PHP & MYSQL NOTES

INTRODUCTION

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

What is a PHP File ?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

What Can PHP Do ?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

Basic PHP Syntax:

A PHP script can be placed anywhere in the document.

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

<?php

// PHP code goes here

?>

Creating (Declaring) PHP Variables:

In PHP, a variable starts with the $ sign, followed by the name of the variable:

Example:

```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

```
Output:
```

Hello world!
5
10.5

PHP echo and print Statements:

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

Echo example:

```
<?php
$txt1 = "Learn PHP";
$txt2 = "vissicomp.com";
$x = 5;
$y = 4;

echo "<h2>$txt1</h2>";
echo "Study PHP at $txt2<br>";
echo $x + $y;
?>
```

Output:

Learn PHP

Study PHP at vissicomp.com
9

Print Example:

```
<?php

print "<h2>PHP is Fun!</h2>";

print "Hello world!<br>";

print "I'm about to learn PHP!";

?>
```

PHP Data Types

PHP Data Types :

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

PHP String :

- A string is a sequence of characters, like "Hello world!".
- A string can be any text inside quotes. You can use single or double quotes:

```
<?php
$x = "Hello world!";
$y = 'Hello world!';

echo $x;
echo "<br>";
echo $y;
?>
```

Output:

```
Hello world!
Hello world!
```

PHP Integer:

An integer is a whole number (without decimals). It is a number between -2,147,483,648 and +2,147,483,647.

Rules for integers:

- An integer must have at least one digit (0-9)
- An integer cannot contain comma or blanks
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

```
<?php

$x = 5985;
var_dump($x);

?>
```

Output:

int(5985)

PHP Float :

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

In the following example $x is a float. The PHP var_dump() function returns the data type and value:

```
<?php
$x = 10.365;
var_dump($x);
?>
```

Output:

float(10.365)

PHP Array :

An array stores multiple values in one single variable.

In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

```
<?php

$cars = array("Volvo","BMW","Toyota");

var_dump($cars);

?>
```

Output:

array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }

PHP Object :

An object is a data type which stores data and information on how to process that data.

In PHP, an object must be explicitly declared.

First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

```
<?php
class Car {

   function Car()
{

      $this->model = "VW";
   }
}

// create an object

$herbie = new Car();

// show object properties

echo $herbie->model;

?>
```

Output :

VW

PHP OPERATORS

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

PHP Arithmetic Operators:

+	Addition

```
<?php

$x = 10;
$y = 6;

echo $x + $y;
?>
```

Output:

16

-	Subtraction

```
<?php
$x = 10;
```

```
$y = 6;

echo $x - $y;
?>
```
Output:
4

*	Multiplication

```
<?php
$x = 10;
$y = 6;

echo $x * $y;
?>
```
Output:
60

/	Division

```
<?php
$x = 10;
$y = 6;

echo $x / $y;

?>
```
Output:

%	Modulus

```
<?php
$x = 10;
$y = 6;

echo $x % $y;
?>
```
Output:
4

PHP Assignment Operators :

The PHP assignment operators are used with numeric values to write a value to a variable.

x = y

```
<?php
$x = 10;
echo $x;
?>
Output:
10
```

x += y	x = x + y	Addition

```
<?php

$x = 20;
$x += 100;

echo $x;
?>
Output:
120
```

x -= y	x = x - y	Subtraction

```
<?php

$x = 50;
$x -= 30;

echo $x;

?>
Output:
20
```

x *= y	x = x * y	Multiplication

```
<?php

$x = 10;

$y = 6;

echo $x * $y;

?>
```

Output:

60

x /= y	x = x / y	Division

```
<?php

$x = 10;
$x /= 5;

echo $x;

?>
```

Output:

2

x %= y	x = x % y	Modulus

```
<?php
$x = 15;
$x %= 4;

echo $x;
?>
```

Output:

3

PHP Comparison Operators

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

==	Equal	$x == $y	Returns true if $x is equal to $y

```
<?php
$x = 100;
$y = "100";

var_dump($x == $y); // returns true because values are equal

?>
Output:
Bool(true)
```

!=	Not equal	$x != $y	Returns true if $x is not equal to $y

```
<?php

$x = 100;
$y = "100";

var_dump($x != $y); // returns false because values are equal
?>

Output:
Bool(false)
```

>	Greater than	$x > $y	Returns true if $x is greater than $y

```
<?php
$x = 100;
$y = 50;

var_dump($x > $y); // returns true because $x is greater than $y
?>

Output:
bool(true)
```

<	Less than	$x < $y	Returns true if $x is less than $y

```
<?php
$x = 10;
$y = 50;

var_dump($x <  $y); // returns true because $x is less than $y
?>
```

Output:

```
Bool(true)
```

>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y

```
<?php

$x = 50;

$y = 50;

var_dump($x >= $y); // returns true because $x is greater than or equal to $y

?>
```

Output:

bool(true)

<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

```
<?php
$x = 50;
$y = 50;

var_dump($x <= $y); // returns true because $x is less than or equal to $y
?>
```

Output:

bool(true)

PHP Increment / Decrement Operators:

The PHP increment operators are used to increment a variable's value.

The PHP decrement operators are used to decrement a variable's value.

++$x	Pre-increment	Increments $x by one, then returns $x

```
<?php
$x = 10;
echo ++$x;
?>
```

Output:

11

$x++	Post-increment	Returns $x, then increments $x by one

```
<?php
$x = 10;
echo $x++;
?>
```

```
Output:

10
```

--$x	Pre-decrement	Decrements $x by one, then returns $x

```
<?php
$x = 10;
echo --$x;
?>
```

```
Output:

9
```

$x--	Post-decrement	Returns $x, then decrements $x by one

```
<?php
$x = 10;
```

```
echo $x--;
?>
```

Output:

10

PHP Logical Operators:

The PHP logical operators are used to combine conditional statements.

&&	And	$x && $y	True if both $x and $y are true

```
<?php
$x = 100;
$y = 50;

if ($x == 100 && $y == 50) {
     echo "Hello world!";
}
?>
```

Output:

Hello world!

||	Or	$x || $y	True if either $x or $y is true

```
<?php


$x = 100;
$y = 50;

if ($x == 100 || $y == 80) {
     echo "Hello world!";
}
?>
```

Output:

Hello world!

!	Not	!$x	True if $x is not true

```
<?php

$x = 100;

if ($x !== 90)

{
     echo "Hello world!";
}

?>
```

Output:

Hello world!

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this. In PHP we have the following conditional statements:

- **if statement** - executes some code only if a specified condition is true
- **if...else statement** - executes some code if a condition is true and another code if the condition is false
- **if...elseif....else statement** - specifies a new condition to test, if the first condition is false
- **switch statement** - selects one of many blocks of code to be executed

PHP - The if Statement

The if statement is used to execute some code **only if a specified condition is true**.

Syntax

```
if (condition) {
    code to be executed if condition is true;
}
```

Example

```
<?php



$color="red";



If($color=="red")
```

```
{


echo "color is red";


}


?>
```

OUTPUT:

color is red

PHP - The if...else Statement

Use the if....else statement to execute some code **if a condition is true and another code if the condition is false**.

Syntax

```
if (condition) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}
```

```
<?php



$color="red";



if($color=="red")
```

```
{

echo "color is red";

}

else



{

echo  "not red color";



}



?>
```

OUTPUT:

color is red

PHP - The if...else if....else Statement

Use the if….else if…else statement to **specify a new condition to test, if the first condition is false.**

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;
}
```

```
<?php

$color="orange";

if($color=="red")
{

echo "color is red";

}

else if($color=="orange")

{

echo "color is orange";

}
```

```
else

{

echo "No color:";



}
?>
```

Output:

Color is orange

The PHP switch Statement

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    case label3:
        code to be executed if n=label3;
        break;
    ...
    default:
```

```
        code to be executed if n is different from all labels;
}
```

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

Example

```
<?php


$color="blue";


switch($color)


{


case "red":

echo "color is red";


break;


case  "blue":




echo "color is blue";


break;
```

```
case "orange":

echo "color is orange";

break;

default:

echo "no color";

break;
}
?>
```

Output:

```
color is blue
```

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.
In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true.
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true.

- **for** - loops through a block of code a specified number of times.
- **foreach** - loops through a block of code for each element in an array.

The PHP for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
    code to be executed;
}
```

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value

Example:

```
<?php


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


{




echo $i;

echo ‘<br>’;
```

```
}



?>
```

Output:

0

1

2

3

4

The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
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, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array ($colors):

Example

```
<?php

$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
```

```
    echo "$value <br>";
}
?>
```

Output:

red
green
blue
yellow

The PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

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

Example:

```
<?php
$i=0;
while($i<5)
{
echo $i;
echo ‘<br>’;
$i++;
}
?>
```

Output:

0

1

2

3

4

The PHP do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax:

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

Example:

```
<?php

$i=0;

do

{


echo $i;


$i++;
```

```
}

while($i<5);

?>
```

Output :

0

1

2

3

4

Array-in-PHP :

What is an Array?
An array is a special variable, which can hold more than one value at a time.
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
The solution is to create an array!
An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP **:**
In PHP, the array() function is used to create an array:
array();

In PHP, there are three types of arrays:
Indexed arrays - Arrays with a numeric index.
Associative arrays - Arrays with named keys.
Multidimensional arrays - Arrays containing one or more arrays.

(1)PHP Indexed Arrays :
There are two ways to create indexed arrays:
The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
```

```
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

Example:

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

<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>

</body>
</html>
```

(2)PHP Associative Arrays:

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

There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

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

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>

</body>
```

```
</html>
```

(3)Multidimensional Associative Array:

```
<?php
$marks=array(
"mohammad"=>array
(
"physics"=> 35,
"maths"=> 30,
"chemistry"=>39
),

"zara"=>array
(
"physics"=> 35,
"maths"=> 40,
"chemistry"=> 21
),
"quara"=>array
(
"physics"=> 45,
"maths"=> 48,
"chemistry"=> 45
)
);

echo "value of mohammad array".'<br>';
echo $marks["mohammad"]["physics"].'<br>';
echo $marks["mohammad"]["maths"].'<br>';
echo $marks["mohammad"]["chemistry"].'<br>';
echo" value of zara array".'<br>';
echo $marks ["zara"]["physics"].'<br>';
echo $marks["zara"]["maths"].'<br>';
echo $marks["zara"]["chemistry"].'<br>';
echo"value of quara array".'<br>';
echo $marks["quara"]["physics"].'<br>';
echo $marks["quara"]["maths"].'<br>';
```

```
echo $marks["quara"]["chemistry"].'<br>';
?>
```

OUTPUT :

value of mohammad array
35
30
39
value of zara array
35
40
21
value of quara array
45
48
45

(4)Multidimensional Index Array :

```
<?php
$abc=array
(
array(1,2,3),
array(4,5,6),
array(6,7,8)
);
echo"value of position 0,0=".$abc[0][0].'<br>';
echo "value of position 0,1=".$abc[0][1].'<br>';
echo "value of position 0,2=".$abc[0][2].'<br>';
echo "value of position 1,0=".$abc[1][0].'<br>';
echo "value of position 1,1=".$abc[1][1].'<br>';
echo "value of position 1,2=".$abc[1][2].'<br>';
echo "value of position 2,0=".$abc[2][0].'<br>';
echo "value of position 2,1=".$abc[2][1].'<br>';
echo "value of position 2,2=".$abc[2][2];

?>
```

OUTPUT :

value of position 0,0=1
value of position 0,1=2
value of position 0,2=3
value of position 1,0=4
value of position 1,1=5

value of position 1,2=6
value of position 2,0=6
value of position 2,1=7
value of position 2,2=8

PHP Functions

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

PHP User Defined Functions

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

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

A function will not execute immediately when a page loads.

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

Create a User Defined Function in PHP

A user defined function declaration starts with the word "function":

Syntax

```
function functionName() {
    code to be executed;
}
```

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

```
<?php

function writeMsg()
 {
   echo "Hello world!";
}
```

```
writeMsg(); // call the function

?>
```

Output:

Hello world!

PHP Function Arguments:

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just seperate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```
<?php
function familyName($fname) {
   echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

Output:

Jani Refsnes.
Hege Refsnes.
Stale Refsnes.
Kai Jim Refsnes.
Borge Refsnes.

The following example has a function with two arguments ($fname and $year):

Example

```
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege", "1975");


familyName("Stale", "1978");


familyName("Kai Jim", "1983");


?>
```

Output:

Hege Refsnes. Born in 1975
Stale Refsnes. Born in 1978
Kai Jim Refsnes. Born in 1983

PHP Functions - Returning values

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

Example

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

echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

Output:

5 + 10 = 15
7 + 13 = 20
2 + 4 = 6

PHP FORM HANDLING :

The PHP superglobals $_GET and $_POST are used to collect form-data.
PHP - A Simple HTML Form

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

Example

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

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

</body>
</html>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.
To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

The output could be something like this:
Welcome John
Your email address is john.doe@example.com
The same result could also be achieved using the HTTP GET method:
Example

```
<html>
<body>

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

</body>
</html>
```

and "welcome_get.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>
```

</body>
</html>
The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.
Think SECURITY when processing PHP forms!

This page does not contain any form validation, it just shows how you can send and retrieve form data.
However, the next pages will show how to process PHP forms with security in mind! Proper validation of form data is important to protect your form from hackers and spammers!
GET vs. POST

Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.
Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
$_GET is an array of variables passed to the current script via the URL parameters.
$_POST is an array of variables passed to the current script via the HTTP POST method.
When to use GET?

Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
GET may be used for sending non-sensitive data.
Note: GET should NEVER be used for sending passwords or other sensitive information!
When to use POST?

Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send.
Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

FORM VALIDATION USING PHP :

(1)first write code for **"form.html"** file.

<form action="validation.php" method="post">

```
Name     <input type="text"         name="name">

Contact    <input type="text"          name="contact">

Email    <input type="text"             name="email">

<input type="submit"    name="submit"     value="submit"  >

</form>
```

(2)WRITE CODE FOR **“validation.php”** file:

```
<?php


$msg=array();




if(empty($_POST['email']))


{


$msg="enter your email";

}


$regex = "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$^";


if(!preg_match( $regex, $_POST['email'] ))


{

$msg="enter your valid email";



}
```

```
if(empty($_POST['contact']))

{

$msg="enter your contact";

}

if(strlen($_POST['contact'])>10)

{

$msg="accept only 10 digit";

}

if(!is_numeric($_POST['contact']))

{

$msg="accept only numbers only";

}

if(empty($_POST['name']))

{

$msg="enter your name";
```

```
}


if(empty($msg))

{


echo "successful";

}

else

{

echo $msg;


}


?>
```

PHP Cookies & Session Handling :

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax

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

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

PHP Create/Retrieve a Cookie

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

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

Example

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

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

</body>
</html>
```

Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>
```

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

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

Start a PHP Session

A session is started with the session_start() function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example

```
<?php
// Start the session
session_start();
```

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

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Also notice that all session variable values are stored in the global $_SESSION variable:

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
```

```
?>

</body>
</html>
```

Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

BASIC MYSQL INTRODUCTION

What is Database?

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds.

Other kinds of data stores can be used, such as files on the file system or large hash tables in memory but data fetching and writing would not be so fast and easy with those types of systems.

So nowadays, we use relational database management systems (RDBMS) to store and manage huge volume of data. This is called relational database because all the data is stored into different tables and relations are established using primary keys or other keys known as foreign keys.

A **Relational DataBase Management System (RDBMS)** is a software that:

- Enables you to implement a database with tables, columns and indexes.
- Guarantees the Referential Integrity between rows of various tables.
- Updates the indexes automatically.
- Interprets an SQL query and combines information from various tables.

RDBMS Terminology:

Before we proceed to explain MySQL database system, let's revise few definitions related to database.

- **Database:** A database is a collection of tables, with related data.
- **Table:** A table is a matrix with data. A table in a database looks like a simple spreadsheet.
- **Column:** One column (data element) contains data of one and the same kind, for example the column postcode.
- **Row:** A row (= tuple, entry or record) is a group of related data, for example the data of one subscription.
- **Redundancy:** Storing data twice, redundantly to make the system faster.
- **Primary Key:** A primary key is unique. A key value can not occur twice in one table. With a key, you can find at most one row.
- **Foreign Key:** A foreign key is the linking pin between two tables.
- **Compound Key:** A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.
- **Index:** An index in a database resembles an index at the back of a book.
- **Referential Integrity:** Referential Integrity makes sure that a foreign key value always points to an existing row.

What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

.Mysql is the most popular open-source database system.

What is Mysql ?

Open source database system

like most modern Database Management Systems is based on the relational model, RDBMS.

Free

Run on Linux, Windows, Netware, AIX, and so on.

easily accessible through programming languages like PHP

Why Mysql?

Open source database

consistent fast performance & high reliability

ease of use

High Availability

Comprehensive Application Development

Management Ease

Lowest Total Cost of Ownership

Where You can get it?

You can download at mysql official website, mysql.com

You can get mysql that packaged with php like in AppServ

For portable application, you can try XAMPP

PHP & MySQL QUERIES

```
<?php

mysql_connect("localhost", "admin", "1admin") or die(mysql_error());

echo "Connected to MySQL<br />";

?>
```

Display:

Connected to MySQL

If you load the above PHP script to your webserver and everything works properly, then you should see "Connected to MySQL" displayed when you view the .php page.

The mysql_connect function takes three arguments. Server, username, and password. In our example above these arguments were:

- Server - localhost
- Username - admin
- Password - 1admin

The "or die(mysql..." code displays an error message in your browser if --you've probably guessed it -- there is an error in processing the connection! Double-check your username, password, or server if you receive this error.

choosing the working database

After establishing a MySQL connection with the code above, you then need to choose which database you will be using with this connection. This is done with themysql_select_db function.

```
<?php

mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
```

```
echo "Connected to MySQL<br />";

mysql_select_db("test") or die(mysql_error());

echo "Connected to Database";

?>
```

Display:

Connected to MySQL

Connected to Database

Now simple Connectivity Code for MYSQL DATABASE.

here in given below code localhost is server name and root is username, password is ""(blank).
note: when we upload it on server then we require following:

(1)databse server name.
(2)username.
(3)password.
(4)database name.

```
<?php

$con=mysql_connect("localhost","root","");

mysql_select_db("needa",$con);

if($con)
{
echo  "connection is successful";
}
else
{
    echo "not connected";
}
```

```
?>
```

Insert , update , delete ,search in PHP :

First OF ALL create table details :

```
Create table details
(id int primary key auto_increment ,
name varchar(200),
city varchar(200));
```

((1) first write code for **insert.html** :

```
<form action="insert.php" method="post">
<input type="text" name="id">
<input type="text" name="name">
<input type="text" name="city">
<input name="myBtn" type="submit" value="Submit Data">
</form>
```

(2) write code for **insert.php** :

```
<?php
$id=$_POST['id'];
$nm=$_POST['name'];
$cy=$_POST['city'];
$con=mysql_connect('localhost','root','' );
mysql_select_db('needa',$con);
$sql="insert into details (id,name,city) values('$id','$nm','$cy')";
mysql_query($sql);
echo "record inserted successfully";

?>
```

Now write code for update in php:

(1)write code for **update.html** :

```
<form action="update.php" method="post">
<input type="text" name="id">
<input type="text" name="name">
<input type="text" name="city">
<input name="myBtn" type="submit" value="Submit Data" >
</form>
```

(2)write code for **update.php** file:

```
<?php

$id=$_POST['id'];

$nm=$_POST['name'];

$cy=$_POST['city'];


$con=mysql_connect('localhost','root','' );

mysql_select_db('needa',$con);

$sql="update details set name ='$nm', city='$cy' where id='$id'";

mysql_query($sql);
echo "record updated successfully";
?>
```

Now write code delete in php :

(1)write code for delete.html:

```
<form action="delete.php" method="post">
<input type="text" name="id">
<input name="myBtn" type="submit" value="Submit Data" >
</form>
```

((2) write code for **“delete.php”** file :

```
<?php

$id=$_POST['id'];

$con=mysql_connect('localhost','root','' );

mysql_select_db('needa',$con);

$sql="delete from details where id='$id'";

mysql_query($sql);

echo "record deleted successfully";


?>
```

Now write code for search example :

(1) write code for **search.html** :

```
<form action="search.php" method="post">
<input type="text" name="name">
<input name="myBtn" type="submit" value="Submit Data" >
</form>
```

(2)write code for **search.php** :

```
<?php
$nm=$_POST['name'];
$con=mysql_connect('localhost','root','' );
mysql_select_db('needa',$con);
$sql="select * from details where name='$nm'";
$result=mysql_query($sql);
while($row=mysql_fetch_array($result))
{
echo $row['id'];
echo $row['name'];
echo $row['city'];

}


?>
```