**Introduction**

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

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

**Common Uses of PHP**

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

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

**Characteristics of PHP**

Five important characteristics make PHP's practical nature possible:

1. Simplicity
2. Efficiency
3. Security
4. Flexibility
5. Familiarity

**"Hello World" Script in PHP**

To get a feel of PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script. As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this:

**Example:**

<html>

<head>

<title>Hello World</title>

<body>

<?php echo "Hello, World!";?>

</body>

</html>

It will produce the following result: Hello, World!

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

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

<?php PHP code goes here ?>

<? PHP code goes here ?>

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

Most common tag is the <?php...?> and we will also use the same tag in our tutorial. From the next chapter, we will start with PHP Environment Setup on your machine and then we will dig out almost all concepts related to PHP to make you comfortable with the PHP language.

**PHP ─ Environment Setup**

In order to develop and run PHP Web pages, three vital components need to be installed on your computer system.

**Web Server** - PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server. Download Apache for free here: <http://httpd.apache.org/download.cgi>.

**Database -** PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database. Download MySQL for free here: <http://www.mysql.com/downloads/index.html>.

**PHP Parser -** In order to process PHP script instructions, a parser must be installed to generate HTML output that can be sent to the Web Browser.

**PHP ─ Syntax**

**Escaping to PHP**

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP.'

There are four ways to do this:

Canonical PHP tags

The most universally effective PHP tag style is: <?php...?>

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

**Short-open (SGML-style) tags**

Short or short-open tags look like this: <? .............................................. ?>

**HTML script tags**

HTML script tags look like this:

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

**PHP ─ Variable Types**

Main way to store information in the middle of a PHP program is by using a variable.

Here are the most important things to know about variables in PHP.

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

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

1. **Integers:** are whole numbers, without a decimal point, like 4195. They are the simplest

type .they correspond to simple whole numbers, both positive and negative. Integers can

be assigned to variables, or they can be used in expressions, like so:

$int\_var = 12345;

$another\_int = -12345 + 12345;

1. **Doubles:** are floating-point numbers, like 3.14159 or 49.1. Default, doubles print with the minimum number of decimal places needed. For example, the code:

$many = 2.2888800;

$many\_2 = 2.2111200;

$few = $many + $many\_2;

print(.$many + $many\_2 = $few<br>.);

It produces the following browser output: 2.28888 + 2.21112 = 4.5

1. **Booleans:** have only two possible values either true or false.

PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so:

if (TRUE)

print("This will always print<br>");

else

print("This will never print<br>");

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

$my\_var = NULL;

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed: $my\_var = null;

1. **Strings:** are sequences of characters, like 'PHP supports string operations.' They are sequences of characters, like "PHP supports string operations". Following are valid examples of string:

$string\_1 = "This is a string in double quotes";

$string\_2 = "This is a somewhat longer, singly quoted string";

$string\_39 = "This string has thirty-nine characters";

$string\_0 = ""; // a string with zero characters

1. **Arrays:** are named and indexed collections of other values.
2. **Objects:** are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
3. **Resources:** are special variables that hold references to resources external to PHP (such as database connections).

The first five are *simple types*, and the next two (arrays and objects) are compound – the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot. We will explain only simile data type in this chapters. Array and Objects will be explained separately.

**Variable Naming**

Rules for naming a variable are:

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

**PHP – Variables**

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

* Local variables
* Function parameters
* Global variables
* Static variables

**PHP Local Variables**

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

<?

$x = 4;

function assignx () {

$x = 0;

print "\$x inside function is $x.

";

}

assignx();

print "\$x outside of function is $x.

";

?>

This will produce the following result.

$x inside function is 0.

$x outside of function is 4.

**PHP Function Parameters**

PHP Functions are covered in detail in PHP Function Chapter. In short, a function is a small unit of program which can take some input in the form of parameters and does some processing and may return a value. Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be:

<?

// multiply a value by 10 and return it to the caller

function multiply ($value) {

$value = $value \* 10;

return $value;

}

$retval = multiply (10);

Print "Return value is $retval\n";

?>

This will produce the following result.

Return value is 100

**PHP Global Variables**

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

<?

$somevar = 15;

function addit() {

GLOBAL $somevar;

$somevar++;

print "Somevar is $somevar";

}

addit();

?>

This will produce the following result.

Somevar is 16

**PHP Static Variables**

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

<?

function keep\_track() {

STATIC $count = 0;

$count++;

print $count;

print "

";

}

keep\_track();

keep\_track();

keep\_track();

?>

This will produce the following result.

1

2

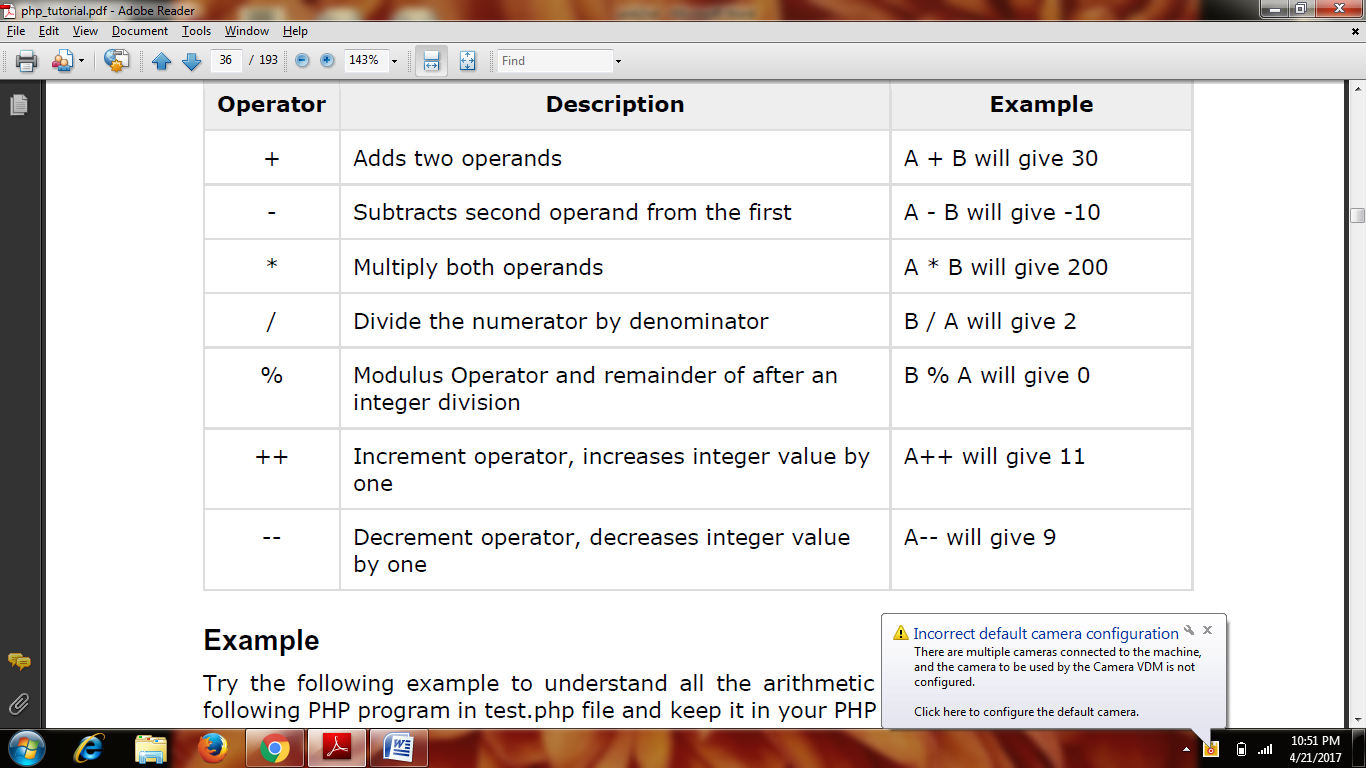
3

**PHP ─ Operator Types**

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

* + Arithmetic Operators

The following arithmetic operators are supported by PHP language: Assume variable A holds 10 and variable B holds 20 then:



**Example:**

<html>

<head><title>Arithmetical Operators</title><head>

<body>

<?php

$a = 42;

$b = 20;

$c = $a + $b;

echo "Addition Operation Result: $c <br/>";

$c = $a - $b;

echo "Subtraction Operation Result: $c <br/>";

$c = $a \* $b;

echo "Multiplication Operation Result: $c <br/>";

$c = $a / $b;

echo "Division Operation Result: $c <br/>";

$c = $a % $b;

echo "Modulus Operation Result: $c <br/>";

$c = $a++;

echo "Increment Operation Result: $c <br/>";

$c = $a--;

echo "Decrement Operation Result: $c <br/>";

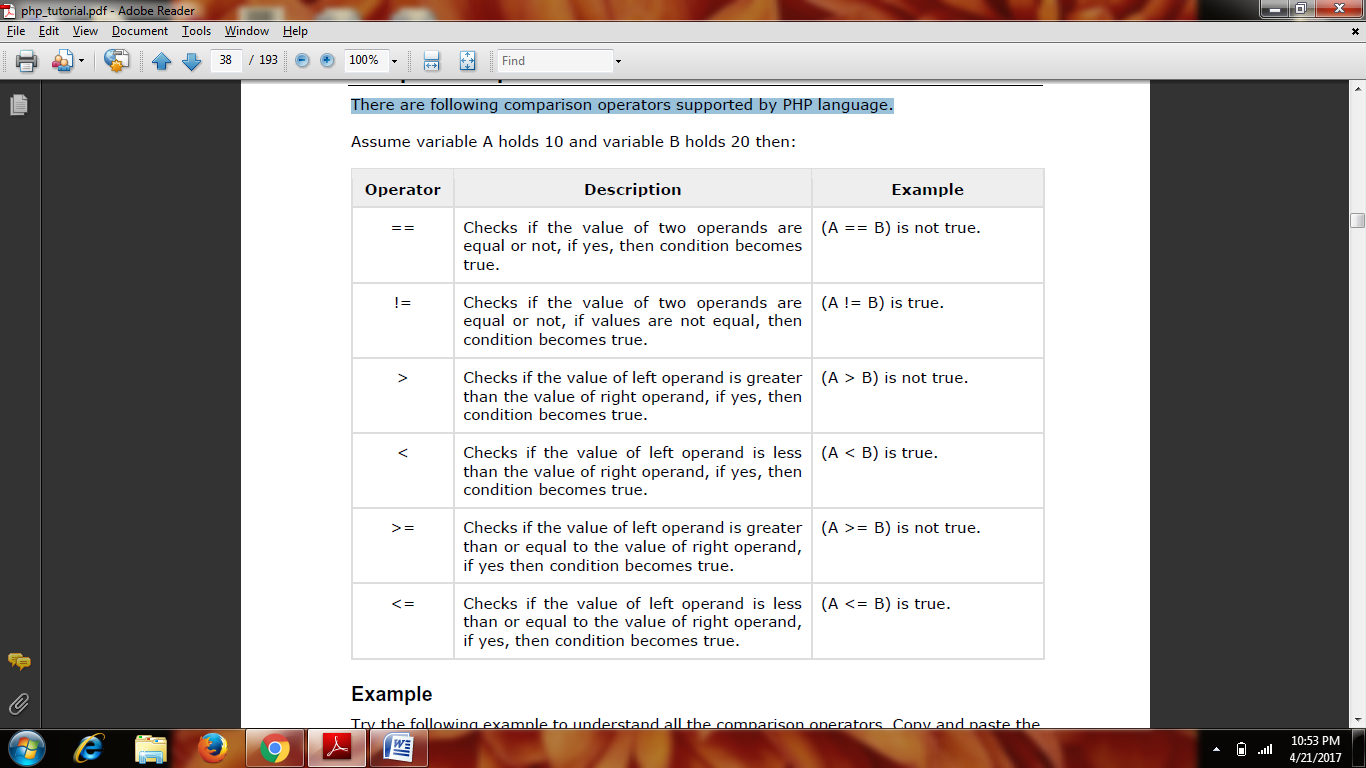
?>

</body>

</html>

* + Comparison Operators

There are following comparison operators supported by PHP language. Assume variable A holds 10 and variable B holds 20 then:



**Example**

<html>

<head><title>Comparison Operators</title><head>

<body>

<?php

$a = 42;

$b = 20;

if( $a == $b ){

echo "TEST1 : a is equal to b<br/>";

}else{

echo "TEST1 : a is not equal to b<br/>";

}

if( $a > $b ){

echo "TEST2 : a is greater than b<br/>";

}else{

echo "TEST2 : a is not greater than b<br/>";

}

if( $a < $b ){

echo "TEST3 : a is less than b<br/>";

}else{

echo "TEST3 : a is not less than b<br/>";

}

if( $a != $b ){

echo "TEST4 : a is not equal to b<br/>";

}else{

echo "TEST4 : a is equal to b<br/>";

}

if( $a >= $b ){

echo "TEST5 : a is either greater than or equal to b<br/>";

}else{

echo "TEST5 : a is neither greater than nor equal to b<br/>";

}

if( $a <= $b ){

echo "TEST6 : a is either less than or equal to b<br/>";

}else{

echo "TEST6 : a is neither less than nor equal to b<br/>";

}

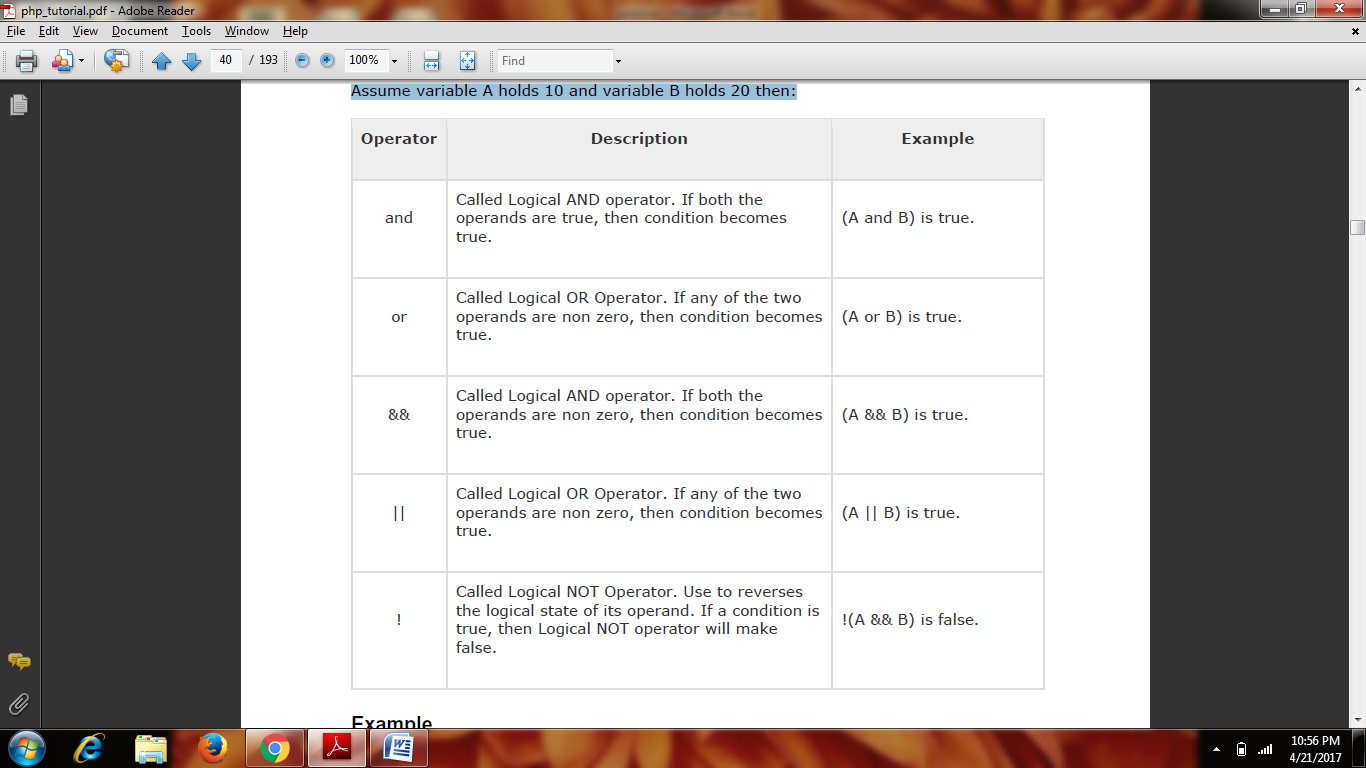
?>

</body>

</html>

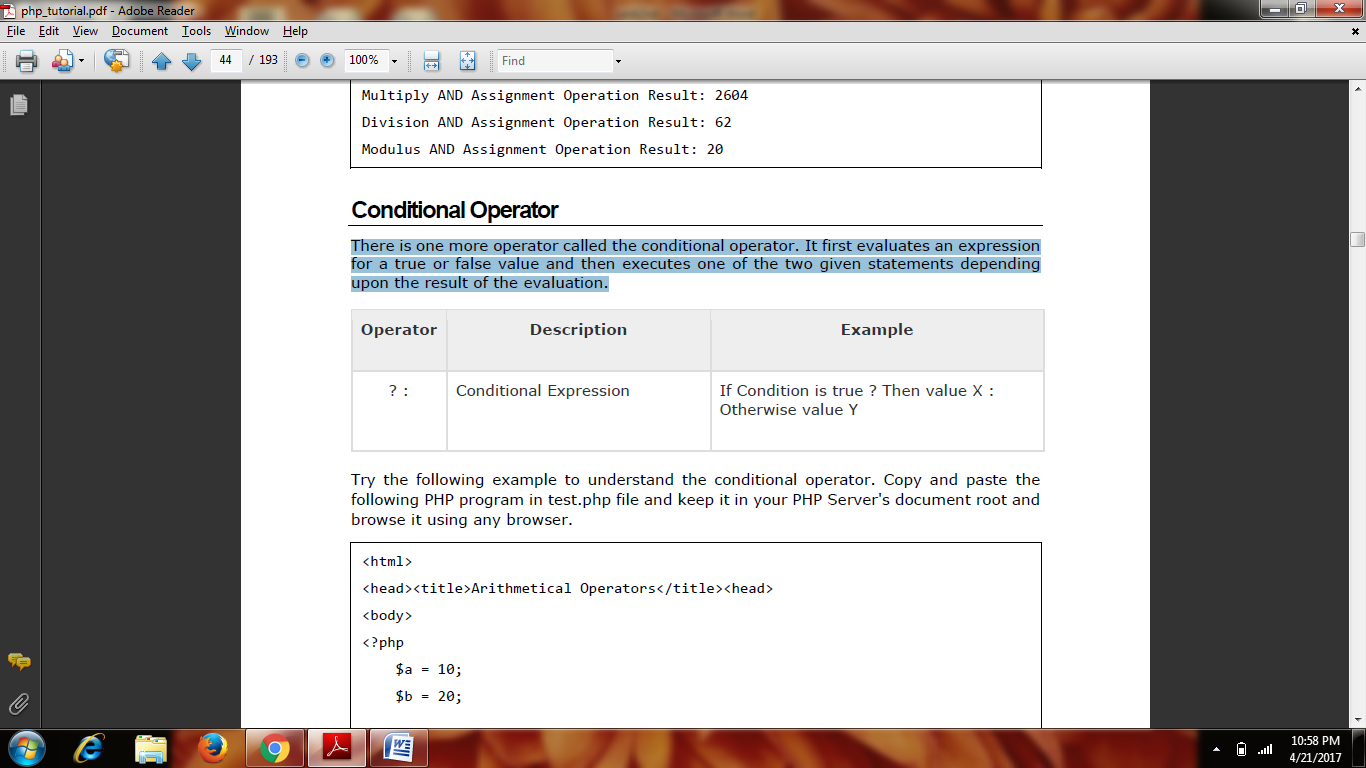
* + Logical (or Relational) Operators

The following logical operators are supported by PHP language. Assume variable A holds 10 and variable B holds 20 then: Assignment Operators



* + Conditional (or ternary) Operators

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



**PHP ─ Decision Making**

The if, elseif ...else and switch statements are used to take decision based on the different condition. You can use conditional statements in your code to make your decisions. PHP supports the following three decision making statements:

**The If...Else Statement**

If you want to execute some code if a condition is true and another code if a condition is

false, use the if....else statement.

**Syntax**

if (condition)

code to be executed if condition is true;

else

code to be executed if condition is false;

**Example**

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

otherwise it will output "Have a nice day!":

<html>

<body>

<?php

$d=date("D");

if ($d=="Fri")

echo "Have a nice weekend!";

else

echo "Have a nice day!";

?>

</body>

</html>

**The ElseIf Statement**

If you want to execute some code if one of the several conditions is true, then use the elseif statement.

**Syntax**

if (condition)

code to be executed if condition is true;

elseif (condition)

code to be executed if condition is true;

else

code to be executed if condition is false;

**Example:**

<html>

<body>

<?php

$d=date("D");

if ($d=="Fri")

echo "Have a nice weekend!";

elseif ($d=="Sun")

echo "Have a nice Sunday!";

else

echo "Have a nice day!";

?>

</body>

</html>

**The Switch Statement**

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

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

Syntax

switch (expression)

{

case label1:

code to be executed if expression = label1;

break;

case label2:

code to be executed if expression = label2;

break;

default:

code to be executed

if expression is different

from both label1 and label2;

}

**Example:**

<html>

<body>

<?php

$d=date("D");

switch ($d)

{

case "Mon":

echo "Today is Monday";

break;

case "Tue":

echo "Today is Tuesday";

break;

case "Wed":

echo "Today is Wednesday";

break;

case "Thu":

echo "Today is Thursday";

break;

case "Fri":

echo "Today is Friday";

break;

case "Sat":

echo "Today is Saturday";

break;

case "Sun":

echo "Today is Sunday";

break;

default:

echo "Wonder which day is this ?";

}

?>

</body>

</html>

**PHP ─ Loop Types**

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

**The for loop statement**

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

**Syntax**

for (initialization; condition; increment)

{

code to be executed;

}

**Example:**

<html>

<body>

<?php

$a = 0;

$b = 0;

for( $i=0; $i<5; $i++ )

{

$a += 10;

$b += 5;

}

echo ("At the end of the loop a=$a and b=$b" );

?>

</body>

</html>

This will produce the following result: At the end of the loop a=50 and b=25

**The while loop statement**

The while statement will execute a block of code if and as long as a test expression is true. If the test expression is true, then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

**Syntax**

while (condition)

{

code to be executed;

}

Example: <html>

<body>

<?php

$i = 0;

$num = 50;

while( $i < 10)

{

$num--;

$i++;

}

echo ("Loop stopped at i = $i and num = $num" );

?>

</body>

</html>

This will produce the following result:

Loop stopped at i = 10 and num = 40

**The do...while loop statement**

The do...while statement will execute a block of code at least once - it will then repeat the

loop as long as a condition is true.

**Syntax**

do

{

code to be executed;

}while (condition);

**Example:**

<html>

<body>

<?php

$i = 0;

$num = 0;

do

{

$i++;

}while( $i < 10 );

echo ("Loop stopped at i = $i" );

?>

</body>

</html>

This will produce the following result:

Loop stopped at i = 10

**The foreach loop statement**

The foreach statement is used to loop through arrays. For each pass the value of the

current array element is assigned to $value and the array pointer is moved by one and in

the next pass next element will be processed.

**Syntax**

foreach (array as value)

{

code to be executed;

}

Example:

<html>

<body>

<?php

$array = array( 1, 2, 3, 4, 5);

foreach( $array as $value )

{

echo "Value is $value <br />";

}

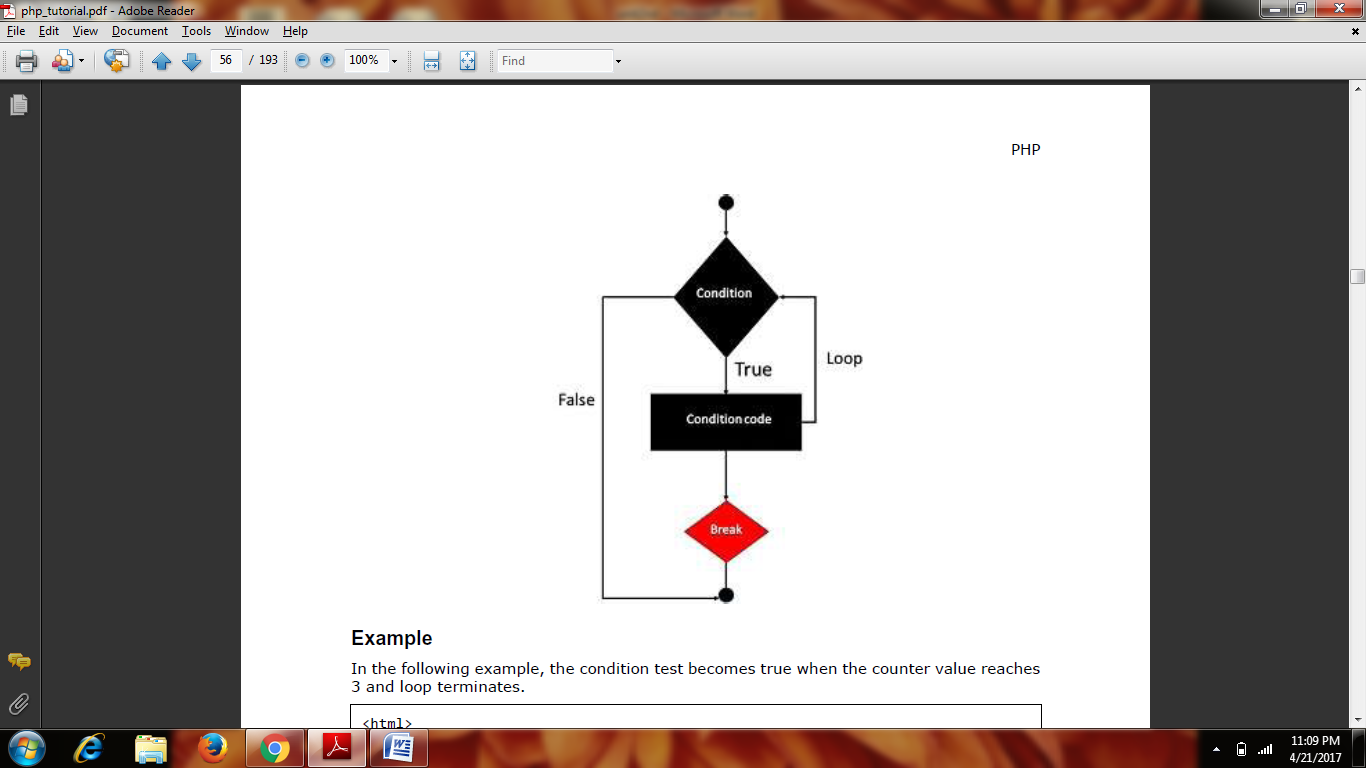
?>

</body>

</html>

**The break statement**

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



Example:

<html>

<body>

<?php

$i = 0;

while( $i < 10)

{

$i++;

if( $i == 3 )break;

}

echo ("Loop stopped at i = $i" );

?>

</body>

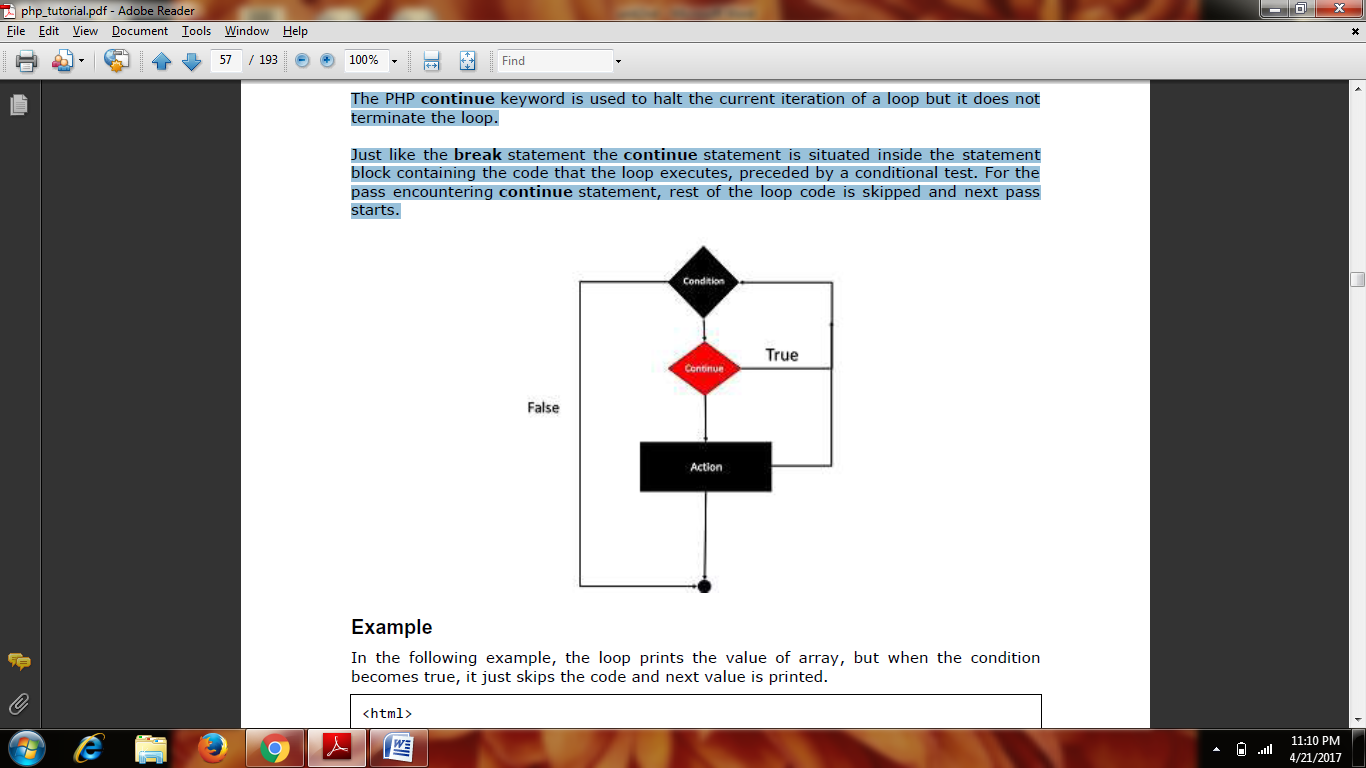
</html>

This will produce the following result:

Loop stopped at i = 3

**The continue statement**

The PHP continue keyword is used to halt the current iteration of a loop but it does not terminate the loop. Just like the break statement the continue statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering continue statement, rest of the loop code is skipped and next pass starts.



**Example:**

<html>

<body>

<?php

$array = array( 1, 2, 3, 4, 5);

foreach( $array as $value )

{

if( $value == 3 )continue;

echo "Value is $value <br />";

}

?>

</body>

</html>

**PHP ─ Arrays**

An array is a data structure that stores one or more similar type of values in a single value. For example, if you want to store 100 numbers, then instead of defining 100 variables, it is easy to define an array of 100 length. There are three different kinds of arrays and each array value is accessed using an ID c which is called array index.

**Numeric Array**

These arrays can store numbers, strings and any object but their index will be represented

by numbers. By default, the array index starts from zero.

**Example**

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

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

<html>

<body>

<?php

/\* First method to create array. \*/

$numbers = array( 1, 2, 3, 4, 5);

foreach( $numbers as $value )

{

echo "Value is $value <br />";

}

/\* Second method to create array. \*/

$numbers[0] = "one";

$numbers[1] = "two";

$numbers[2] = "three";

$numbers[3] = "four";

$numbers[4] = "five";

foreach( $numbers as $value )

{

echo "Value is $value <br />";

}

?>

</body>

</html>

**Associative Arrays**

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values. To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

Example

<html>

<body>

<?php

/\* First method to associate create array. \*/

$salaries = array(

"mohammad" => 2000,

"qadir" => 1000,

"zara" => 500

);

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";

echo "Salary of qadir is ". $salaries['qadir']. "<br />";

echo "Salary of zara is ". $salaries['zara']. "<br />";

/\* Second method to create array. \*/

$salaries['mohammad'] = "high";

$salaries['qadir'] = "medium";

$salaries['zara'] = "low";

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";

echo "Salary of qadir is ". $salaries['qadir']. "<br />";

echo "Salary of zara is ". $salaries['zara']. "<br />";

?>

</body>

</html>

**Multidimensional Arrays**

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

**Example:**

<html>

<body>

<?php

$marks = array(

"mohammad" => array

(

"physics" => 35,

"maths" => 30,

"chemistry" => 39

),

"qadir" => array

(

"physics" => 30,

"maths" => 32,

"chemistry" => 29

),

"zara" => array

(

"physics" => 31,

"maths" => 22,

"chemistry" => 39

)

);

/\* Accessing multi-dimensional array values \*/

echo "Marks for mohammad in physics : " ;

echo $marks['mohammad']['physics'] . "<br />";

echo "Marks for qadir in maths : ";

echo $marks['qadir']['maths'] . "<br />";

echo "Marks for zara in chemistry : " ;

echo $marks['zara']['chemistry'] . "<br />";

?>

</body>

</html>

**PHP ─ Strings**

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

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

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

The escape-sequence replacements are:

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

**String Concatenation Operator**

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

<?php

$string1="Hello World";

$string2="1234";

echo $string1 . " " . $string2;

?>

This will produce the following result:

Hello World 1234

**Using the strlen() function**

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

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

<?php

echo strlen("Hello world!");

?>

This will produce the following result: 12

**Using HTML Forms**

The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will automatically be available to your PHP scripts. Try out the following example by putting the source code in test.php script.

<?php

if( $\_POST["name"] || $\_POST["age"] )

{

echo "Welcome ". $\_POST['name']. "<br />";

echo "You are ". $\_POST['age']. " years old.";

exit();

}

?>

<html>

<body>

<form action="<?php $\_PHP\_SELF ?>" method="POST">

Name: <input type="text" name="name" />

Age: <input type="text" name="age" />

<input type="submit" />

</form>

</body>

</html>

* The PHP default variable $\_PHP\_SELF is used for the PHP script name and when you click "submit" button, the same PHP script will be called and will produce following result:
* The method = "POST" is used to post user data to the server script. There are two methods of posting data to the server script which are discussed in PHP GET & POST chapter.

**PHP ─ GET and POST Methods**

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

* The GET Method
* The POST Method

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

name1=value1&name2=value2&name3=value3

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

**The GET Method**

The GET method sends the encoded user information appended to the page request. The

page and the encoded information are separated by the ? character.

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

The GET method produces a long string that appears in your server logs, in the browser's Location: box.

* The GET method is restricted to send up to 1024 characters only.
* Never use GET method if you have password or other sensitive information to be sent to the server.
* GET can't be used to send binary data, like images or word documents, to the server.
* The data sent by GET method can be accessed using QUERY\_STRING environment variable.
* The PHP provides $\_GET associative array to access all the sent information using GET method.

**The POST Method**

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

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

**Example**

<?php

if( $\_POST["name"] || $\_POST["age"] )

{

echo "Welcome ". $\_POST['name']. "<br />";

echo "You are ". $\_POST['age']. " years old.";

exit();

}

?>

<html>

<body>

<form action="<?php $\_PHP\_SELF ?>" method="POST">

Name: <input type="text" name="name" />

Age: <input type="text" name="age" />

<input type="submit" />

</form>

</body>

</html>

**PHP ─ Functions**

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value. You already have seen many functions like fopen() and fread() etc. They are built-in functions but PHP gives you option to create your own functions as well. There are two parts which should be clear to you:

* Creating a PHP Function
* Calling a PHP Function

**Creating PHP Function**

It is very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. The following example creates a function called writeMessage() and then calls it just after creating it. Note that while creating a function its name should start with keyword function and all the PHP code should be put inside { and } braces as shown in the following example below:

<html>

<head>

<title>Writing PHP Function</title>

</head>

<body>

<?php

/\* Defining a PHP Function \*/

function writeMessage()

{

echo "You are really a nice person, Have a nice time!";

}

/\* Calling a PHP Function \*/

writeMessage();

?>

</body>

</html>

**PHP Functions with Parameters**

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

<html>

<head>

<title>Writing PHP Function with Parameters</title>

</head>

<body>

<?php

function addFunction($num1, $num2)

{

$sum = $num1 + $num2;

echo "Sum of the two numbers is : $sum";

}

addFunction(10, 20);

?>

</body>

</html>

**PHP ─ Sessions**

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

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit. The location of the temporary file is determined by a setting in the php.ini file called session.save\_path. Bore using any session variable make sure you have setup this path. When a session is started, the following actions take place:

* PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
* A cookie called PHPSESSID is automatically sent to the user's computer to store unique session identification string.
* A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess\_ ie sess\_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values. A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

**Starting a PHP Session**

A PHP session is easily started by making a call to the **session\_start()** function. This function first checks if a session is already started and if none is started, then it starts one. It is recommended to put the call to **session\_start()** at the beginning of the page. Session variables are stored in associative array called **$\_SESSION[]**. These variables can be accessed during lifetime of a session. The following example starts a session and then registers a variable called **counter** that is incremented each time the page is visited during the session. Make use of **isset()** function to check if session variable is already set or not.

<?php

session\_start();

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

{

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

}

else

{

$\_SESSION['counter'] = 1;

}

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

$msg .= "in this session.";

?>

**Example:**

<html>

<head>

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

</head>

<body>

<?php echo ( $msg ); ?>

</body>

</html>

**Destroying a PHP Session**

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

Here is the example to unset a single variable:

<?php

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

?>

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

<?php

session\_destroy();

?>

**PHP ─ File Uploading**

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.

Information in the phpinfo.php page describes the temporary directory that is used for file uploads as upload\_tmp\_dir and the maximum permitted size of files that can be uploaded is stated as upload\_max\_filesize. These parameters are set into PHP configuration filephp.ini The process of uploading a file follows these steps:

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

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

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

**Creating an Upload Form**

<html>

<head>

<title>File Uploading Form</title>

</head>

<body>

<h3>File Upload:</h3>

Select a file to upload: <br />

<form action="/php/file\_uploader.php" method="post"

enctype="multipart/form-data">

<input type="file" name="file" size="50" />

<br />

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

</form>

</body>

</html>

**Creating an upload script**

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

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

**PHP Database ODBC**

ODBC is an Application Programming Interface (API) that allows you to connect to a data source (e.g. an MS Access database).

**Create an ODBC Connection**

With an ODBC connection, you can connect to any database, on any computer in your network, as long as an ODBC connection is available.

Here is how to create an ODBC connection to a MS Access Database:

1. Open the **Administrative Tools** icon in your Control Panel.
2. Double-click on the **Data Sources (ODBC)** icon inside.
3. Choose the **System DSN** tab.
4. Click on **Add**in the System DSN tab.
5. Select the **Microsoft Access Driver**. Click **Finish.**
6. In the next screen, click **Select** to locate the database.
7. Give the database a **Data Source Name (DSN)**.
8. Click **OK**.

**Connecting to an ODBC**

The odbc\_connect() function is used to connect to an ODBC data source. The function takes four parameters: the data source name, username, password, and an optional cursor type.

The odbc\_exec() function is used to execute an SQL statement.

An ODBC Example

The following example shows how to first create a database connection, then a result-set, and then display the data in an HTML table.

**Example:**

<html>

<body>

<?php

$conn=odbc\_connect('northwind','','');

if (!$conn)

{exit("Connection Failed: " . $conn);

}

$sql="SELECT \* FROM customers";

$rs=odbc\_exec($conn,$sql);

if (!$rs)

{

exit("Error in SQL");

}

echo "<table><tr>";

echo "<th>Companyname</th>";

echo "<th>Contactname</th></tr>";

while (odbc\_fetch\_row($rs))

{

$compname=odbc\_result($rs,"CompanyName");

$conname=odbc\_result($rs,"ContactName");

echo "<tr><td>$compname</td>";

echo "<td>$conname</td></tr>";

}

odbc\_close($conn);

echo "</table>";

?>

</body>

</html>