Introduction to PHP

PHP is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features thrown in. Another way to think of PHP is a powerful, behind the scenes scripting language that visitors won't see!

The goal of the language is to allow web developers to write dynamically generated pages quickly. When someone visits a PHP webpage, web server processes the PHP code. It then sees which parts it needs to show to visitors (content and pictures) and hides the other stuffs (file operations, math calculations, etc.) then translates PHP into HTML. After the translation into HTML, it sends the webpage to the visitor's web browser.

What is PHP?

- PHP stands for **P**HP: **H**ypertext **P**reprocessor
- PHP is a server-side scripting language, like ASP, JSP , Perl
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is an open source software
- PHP is free to download and use

Why PHP?

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

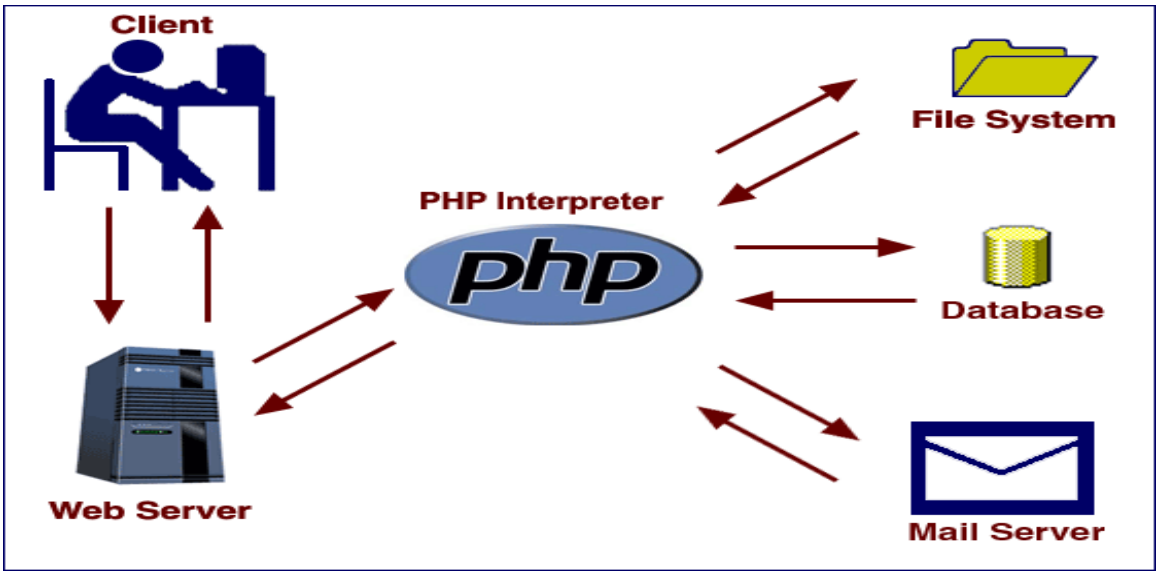
Where to Start?

To get access to a web server with PHP support, you can:

- Install Apache (or IIS) on your own server, install PHP, and MySQL
- Or find a web hosting plan with PHP and MySQL support

How it Works

When a user navigates in her browser to a page that ends with a .php extension, the request is sent to a web server, which directs the request to the PHP interpreter. As shown in the diagram, the PHP interpreter processes the page, communicating with file systems, databases, and email servers as necessary, and then delivers a web page to the web server to return to the browser.



PHP-Syntax

Syntax - The rules that must be followed to write properly structured code. PHP's syntax and semantics are similar to most other programming languages (C, Java, Perl) with the addition that all PHP code is contained within **<?php** ..... **?>** tag. That is, a PHP scripting block always starts with **<?php** and ends with **?>**, and can be placed anywhere in the document. On servers with shorthand support enabled it is possible to start a scripting block with **<?** and end with **?>**.

PHP Code

```
<?php

//write php code here....

?>

<?

//write php code here....

?>
```

Our First Script

Let's jump straight in with a PHP script. To begin, open your favorite text editor. Like HTML documents, PHP files are made up of plain text. You can create them with any text editor, such as Notepad on Windows, Simple Text and BBEdit on MacOS, or VI and Emacs on UNIX operating systems. Most popular HTML editors provide at least some support for PHP.

PHP Code

```
<?php
```

```
echo "Hello world ! it is my first php script";

//using echo for calling php script

?>
```

PHP semicolon

The semicolon signifies the end of a PHP statement and should never be forgotten. For example, if we repeated our "Hello World!" code several times, then we would need to place a semicolon at the end of each statement.

PHP Code

```
<?php

echo "Hello World !";

?>
```

PHP starting and ending script

When writing PHP, you need to inform the interpreter that you want it to execute your commands. If you don't do this, the code you write will be mistaken for HTML and will be output to the browser. Following table shows different ways of enclosing PHP code.

Tag Style	Start Tag	End Tag
Standard tags	<?php	?>
Sort tags	<?	?>
Script tags	<script language="php">	</script>

Let's run through some of the ways in which you can legally write the code in listing.

PHP Code

Code	Result
<?php echo "hello world !"; ?>	Hello world !

<? echo “hello world !”; ?>	Hello world !
<script language =”php”> echo “hello world !”; </script>	Hello world !
<?php echo “hello world !”; ?>	Hello world !

PHP print function

print() is a function that outputs data. In most cases, anything output by *print()* ends up in the browser window. A function is a command that performs an action, usually modified in some way by data provided for it. Data sent to a function is almost always placed in parentheses after the function name. In this case, you sent the *print()* function a collection of characters, or string. Strings must always be enclosed by quotation marks, either single or double.

Combining HTML and PHP

You can incorporate this into an HTML document simply by adding HTML outside the PHP start and end tags, as shown in example code.

As you can see, incorporating HTML into a PHP document is simply a matter of typing in the code. The PHP interpreter ignores everything outside PHP open and close tags. If you were to view with a browser, as shown in Figure, you would see the string "this is a php code". If you were to view the document source, as shown in Figure the listing would look exactly like a normal HTML document.

PHP Code

Code	***Result***

<?php
 //echo style
 print “hello world !
”;
 //function style
 print (“hello dude !”);
 ?>
	Hello world!
	 Hello dude !

PHP Code

Code	*Result*
<html> <head><title>HTML-PHP</title></head> <body> This is a html code <?php print “this is a php code”; ?> </body> </html>	This is a html code This is a php code

PHP Comments

Adding comments to your code as you write can save you time later on and make it easier for other programmers to work with your code.

Single line comments begin with two forward slashes (//) or a single hash sign (#).All text from either of these marks until either the end of the line or the PHP close tag is ignored.

Multiline comments begin with a forward slash followed by an asterisk (/*) and end with an asterisk followed by a forward slash (*/).

```
// this is a single line comment

# this is another single line coment

/*

this is a php multi line coment

this is a php multi line coment

this is a php multi line coment

*/
```

PHP Variables

Variables are used for storing values, like text strings, numbers or arrays. When a variable is declared, it can be used over and over again in your script.

All variables in PHP start with a $ sign symbol. The correct way of declaring a variable in PHP is *$var_name = value;*

NOTE: New PHP programmers often forget the $ sign at the beginning of the variable. In that case it will not work.

Following example creates a variable containing a string, and a variable containing a number:

```
<?php
$txt="Hello_World!";
$x=16;
?>
```

PHP is a loosely typed language. In PHP, a variable does not need to be declared before adding a value to it. In the example above, you see that you do not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value. In case of a strongly typed programming language, you have to declare (define) the type and name of the variable before using it. In PHP, the variable is declared automatically when you use it.

Naming Rules for Variables

- A variable name must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and underscores (a-z, A-Z, 0-9, and _)
- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore ($my_string), or with capitalization ($myString)

PHP Strings

Before you can use a string you have to create it. A string can be used directly in a function or it can be stored in a variable. Below we create the exact same string twice: first storing it into a variable and in the second case we send the string directly to echo.

PHP Code

Code	Result

<?php
//set string
$my_string = "Hello every body !";
$_mystring="Hello guys !";
$mystring="Hello dude !";
$mystring1="Hello web !";
 	
echo "Hello every body !";
echo "
";
//
 for line break
//call for out put the string
echo $my_string;
echo "
";
echo $_mystring;
echo "
";
echo $mystring;
echo "
";
echo $mystring1;
?>
	Hello every body !
	Hello every body !
	Hello guys !
	Hello dude !
	Hello web !

Constants

Constants are like variables except that, once assigned a value, they cannot be changed. Constants are created using the *define()* function and by convention (but not by rule) are in all uppercase letters. Constants can be accessed from anywhere on the page.

Syntax

define ('CONST_NAME',VALUE);

PHP Code

Code	Result

<?php
 	
define("name","Prince");
 	
define("number","100");
 	
print "my name is ".name."
";
 	
print "my number is ".number;
 	
?>
	my name is Prince
	
	my number is 100

Double-Quotes and Single-Quotes

In JavaScript, you can use either use single (') or double (") quotes and they behave the same. In php, it is different. Double quotes and single quotes aren't exactly the same.

```
$something="Oh something"; echo "My answer is $something.<br>";

//result is: My answer is Oh something
```

In the above, using double quotes, the variable $something is evaluated within the quotes and the result isn't what is expected: the variable is evaluated to Oh something. Variables, but not functions or constants, are evaluated even when enclosed in double quotes.

```
echo 'My answer is $something.<br>';  //result is: My answer is $something.
```

When single quotes are used, as above, the variable isn't evaluated and is printed on the screen literally as $something.

Escaping with the Backslash

The backslash character (\) can be used to escape quotes and variables. The next example repeats the idea that things are evaluated within double quotes.

```
$something="Oh something!"; echo "My answer is $something";

//result: My answer is Oh something!
```

So nothing new here, just a reminder. Next we use the backslash to escape the variable $something and also the double quotes around "is":

```
echo "<br>";  echo "My answer \"is\" \$something";
```

```
//result: My answer "is" $something
```

The program doesn't crash on finding more double quotes, but understands that these are just to emphasize "is". And the program does not explode $something, but prints it literally, because it is escaped

PHP Array

PHP arrays allow you to store groups of related data in one variable (as opposed to storing them in separate variables). Storing data like this in an array has many benefits. For example, if you have a lot of data, you could populate the array programmatically. You could also output the contents programmatically - all you need to know is the name of the array.

There are 3 different types of PHP arrays:

- Numeric arrays
- Associative arrays
- Multidimensional arrays

Numeric array

Numeric arrays use a number as the "key". The key is the unique identifier, or ID, of each item within the array. We can use this ID later when working with the contents within the array - we don't need to know the item's value, just the ID.

Note that numbering starts at zero.

You can choose between the following examples when creating an array. Both of these examples have the same result.

Example: Manual key assignment

When creating an array this way, you assign each "key" as a number within open and closing square brackets ([and]).

```
$arrayName[0] = "Value1";
$arrayName[1] = "Value2";
$arrayName[2] = "Value3";
```

Another Example: Automatic key assignment

When using this method to create an array, you don't need to assign the key - PHP will do that for you.

```
$arrayName = array("Value1", "Value2", "Value3");
```

You can work with each value in an array by referring to its key. For example, to select the 2nd value in an array, we would do this `$arrayName[1]`. Just a reminder that numbering starts at zero, so the 2nd item actually uses the number one as it's key.

PHP Code

Code	Result
<?php $fruit = array("Apples", "Strawberries", "Blackberries"); echo $fruit[1]; ?>	Strawberries

Associative array

Associative arrays are similar to numeric arrays but, instead of using a number for the key, we use a value. We then assign another value to the key - you could think of it as two values for the price of one!

As with creating numeric arrays, there are two options for creating associative arrays in PHP (although in both cases, we have to create the key manually).

Example: *$arrayName['keyName'] = "Value1";*

$arrayName['keyName'] = "Value2";
$arrayName['keyName'] = "Value3";

Another Example

$arrayName = array("keyName"=>"Value1", "keyName"=>"Value2", "keyName"=>"Value3");

You can display the contents of associative arrays just as you would with numeric arrays - by referring to it's key.

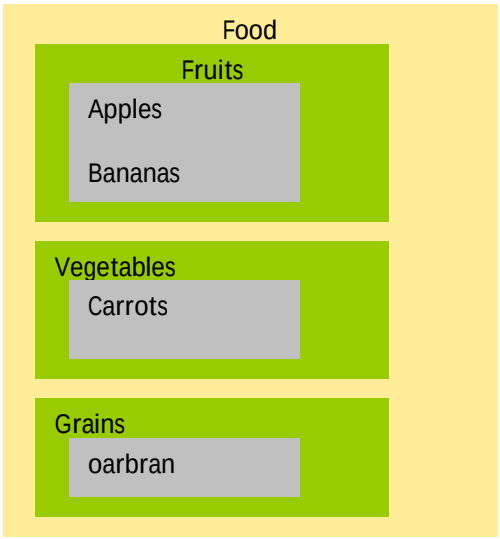
PHP Code

Code	Result
<?php $vegetables=array("Gourd"=>"40kilojoules", "Artichoke"=>"105 kilojoules", "Cassava"=>"550 kilojoules"); echo "Artichoke: " . $vegetables["Artichoke"]; ?>	Artichoke: 105 kilojoules

Multidimensional arrays

Multidimensional arrays allow you to put an array inside another array. In other words, the contents of the array are another array. You can do this as many times as you wish - you could have an array inside another array, which is inside another array, etc

The following diagram demonstrates this. We have an array called "Food", which contains 3 arrays (called "Fruit", "Vegetables", "Grains"). Each of these arrays contains their own arrays (one for each food within that group).



We can create the above array using the following code:

Code	Result
<?php *$Food = array ("Fruit"=>array ("Apples", "Bananas", "Oranges",), "Vegetables"=>array ("Carrots", "Potatoes",), "Grains"=>array ("Oatbran",)* *);* *echo "Best Food is ". $Food[Vegetables][1]."."* *?>*	Best Food is Potatoes.

PHP Operator

In all programming languages, operators are used to manipulate or perform operations on variables and values. There are many operators used in PHP, which can be separated into the following categories to make it easier to learn them all.

- **Assignment** Operators
- **Arithmetic** Operators
- **Comparison** Operators
- **String** Operators
- **Combination Arithmetic & Assignment** Operators
- **Logical** Operators

Assignment Operators

Assignment operators are used to set a variable equal to a value or set a variable to another variable's value. Such an assignment of value is done with the "=", or equal character. Example:

$my_var = 4;

$another_var = $my_var;

Now both *$my_var* and *$another_var* contain the value 4. Assignments can also be used in conjunction with arithmetic operators.

Arithmetic Operators

Operator	Name	Example
+	Addition	2+4
-	Subtraction	10-5
*****	Multiplication	5*5
/	Division	10/2
%	Modulus	43%10

PHP Code

Code	Result
<?php *$addition = 2+4;* *$subtraction = 10-5;* *$multiplication = 5*5;* *$division = 10/2;* *$modulus = 43%10;* *echo "addition: 2 + 4 = ".$addition." ";* *echo "subtraction: 10 - 5 = ".$subtraction." ";* *echo "multiplication: 5 *5 = ".$multiplication." ";* *echo "division: 10 / 2 = ".$division." ";* *echo "modulus: 43 % 10 = " . $modulus ;* *?>*	addition: 2 + 4 = 6 subtraction: 10 - 5 = 5 multiplication: 5 *5 = 25 division: 10 / 2 = 5 modulus: 43 % 10 = 3

Comparison Operators

Comparisons are used to check the relationship between variables and/or values. Comparison operators are used inside conditional statements and evaluate to either true or false. Here are the most important comparison operators of PHP. **Asume $x = 4 and $y = 5**

Operator	*Name*	*Example*	*Result*
= =	Equal To	$x = = $y	False
!=	Not Equal To	$x != $y	True
<	Less Than	$x < $y	True
>	Greater Than	$x > $y	False
<=	Less Than or Equal To	$x <= $y	True
>=	Greater Than or Equal To	$x >= $y	False

String Operators

There is only one string operator in PHP. As we have already seen in the Arithmetic Operators Lesson, the period "." is used to add two strings together, or more technically, the period is the concatenation operator for strings.

PHP Code

Code	*Result*
<?php *$_a = “hello”;* *$_b = “world”;* *$_new = $_a . $_b;* *print $_new . “!”;* *?>*	Hello world !

Combination of Arithmetic and Assignment Operators

In programming it is a very common task to have to increment a variable by some fixed amount. The most common example of this is a counter. Say you want to increment a counter by 1, you would have:

$counter = $counter + 1;

However, there is a short-hand for doing this.

$counter += 1;

This combination assignment/arithmetic operator would accomplish the same task. The downside to this combination operator is that it reduces code readability to those programmers who are not used to such an operator. Here are some examples of other common shorthand operators. In general, "+=" and "-=" are the most widely used combination operators.

Operator	*Name*	*Example*	*Equivalent operation*
+=	Plus Equals	$x += 2	$x = $x + 2
-=	Minus Equals	$x -= 2	$x = $x – 2
*=	Multiply Equals	$x *=2	$x = $x * 2
/=	Divide Equals	$x /=2	$x = $x / 2
%=	Modulo Equals	$x %= 2	$x = $x % 2
.=	Concatenate Equals	$_a .= $b	$_a = $_a . $_b

Logical Operators

The logical operators test combinations of Booleans. The or (||) operator, for example returns true if either the left or the right operand is true.

true || false would return true

The and operator only returns true if both the left and right operands are true.

true && false would return false

It's unlikely that you would use a logical operator to test Boolean constants, however. It would make more sense to test two or more expressions that resolve to a Boolean. For example,

($x > 2) && ($x < 15) would return true if $x contained a value that is greater than 2 and smaller than 15. We include the parentheses to make the code easier to read.

Operator	Name	Return True If...	Example	Result
||	Or	Left or right is true	true || false	True
&&	And	Left and right are true	true && false	False
!	Not	The single operand is not true	! true	False

Increment and Decrement Variable

This may seem a bit absurd, but there is even shorter shorthand for the common task of adding 1 or subtracting 1 from a variable. To add one to a variable or "increment" use the

"++" operator: *$x++;* which is equivalent to *$x += 1;* or *$x = $x + 1;*

To subtract 1 from a variable, or "decrement" use the "--" operator. *$x--;* which is equivalent to *$x -= 1;* or *$x = $x - 1;*

In addition to this "shorter hand" technique, you can specify whether you want to increment before the line of code is being executed or after the line has executed. Our PHP code below will display the difference.

PHP Code

Code	Result
<?php *$x = 4;*	4

// The value of x with post-plusplus *echo $x++.” ”;* *// The value of x after the post-plusplus is* *echo $x.” ”;* *$x = 4;* *// The value of x with with pre-plusplus* *echo ++$x.” ”;* *// The value of x after the pre- plusplus is* *echo $x;* *?>*	5 5 5

Switching Flow

Most scripts evaluate conditions and change their behavior accordingly. The facility to make decisions makes your PHP pages dynamic, capable of changing their output according to circumstances. Like most programming languages, PHP allows you to do this with an *if* statement.

The if Statement

The **if** statement evaluates an expression between parentheses. If this expression results in a true value, a block of code is executed. Otherwise, the block is skipped entirely. This enables scripts to make decisions based on any number of factors.

*if (**expression**)*

{

// code to execute if the expression evaluates to true

}

PHP Code

Code	***Result***
<?php *$number = 100;*	Correct number

if($number == 100) *{ print "Correct number"; }* *?>*	

You use the comparison operator == to compare the variable $number with the value 100. If they match, the expression evaluates to true, and the code block below the *if* statement is executed. Although the code block is wrapped in braces in the example, this is only necessary if the block contains more than one line. The following fragment, therefore, would be acceptable.

```
if ( $number == 100 )
        print "Correct number";
```

If you change the value of $number to "50" and run the script, the expression in the if statement evaluates to false, and the code block is skipped. The script remains sulkily silent.

PHP Code

Code	*Result*
<?Php *$number = 50;* *if($number = = 100)* *{ print "Correct number";}* *?>*	

Using *else* with *if* Statement

When working with *if* statement, you will often want to define an alternative block of code that should be executed if the expression you are testing evaluates to false. You can do this by adding **else** to *if* statement followed by a further block of code.

```
if ( expression )
{
// code to execute if the expression evaluates to true
}
```

```
else
{
// code to execute in all other cases
}
```

PHP Code

Code	Result

<?php
$number = 50;
if($number = = 100)
{
print “Correct number”;
}
else
{
print “Wrong number”;
}
?>
	Wrong number

$number contains the 50, which is not equivalent to 100, so the expression in *if* statement evaluates to false. This means that the first block of code is skipped. The block of code after else, therefore, is executed, and the message "Wrong number" is printed to the browser.

Using *elseif* with *if* Statement

You can use an ***if-elseif-else*** construct to test multiple expressions before offering a default block of code.

```
if ( expression )
{
// code to execute if the expression evaluates to true
```

```
}

elseif ( another expression )

{

// code to execute if the previous expression failed and this one evaluates to true

}

else

{

// code to execute in all other cases

}
```

If the first expression does not evaluate to true, then the first block of code is ignored. The *elseif* clause then causes another expression to be evaluated. Once again, if this expression evaluates to true, then the second block of code is executed. Otherwise, the block of code associated with the else clause is executed. You can include as many *elseif* clauses as you want, and if you don't need a default action, you can omit the *else* clause.

PHP Code

Code	Result

<?Php
$number = 50;
if ($number = = 100){ print "Correct number"; }
elseif ($number = = 50)
{ print "Else If is a Correct "; }
else
{ print "Wrong number";}
?>
	Else If is a Correct

Once again, *$number* holds value 50, This is not equivalent to 100, so the first block is ignored. The *elseif* clause tests for equivalence between the contents of *$number* and 50, which evaluates to true. This block of code is therefore executed.

The *switch* Statement

With the use of the switch statement you can check for all these conditions at once, and the great thing is that it is actually more efficient programming to do this.

The way the Switch statement works is it takes a single variable as input and then checks it against all the different cases you set up for that switch statement. Instead of having to check that variable one at a time, as it goes through a bunch of If Statements, the Switch statement only has to check one time.

In our example the single variable will be $destination and the cases will be: Las Vegas, Amsterdam, Egypt, Tokyo, and the Caribbean Islands.

PHP Code

Code	***Result***
$destination = "Tokyo"; *echo "Traveling to $destination ";* *switch ($destination)* *{* *case "Las Vegas":* *echo "Bring an extra $500";* *break;* *case "Amsterdam":* *echo "Bring an open mind";* *break;* *case "Egypt":* *echo "Bring15 bottles of SPF 50 Sunscreen";* *break;*	Traveling to Tokyo Bring lots of money

case "Tokyo": *echo "Bring lots of money";* *break;* *case "Caribbean Islands":* *echo "Bring a swimsuit";* *break;* *}* *?>*	

The value of $destination was Tokyo, so when PHP performed the switch operating on $destination in immediately did a search for a case with the value of "Tokyo". It found it and proceeded to execute the code that existed within that segment.

You might have noticed how each case contains a break; at the end of its code area. This break prevents the other cases from being executed. If the above example did not have any break statements then all the cases that follow Tokyo would have been executed as well. Use this knowledge to enhance the power of your switch statements!

The form of the switch statement is rather unique, so spend some time reviewing it before moving on. Note: Beginning programmers should always include the break; to avoid any unnecessary confusion.

switch Statement Default Case

You may have noticed the lack of a place for code when the variable doesn't match our condition. *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. Below is a variation of our example that will result in none of the cases being used causing our *switch* statement to fall back and use the *default* case

PHP Code

Code	***Result***
<?php *$destination = "New York";*	Bring lots of underwear

```
echo "Traveling to $destination<br />";

switch ($destination){

   case "Las Vegas":

       echo "Bring an extra $500";

       break;

   case "Amsterdam":

      echo "Bring an open mind";

      break;

   case "Egypt":

      echo "Bring15bottles of SPF 50
Sunscreen";

     break;

  case "Tokyo":

     echo "Bring lots of money";

     break;

  case "Caribbean Islands":

     echo "Bring a swimsuit";

     break;

  default:

    echo "Bring lots of underwear!";

    break;

}

?>
```

Using the ?: Operator

The ***?:*** or ternary operator is similar to the if statement but returns a value derived from one of two expressions separated by a colon. Which expression is used to generate the value returned depends on the result of a test expression:

(***expression***) ? ***returned_if_expression_is_true : returned_if_expression_is_false;***

If the test expression evaluates to true, the result of the second expression is returned; otherwise, the value of the third expression is returned.

PHP Code

Code	*Result*
<html> *<head><title>Example</title></head>* *<body>* *<?php* *$_ct = "500";* *$_logic = (100 > $_ct)?"your number is correct":"your number is wrong";* *print $_logic;* *?>* *</body>* *</html>*	Your number is wrong

Looping

So far you've looked at decisions that a script can make about what code to execute. Scripts can also decide how many times to execute a block of code. Loop statements are designed to enable you to achieve repetitive tasks. Almost without exception, a loop continues to operate until a condition is achieved, or you explicitly choose to exit the loop.

While Loop

Repetitive tasks are always a burden to us. Deleting spam email, sealing 50 envelopes, and going to work are all examples of tasks that are repeated. The nice thing about programming is that you can avoid such repetitive tasks with a little bit of extra thinking. Most often these repetitive tasks are conquered in the loop.

The idea of a loop is to do something over and over again until the task has been completed. Before we show a real example of when you might need one, let's go over the structure of the PHP while loop.

Example

The function of the while loop is to do a task over and over as long as the specified conditional statement is true. This logical check is the same as the one that appears in a PHP if statement to determine if it is true or false. Here is the basic structure of a PHP while loop.

```
while ( conditional statement)
{
        //do this code;
}
```

It displays how the while loop is structured. The conditional statement is checked:

- If it is true, then code within the while loop is executed.
- If the conditional statement is false, then code following the while loop is executed like normal.

Example

Imagine that you are running an art supply store. You would like to print out the price chart for number of brushes and total cost. You sell brushes at a flat rate, but would like to display how much different quantities would cost. This will save your customers from having to do the mental math themselves.

You know that a while loop would be perfect for this repetitive and boring task. Here is how to go about doing it.

PHP Code

Code	*Result*
<?php $brush_price = 5; $counter = 10; echo "<table border=\"1\" align=\"center\">";	**Quantity** | **Price** 10 | 50 20 | 100

```
echo "<tr><th>Quantity</th>";
echo "<th>Price</th></tr>";
while ( $counter <= 100 ) {
        echo "<tr><td>";
        echo $counter;
        echo "</td><td>";
        echo $brush_price * $counter;
        echo "</td></tr>";
        $counter = $counter + 10;
}
echo "</table>";
?>
```

30	150
40	200
50	250
60	300
70	350
80	400
90	450
100	500

The loop created a new table row and its respective entries for each quantity, until our counter variable grew past the size of 100. When it grew past 100 our conditional statement failed and the loop stopped being used. Let's review what is going on.

- We first made a $brush_price and $counter variable and set them equal to our desired values.
- The table was set up with the beginning table tag and the table headers.
- The while loop conditional statement was checked, and $counter (10) was indeed smaller or equal to 100.
- The code inside the while loop was executed, creating a new table row for the price of 10 brushes.
- We then added 10 to $counter to bring the value to 20.
- The loop started over again at step 3, until $counter grew larger than 100.
- After the loop had completed, we ended the table.

Do While Loop

A "do while" loop is a slightly modified version of the while loop. But in this case first the statements within the loop is executed and then the while condition is check. The basic structure of PHP *do while* loop is as follows:

```
do
{
```

```
    //do this code;
} while ( conditional statement);
```

While and Do While Contrast

A simple example that illustrates the difference between these two loop types is a conditional statement that is always false. First the while loop:

PHP Code

Code	Result
<?php $_ex = 0; while($_ex >1) { echo “good example baby”; } ?>	

As you can see, this while loop's conditional statement failed (0 is not greater than 1), which means the code within the while loop was not executed. Now, can you guess what will happen with a do-while loop?

PHP Code

Code	Result
<?php $_ex = 0; do { echo “good example baby”; }while($_ex >1); ?>	Good example baby

The code segment "good example baby" was executed even though the conditional statement was false. This is because a do-while loop first do's and secondly checks the while condition!

For Loop

for loop is simply a while loop with a bit more code added to it. The common tasks that are covered by a *for* loop are:

- Set a counter variable to some initial value
- Check to see if the conditional statement is true
- Execute the code within the loop
- Increment a counter at the end of each iteration through the loop

for loop allows you to define these steps in one easy line of code.

Example

Let us take the example from the while loop Paragraph and see how it could be done in a *for* loop. The basic structure of *for* loop is as follows:

```
for ( initialize a counter; conditional statement; increment a counter)
{
        //do this code;
}
```

Note: how all the steps of the loop are taken care of in the *for* loop statement. Each step is separated by a semicolon: initiliaze counter, conditional statement, and the counter increment. A semicolon is needed because these are separate expressions. However, notice that a semicolon is not needed after the "increment counter" expression.

Here is the example of the brush prices like while loop another example

PHP Code

Code	*Result*

```
<?php
$brush_price = 5;
echo "<table border='1' align='center'>";
echo "<tr><th>Quantity</th>";
echo "<th>Price</th></tr>";
for ( $counter = 10; $counter <= 100; $counter += 10)
{
        echo "<tr><td>";
        echo $counter;
        echo "</td><td>";
        echo $brush_price * $counter;
        echo "</td></tr>";
}
echo "</table>";
?>
```

Quantity	Price
10	50
20	100
30	150
40	200
50	250
60	300
70	350
80	400
90	450
100	500

Breaking Out of Loops with break Statement

Both while and for statements incorporate a built-in test expression with which you can end a loop. The **break** statement, though, enables you to break out of a loop according to additional tests. This can provide a safeguard against error.

PHP Code

Code	*Result*

<?php
$_cn = -4;
for(;$_cn <= 10;$_cn++)
{ if($_cn == 0) { break; }
print $_cn."
";
}
	-4
	-3
	-2
	-1

?>	

Foreach Loop

The *foreach* loop is used to loop through arrays.

Example

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

For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

PHP Code

Code	Result
<?php $x=array("one","two","three"); foreach ($x as $value) { echo $value . " "; } ?>	one two three

Another Example

This crazy statement roughly translates into: For each element of the $array associative array I want to refer to the key as $key and the value as $value.

```
foreach ($array as $key=>$value)
{
//code to be executed;
}
```

The operator "=>" represents the relationship between a key and value. You can imagine that the key points => to the value. In our example we named the key $key and the value $value. However, it might be easier to think of it as $name and $age. Below our example does this and notice how the output is identical because we only changed the variable names that refer to the keys and values.

PHP Code

Code	*Result*
<?php *$x=array(0=>"one",1=>"two",2=>"three");* *foreach ($x as $key=>$value)* *{* *echo "key = " .$key ." value = ".$value. " ";* *}* *?>*	key = 0 value = one key = 1 value = two key = 2 value = three

PHP Function

What is Function?

You can think of a function as a machine. A machine takes the raw materials you feed it and works with them to achieve a purpose or to produce a product. A function accepts values from you, processes them, and then performs an action (printing to the browser, for example) or returns a new value, possibly both. If you needed to bake a single cake, you would probably do it yourself. If you needed to bake thousands of cakes, you would probably build or acquire a cake-baking machine. Similarly, when deciding whether to create a function, the most important factor to consider is the extent to which it can save you from repetition.

A function, then, is a self-contained block of code that can be called by your scripts. When called, the function's code is executed. You can pass values to functions, which they will then work with. When finished, a function can pass a value back to the calling code.

NEW TERM: A *function* is a block of code that is not immediately executed but can be called by your scripts when needed. Functions can be built-in or user-defined. They can require information to be passed to them and usually return a value.

Calling Functions

Functions come in two flavors— those built in to the language and those you define yourself. PHP has hundreds of built-in functions. The very first script in this tutorial consisted of a single function call:

print("Hello Web");

Note: *print ( )* is not a typical function in that it does not require parentheses in order to run successfully. *print ("Hello Web");* and *print "Hello Web";* are equally valid. This is an exception. All other functions require parentheses, whether or not they accept arguments.

In this example, we called the *print ( )* function, passing it the string "Hello Web". The function then went about the business of writing the string. A function call consists of the function name, *print* in this case, followed by parentheses. If you want to pass information to the function, you place it between these parentheses. A piece of information passed to a function in this way is called an **argument**. Some functions require that more than one argument be passed to them. Arguments in these cases must be separated by commas.

some_function($an_argument, $another_argument);

print ( ) is typical in that it returns a value. Most functions give you some information back when they've completed their task, if only to tell whether their mission was successful. *print ( )* returns a boolean, therefore. The *abs ( )* function, for example, requires a signed numeric value and returns the absolute value of that number.

PHP Code

Code	***Result***
<?php *$_number = -1200;* *$new_number = abs($_number);* *print $new_number;* *?>*	1200

In this example, we assign the value – 1200 to a variable *$_number*. We then pass that variable to the *abs ( )* function, which made the necessary calculation and returned a new value. We assign this to the

variable *$new_num* and print the result. In fact, we could have dispensed with temporary variables altogether, passing our number straight to *abs ( )*, and directly printing the result.

```
print ( abs( –1200 ) );
```

Create a Function

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

Example

```
function function_Name( )
{
//code to be executed;
}
```

PHP function guidelines

- Give the function a name that reflects what the function does.
- The function name can start with a letter or underscore (not a number).

Example: A simple function that *my_name* when it is called as the following code.

PHP Code

Code	Result

<?php
function myname()
{
echo “Prince”;
}
echo “my name is ”;
myname();
?>
	My name is prince

Functions - adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable. Parameters are specified after the function name, inside the parentheses.

Example: The following example will write different names.

PHP Code

Code	Result
<?php *function myname($_name)* *{ echo $_name; }* *echo "my name is ";* *myname("prince ");* *echo " my friend name is ";* *myname("john ");* *echo " my girl friend name is ";* *myname("shely");* *?>*	my name is prince my friend name is john my girl friend name is shely

Functions - return values

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

PHP Code

Code	Result
<?php *function tm($a,$b)* *{* *$total = $a + $b;* *return $total;* *}*	The total mark is 75

echo "The total mark is "; *tm(50,25);* *?>*	

Variable Scope

A variable declared within a function remains local to that function. In other words, it will not be available outside the function or within other functions. In larger projects, this can save you from accidentally overwriting the contents of a variable when you declare two variables of the same name in separate functions.

PHP Code

Code	Result
<?php *function test()* *{* *$testvariable = "this is a test variable";* *}* *print "test variable: $testvariable ";* *//$test variable will not run* ***?>***	test variable:

You can see the output of the script. The value of the variable *$testvariable* is not printed. This is because no such variable exists outside the *test ( )* function. Note that attempting to access a nonexistent variable does not cause an error. Similarly, a variable declared outside a function will not automatically be available within it.

Accessing variables with the global statement

From within a function, it is not possible by default to access a variable that has been defined elsewhere. If you attempt to use a variable of the same name, you will set or access a local variable only.

PHP Code

Code	Result
<?php	3

$a = 1;* *$b = 2; *function Sum()* *{* *global $a, $b;* *$b = $a + $b;* *}* *Sum();* *echo $b;* *?>*	

The above script will output 3. By declaring *$a* and *$b* global within the function, all references to either variable will refer to the global version. There is no limit to the number of global variables that can be manipulated by a function.

Usage of Important Built-in Function and String in PHP

Being able to manipulate function and strings is a valuable skill, especially in PHP. You'll most likely come across a programming problem that requires you to find some data in a string. The beginning of a lot of your string manipulation expertise will begin with the *strops ( )* function, which allows you to find data in your string.

Searching a string with strops

The way *strops ( )* works is it takes some string you want to search in as its first argument and another string, which is what you are actually searching for, as the second argument. If the function can find a search match, then it will return the position of the first match. However, if it can't find a match it will return false.

To make this function crystal clear, let's search a numbered, in-order string, for the number five.

PHP Code

Code	***Result***
<?php *$numberedString = "1234567890";* *// 10 numbers from 1 to 0*	The position of 5 in our string was 4

$fivePos = strpos($numberedString, "5"); *echo "The position of 5 in our string was $fivePos";* *?>*	

Notice that the position is 4, which may seem confusing at first, until you realize that PHP starts counting from 0.

- The number 1 - Position 0 - **No match**
- The number 2 - Position 1 - **No match**
- The number 3 - Position 2 - **No match**
- The number 4 - Position 3 - **No match**
- The number 5 - Position 4 – **Match**

Although we only searched for a single character, you can use this function to search for a string with any number of characters. Also, it is important to note that this function will return the position of the start of the first match. So if we had searched the same string for "567890" we would again find a match and position 4 because that is where the match starts.

The strlen function

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

Let's find the length of a string-

<?php *echo strlen("Hello world!");* *?>*
The output of the code above will be 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).

PHP str_replace function

Another key tool to have in your programming toolbox is the ability to quickly replace parts of a PHP string with new values. The *str_replace* function is similar to a word processor's "Replace All" command that lets you specify a word and what to replace it with then replaces every occurrence of that word in the document.

str_replace parameters: *str_replace* has three parameters that are required for the function to work properly. ***str_replace (search, replace, originalString)***

- **search** - This is what you want to search your string for. This can be a string or an array.
- **replace** - All matches for search will be replaced with this value. This can be a string or an array.
- **originalString** - This is what search and replace will be operating on. The str_replace function will return a modified version of originalString when it completes.

Example

Imagine we are working at a school district and need to create a webpage for the students' parents. The webpage has an introduction string that we need to customize depending on if the student is male or female. With *str_replace* this is mighty easy.

PHP Code

Code	***Result***
<?php *//string that needs to be customized* *$rawstring = "Welcome Birmingham parents. Your replaceme is a pleasure to have!";* *//male string* *$malestr = str_replace("replaceme", "son", $rawstring);* *//female string* *$femalestr = str_replace("replaceme", "daughter", $rawstring);* *echo "Son: ". $malestr . " ";* *echo "Daughter: ". $femalestr;* *?>*	Son: Welcome Birmingham parents. Your son is a pleasure to have! Daughter: Welcome Birmingham parents. Your daughter is a pleasure to have!

With these two gender customized strings created we could then provide a more engaging experience for the student's parents when they logged into the school website with their kid's credentials.

str_replace arrays: multiple replaces in one

PHP Code

Code	Result
<?php *$rawstring = "Welcome Birmingham parents. Your replaceme is a pleasure to have!";* *$arr=array('Barmingham','replaceme')* *$arr1=array('Wasington','son')* *$malestr = str_replace($arr, $arr1, $rawstring);* */* *$femalestr = str_replace("replaceme", "daughter", $rawstring);* **/* *echo "Son: ". $malestr . " ";* *#echo "Daughter: ". $femalestr;* *?>*	Son: Welcome washington parents. Your son is a pleasure to have

substr_replace function

The function *substr_replace* introduces some additional functionality to compliment *str_replace*. *substr_replace* is a more mathematically based replace function, which relies on starting points and lengths to replace parts of strings, as opposed to searching and replacing.

substr_replace's four parameters: There are three required parameters for the *substr_replace* function (original string, replacement string, starting point) and one that's optional (length)

- **original string** - This is your original string that will be operated on.
- **replacement** string - This string will be used to replace everything in the string from the starting point to the ending point (specified by length).
- **starting point** - This is the place in the original string that will be used to mark the replacement's beginning. A negative value specifies the number of characters from the end of the string.
- **optional length** - How many characters from the original string will be replaced. If no length is specified then the end of the string is used. If a value of 0 is used then no characters will be replaced and an insert is performed. A negative value specifies the number of characters from the end of the string.

Example: This example of *substr_replace* shows what happens when you omit the length parameter at various starting points

PHP Code

Code	Result
<?php *//string that needs to be customized* *$original = "ABC123 Hello Mr. Cow! DEF321";* *//starting point 5* *$sp5 = substr_replace($original, "Five", 5);* *//starting point 12* *$sp12 = substr_replace($original, "Twelve", 12);* *//starting point 0* *$sp0 = substr_replace($original, "Zero", 0);* *//starting point -1* *$spneg1 = substr_replace($original, "Negative 1", -1);* *//Echo each string* *echo "Original String: $original ";* *echo "Starting Point 5: $sp5 ";* *echo "Starting Point 12: $sp12 ";* *echo "Starting Point 0: $sp0 ";* *echo "Starting Point -1: $spneg1 ";* *?>*	Original String: ABC123 Hello Mr. Cow! DEF321 Starting Point 5: ABC12Five Starting Point 12: ABC123 HelloTwelve Starting Point 0: Zero Starting Point -1: ABC123 Hello Mr. Cow! DEF32Negative **1**

As you can see, when you don't specify the fourth parameter, length, everything after the starting point is replaced by the second parameter replacement string.

Note: The first replacement occurred at position 5, which in $original was the character 3. This 3 and everything onward was replaced with the replacement string. Remember that you start counting character to begin from zero. The $original string could be labeled as so:

- Letter A - Position 0
- Letter B - Position 1
- Letter C - Position 2
- Letter 1 - Position 3
- Letter 2 - Position 4
- Letter 3 - Position 5

substr_replace specifying a length: If you want to get any sort of precision out of this function you're going to have to get into the nitty gritty of specifying the exact length of characters you want replaced in your original string.

Imagine that you want to get rid of those ugly pseudo references (ABC123, DEF321) at the beginning and end of the string. Since both of those strings are a length of 6 and we know one is at the very beginning of the string and the other is at the very end of the string we should probably use a starting point of 0 for ABC123 and a value of -6 for DEF321. By having a replacement string of nothing "" we can do something similar to select and delete that we often do in a word processor.

PHP Code

Code	***Result***
<?php *//string that needs to be customized* *$original = "ABC123 Hello Mr. Cow! DEF321";* *//remove ABC123 and store in $cleanedstr* *$cleanedstr = substr_replace($original, "", 0, 6);* *//remove DEF321 from $cleanedstr* *$cleanedstr2 = substr_replace($cleanedstr, "", -6, 6);* *//Echo each string* *echo "Original String: $original ";* *echo "Clean #1: $cleanedstr ";* *echo "Clean #2: $cleanedstr2";*	Original String: ABC123 Hello Mr. Cow! DEF321 Clean #1: Hello Mr. Cow! DEF321 Clean #2: Hello Mr. Cow!

?>	

Make sure that you play around with this function some on your own so you can get a feel for how the starting point and length parameters affect this function.

String capitalization functions

PHP has three primary capitalization related functions: *strtoupper, strtolower and ucwords*. The function names are pretty self-explanatory, but why they are useful in programming might be new to you.

Converting a String to Upper Case – strtoupper

The *strtoupper* function takes one argument, the string you want converted to upper case and returns the converted string. Only letters of the alphabet are changed, numbers will remain the same.

PHP Code

Code	***Result***
<?php *$originalString = "String Capitalization 1234";* *$upperCase = strtoupper($originalString);* *echo "Old string - $originalString ";* *echo "New String - " . $upperCase;* *?>*	Old string - String Capitalization 1234 New String - STRING CAPITALIZATION 1234

One might use this function to increase emphasis of a important point or in a title. Another time it might be used with a font that looks very nice with all caps to fit the style of the web page design.

A more technical reason would be to convert two strings you are comparing to see if they are equal. By converting them to the same capitalization you remove the possibility that they won't match simply because of different capitalizations.

Converting a String to Lower Case – strtolower

The *strtolower* function also has one argument: the string that will be converted to lower case.

PHP Code

Code	Result
<?php *$originalString = "String Capitalization 1234";* *$lowerCase = strtolower($originalString);* *echo "Old string - $originalString ";* *echo "New String - $lowerCase";* *?>*	Old string - String Capitalization 1234 New String - string capitalization 1234

Capitalizing the First Letter – ucwords

Titles of various media types often capitalize the first letter of each word and PHP has a time-saving function that will do just this.

PHP Code

Code	***Result***
<?php *$titleString = "a title that could use some hELP";* *$ucTitleString = ucwords($titleString);* *echo "Old title - $titleString ";* *echo "New title - $ucTitleString";* *?>*	Old title - a title that could use some hELP New title - A Title That Could Use Some HELP

String *explode*

The PHP function *explode* lets you take a string and blow it up into smaller pieces. For example, if you had a sentence you could ask explode to use the sentence's spaces " " as dynamite and it would blow up the sentence into separate words, which would be stored in an array. The sentence "Hello, I would like to lose weight." would look like this after explode got done with it:

- Hello,
- I
- would
- like
- to
- lose
- weight

The dynamite (the space character) disappears, but the other stuff remains, but in pieces. With that abstract picture of the explode function in mind, lets take a look at how it really works.

The explode function: The first argument that *explode* takes is the delimiter (our dynamite) which is used to blow up the second argument, the original string. explode returns an array of string pieces from the original and they are numbered in order, starting from 0. Let's take a phone number in the form ###-###-#### and use a hyphen "-" as our dynamite to split the string into three separate chunks.

PHP Code

Code	Result
<?php *$rawPhoneNumber = "800-555-5555";* *$phoneChunks = explode("-", $rawPhoneNumber);* *echo "Raw Phone Number = $rawPhoneNumber ";* *echo "First chunk = $phoneChunks[0] ";* *echo "Second chunk = $phoneChunks[1] ";* *echo "Third Chunk chunk = $phoneChunks[2]";* *?>*	Raw Phone Number = 800-555-5555 First chunk = 800 Second chunk = 555 Third Chunk chunk = 5555

Explode Function - Setting a Limit: If you want to control the amount of destruction that explode can wreak on your original string, consider using the third (optional) argument which allows you to set the

number of pieces explode can return. This means it will stop exploding once the number of pieces equals the set limit. Below we've blown up a sentence with no limit and then with a limit of 4.

PHP Code

Code	Result
<?PHP *$someWords = "Please don't blow me to pieces.";* *$wordChunks = explode(" ", $someWords);* *for($i = 0; $i < count($wordChunks); $i++){* *echo "Piece $i = $wordChunks[$i] ";* *}* *$wordChunksLimited = explode(" ", $someWords, 4);* *for($i = 0; $i < count($wordChunksLimited); $i++){* *echo "Limited Piece $i = $wordChunksLimited[$i] ";* *}* *?>*	Piece 0 = Please Piece 1 = don't Piece 2 = blow Piece 3 = me Piece 4 = to Piece 5 = pieces. Limited Piece 0 = Please Limited Piece 1 = don't Limited Piece 2 = blow Limited Piece 3 = me to pieces.

The limited explosion has 4 pieces (starting from 0, ending at 3).

Array implode

The PHP function implode operates on an array and is known as the "undo" function of explode. If you have used explode to break up a string into chunks or just have an array of stuff you can use implode to put them all into one string.

PHP implode - Repairing the Damage: The first argument of implode is the string of characters you want to use to join the array pieces together. The second argument is the array (pieces).

PHP Code

Code	Result

<?php *$pieces = array("Hello", "World,", "I", "am", "Here!");* *$gluedTogetherSpaces = implode(" ", $pieces);* *$gluedTogetherDashes = implode("-", $pieces);* *for($i = 0; $i < count($pieces); $i++){* *echo "Piece #$i = $pieces[$i] ";* *}* *echo "Glued with Spaces= $gluedTogetherSpaces ";* *echo "Glued with Dashes = $gluedTogetherDashes";* *?>*	Piece #0 = Hello Piece #1 = World, Piece #2 = I Piece #3 = am Piece #4 = Here! Glued with Spaces = Hello World, I am Here! Glued with Dashes = Hello-World,-I-am-Here!

The implode function will convert the entire array into a string and there is no optional argument to limit this as there was in the explode function.

Working with Files

Include and Require Function

You can insert the content of one PHP file into another PHP file before the server executes it, with *include ( )* or *require ( )* function. The two functions are identical in every way, except how they handle errors.

- include() generates a warning, but the script will continue execution
- require() generates a fatal error, and the script will stop

These two functions are used to create functions, headers, footers, or elements that will be reused on multiple pages.

Server side includes saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. When the header needs to be updated, you can only update the include file, or when you add a new page to your site, you can simply change the menu file (instead of updating the links on all your web pages).

Include function: *include ( )* function takes all the content in a specified file and includes it in the current file. If an error occurs, *include ( )* function generates a warning, but the script will continue execution.

Example: All pages in the Web site should include this header.php file. Here is how it can be done:

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

<a href="#"> Home      </a>  
<a href="#"> About Us  </a>   
<a href="#"> Contact Us</a>

</body>
</html>
```

Assume that you have a standard header file, called "header.php". To include the header file in a page, use *include ( )* function:

PHP Code

Code	Result
<html> <body> <?php include("header.php"); ?> Welcome to my home page! Some text. </body>	Home About Us Contact Us Welcome to my home page! Some text.

</html>	

If you look at the source code of the page above (in a browser), it will look like this:

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

<a href="#"> Home       </a>  
<a href="#"> About Us  </a>  
<a href="#"> Contact Us</a>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>

</body>
</html>
```

Require function: *require ( )* function is identical to *include ( )*, except that it handles errors differently. If an error occurs, *include ( )* function generates a warning, but the script will continue execution. r*equire ( )* generates a fatal error, and the script will stop.

Error Example *include ( )* Function

```
<html>
<body>

<?php
//wrong file name
include("wrongFile.php");
```

```
echo "Hello World!";
?>


</body>
</html>
```

Error message

Warning: include(wrongFile.php) [function.include]:

failed to open stream:

No such file or directory in C:\home\website\test.php on line 5

Warning: include() [function.include]:

Failed opening 'wrongFile.php' for inclusion

(include_path='.;C:\php5\pear')

in C:\home\website\test.php on line 5

Hello World!

Notice that the echo statement is executed! This is because a Warning does not stop the script execution.

Error Example *require ( )* Function: Now, let's run the same example with the *require ( )* function

```
<html>
<body>


<?php
//wrong file name
```

```
require("wrongFile.php");

echo "Hello World!";

?>

</body>

</html>
```

Error message

Warning: require(wrongFile.php) [function.require]:

failed to open stream:

No such file or directory in C:\home\website\test.php on line 5

Fatal error: require() [function.require]:

Failed opening required 'wrongFile.php'

(include_path='.;C:\php5\pear')

in C:\home\website\test.php on line 5

The echo statement is not executed, because the script execution stopped after the fatal error. It is recommended to use *require ( )* function instead of *include ( )*, because scripts should not continue after an error.

Working with Forms

Many website applications rely on HTML forms so that users can perform their tasks. For example, most Content Management Systems allow users to provide content for a website by entering the content into a textarea form field, then clicking a "Save" button. When the user clicks the Save button, the form is submitted to the *action* page. The action page is normally specified with the *action* attribute of the form tag.

Once a form has been submitted, the form fields are made available to the action page as a special type of variable. You, as the programmer, can read these form variables by using the appropriate syntax for form variables. Form variables are stored as an array..

Forms can be submitted using one of two methods: get or post. By default, a form will be submitted using the "get" method. This results in each form variable being passed via the URL. If you decide to use the "post" method, you specify this using the method attribute (method="post") of the form tag.

The Get Method

If you use the *get* method, your action page can access each form field using $_GET["variableName"] (where "variableName" is the name of the form field).

Example

Form page (formpage.html)

HTML Code

Code
<html> *<head></title>php form processing</title></head>* *<form action="actionpage.php" method="get">* *<input type="text" name="firstName" /> * *<input type="text" name="lastName" /> * *<input type="submit" />* *</form>* *</body>* *</html>*

Action page (actionpage.php): Here, the action page outputs the contents of the form variables that were passed from the form.

PHP Code

```
<html>
<head>
```

```
<title> php form processing </title>
</head>
<body>

<$php
echo “First Name:” . $_GET["firstName"] . “<br>”;
echo “Last Name:” . $_GET["lastName"] ;
?>

<body>
</html>
```

PHP Post Method

If your form uses the post method, you use $_POST["variableName"].

HTML Code

Code
<html> <head></title>php form processing</title></head>   <form action="actionpage.php" method="post"> <input type="text" name="firstName" /> <input type="text" name="lastName" /> <input type="submit" /> </form>   </body> </html>

Action page (actionpage.php): Here, the action page outputs the contents of the form variables that were passed from the form.

PHP Code

```
<html>
<head>
<title> php form processing </title>
</head>
<body>

<$php
echo “First Name:” . $_POST["firstName"] . “<br>”;
echo “Last Name:” . $_POST["lastName"] ;
?>
<body>
</html>
```

Difference between Get and Post Method

Get Method

- data is submitted as a part of url
- data is visible to the user
- it is not secure but fast and quick
- We can bookmark link with this method
- Server can log all action

Post Method

- data is submitted as a part of http request
- data is not visible in the url
- it is more secure but slower as compared to GET
- POST method have security of data.
- We cannot bookmark page link

The Request Method

This variable contains the content of both $_GET, $_COOKIE, and $_POST. This can get the values passed from the form with both the GET and POST methods.

Example

HTML Code

```
<html>
<head></title>php form processing</title></head>

<form action="actionpage.php" method="get">
<input type="text" name="firstName" /><br />
<input type="text" name="lastName" /><br />
<input type="submit" />
</form>

</body>
</html>

OR

<html>
<head></title>php form processing</title></head>

<form action="actionpage.php" method="post">
<input type="text" name="firstName" /><br />
<input type="text" name="lastName" /><br />
<input type="submit" />
</form>
```

```
</body>

</html>
```

Action page (actionpage.php): Here, the action page outputs the contents of the form variables that were passed from the form.

PHP Code

```
<html>

<head>

<title> php form processing </title>

</head>

<body>


<$php

echo “First Name:” . $_REQUEST["firstName"] . “<br>”;

echo “Last Name:” . $_REQUEST["lastName"] ;

?>


<body>

</html>
```

Connecting to the database server in PHP

Before you can access and work with data in a database, you must create a connection to the database. In PHP, this is done with the mysql_connect() function.

Syntax: *mysql_connect (servername,username,password);*

Parameter	*Description*
servername	Optional. Specifies the server to connect to. Default value is "localhost:3306"
username	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
password	Optional. Specifies the password to log in with. Default is ""

Note: There are more available parameters, but the ones listed above are the most important.

PHP Code

Code	*Result*
<?php *$cn = mysql_connect (“localhost”,”root”,””)* */*if the server is localhost ,the user name is root & the password is empty* */* *if (!$cn)* *{* *die(“could not connect”.mysql_error())* */* The "die" part will be executed if the connection fails:* */* *}* *else* *{* *echo “connected to the server successfully”* *}* *?>*	connected to the server successfully

Closing a Connection

The connection will be closed as soon as the script ends. To close the connection before, use the mysql_close() function.

PHP Code

```
<?php
$cn = mysql_connect (“localhost”,”root”,””)
if ( !$cn)
{
  die(“could not connect”.mysql_error())
}
else
{
  echo “connected to the server successfully”
}
mysql_close($cn)
?>
```

Create Database and Tables

Create a Database

The CREATE DATABASE statement is used to create a database in MySQL.

Syntax: CREATE DATABASE database_name

To get PHP to execute the statement above you must use the **mysql_query()** function. This function is used to send a query or command to a MySQL connection.

PHP Code

Code	*Result*
<?php *$cn = mysql_connect (“localhost”,”root”,””)* *if (!$cn)* *{* *die(“could not connect”.mysql_error())* *}* *else*	connected to the server successfully Database created successfully

{ *echo "connected to the server successfully"* *}* *echo " ";* *//database name is m_db* *if (! mysql_query("create database m_db",$cn))* *{* *echo "Could not created database".mysql_error())* *}* *else* *{* *echo "Database created successfully"* *}* *mysql_close($cn)* *?>*	

Create a Table

The CREATE TABLE statement is used to create a database table in MySQL.

Syntax: *CREATE TABLE table_name (column_name1 data_type, column_name2 data_type, column_name3 data_type, ...)*

We must add the CREATE TABLE statement to the *mysql_query ( )* function to execute the command.

The following PHP code shows how you can create a table named "person", with three columns. The column names will be "FirstName", "LastName" and "Age".

PHP Code

Code	***Result***
<?php *$cn = mysql_connect ("localhost","root","")* *if (!$cn)* *{*	connected to the server successfully table created successfully

```
  die("could not connect".mysql_error())
}
else
{
  echo "connected to the server successfully"
}
echo "<br>";
mysql_select_db("m_db",$cn)
// selected database & server
$q =" CREATE TABLE person
(
FirstName varchar(15),
LastName varchar(15),
Age int
)";
$mq = mysql_query($q,$cn)
if ( !$mq)
{
 echo "Could not created table".mysql_error())
}
else
{
 echo "table created successfuly"
}
mysql_close($cn)
?>
```

Important: A database must be selected before a table can be created. The database is selected with the *mysql_select_db ( )* function.

Note: When you create a database field of type varchar, you must specify the maximum length of the field, e.g. varchar(15).

Primary Keys and Auto Increment Fields: Each table should have a primary key field.

A primary key is used to uniquely identify the rows in a table. Each primary key value must be unique within the table. Furthermore, the primary key field cannot be null because the database engine requires a value to locate the record.

The primary key field is always indexed. There is no exception to this rule! You must index the primary key field so the database engine can quickly locate rows based on the key's value.

The following example sets the personID field as the primary key field. The primary key field is often an ID number, and is often used with the AUTO_INCREMENT setting. AUTO_INCREMENT automatically increases the value of the field by 1 each time a new record is added. To ensure that the primary key field cannot be null, we must add the NOT NULL setting to the field.

PHP Code

```
$sql = "CREATE TABLE person
(
 personID int NOT NULL AUTO_INCREMENT,
 PRIMARY KEY(personID),
 FirstName varchar(15),
 LastName varchar(15),
 Age int
)";
mysql_query($sql,$con);
```

Adding Data to a Table

The following PHP code adds two new records to the "Person" table:

PHP Code

Code	*Result*
<?php *$cn = mysql_connect ("localhost","root","")* *if (!$cn)* *{* *die("could not connect".mysql_error())* *}*	connected to the server successfully data inserted successfuly

```
else
{
  echo “connected to the server successfully”
}
echo “<br>”;
mysql_select_db(“m_db”,$cn)
$q ="INSERT INTO person (FirstName, LastName, Age)
      VALUES ('Peter', 'Griffin', '35')";
$q2 ="INSERT INTO person (FirstName, LastName, Age)
       VALUES ('Glenn', 'Quagmire', '33')";
$mq = mysql_query($q,$cn)
$mq2 = mysql_query($q2,$cn)
$mq = $mq . $mq2
if ( !$mq)
{
  echo “Could not inserted data”.mysql_error())
}
else
{
  echo “data inserted  successfuly”
}
mysql_close($cn)
?>
```

Insert Data from a Form into a Database

Now we will create an HTML form that can be used to add new records to the "Person" table.

HTML form

```
<html>
<body>
<form action="insert.php" method="post">
 Firstname: <input type="text" name="firstname" />
 Lastname: <input type="text" name="lastname" />
 Age: <input type="text" name="age" />
<input type="submit" />
</form>
</body>
</html>
```

When a user clicks the submit button in the HTML form in the example above, the form data is sent to "insert.php". The "insert.php" file connects to a database, and retrieves the values from the form with the PHP $_POST variables. Then, the mysql_query() function executes the INSERT INTO statement, and a new record will be added to the database table.

Below is the **PHP Code** in the "insert.php" page.

Code	Result

<?php
$cn = mysql_connect ("localhost","root","")
if (!$cn)
{
die("could not connect".mysql_error())
}
/*
else
{
echo "connected to the server successfully"
	data inserted successfully

```
}
*/
echo “<br>”;
mysql_select_db(“m_db”,$cn)
$q =”INSERT INTO person (FirstName, LastName, Age)
    VALUES ('$_POST[firstname]', '$_POST[lastname]', '$_POST[age]')”;
$mq = mysql_query($q,$cn)
if ( !$mq)
{
 echo “Could not inserted data”.mysql_error())
}
else
{
 echo “data inserted  successfuly”
}
mysql_close($cn)
?>
```

MySQL Data Operation using PHP

MySQL Select Data

The following PHP code selects all the data stored in the "Person" table (The * character selects all of the data in the table).

PHP Code

Code	Result

<?php
$cn = mysql_connect (“localhost”,”root”,””)
if (!$cn)
{
	Peter Griffin
	Glenn Quagmire

```
 die("could not connect".mysql_error())
}
echo "<br>";
mysql_select_db("m_db",$cn)
$q =" SELECT * FROM person";
//* = All
$mq = mysql_query($q,$cn)
while($row = mysql_fetch_array($mq))
 {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
 }
mysql_close($cn);
?>
```

The code above stores the data returned by the mysql_query() function in the $mq variable. Next, we use the **mysql_fetch_array()** function to return the first row from the recordset as an array. Each subsequent call to mysql_fetch_array() returns the next row in the recordset. The while loop circle through all the records in the recordset. To print the value of each row, we use the PHP $row variable ($row['FirstName'] and $row['LastName']).

Display the Result in an HTML Table

The following PHP code selects the same data as the example above, but will display the data in an HTML table.

PHP Code

Code

```
<?php
$cn = mysql_connect ("localhost","root","")
if ( !$cn)
{
  die("could not connect".mysql_error())
}
echo "<br>";
mysql_select_db("m_db",$cn)
```

Result

Firstname	Lastname
Glenn	Quagmire
Peter	Griffin

```
$q =" SELECT * FROM person";
$mq = mysql_query($q,$cn)
echo "<table border='1'>";
echo “<tr>”;
echo “<th>Firstname</th><th>Lastname</th>”;
echo “</tr>";
while($row = mysql_fetch_array($mq))
 {
   echo "<tr>";
   echo "<td>" . $row['FirstName'] . "</td>";
   echo "<td>" . $row['LastName'] . "</td>";
   echo "</tr>";
 }
echo "</table>";
mysql_close($cn)
?>
```

MySQL Where Clause

The following operators can be used with the WHERE clause.

Operator	Description
=	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

Note: SQL statements are not case sensitive. WHERE is the same as where.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

The following PHP code will select all rows from the "Person" table, where FirstName='Peter':

PHP Code

Code	Result

<?php
$cn = mysql_connect (“localhost”,”root”,””)
if (!$cn)
{
die(“could not connect”.mysql_error())
}
echo “
”;
mysql_select_db(“m_db”,$cn)
$q =" SELECT * FROM person WHERE FirstName='Peter' ";
$mq = mysql_query($q,$cn);
while($row = mysql_fetch_array($mq))
{
echo $row['FirstName'] . " " . $row['LastName'];
echo "
";
}
mysql_close($cn)
?>
	Peter Griffin

Order by Keyword

The following PHP code selects all the data stored in the "Person" table, and sorts the result by the "Age" column.

PHP Code

Code	Result

<?php
$cn = mysql_connect (“localhost”,”root”,””)
if (!$cn)
{
die(“could not connect”.mysql_error())
}
echo “
”;
	Glenn Quagmire 33
	Peter Griffin 35

mysql_select_db("m_db",$cn) *$q =" SELECT * FROM person ORDER BY age ";* *$mq = mysql_query($q,$cn)* *while($row = mysql_fetch_array($mq))* *{* *echo $row['FirstName'] . " " . $row['LastName'];* *echo " ";* *}* *mysql_close($cn)* *?>*	

Update Data

Let we have a table named "Person". Here is how it looks:

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

The following PHP code updates some data in the "Person" table.

Code	Result
<?php *$cn = mysql_connect ("localhost","root","")* *if (!$cn)* *{* *die("could not connect".mysql_error())* *}* *echo " ";* *mysql_select_db("m_db",$cn)* *$q =" UPDATE Person SET Age = '36'* *WHERE FirstName = 'Peter' AND LastName = 'Griffin') ";* *$mq = mysql_query($q,$cn)*	data updated successfuly

if (!$mq)* *{* *echo "Could not updated data".mysql_error())* *}* *else* *{* *echo "data updated successfuly"* *}* *mysql_close($cn); *?>*	

After the update, the "Person" table will look like this:

FirstName	LastName	Age
Peter	Griffin	36
Glenn	Quagmire	33

Delete Data from Database

Earlier in the tutorial we created and updated a table named "Person". Here is how it looks:

FirstName	LastName	Age
Peter	Griffin	36
Glenn	Quagmire	33

The following PHP Code deletes all the records in the "Person" table where LastName='Griffin':

PHP Code

Code	Result
<?php *$cn = mysql_connect ("localhost","root","");* *if (!$cn)* *{*	data deleted successfully

```
 die("could not connect".mysql_error());
}
echo "<br>";
mysql_select_db("m_db",$cn);
$q =" DELETE FROM Person WHERE LastName='Griffin' ";
$mq = mysql_query($q,$cn)
if ( !$mq)
{
 echo "Could not deleted data".mysql_error())
}
else
{
 echo "data deleted  successfuly"
}
mysql_close($cn);
?>
```

After the deletion, the table will look like this:

FirstName	LastName	Age
Glenn	Quagmire	33

PHP Session and Cookie

PHP Session

Sessions can transfer data between two or more pages. It`s somehow like a variable that keeps its value even if you call it from an another file. Let`s say that we have a website and in the index.php file we are asking for the name of the visitor and we store it in the *$name variable (e.g.:$name = 'Tom')*.

If the visitor navigates to an another page the value of *$name* will be lost, so we would need to ask for the name on each page so in this case is way better to use sessions. In order to use a session first we need to "start" it.

PHP Code

```
<?php
```

```
session_start();
?>
```

Note: The session_start(); function must be the at the very beginning of your code, even before the <html> tag. A session variable look`s and works like an element of an associative array.

$_SESSION ['name'] = 'Tom';

echo $_SESSION ['name'];

And because sessions are keeping their value even if you go to another page.

page1.php

```
<?php
session_start();
$_SESSION['name'] = 'Tom';
?>
```

page2.php

Code	Result
<?php session_start(); echo $_SESSION['name'] ; ?>	Tom

Let`s say that we have a page that is displayed only to members. If somebody is logged in than it`s username is stored in a session. In order to show the content of the page to the visitor we need to check if the username session exists or not. This can be easily done using this function:

```
isset($_SESSION['username']);
 //Returns True if exists and False if not
```

So if we use the isset function our code will look like this.

PHP Code

```
<?php
session_start();
if(isset($_SESSION['username'])){
echo 'Welcome ' . $_SESSION ['username'];
```

```
}
else{
echo 'You are not logged in!';
}
?>
```

When the user leaves your website or closes the browser the sessions are automatically deleted, but if you want to delete a session yourself use the following code:

```
unset($_SESSION['username']);
```

This is useful when you want to delete only a single session. If you want to delete all the sessions use the following code:

```
session_destroy();
```

Note: After this function you can`t use more sessions on the page. Example of using the session_destroy function is as follows:

log-out.php

```
session_start();
session_destroy();
```

By destroying the sessions the user is logged out.

Using Cookies in PHP

Cookies is small text files saved locally on the user's computer by the website, when the website later on wants the information it just reads the cookie on the user's computer. An example on what you can use cookies for is to see if the user has visited the site before, if it has then a cookie you saved the last time lies on the computer so therefore you can only check if it does.

Creating cookie

When creating a cookie you have to do it before sending anything to the page, so you can't have a html tag before it. To create a cookie you do like this:

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

Name	The name of the cookie, you use this to receive the cookie later.

Value	The value that you'll store in the cookie.
Expire	When the cookie will expire.
Path	The path on the server where the cookie is available on, leaving this empty will allow whole domain to access it.
Domain	The domain which can access the cookie, leaving this empty will allow all your domains to access it.

Here's an example on how to create a cookie:

```
$expire=time()+60*60*24*3;
setcookie("Cookie_Name", "Cookie_Value", $expire);
```

The above example will create a cookie called "Cookie_Name" with the value "Test_Value" which will expire after 3 days.

Reading Cookies

To read a cookie you use *$_COOKIE["<Name of cookie>"];* and replace **<Name of cookie>** with the name of the cookie you want to read.

So if we want to read the cookie we created above we should do like this.

```
$_COOKIE["Cookie_Name"];
```

Here comes a simple example on reading a cookie.

```
if (isset($_COOKIE["Cookie_Name"]))
{
 $Value = $_COOKIE["Cookie_Name"];
 $expire=time()+60*60*24*3;
 setcookie("Cookie_Name", $Value, $expire);
 $Value = "The Value of Cookie_Name is " . $Value;
 }
else
{
 $Value = "Cookie not set";
}
echo $Value;
```

So first of all we check if the cookie exists, if it do we reads it. Then recreate the cookie with an expire time of 3 days, this is so each time the user visit the site we will update the expire date, then we set the value to be a message telling what the value is, and if the cookie didn't existed we stored "Cookie not set" in the variable called Value. At the the end we just echo the result.