate_ip	275
string	513
stripped	513
encoded	514
special_chars	515
full_special_chars	522
unsafe_raw	516
email	517
url	518
number_int	519
number_float	520
magic_quotes	521
callback	1024

Why Use Filters?

Many web applications receive external input. External input/data can be:

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

PHP filter_var() Function:

The filter_var() function both validate and sanitize data.

The filter_var() function filters a single variable with a specified filter. It takes two pieces of data:

- The variable you want to check

- The type of check to use

Sanitize a String:

The following example uses the filter_var() function to remove all HTML tags from a string:

Code:

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

<?php
$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str, FILTER_SANITIZE_STRING);
echo $newstr;
?>

</body>
</html>
```

Output:

Hello World!

Validate an Integer:

The following example uses the filter_var() function to check if the variable $int is an integer. If $int is an integer, the output of the code above will be: "Integer is valid". If $int is not an integer, the output will be: "Integer is not valid":

Code:

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

<?php
$int = 100;

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

</body>
</html>
```

Output:

Integer is valid

Validate an IP Address:

The following example uses the filter_var() function to check if the variable $ip is a valid IP address:

Code:

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

<?php
$ip = "127.0.0.1";

if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {
   echo("$ip is a valid IP address");
} else {
   echo("$ip is not a valid IP address");
}
?>

</body>
</html>
```

Output:

127.0.0.1 is a valid IP address

Sanitize and Validate an Email Address:

The following example uses the filter_var() function to first remove all illegal characters from the $email variable, then check if it is a valid email address:

Code:

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

<?php
$email = "john.doe@example.com";

// Remove all illegal characters from email
$email = filter_var($email, FILTER_SANITIZE_EMAIL);

// Validate e-mail
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
   echo("$email is a valid email address");
} else {
   echo("$email is not a valid email address");
}
?>

</body>
</html>
```

Output:

john.doe@example.com is a valid email address

Sanitize and Validate a URL:

The following example uses the filter_var() function to first remove all illegal characters from a URL, then check if $url is a valid URL:

Code:

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

<?php
$url = "http://www.w3schools.com";

// Remove all illegal characters from a url
$url = filter_var($url, FILTER_SANITIZE_URL);

// Validate url
if (!filter_var($url, FILTER_VALIDATE_URL) === false) {
    echo("$url is a valid URL");
} else {
    echo("$url is not a valid URL");
}
?>

</body>
</html>
```

Output:

http://www.w3schools.com is a valid URL

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 are sent to the browser.

PHP Error Handling:

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

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

We will show different error handling methods:

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

Basic Error Handling: Using the die() function:

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

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

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

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

To 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.

PHP Exception Handling:

What is an Exception:

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

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

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

We will show different error handling methods:

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

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

Basic Use of Exceptions:

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

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

Let's 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
```