PHP FULL TUTORIAL GUIDE

WWW.COMPUTINGTUTORIAL.COM

Table of Contents

INTRODUCTION

I want to thank you and congratulate you for choosing this book. This book contains elaborate steps, strategies and guidelines on how to create codes using the correct syntax, keywords, and punctuation marks. It also provides tips on how to correctly use arrays, forms, conditional statements, sessions etc. You will also learn how to open, read, write, and close files in PHP. There are sample programs(examples) that can serve as your guidelines when writing your code. This book also contains useful information about PHP, including their features. I hope it will be of help to you!

Chapter 1:PHP INTRODUCTION

Requirement

Before you proceed to study php, it is recommended that you have a basic understanding of the following:

.CSS

· HTML

What is PHP

· PHP is an acronym for Hypertext Preprocessor

.PHP is a HTML embedded scripting language

· PHP scripts are executed on the server

· PHP files have extension “.php”

What PHP can do

It is also helpful to think of PHP in terms of what it can do for you.

. Reduce the time to create large websites.

. Create a customized user experience for visitors based on information that you have gathered from them.

· can generate dynamic page content

· can collect form data

· can send and receive cookies

· can add, delete, modify data in your databases

· can create, open, read, write, and close files on the server

· can encrypt data

Why choose PHP

· PHP runs on various operating system platforms (Windows, Unix, etc.)

· PHP is easy to learn and runs efficiently on the server side

· PHP is compatible with almost all servers used today (Apache,etc.)

· PHP supports a wide range of databases

· PHP is freely available

PHP Syntax

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

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

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

As you notice in the below example, there was a semicolon after the line of PHP code. The semicolon signifies the end of a PHP statement and should never be forgotten. The semi colon terminates that statement.

Here is an example of a simple PHP file, with a PHP script that uses a PHP function (built-in)“echo” to output the text “Hello World!”

```
<html>
<head>
<title>My First PHP Page</title>
</head>
<body>
<?php
echo “Hello World!”;
?>
</body>
</html>
```

Display:

```
Hello World!
```

Comments in PHP

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

PHP supports three forms or ways of expressing a comment

Comments are useful for:

· To remind yourself what you did - Most programmers have experienced coming back to their Comments,this remind you of what you were thinking when you wrote the code

· To let others understand what you are doing - Comments let other programmers understand each of your steps

// This is a single line comment

# This is also a single line comment

/*

This is a multiple lines comment block that spans over more than one line

*/

Case sensitivity in PHP

In PHP, all variables are case-sensitive.

In the next chapter is a detailed discussion of the variables

In the example below, only the first statement will display the value of the $pet variable (because $pet, $PeT, and $PET are treated as three totally different variables):

Example

```
<html>
<body>
<?php
$pet="cat";
echo "My animal is a " . $pet . "<br>";
echo "My animal is a " . $PeT . "<br>";
echo "My animal is a " . $PET. "<br>";
?>
</body>
</html>
```

In PHP, all user-defined functions, keywords, and classes are case-insensitive.

In the example below, all two echo statements below are legal:

Example

```
<html>
<body>
<?php
echo "Hello World!<br>";
ECHO "Hello World!<br>";
?>
-</body>
</html>
```

White spaces

With HTML, whitespace is ignored between PHP statements. This is to mean that there can be one line of PHP code, then 2 or more lines of blank space before the next line of PHP code. Because of this you can indent your code and the PHP interpreter will ignore those spaces as well.

Here is an elaborate example

```
<html>

<head>

<title>My First PHP Page</title>

</head>

<body>

<?php

echo “Hello World!”;

echo “Hello World!”;

?>

</body>

</html>
```

Display:

```
Hello World!Hello World!
```

Chapter 2:PHP VARIABLES

Variable defination

A variable is a means of storing a value, such as text string “Hello World!” or the integer value 30. A variable can then be reused throughout your code, instead of having to type out the actual value over and over again,in other words variables can be termed as “containers” for storing information

In PHP you define a variable with the following form:

• $variable_name = Value;

just Like in mathematics

r=1

s=2

t=r+s

the letters r,s hold the numbers 1,2 respectively.

From the expression t=r+s above, we can calculate the value of r to be 3.

In PHP these letters are called **variables.**

Example

```
<?php
$r=1;
$s=2;
$t=$r+$s;
echo $t;
?>
```

Example

```
<?php
$txt=“Hello world”;
?>
```

Conventions for assigning variables

. Variables with more than one word should be separated with underscores. $my_variable

· A variable starts with the $ sign, followed by the name of the variable

· Variable names are case sensitive ($r and $R are two different variables)

· A variable name cannot start with a number

· A variable name must start with a letter or the underscore character

· A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

Chapter 3: PHP DATA TYPES

PHP Data Types

String, Integer, Floating point numbers, Boolean, Array.

String

A string is a sequence of characters, like “Hello world!”.

A string can be any text inside quotes. You can use single or double quotes,either will work efficiently

Example

```
<?php

$x = “Hello world!”;

echo $x;

?>
```

Integers

An integer is a number without decimals.

Rules for integers:

· An integer must have at least one digit (0-9)

· An integer cannot contain comma or blanks

· An integer must not have a decimal point

· An integer can be either positive or negative

· Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

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

Example

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

Floating point numbers

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

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

Example

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

Booleans

Booleans can be either TRUE or FALSE.

```
var x=true;

var y=false;
```

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

Arrays

An array stores multiple values in one single variable.

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

Example

```
<?php

$fruits=array(“mango”,“apple”,“ovacado”,’’banana’’);

var_dump($fruits);

?>
```

You will learn a lot more about arrays in later chapters of this tutorial.

Chapter 4:PHP OPERATORS

PHP Operators

This chapter shows the different operators that can be used in PHP scripts.

PHP arithmetic operators

1. Addition or summation (+)

$x + $y

Sum of $x and $y

2. Subtraction (-)

$x - $y

Difference of $x and $y

3. Multiplication (*)

$x * $y

Product of $x and $y

4. Division (/)

$x / $y

Quotient of $x and $y

5. Modulus (%)

$x % $y

Remainder of $x divided by $y

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

Example

```
<?php
$x=5;
$y=3;
echo ($x + $y); // outputs 8
echo ($x - $y); // outputs 2
echo ($x * $y); // outputs 15
echo ($x / $y); // outputs 1.6666666666667
echo ($x % $y); // outputs 2
?>
```

PHP assignment operators

The PHP assignment operator is used to write a value to a variable.

The basic assignment operator in PHP is “=”. It means that the left operand gets set to the value of the assignment expression on the right.

x = y and is the same as x = y

The left operand gets set to the value of the expression on the right

1. Addition

x += y

x = x + y

2. Subtraction

x -= y

x = x - y

3. Multiplication

x *= y

x = x * y

4. Division

x /= y

x = x / y

5. modulus

x %= y

x = x % y

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

Example

```
<?php
$x=10;
echo $x; // outputs 10

$y=10;
$y += 10;
echo $y; // outputs 20

$z=40;
```

```
$z -= 25;
echo $z; // outputs 15

$i=5;
$i *= 9;
echo $i; // outputs 45

$j=12;
$j /= 6;
echo $j; // outputs 2

$k=15;
$k %= 4;
echo $k; // outputs 3
?>
```

PHP string Operators

1. Concatenation (.)

```
$txt1 = “Hello”
$txt2 = $txt1 . ” world!”
```

Output is hello world

2. Concatenation assignment(.=)

```
$txt1 = “Hello”
```

Now $txt1 contains “Hello world!”

```
$txt1 .= ” world!”
```

The example below shows the results of using the string operators:

Example

```
<?php
$x = “Hello”;
$y = $a . ” world!”;
echo $y; // outputs Hello world!

$x=“Hello”;
$x .= ” world!”;
echo $x; // outputs Hello world!
?>
```

PHP Increment/Decrement Operators

1. Pre-increment (++$x)

Increments $x by one, then returns $x

2. Post-increment ($x++)

Returns $x, then increments $x by one

3. Pre-decrement (—$x)

Decrements $x by one, then returns $x

4. Post-decrement ($x—)

Post-decrement

Returns $x, then decrements $x by one

The example below shows the different results of using the different increment/decrement operators: Example

```
<?php
$x=9;
echo ++$x; // outputs 10

$y=10;
echo $y++; // outputs 10

$z=6;
echo —$z; // outputs 5

$i=5;
echo $i—; // outputs 5
?>
```

PHP comparison Operators

The PHP comparison operators are used to compare two values (number or string):

1. Equal (==)

$x == $y

True if $x is equal to $y

2. Identical (===)

$x === $y

True if $x is equal to $y, and they are

of the same type

3. Not equal (!=)

$x != $y

True if $x is not equal to $y

4. Not equal (<>)

$x <> $y

True if $x is not equal to $y

5. Not identical (!==)

$x !== $y

True if $x is not equal to $y, or they

are not of the same type

6. Greater than (>)

$x > $y

True if $x is greater than $y

7. Less than (<)

$x < $y

True if $x is less than $y

8. Greater than or equal to (>=)

$x >= $y

True if $x is greater than or equal to $y

9. Less than or equal to (<=)

$x <= $y

True if $x is less than or equal to $y

The example below shows the different results of using some of the comparison operators:

Example

```
<?php
$x=100;
$y=“100”;

var_dump($x == $y);
echo “<br>”;
var_dump($x === $y);
echo “<br>”;
var_dump($x != $y);
echo “<br>”;
var_dump($x !== $y);
echo “<br>”;

$a=50;
$b=90;
var_dump($a > $b);
echo “<br>”;
var_dump($a < $b);
?>
```

PHP logical Operators

1.and

$x and $y

True if both $x and $y are true

2.or

$x or $y

True if either $x or $y is true

3.xor

$x xor $y

True if either $x or $y is true, but not

4.both (&&)

$x && $y

True if both $x and $y are true

5.|| (Or)

$x || $y

True if either $x or $y is true

6.! (Not)

!$x

True if $x is not true

PHP array Operators

The PHP array operators are used to compare arrays:

1. Union (+)

$x + $y

Union of $x and $y (but duplicate keys are not overwritten)

2. Equality (==)

$x == $y

True if $x and $y have the same key/value pairs

3.Identity (===)

$x === $y

True if $x and $y have the same key/value pairs in the same order and of the same types

4. Inequality (!=)

$x != $y

True if $x is not equal to $y

5. Inequality (<>)

$x <> $y

True if $x is not equal to $y

6. Non-identity (!==)

$x !== $y

True if $x is not identical to $y

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

Example

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

```
?>
```

Chapter 5:CONTROL STRUCTURES

If statement

The PHP *if statement* is very similar to other programming languages use of the *if statement*, check the following:

This simple kind of if/then statement is very common in every day life and also appears in programming quite often. Whenever you want to make a decision given that something is true and be sure that you take the appropriate action, you are using an if/then relationship.

example

```
$my_name = “william”;

if ( $my_name == “william” ) {
echo “Your name is william!<br />”;
}
```

Display:

```
Your name is william!
```

explanation

Did you get that we were comparing the variable $my_name with “william” to see if they were equal? In PHP you use the double equal sign (==) to compare values. Additionally, notice that because the if statement turned out to be true, the code segment was executed, printing out “Your name is william!”. Let’s go a bit more into details

• We first set the variable $my_name equal to “william”.

• We next used a PHP if statement to check if the value contained in the variable $my_name was equal to “william”

• The comparison between $my_name and “william” was done with a double equal sign “==”, not a single equals”=”! A single equals is for assigning a value to a variable, while a double equals is for checking if things are equal.

• $my_name is indeed equal to “william” so the echo statement is executed

Suppose $my_name is not equal to”william” what would have happened

PHP Code:

```
$my_name = “james”;

if ( $my_name == “william” ) {

echo “Your name is william!<br />”;
}
```

```
echo “Welcome to mysite!”;
```

Display:

```
Welcome to mysite!
```

If /else statement

Has someone ever told you, “if you study smart, then you will pass”? And what happens if you do not study smart? Well, you fail! This is an example of an if/else conditional statement.

PHP Code:

```
$number_four == 4;

if ( $number_four == 4 ) {

echo ” true”;

} else {

echo ” false”;

}
```

Display:

```
true
```

explanation

• We first made a PHP variable called $number_four and set it equal to 4.

• In this example we compared a variable to an integer value. To do such a comparison we use ” ==“.

• $number_four is indeed equal to 4 and so this statement will evaluate to true.

• All code that is contained between the opening curly brace “{” that follows the if statement and the closing curly brace “}” will be executed when the if statement is true.

• The code contained within the else segment will not be used.

Using else if with if..else

example

PHP Code:

```
$teacher = “george”;
if($teacher == “anne”){

echo “Hello madam”;
} elseif ($teacher == “george”){
echo “hello Sir!”;
}else {
echo
“Morning”;
}
```

Display:

```
hello Sir!
```

PHP first checked to see if *$teacher* was equal to “anne”, which evaluated to false. Next, PHP checked the first elseif statement *$teacher* did in fact equal “george” so the phrase george Sir!” was printed out. If we wanted to check for more teachers names we could insert more elseif statements! ,note that an elseif statement cannot be used unless it is preceded by an if statement.

Switch statement

There are times when an if statement is not the most efficient way to check for certain conditions. With the use of the *switch* statement you can check for all these conditions at once,

This is how it works,first we have a single expression (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the code structure. If there is a match, the block of code associated with that case is executed. The use of break is to prevent the code from running into the next case automatically. The default statement is used if no match is found.

Example

```
<?php
$color=“pink”;
switch ($color)
{
case “yellow”:
echo “Your favorite color is yellow!”;
break;
case “red”:
echo “Your favorite color is red!”;
break;
case “pink”:
echo “Your favorite color is pink!”;
break;
default:
echo “Your favorite color is not known!”;
}
?>
```

The *if* statement has the *else* clause and the *switch* statement has the *default* case. It’s usually a good idea to always include the *default* case in all your switch statements.

while Loops

PHP while loops execute a block of code while the specified condition is true. Often when you write code, you want the same block of code to run over and over again in a row.Instead of adding several (which is tedious)almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, the following looping statements are used:

· **while** - loops through a block of code and executes as long as the specified condition is true

Syntax

while (*condition is true*)

{

code to be executed;

}

The example below first sets a variable $x to 1 ($x=1;). Then, the while loop will continue to run as long as $x is less than, or equal to 10. $x will increase by 1 each time the loop runs ($x++;): Example

```
<?php
$x=1;
while($x<=10)
{
echo “The number is: $x <br>”;
$x++;
}
?>
```

· **do…while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true,the do…while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

do

{

code to be executed;

}

while (*condition is true*);

The example below first sets a variable $x to 1 ($x=1;). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked and the loop will continue to run as long as $x is less than, or equal to 10:

Example

```
<?php
$x=1;
do
{
echo “The number is: $x <br>”;
$x++;
}
while ($x<=10)
?>
```

Note that in a do while loop the condition is tested after executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition fails the first time.

For Loop

PHP for loops execute a block of code a specified number of times. The for loop is used when you know in advance how many times the script should run.

Syntax

for (*init counter; test counter; increment counter*)

{

code to be executed;

}

explanation

· *init counter*: Initialize the loop counter value

· *test counter*: Evaluated for each loop iteration. If it evaluates to true, the loop continues. If it evaluates to false, the loop ends.

· *increment counter*: Increases the loop counter value

The example below displays the numbers from 0 to 100:

Example

```
<?php
for ($x=0; $x<=100; $x++)
{
echo “The number is: $x <br>”;
}
?>
```

Foreach Loop

The foreach loop works only on arrays, and is used to loop through each key pair in an array. For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element. The following example demonstrates a loop that will output the values of the given array ($fruits):

Syntax

foreach ($ *array* as $ *value*)

{

code to be executed;

}

Example

```
<?php
$fruits = array(“mango”,“orange”,“pawpaw”,“ovacado”);
foreach ($fruits as $value)
{
echo “$value <br>”;
}
?>
```

Chapter 6:PHP ARRAYS

Array definition

An array stores multiple values in one single variable,hence an array is a special variable, which can hold more than one value at a time.

If you have a list of items (example a list of book names), storing the books in single variables could look like this:

$books1=“computing”;

$books2=“religion”;

$books3=“history”;

$books4=“philosophy”;

what if you want to loop through the books and find a specific one? And what if you had not 4 books, but 4000? The best solution is to create an array,an array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an array

The array() function is used to create an array:

There are three types of arrays:

· **Indexed arrays** - Arrays with numeric index

· **Associative arrays** - Arrays with named keys

· **Multidimensional arrays** - Arrays containing one or more arrays

Indexed arrays

There are two ways to create indexed arrays:

Note:index always starts at 0

1.The index can be assigned automatically
$books=array(“computing”,“religion”,“history”);

2. The index can be assigned manually:

$books [0]=“computing”;

$books [1]=“religion”;

$books [2]=“history”;

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

Example

```
<?php

$books=array(“computing”,“religion”,“history”);


echo “I love ” . $books[0] . “, ” . $books[1] . ” and ” .$books[2] .”;

?>
```

Looping in indexed array

To loop through and echo all the values of an indexed array, you could use a for loop Example

```
<?php

$books=array(“computing”,“religion”,“history”);

$arraylength=count($books);

for($x=0;$x<$arraylength;$x++)

{

echo $books[$x];

echo “<br>”;

}

?>
```

Associative array

Associative arrays are arrays that use named keys that you assign to them.

The named keys can then be used in a script:

There are two ways to create an associative array:

```
1.$year=array("benson"=>"1975","anne"=>"1937","Joel"=>"2000"); or
2.$year['benson']="1975";
  $year['anne']="1937";
  $year['Joel']="2000";
```

Example

```
<?php
$year=array("benson"=>"1975","anne"=>"1937","Joel"=>"2000");
echo "benson was born in " . $year['benson'] . ;
?>
```

Looping in associative array

To loop through and echo all the values of an associative array, you could use a foreach loop

Example

```
<?php
$year=array(“benson”=>“1975”,“anne”=>“1937”,“Joel”=>“2000”);
foreach($year as $x=>$x_value)
{
echo “Key=” . $x . “, Value=” . $x_value;
echo “<br>”;
}
?>
```

Get the length of an array

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

Example

```
<?php
$books=array(“computing”,“religion”,“history”);
echo count($books);
?>
```

Chapter 7:PHP FUNCTIONS

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

Note:

.Besides the built-in PHP functions, we can create our own functions.

.A function is a block of statements that can be used repeatedly in a program.

.A function will not execute immediately when a page loads.

.A function will be executed by a call to the function.

Creating user defined Functions

.A user defined function declaration always starts with the word “function”

.Function names are case-insensitive.

.A function name can start with a letter or underscore (but not a number).

.Give the function a name that reflects what the function does

Remember that the opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function.

Syntax

function *functionName*()

{

code to be executed;

}

example

In the example below, we create a function named “displayname”. The function outputs “Hello world!”. To call the function, just write its name: Example

```
<?php
function displayname()
{
echo “Hello world!”;
}
displayname(); // call the function
?>
```

Functions arguments

An argument is just like a variable,through which information is passed to functions,arguments are specified after the function name, inside the parentheses.

You can add as many arguments as you want however you seperate them with a comma.

The following example has a function with two argument . When the detail() function is called, we also pass along detail (e.g. james,1937), and the detail is used inside the function, which outputs several different details

Example

```
<?php
function detail($name,$year)
{
echo “$name Refsnes. born in $year <br>”;
}
detail(“james”,“1937”);
detail(“kate”,“1971”);
detail(“anne”,“1963”);
?>
```

Default argument value

The following illustrates how to use a default parameter. If we call the function width() without arguments it takes the default value as argument

Example

```
<?php

function width($w=60)

{

echo “The width is : $w <br>”;

}

width(65);

width(); // will use the default value of 60

width(15);

width(90);

?>
```

Returning value

To let a function return a value, use the return statement

Example

```
<?php
function div($x,$y)
{
$z=$x/$y;
return $z;
}

echo “15 / 2 = ” . div(15,5) . “<br>”;
echo “14/2= ” . div(14,2) . “<br>”;
echo “12 / 4 = ” . div(12,4);
?>
```

Chapter 8:PHP Form Handling

A Simple HTML Form description

HTML Forms are required when you want to collect some data from the site visitor. For example during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML **<form>** tag is used to create an HTML form and it has following syntax:

```
<form action="Script URL" method="GET|POST">
    form elements like input, textarea etc.
</form>
```

Action	Backend script ready to process your passed data.
Method	Method to be used to upload data. The most frequently used are GET and POST methods.

HTML Form Controls

There are different types of form controls that you can use to collect data using HTML form:

- Checkboxes Controls
- Radio Box Controls
- Select Box Controls
- File Select boxes
- Hidden Controls
- Clickable Buttons
- Submit and Reset Button

Checkbox Control

Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to checkbox

Here is an example HTML code for a form with two checkboxes:

```
<html>
<head>
<title>Checkbox Control</title>
</head>
<body>
<form>
<input type=“checkbox” name=“maths” value=“on”> Maths
<input type=“checkbox” name=“physics” value=“on”> Physics
</form>
</body>
</html>
```

Radio Button Control

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to radio.

Here is example HTML code for a form with two radio buttons:

```
<html>
<head>
<title>Radio Box Control</title>
</head>
<body>
<form>
<input type=“radio” name=“subject” value=“maths”> Maths
<input type=“radio” name=“subject” value=“physics”> Physics
</form>
</body>
</html>
```

Select Box Control

A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

Here is example HTML code for a form with one drop down box

```
<html>
<head>
<title>Select Box Control</title>
</head>
<body>
<form>
<select name="dropdown">
<option value="Maths" selected>Maths</option>
<option value="Physics">Physics</option>
</select>
</form>
</body>
</html>
```

File Upload Box

If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the <input> element but type attribute is set to file.

Here is example HTML code for a form with one file upload box:

```
<html>
<head>
<title>File Upload Box</title>
</head>
<body>
<form>
<input type=“file” name=“fileupload” accept=“image/*” />
</form>
</body>
</html>
```

Button Controls

There are various ways in HTML to create clickable buttons. You can also create a clickable button using <input> tag by setting its type attribute to button. The type attribute can take the following values:

Here is example HTML code for a form with three types of buttons:

```
<html>
<head>
<title>button control</title>
</head>
<body>
<form>
<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset"  value="Reset" />
<input type="button" name="ok" value="OK"  />
<input type="image" name="imagebutton" src="/html/images/logo.png" />
</form>
</body>
</html>
```

Hidden Form Controls

Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number. When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page has be displayed next based on the passed current page.

Here is example HTML code to show the usage of hidden control:

```
<html>

<head>

<title>hidden form control</title>

</head>

<body>

<form>

<p>This is number 1</p>

<input type=“hidden” name=“pagenumber” value=“1” />

<input type=“submit” name=“submit” value=“Submit” />

<input type=“reset” name=“reset”  value=“Reset” />

</form>

</body>

</html>
```

Post and get methods

The example below contains a simple HTML form with two input fields and a submit button, using the post method:

Example

```
<html>
<body>
<form action=“process.php” method=“post”>
Name: <input type=“text” name=“surname”><br>
Age: <input type=“text” name=“age”><br>
<input type=“submit” name=”submit”>
</form>
</body>
</html>
```

When a user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named “process.php”. The form data is sent with method=“post”.

“process.php” looks like this:

```
<html>
<body>
Welcome <?php echo $_POST[“surname”]; ?>!!!<br>
You are <?php echo $_POST[“age”]; ?> years old.
</body>
</html>
```

The output:

```
Welcome oliver!!!
You are 36 years old.
```

The same result could also be achieved using method=“get”:

Example

```
<html>
<body>
<form action=“process.php” method=“get”>
Name: <input type=“text” name=“surname”><br>
Age: <input type=“text” name=“age”><br>
<input type=“submit” name=”submit”>
```

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

and “process.php” looks like this:

```
<html>
<body>
Welcome <?php echo $_GET[“surname”]; ?>!!!<br>
You are <?php echo $_GET[“age”]; ?> years old.
</body>
</html>
```

Review of the post method

Code Excerpt:

```
<form action=“process.php” method=“post”>

Name: <input type=“text” name=“surname”><br>

Age: <input type=“text” name=“age”><br>
```

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

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

Code Excerpt:

```
$y = $_POST[‘surname’];

$x = $_POST[‘age’];
```

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

Review of the get method

As we mentioned above, the alternative to the *post* method is *get*. If we were to change our HTML form to the *get* method, it would look like this:

Code Excerpt:

```
<form action=“process.php” method=“get”>

Name: <input type=“text” name=“surname”><br>

Age: <input type=“text” name=“age”><br>
```

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

“?item=##&quantity=##”

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

Code Excerpt:

```
$y = $_GET[‘surname’];

$x = $_GET[‘age’];
```

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

Security precaution

Whenever you are taking user input and using you need to be sure that the input is safe. If you are going to insert the data into a MySQL database, then you should be sure you have thought about preventing *MySQL* injection,if you are going to make a user input available to the public,the you should think about PHP htmlentities.

Chapter 9:GLOBAL VARIABLES

Global Variables - Superglobals

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

The PHP superglobal variables are:

· $GLOBALS

· $_SERVER

· $_REQUEST

· $_POST

· $_GET

· $_FILES

· $_ENV

· $_COOKIE

· $_SESSION

In this chapter will explain some of the superglobals,

$GLOBAL

$GLOBAL is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.

The example below shows how to use the super global variable $GLOBAL:

Example

```
<?php
$x = 15;
$y = 5;
functionmultipyl()
{
$GLOBALS[‘z’] = $GLOBALS[‘x’] * $GLOBALS[‘y’];
}
multiply();
echo $z;
?>
```

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

$_SERVER

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

The example below shows how to use some of the elements in $_SERVER:

Example

```
<?php
echo $_SERVER[‘PHP_SELF’];
echo “<br>”;
echo $_SERVER[‘SERVER_NAME’];
echo “<br>”;
echo $_SERVER[‘HTTP_HOST’];
echo “<br>”;
echo $_SERVER[‘HTTP_REFERER’];
echo “<br>”;
echo $_SERVER[‘HTTP_USER_AGENT’];
echo “<br>”;
echo $_SERVER[‘SCRIPT_NAME’];
?>
```

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

Element/Code	
$_SERVER[‘PHP_SELF’]	Returns the filename of the currently executing script
$_SERVER[‘GATEWAY_INTERFACE’]	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER[‘SERVER_ADDR’]	Returns the IP address of the host server
$_SERVER[‘SERVER_NAME’]	Returns the name of the host server (such as www.w3schools.com)
$_SERVER[‘SERVER_SOFTWARE’]	Returns the server identification string (such as Apache/2.2.24)
$_SERVER[‘SERVER_PROTOCOL’]	Returns the name and revision of the information protocol (such as HTTP/1.1)

$_SERVER[‘REQUEST_METHOD’]

Returns the request method used to access the page (such as POST)

$_SERVER[‘REQUEST_TIME’]

Returns the timestamp of the start of the request (such as 1377687496)

$_SERVER[‘QUERY_STRING’]

Returns the query string if the page is accessed via a query string

$_SERVER[‘HTTP_ACCEPT’]

Returns the Accept header from the current request

$_SERVER[‘HTTP_ACCEPT_CHARSET’]

Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)

$_SERVER[‘HTTP_HOST’]

Returns the Host header from the current request

$_SERVER[‘HTTP_REFERER’]

Returns the complete URL of the current page (not reliable because not all user-agents support it)

$_SERVER[‘HTTPS’]

Is the script queried through a secure HTTP protocol

$_SERVER[‘REMOTE_ADDR’]

Returns the IP address from where the user is viewing the current page

$_SERVER[‘REMOTE_HOST’]

Returns the Host name from where the user is viewing the current page

$_SERVER[‘REMOTE_PORT’]

Returns the port being used on the user’s machine to communicate with the web server

$_SERVER[‘SCRIPT_FILENAME’]

Returns the absolute pathname of the currently executing script

$_SERVER[‘SERVER_ADMIN’]

Returns the value given to the SERVER_ADMIN directive in the web server

configuration file (if your script runs on a virtual host, it will be the value

defined for that virtual host) (such as someone@w3scholls.com)

$_SERVER[‘SERVER_PORT’]

Returns the port on the server machine being used by the web server for communication (such as 80)

$_SERVER[‘SERVER_SIGNATURE’]

Returns the server version and virtual host name which are added to server-generated pages

$_SERVER[‘PATH_TRANSLATED’]

Returns the file system based path to the current script

$_SERVER[‘SCRIPT_NAME’]

Returns the path of the current script

$_SERVER[‘SCRIPT_URI’]

Returns the URI of the current page

$_REQUEST

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

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on “Submit”, the form data is sent to the file specified in the action attribute of the

<form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

Example

```
<html>
<body>
<form method=“post” action=”<?php echo $_SERVER[‘PHP_SELF’];?>”> Name: <input type=“text” name=“name”>
<input type=“submit”>
</form>
<?php
$name = $_REQUEST[‘fname’];
echo $name;
?>
</body>
</html>
</html>
```

Chapter 10:SESSIONS

Sessions overview

As a website becomes more sophisticated, so must the code that backs it. When you get to a stage where your website need to pass along user data from one page to another, it might be time to start thinking about using PHP sessions.

A normal HTML website will not pass data from one page to another. In other words, all information is forgotten when a new page is loaded. This makes it quite a problem for tasks like a shopping cart, which requires data(the user's selected product) to be remembered from one page to the next.

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping cart items, etc). However, this session information is temporary and is usually deleted very quickly after the user has left the website that uses sessions.

It is important to ponder if the sessions' temporary storage is applicable to your website. If you require a more permanent storage you will need to find another solution, like a MySQL database.

Sessions work by creating a unique identification (UID) number for each visitor and storing variables based on this ID. This helps to prevent two users' data from getting confused with one another when visiting the same webpage.

Note:If you are not experienced with session programming it is not recommended that you use sessions on a website that requires high-security, as there are security holes that take some advanced techniques to plug.

Starting a session

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

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

session.

PHP Code:

```
<?php

session_start(); // start up your PHP session!

?>
```

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

Storing a session variable

When you want to store user data in a session use the $_SESSION *associative array. This is w*here you both store and retrieve session data. In previous versions of PHP there were other ways to perform this store operation, but it has been updated and this is the correct way to do it.

PHP Code:

```
<?php
session_start();
$_SESSION[‘views’] = 1; // store session data
echo “Pageviews = “. $_SESSION[‘views’]; //retrieve data
?>
```

Display:

```
Pageviews = 1
```

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

Using PHP’s *isset* Function

Now that you know can easily store and retrieve data from the $_SESSION array, we can now explore some of the real functionality of sessions. When you create a variable and store it in a session, you probably want to use it in the future. However, before you use a session variable it is necessary that you check to see if it exists already!

This is where PHP’s *isset* function comes in handy, *isset* is a function that takes any variable you want to use and checks to see if it has been set. That is, it has already been assigned a value.

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

PHP Code:

```
<?php
session_start();
if(isset($_SESSION[‘views’]))
$_SESSION[‘views’] = $_SESSION[‘views’]+ 1;
else
$_SESSION[‘views’] = 1;
echo “views = “. $_SESSION[‘views’];
?>
```

The first time you run this script on a newly opened browser the *if statement* will fail because no session variable *views* would have been stored yet. However, if you were to refresh the page the *if statement* would be true and the counter would increment by one. Each time you reran this script you would see an increase in *view* by one.

Destroying a Session

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

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

PHP Code:

```
<?php

session_start();

if(isset($_SESSION[‘cart’]))

unset($_SESSION[‘cart’]);

?>
```

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

PHP Code:

```
<?php

session_start();

session_destroy();

?>
```

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

Session remarks

Cookies have been around for quite some time on the internet. They were invented to allow webmaster’s to store information about the user and their visit on the user’s computer.

At first they were feared by the general public because it was believed they were a serious privacy risk.

Nowadays nearly everyone has cookies enabled on their browser, partly because there are worse things to worry about and partly because all of the “trustworthy” websites now use cookies.

This lesson will teach you the basics of storing a cookie and retrieving a cookie, as well as explaining the various options you can set with your cookie.

Chapter 11:COOKIES

Creating a cookie

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

1. **name**: The name of your cookie. You will use this name to later retrieve your cookie, so don’t forget it!

2. **value**: The value that is stored in your cookie. Common values are username(string) and last visit(date).

3. **expiration**: The date when the cookie will expire and be deleted. If you do not set this expiration date, then it will be treated as a session cookie and be removed when the browser is restarted.

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

PHP Code:

```
<?php

//Calculate 60 days in the future

//seconds * minutes * hours * days + current time

$inTwoMonths = 60 * 60 * 24 * 60 + time();

setcookie(lastVisit, date(“G:i - m/d/y”), $inTwoMonths);

?>
```

Don’t worry if you can’t follow the somewhat involved date calculations in this example. The important part is that you know how to set a cookie, by specifying the three important arguments: name, value and expiration date.

Retrieving a Cookie

If the cookie has not expired yet, let’s retrieve it from the user’s PC using the *$_COOKIE* associative array. The name of your stored cookie is the key and will let you retrieve your stored cookie value!

PHP Code:

```
<?php
if(isset($_COOKIE[‘lastVisit’]))
$visit = $_COOKIE[‘lastVisit’];
else
echo “You’ve got some stale cookies!”;
echo “Your last visit was - “. $visit;
?>
```

This handy script first uses the *isset* function to be sure that our “lastVisit” cookie still exists on the user’s PC, if it does, then the user’s last visit is displayed. If the user visited our site on may 14, 2015 it might look something like this:

Display:

```
Your last visit was - 10:41- 05/14/15
```

Chapter 12:WORKING WITH FILES

Opening and closing files

In PHP,the function fpen() is normally used to open files. It requires two arguments that state the file name and the mode in which it has to operate. In the event the opening of a file is not successful the function fopen() returns the value of false, otherwise it returns a file pointer to read or write to that particular file.

File modes can be specified as stated below

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

Reading a file

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

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

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

- Open a file using fopen() function.
- Get the file's length using filesize() function.
- Read the file's content using fread() function.
- Close the file with fclose() function.

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

```
<?php
$filename = “example.txt”;
$file = fopen( $filename, “r” );

if( $file == false )
{
echo ( “Error occurred in opening file” );
exit();
}

$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );
fclose( $file );

echo ( “File size is : $filesize bytes” );
echo ( “<pre>$filetext</pre>” );
?>
```

writing a file

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

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

```
<?php
    $filename = “/home/user/guest/newfile.txt”;
    $file = fopen( $filename, “w” );

    if( $file == false )
    {
    echo ( “Error occurred in opening new file” );
    exit();
    }
    fwrite( $file, “This is  a simple file\n” );
    fclose( $file );
?>


    <?php
    $filename = “newfile.txt”;
    $file = fopen( $filename, “r” );

    if( $file == false )
    {
    echo ( “Error occured in opening file” );
    exit();
    }

    $filesize = filesize( $filename );
    $filetext = fread( $file, $filesize );

    fclose( $file );

    echo ( “File size is: $filesize bytes” );
    echo ( “$filetext” );
    echo(“file name: $filename”);
    ?>

    </body>
</html>
```

Chapter 13:OBJECT ORIENTED PHP

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

Object oriented concepts

Class − This is a programmer-defined data type, 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.

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 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.

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.

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 a simple 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 $ 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.

example

```
<?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 ie. itself.

Creating objects

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.

```
$biology = 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.

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 behaviour, 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( “biology”, 10 );
$maths = new Books ( “maths”, 14 );
$chemistry = new Books (“chemistry”, 17 );

/* Get those set values */
$physics->getTitle();
$chemistry->getTitle();
$maths->getTitle();

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

This will produce the following result −

```
 biology
 maths
 chemistry
 10
 14
 17
```

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.

Example:

```
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.

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.

Example:

getPrice and getTitle functions are overridden to return 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 infront of the member.

```
class MyClass {
    private $car = “audi”;
    $driver = “willy”;

    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 = “audi”;
    $driver = “willy”;

    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 implementers. 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 email {
   public function sendemail();
}
```

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

```
class Report implements email {
   // sendemail() Definition goes here
}
```

CONCLUSION

Thank you again for choosing to read this book. I hope this book was able to help you learn and understand more about PHP, and how to write PHP codes. The next step is now upon you to apply what you have learned in this book.

Finally, if you enjoyed this book, please with humble request take the time to share your thoughts and post a review on Amazon. It would be greatly appreciated.