Web Engineering
(CS 603)

Credits: 4 (Lect: 3+ Lab: 2)

By:
Prof. Rajesh K. Ahir
Assistant Professor,

विद्या विनयेन शोभते

Hypertext Pre-processor (PHP)

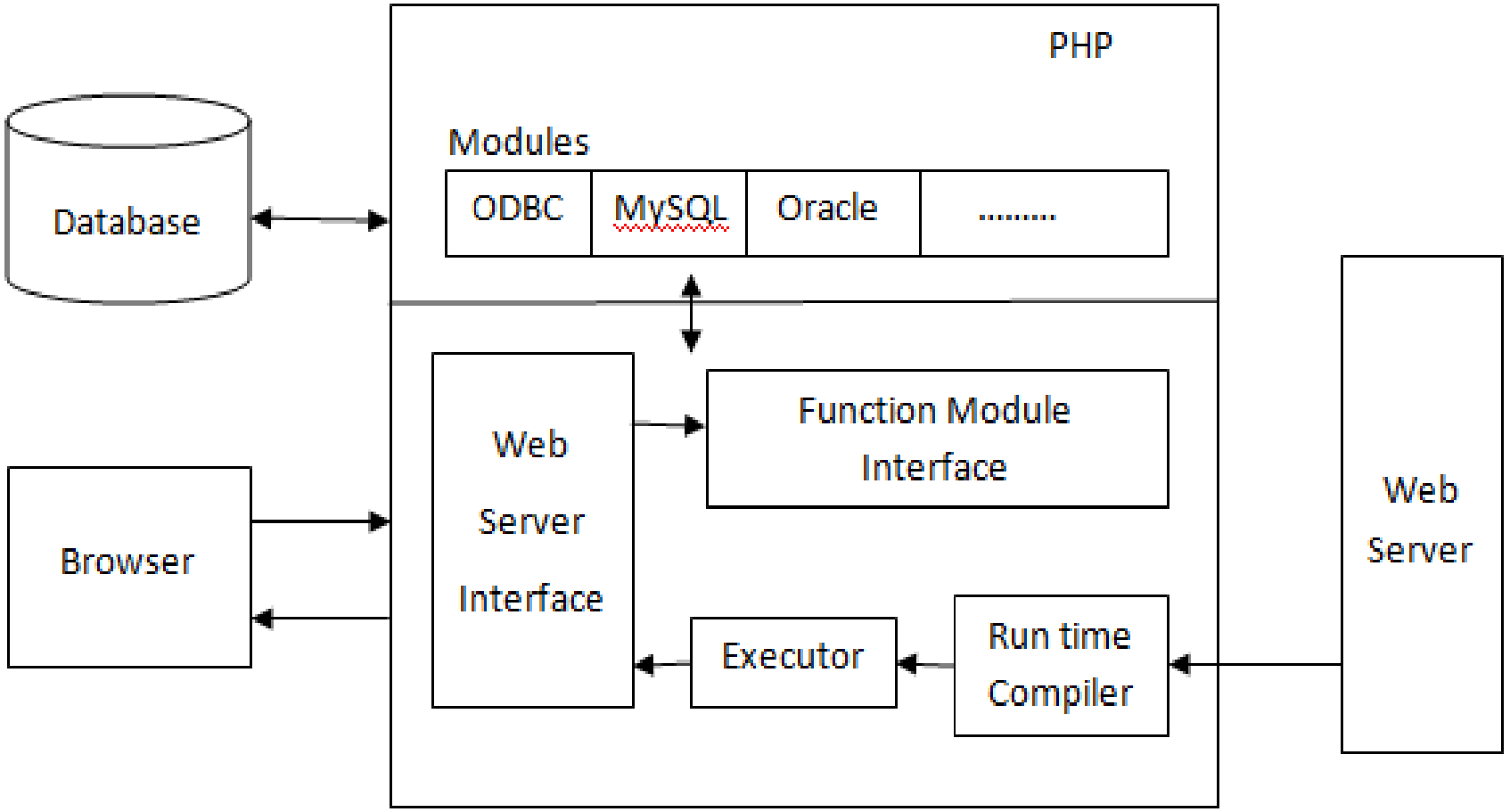
Introduction to PHP

- PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.
- PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.
- **Hypertext pre-processor** - Interpreter of PHP code, processes PHP code invoked by web server and send pure HTML code to browser

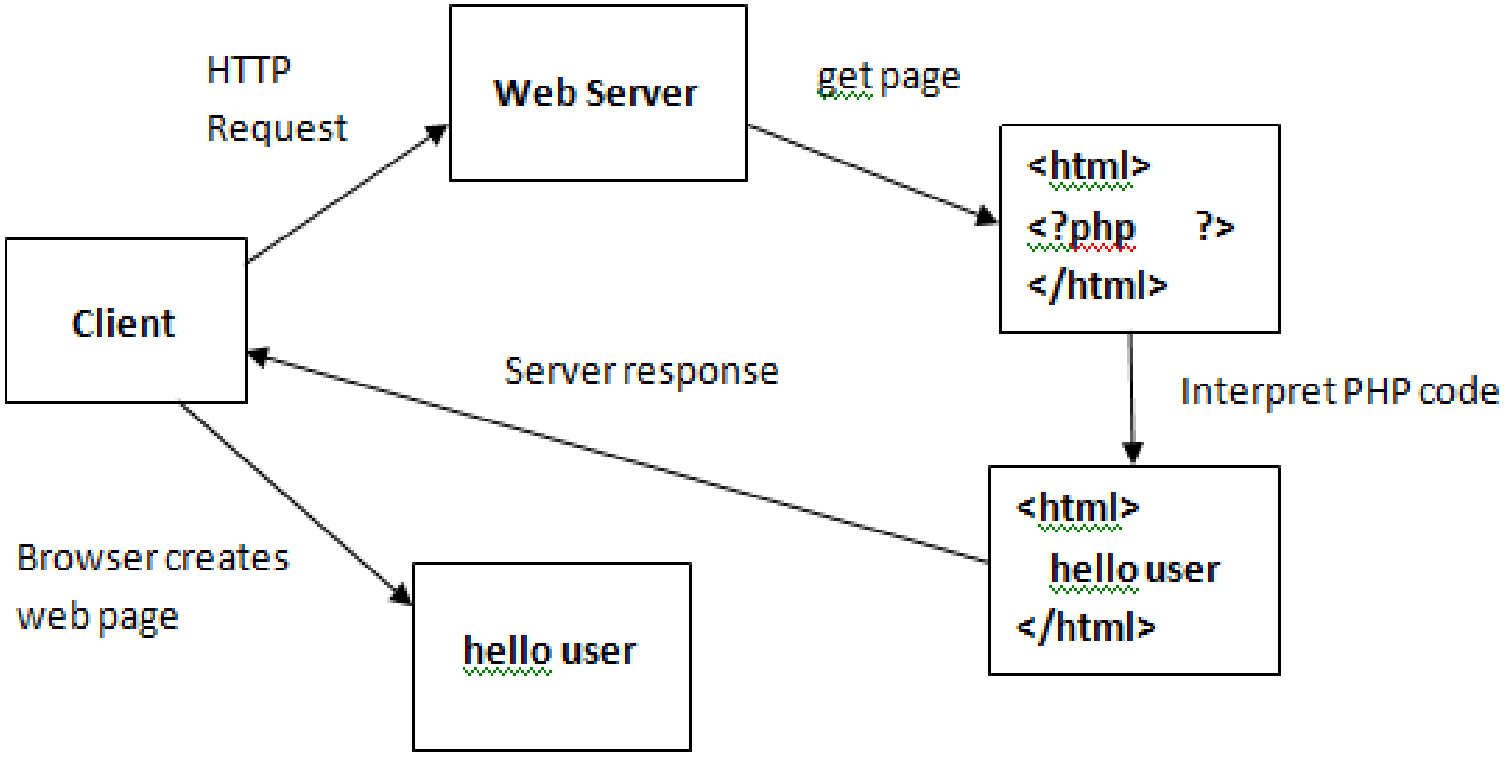
Features of PHP

- Free
- Platform compatibility
- Simple syntax
- Supports many databases
- Broad functionality for file system support, XML, Macromedia Flash etc.
- A vibrant and supportive community

Internal Structure



How PHP script works?



Basic Syntax of PHP

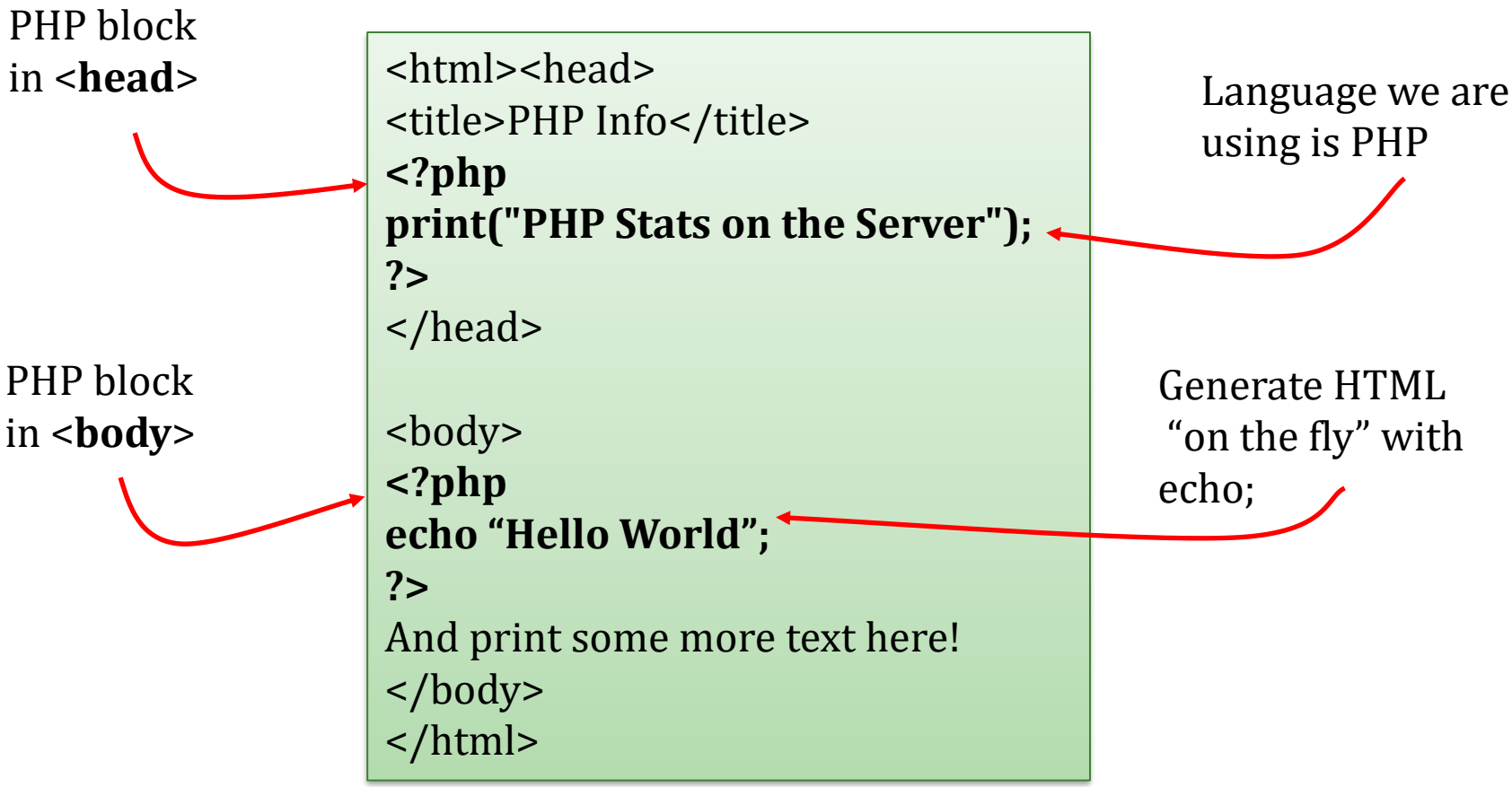
- PHP files may contain text, HTML tags and scripts
- PHP files are returned to the browser as plain HTML
- PHP files have a file extension of **".php"**, **".php3"** or **".phtml".**
- At the end of each PHP statement we need to put a semicolon, i.e. **"; "** which indicate the end of a PHP statement and should never be forgotten.

```
<?php
    //your php code goes here
?>
```

OR

```
<?
    //your php code goes here
?>
```

HTML in a PHP page



What is a MySQL?

- MySQL is a database server.
- MySQL is ideal for both small and large applications.
- MySQL supports standard SQL.
- MySQL is free to download and use.
- WAMP Server: Window Apache MySQL PHP
  - **PHP files** must be stored at **"c:\wamp\www\<websitedirname>"**
- XAMPP Server:
  - PHP files must be stored at **"c:\xampp\htdocs\<websitedirname>"**

PHP Comments

```
<html>
    <body>

        <?php
            //This is a comment
            /* This is a comment block */
        ?>

    </body>
</html>
```

PHP – working with variable

- **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 – working with variable

- **Example:**

```
<?php
        $str="I love PHP";
        echo $str;
        $num = 100;
        echo $num;
        echo $str;
        echo $num; //reuse the $num to print its value again in our code
?>
```

PHP – working with variable

- **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 – working with variable

- **Variables types:**
  - **Functions to check type:**
    - gettype(varname)
    - is_string(varname)
    - is_long(varname)
    - is_double(varname)
    - is_array(varname)
    - is_object(varname)

PHP – working with variable

- **Constant:**
  - A constant is an identifier (name) for a simple value.
  - As the name suggests, that value cannot change during the execution of the script.
  - A constant is case-sensitive by default.
  - By convention, constant identifiers are always uppercase.

```
<?php
    // Valid constant names
                define(“A", "something");
                define(“A2", "something else");
                define(“A_B", "something more");
    // Invalid constant names
                define("2A", "something");
?>
```

PHP –String Manipulation

- **String- Concatenation Operator:**
  - There is only one string operator in PHP.
  - The concatenation operator **(.)** and concatenation with assignment **(.=)** is used to put two string values together.
- **String Functions:**
  - strlen("Hello world!"); // outputs: 12
  - str_word_count("Hello world!"); // outputs: 2
  - strrev("Hello world!"); // outputs: **!dlrow olleH**
  - strpos("Hello world!", "world"); // outputs: **6**
  - str_replace("world", “GCET", "Hello world!"); // outputs: **Hello GCET!**
  - strstr($string1,$string2) // It will find string 2 inside string 1.

PHP –String Manipulation

- **String Functions:**
  - **substr($string1,startpos,endpos):** returns string from either start position to end or the section given by startpos to endpos.
  - **trim($string) :** trim away white spaces including tabs , newlines and space from beginning and end of a string
  - **ltrim($string) :** trim spaces from start of a string only
  - **rtrim($string) :** trim spaces from end of a string only

PHP –String Manipulation

- **String Functions:**
  - **substr_replace($string1, $string2, startpos, endpos):** similar to substr but replaces the substring with string 2 at start through to optional end points .
  - **strtolower($string) :** convert string into lowercase
  - **strtoupper($string) :** convert string into uppercase
  - **ucwords($string) :** converts all first letters in a string to uppercase.
  - **explode($delimiters, $string) :** breaks string up into an array at the points marked by the delimiters.

PHP –Operators, Conditional Statements and Loops

- **Operators:** all operators
- **Conditional Statements:**
  - if
  - elseif
  - switch
- **Loops:**
  - for loop
  - while loop
  - do while loop

PHP –Inbuilt and User Defined Functions

- **Inbuilt Functions:** The real power of PHP comes from its functions; it has more than 1000 built-in functions.
- **User Defined Functions:**
  - A user defined function declaration starts with the word "**function**":
  - Syntax:

function *funName()*

{

code to be executed;

}

PHP –Inbuilt and User Defined Functions

- **User Defined Functions:**
  - **Example:**

```
<html>
    <body>
            <?php
                    function writeMsg()
                    {
                            echo "Hello world!";
                    }
            writeMsg();
            ?>
    </body>
</html>
```

PHP –Inbuilt and User Defined Functions

- **User Defined Functions with arguments:**
  - **Example:**

```
<?php
        function Details($fn, $y)
        {
                echo “Mr. $fn Born in $y <br>";
        }

 Details(“Joy", "1975");
 Details(“Bob", "1978");
 Details("Jim", "1983");
?>
```

PHP –working with Array

- An array is a special variable, which can hold more than one value at a time.
- In PHP, the array() function is used to create an array:

```
$car = array("Volvo", "BMW", "Toyota")

echo $car[1];

echo count($car);
```

- **Types of array:**
  - Indexed array
  - Associative array

PHP –working with Array

- **Indexed array:** The keys of an indexed array are integers, beginning at 0.
  - **Example:**

```
<?php
    $cars[0]=“Ferrari";
    $cars[1]="Volvo";
    $cars[2]="BMW";
    $cars[3]="Toyota";
    echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?> //
```

That's an indexed array, with integer indexes beginning at 0.

O/p : Ferrari and Volvo are Swedish cars.

PHP –working with Array

- **Associative array:** An associative array, each ID key is associated with a value.
  - **Example:**

```
<?php
        $price['Wheel'] = 75.25;
        $price['Tire'] = 50.00;
?>
```

- An easier way to initialize an array is to use the **array( )** construct, which builds an array from its arguments:
  - Use the **=>** symbol to separate indexes from values:
  - $price = array('Gaskit' => 15.29, 'Wheel' => 75.25, 'Tire' => 50.00);

PHP –working with Array

- **Extracting Multiple Values:**
  - To insert more values into the end of an existing indexed array, use the [] syntax:
    1. $college= array('IIIT', 'Surat');
    2. $college[] = 'Hello';

- **Getting the Size of an Array**
  - The count( ) functions return the number of elements in the array.
    1. $college= array('IIIT', 'Surat');
    2. $size = count($college);

PHP –working with Array

- **Adding Values to the End of an Array:**
  - To copy all of an array's values into variables, use the list( ) construct:

    list($variable, ...) = $array;
- **Example:**

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' => 'abc');

list($n, $a, $w) = $person; // $n is 'Fred', $a is 35, $w is 'abc'
```

PHP –working with Array

- **array_count_values() Function:**
  - The array_count_values() function **returns an array**, where the keys are the original array's values, and the values is the number of occurrences.

- **Example**

```
<?php
    $a=array("Cat","Dog","Horse","Dog");
    print_r(array_count_values($a));
?>
```

O/P: **Array ([Cat] => 1 [Dog] => 2 [Horse] => 1)**

PHP –working with Array

- **array()** creates an array, with keys and values. If you skip the keys when you specify an array, an integer key is generated, starting at 0 and increases by 1 for each value.

  array(key => value)

- **Example**

```
<?php
    $a=array("a"=>"Dog","b"=>"Cat","c"=>"Horse");
    print_r($a);
?>
```

  o/p : Array ([a] => Dog [b] => Cat [c] => Horse)

- **Example**

```
<?php
    $a=array("Dog","Cat","Horse");
    print_r($a);
?>
```

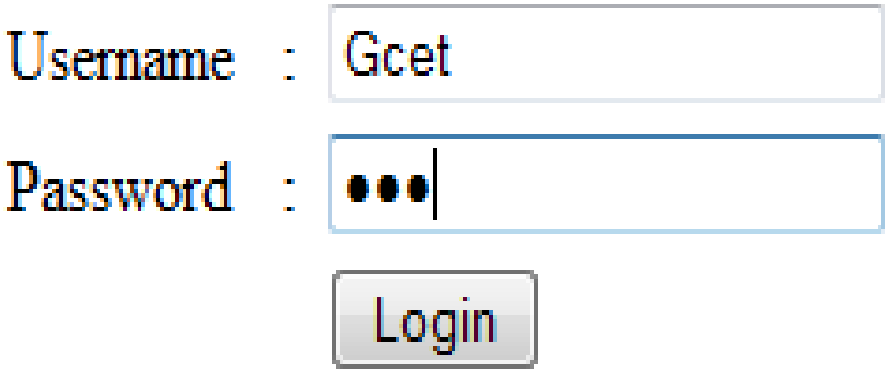
  o/p : Array ([0] => Dog [1] => Cat [2] => Horse

PHP – The $_GET Function

- The built-in **$_GET** function is used to collect values from a form, sent with method="GET".
- When using **method="GET"** in HTML forms, all variable names and values are displayed in the URL.
- Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.
- **Note:** This method should not be used when sending passwords or other sensitive information!
- Because the variables are displayed in the URL.
- **Note:** The GET method is not suitable for very large variable values. It should not be used with values exceeding 2000 characters.

PHP – The $_GET Function

- **Example:**




```
<html >
 <head>
<title>Untitled Document</title>
</head>
<body>
<form name="form1" method="GET" action="display.php" >
Username: <input name="myusername" type="text" id="myusername">
Password: <input name="mypassword" type="password" id="mypassword >
<input type="submit" name="Submit" value="Login">
</form>
</body>
</html>
```

PHP – The $_GET Function

- **Example:** display.php

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

```
<?php
$u= $_GET['myusername'];
$p= $_GET['mypassword'];
echo Username: $u "<br />" Password: $p;
?>
```

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

Output:

Username: Gcet

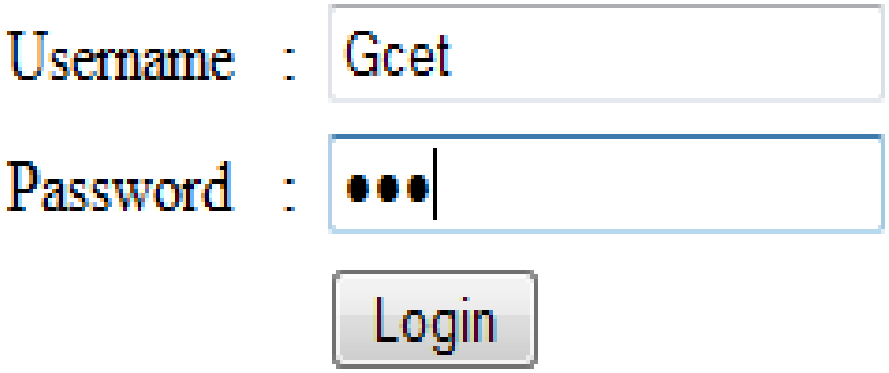
Password: abc

PHP –The $_POST Function

- The built-in **$_POST** function is used to collect values from a form sent with **method="POST".**
- Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.
- **Note:** However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the **post_max_size** in the **php.ini** file).

PHP – The $_POST Function

- **Example:**




```
<html >
 <head>
<title>Untitled Document</title>
</head>
<body>
<form name="form1" method="POST" action="display.php" >
Username: <input name="myusername" type="text" id="myusername">
Password: <input name="mypassword" type="password" id="mypassword >
<input type="submit" name="Submit" value="Login">
</form>
</body>
</html>
```

PHP – The $_POST Function

- **Example:** display.php

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

```
<?php
$u= $_POST['myusername'];
$p= $_POST['mypassword'];
echo Username: $u "<br />" Password: $p;
?>
```

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

Output:

Username: Gcet

Password: abc

PHP –The $_REQUEST Function

- The PHP built-in **$_REQUEST** function contains the contents of both **$_GET, $_POST, and $_COOKIE.**
- The $_REQUEST function can be used to collect form data sent with both the GET and POST methods.

PHP –Date() Function

- **Example:**

```
<?php
    echo date("Y-m-d");
    echo date("Y/m/d");
    echo date("M d, Y");
    echo date("F d, Y");
    echo date("D M d, Y");
    echo date("l F d, Y");
    echo date("l F d, Y, h:i:s");
    echo date("l F d, Y, h:i A");
?>
```

Output:

2011-02-18
2011/02/18
Feb 18, 2011
February 18, 2011
Fri Feb 18, 2011
Friday February 18, 2011
Friday February 18, 2011, 12:49:17
Friday February 18, 2011, 12:49 AM

PHP –Date() Function Exercise

- **Change background color based on day of the week using array.**

```
<html> <head>
<title>Background Colors change based on the day of the week</title>
</head>
<?php
$today = date("w");
$bgcolor = array( "#FEF0C5", "#FFFFFF", "#FBFFC4", "#FFE0DD",
                  "#E6EDFF", "#E9FFE6", "#F0F4F1");
?>
<body bgcolor="<?print("$bgcolor[$today]");?>">
<br>This just changes the color of the screen based on the day of the week
</body>
</html>
```

PHP –include() Function

- **include("filename.html")** - includes and evaluates a specific file; failure results in a Warning.

menu.html

```
<html> <body>
<a href="index.php">Home</a> -
 <a href="about.php">About Us</a> -
<a href="links.php">Links</a> -
<a href="contact.php">Contact Us</a>
 <br />
```

Include.php

```
<?php include("menu.html"); ?>
<p>This is my home page that uses
    a common menu to save my time
    when I add new pages to my
    website!</p>
 </body>
 </html>
```

Home - About us - Links - Contact US

This is my home page that uses a common menu to save my time when I add new pages to my website!

PHP –include_once() Function

- **include_once()** - same as include except if the file has already been included, it will not be included again.
- Use when the same file might be included and evaluated more than once during a particular execution of a script, and you want to be sure that it is included exactly once to avoid problems with function redefinitions, variable value reassignments, etc.
- **Example:**

```
<?php
   include_once ‘header.php’;
?>
```

PHP –include_once() Function

- **include_once() needs:**

GLOBALS.PHP:

```
<?php
    include('FUNCTIONS.PHP');
    foo();
    ?>
```

HEADER.PHP

```
<?php
    include('FUNCTIONS.PHP');
    include('GLOBALS.PHP');
    foo();
?>
```

PHP –require() Function

- **require(“filename.php”)** – includes and evaluates a specific file; failure results in a Fatal Error.
- **Example:**

```
<?php
   require ‘header.php’;
?>
```

PHP –require_once() Function

- **require_once(“filename.php”)** – same as require except if the file has already been included, it will not be included again.
- **Example:**

```
<?php
   require_once ‘header.php’;
?>
```

PHP –Database Connectivity

- PHP 5 and later can work with a MySQL database using:
  - MySQLi extension (the "i" stands for improved)
  - PDO (PHP Data Objects)

PHP –Database Connectivity

- **Both MySQLi and PDO have their advantages:**
  - PDO will work on 12 different database systems, where as MySQLi will only work with MySQL databases.
  - Both are object-oriented, but MySQLi also offers a procedural API.
  - Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

PHP –Database Connectivity

- **Both MySQLi and PDO have their advantages:**
  - PDO_DBLIB (FreeTDS / Microsoft SQL Server / Sybase)
  - PDO_FIREBIRD (Firebird/Interbase 6)
  - PDO_IBM (IBM DB2)
  - PDO_INFORMIX (IBM Informix Dynamic Server)
  - PDO_MYSQL (MySQL 3.x/4.x/5.x)
  - PDO_OCI (Oracle Call Interface)
  - PDO_ODBC (ODBC v3 (IBM DB2, unixODBC and win32 ODBC))
  - PDO_PGSQL (PostgreSQL)
  - PDO_SQLITE (SQLite 3 and SQLite 2)

PHP –Database Connectivity

- **Creating a Database:**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Write a query for creating a database.
  - Step 4: Execute the query using **mysqli_query()** function.
  - Step 5: Close the Connection using **mysqli_close()** function.

PHP –Database Connectivity

- **Creating a Database Example:**

```
<?php
$con = mysqli_connect("localhost","root","");                    //Step 1
if(!$con)                                                        //Step 2
{
    die("Could not connect" . mysqli_connect_error());
}
$qry =  "CREATE DATABASE mydb";                                  //Step 3
$eqry = mysqli_query($con, $qry);                                //Step 4
if($eqry)
{
    echo "Database created";
}
else
{ echo "Error creating Database";}
mysqli_close($con);                                              //Step 5
?>
```

PHP –Database Connectivity

- **Creating a Table:**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Select a database using **mysqli_select_db()** function
  - Step 4: Write a query for creating a database.
  - Step 5: Execute the query using **mysqli_query()** function.
  - Step 6: Close the Connection using **mysqli_close()** function.

PHP –Database Connectivity

- **Creating a Table Example 1:**

```
<?php
$con = mysqli_connect("localhost","root","");                    //Step 1
if(!$con)                                                        //Step 2
{    die("Could not connect" . mysqli_connect_error());    }
mysqli_select_db("mydb");                                        //Step 3
$qry = "CREATE TABLE abc(ID int NOT NULL AUTO_INCREMENT PRIMARY
     KEY(ID), name varchar(10), age int)";                       //Step 4
$eqry = mysqli_query($con, $qry);                                //Step 5
if($eqry)
{    echo "Table created";    }
else
{ echo "Error creating Table";}
mysqli_close($con);                                              //Step 6
?>
```

PHP –Database Connectivity

- **Creating a Table Example 2:**

```
<?php
$con = mysqli_connect("localhost","root","","mydb");                    //Step 1 & 3
if(!$con)                                                               //Step 2
{    die("Could not connect" . mysqli_connect_error());    }
$qry = "CREATE TABLE abc(ID int NOT NULL AUTO_INCREMENT PRIMARY
    KEY(ID), name varchar(10), age int)";                               //Step 4
$eqry = mysqli_query($con, $qry);                                       //Step 5
if($eqry)
{    echo "Table created";    }
else
{ echo "Error creating Table";}
mysqli_close($con);                                                     //Step 6
?>
```

PHP –Database Connectivity

- **Include dbconfig file in PHP page: dbconfig.php**

```
<?php
$dbhost = "localhost";
$dbuser = "root";
$dbpass = "";
$dbname = “myDB";
$con = mysqli_connect($dbhost,$dbuser,$dbpass,$dbname) or die('cannot
    connect to the server');
?>
```

PHP –Database Connectivity

- **Insert the data into Table:**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Write a query for Insert the data into table.
  - Step 5: Execute the query using **mysqli_query()** function.
  - Step 6: Close the Connection using **mysqli_close()** function.

PHP –Database Connectivity

- **Insert the data into Table Example:**

```
<?php
include(“dbconfig.php”);
$qry = “INSERT into abc(ID, name, age) VALUES (1,’X’,21)”;    //Step 3
$eqry = mysqli_query($con, $qry);                              //Step 4
mysqli_close($con);                                            //Step 5
?>
```

PHP –Database Connectivity

- **Insert the data into Table coming from HTML page:**

```
<?php
include("dbconfig.php");
$id=$_POST['idno'];
$n=$_POST['name'];
$a=$_POST['age'];
$qry = "INSERT into abc(ID, name, age) VALUES ('$id', '$n', '$a')";  //Step 3
$eqry = mysqli_query($con, $qry);                                    //Step 4
mysqli_close($con);                                                  //Step 5
?>
```

PHP –Database Connectivity

- **Select and Display the data from table:**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Write a query for Select a data from table..
  - Step 4: Execute the query using **mysqli_query()** function.
  - Step 5: Fetch the data from results of **mysqli_fetch_array()** function.
  - Step 6: Close the Connection using **mysqli_close()** function.

PHP –Database Connectivity

- **Select and Display the data from table :**

```
<?php
include("dbconfig.php");
$qry = "SELECT * FROM abc";                                    //Step 3
$result = mysqli_query($con, $qry);                            //Step 4
echo "<table border=1> <tr><th> ID </th><th> Name</th><th>Age</th></tr>";
while($row = mysqli_fetch_array($result))                      //Step 5
{
    echo "<tr><td>" . $row['ID']. "</td>";
    echo "<td>" . $row['name']. "</td>";
    echo "<td>" . $row['age']. "</td></tr>";
}
echo "</table>";
mysqli_close($con);                                            //Step 6
?>
```

PHP –Database Connectivity

- **Populate the drop down menu with table values:**

```
<?php
include(“dbconfig.php”);
$qry = “SELECT id, sname FROM states”;                    //Step 3
$result = mysqli_query($con, $qry);                       //Step 4
echo “<select name= states>”;
while($row=mysqli_fetch_array($result)){                  //Step 5
echo “<option value=”; echo $row['id']; echo “>”  echo $row[‘sname']; echo
    “</option>”; }
echo “</select>”;
mysqli_close($con);                                       //Step 6
?>
```

PHP –Database Connectivity

- **Update the data:**

```
<?php
include("dbconfig.php");
$qry = "Update states SET sname='$_POST['sname']' WHERE ID='1'";
$result = mysqli_query($con, $qry);
if($result)
{ echo "Updated";}
else
{ echo "Error in Updating"; }
mysqli_close($con);
?>
```

PHP –Database Connectivity

- **Delete the data:**

```
<?php
include("dbconfig.php");
$qry = "delete from abc WHERE sname='$_POST['sn']'";
$result = mysqli_query($con, $qry);
if($result)
{ echo "Deleted";}
else
{ echo "Error in Deleting"; }
mysqli_close($con);
?>
```

PHP –Database Connectivity

- **Display all the table in database "mydb":**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Fetch the list of tables from database using **mysql_list_tables**() function.
  - Step 4: Fetch the number of rows of tables using **mysql_num_rows()** function.
  - Step 5: Take the table name using **mysql_tablename()** function.
  - Step 6: Print all the tables.

PHP –Database Connectivity

- **Display all the table in database “mydb”:**

```
<?php
include(“dbconfig.php”);
$result = mysql_list_tables(“mydb”);                    //Step 3
$rcount = mysql_num_rows($result);                      //Step 4
$tablist = “”;
for($i=0; $i<$rcount; $i++)
{
    $tabname = mysql_tablename($result, $i);            //Step 5
    $tablist .= $tabname.”<br/>”;
}
echo $tablist;                                          //Step 6
?>
```

PHP –Database Connectivity

- **Display all the databases:**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Fetch the list of database using **mysql_list_dbs()** function
  - Step 4: Fetch the number of rows of databases using **mysql_num_rows()** function.
  - Step 5: Take the database name using **mysql_db_name()** function.
  - Step 6: Print all the tables.

PHP –Database Connectivity

- **Display all the databases:**

```
<?php
include(“dbconfig.php”);
$result = mysql_list_dbs($con);                    //Step 3
$rcount = mysql_num_rows($result);                 //Step 4
$dblist = “”;
for($i=0; $i<$rcount; $i++)
{
    $dbname = mysql_db_name($result, $i);          //Step 5
    $dblist .= $dbname.”<br/>”;
}
echo $dblist;                                      //Step 6
?>
```

PHP –Database Connectivity

- **File Upload on Server:**
  - **Step 1:** Configure The "**php.ini**" File
  - **Step 2:** In your "php.ini" file, search for the **file_uploads** directive, and set it to On:

file_uploads = On

upload_max_filesize = 50M

  - **Step 3:** Creating HTML File:

```
<form action="upload.php" method="post" enctype="multipart/form-data">
    Select image to upload:
    <input type="file" name="ufile" id="ufile">
    <input type="submit" value="Upload Image" name="btn_submit">
</form>
```

PHP –Database Connectivity

- **File Upload on Server:**
  - **Step 4: 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 **ufile**, then PHP would create following **five variables**.
    - $_FILES[‘**ufile**’][‘name’], $_FILES[‘**ufile**’][‘temp_name’], $_FILES[‘**ufile**’][‘type’], $_FILES[‘**ufile**’][‘size’], $_FILES[‘**ufile**’][‘error’],

PHP –Database Connectivity

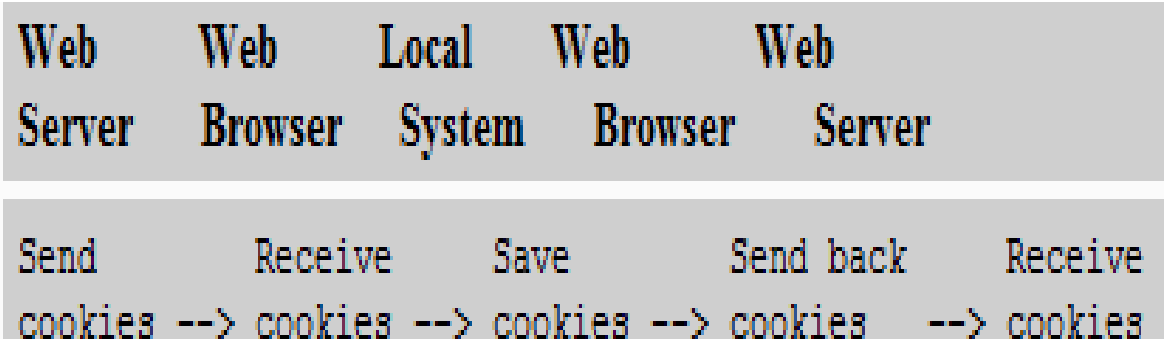
- **File Upload on Server:**
  - **Step 4: Creating an upload script.**
    - $_FILES[‘**ufile**’][‘name’]: This array value specifies the original name of the file, including the file extension. It doesn't include the file path.
    - $_FILES[‘**ufile**’][‘temp_name’]: This array value specifies the temporary name including full path that is assigned to the file once it has been uploaded to the server.
    - $_FILES[‘**ufile**’][‘type’]: The mime type of the file, an example would be *"image/gif".*
    - $_FILES[‘**ufile**’][‘size’]: The size, in bytes, of the uploaded file.
    - $_FILES[‘**ufile**’][‘error’]: The error code associated with the file upload.

PHP –Cookies

- Cookies are small pieces of information about a user that are stored by a Web server in text files on the user's computer.
- The information can really be anything... it can be a name, the number of visits to the site, web based shopping cart information, personal viewing preferences or anything else that can be used to help provide customized content to the user.
- Temporary cookies remain available only for the current browser session
- Persistent cookies remain available beyond the current browser session and are stored in a text file on a client computer

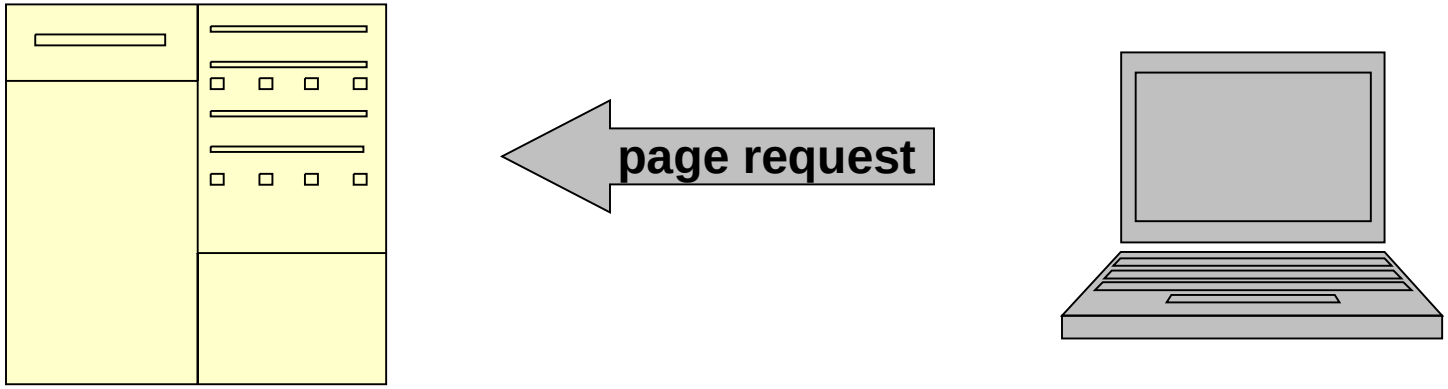
PHP –Cookies

- Each individual server or domain can store only 20 cookies on a user's computer for a single website
- Total cookies per browser cannot exceed 300
- The largest cookie size is 4 kilobytes
- The information is constantly passed in HTTP headers between the browser and web server; the browser sends the current cookie as part of its request to the server and the server sends updates to the data back to the user as part of its response.



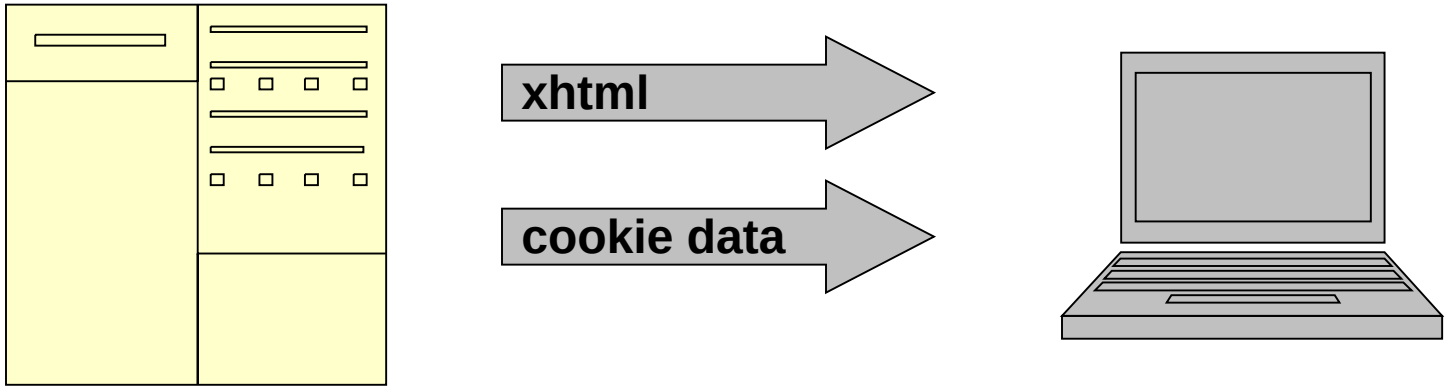
PHP –Cookies

- User sends a request for page at www.example.com for the first time.



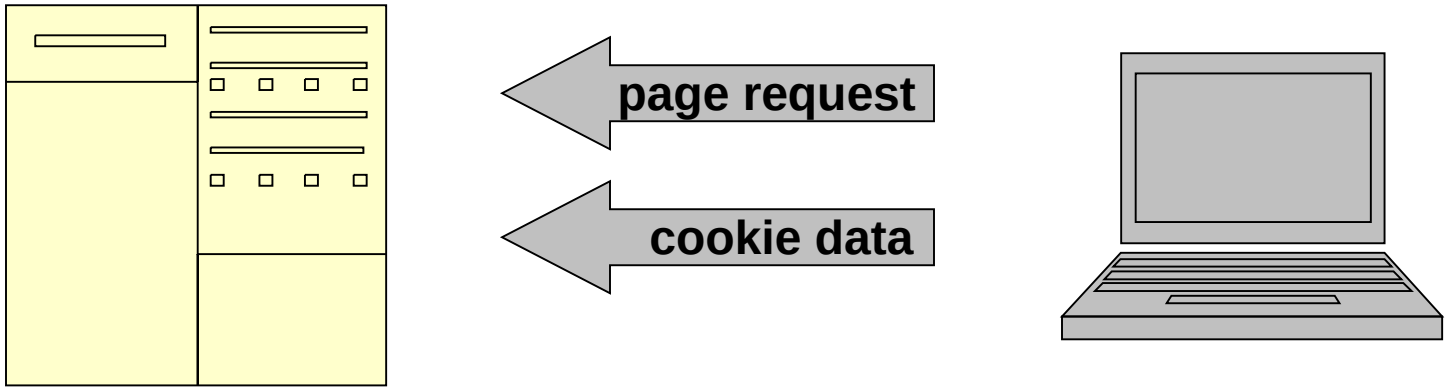
PHP –Cookies

- Server sends back the page xhtml to the browser and stores some data in a cookie on the user's PC.



PHP –Cookies

- At the next page request for domain www.example.com, all cookie data associated with this domain is sent too.



PHP –Cookies

- The syntax for the **setcookie()** function is:

  setcookie(name ,value ,expires,path, domain, secure)

- To skip the value, path, and domain arguments, specify an empty string as the argument value
- To skip the expires and secure arguments, specify 0 as the argument value.

PHP –Cookies

- Call the setcookie() function before sending the Web browser any output, including white space, HTML elements, or output from the echo() or print() statements.
- Cookies can include special characters when created with PHP since encoding converts special characters in a text string to their corresponding hexadecimal ASCII value.
- Cookies created with only the name and value arguments of the setcookie() function are temporary cookies because they are available for only the current browser session.

```
<?php
        setcookie(“firstName”, “abc”);
?>
```

PHP –Cookies

- The expires argument determines how long a cookie can remain on a client system before it is deleted
- Cookies created without an expires argument are available for only the current browser session
- To specify a cookie's expiration time, use PHP's time() function

  setcookie("firstName", "abc", time()+3600);

PHP –Cookies

- The path argument determines the availability of a cookie to other Web pages on a server
- Using the path argument allows cookies to be shared across a server
- A cookie is available to all Web pages in a specified path as well as all subdirectories in the specified path

```
setcookie(“firstName”, “abc”, time()+3600, “/”);
```

PHP –Cookies

- The domain argument is used for sharing cookies across multiple servers in the same domain
- Cookies cannot be shared outside of a domain

```
setcookie(“firstName”, “abc”, time()+3600, “/”, “example.com”);
```

- The secure argument indicates that a cookie can only be transmitted across a secure Internet connection using HTTPS or another security protocol
- To use this argument, assign a value of 1 (for true) or 0 (for false) as the last argument of the setcookie() function

```
setcookie(“firstName”, “abc”, time()+3600, “/”, “example.com”, 1);
```

PHP –Creating Cookies

setcookie(‘age’,’20’,time()+60*60*24*30)

- This command will set the cookie called age on the user’s PC containing the data 20. It will be available to all pages in the same directory or subdirectory of the page that set it (the default path and domain). It will expire and be deleted after 30 days.

PHP –Reading Cookies

- Cookies that are available to the current Web page are automatically assigned to the $_COOKIE autoglobal.
- Access each cookie by using the cookie name as a key in the associative $_COOKIE[] array

```
echo $_COOKIE['firstName'];

$variable = $_COOKIE[‘cookie_name’];

$variable = $HTTP_COOKIE_VARS[‘cookie_name’];
```

PHP –Reading Cookies

- To ensure that a cookie is set before you attempt to use it, use the isset() function

```
setcookie("firstName", "Don");
setcookie("lastName", "Gosselin");
setcookie("occupation", "writer");
if (isset($_COOKIE['firstName']) && isset($_COOKIE['lastName']) &&
    isset($_COOKIE['occupation']))
{
   echo "{$_COOKIE['firstName']} {$_COOKIE['lastName']}
       is a {$_COOKIE['occupation']}.";
}
```

PHP –Reading Cookies

- Use multidimensional array syntax to read each cookie value.

```
setcookie("professional[0]", "Don");
setcookie("professional[1]","Gosselin");
setcookie("professional[2]","writer");
    if (isset($_COOKIE['professional']))
    {
        echo "{$_COOKIE['professional'][0]}
            {$_COOKIE['professional'][1]} is a
             {$_COOKIE['professional'][2]}.";
    }
```

PHP –Deleting Cookies

- By default, the cookies are set to be deleted when the browser is closed. We can override that default by setting a time for the cookie's expiration but there may be occasions when you need to delete a cookie before the user closes his browser, and before its expiration time arrives.
- To delete a persistent cookie before the time assigned to the expires argument elapses, assign a new expiration value that is sometime in the past

PHP –Deleting Cookies

- Do this by subtracting any number of seconds from the time() function.

```
setcookie("firstName", "", time()-3600);

setcookie("lastName", "", time()-3600);

setcookie("occupation", "", time()-3600);
```

PHP –Deleting Cookies

- **There are the following benefits of using cookies for state management:**
  1. No server resources are required as they are stored in client.
  2. They are light weight and simple to use.

- **There are the following limitations of using cookies:**
  1. Some users disable their browser or client device's ability to receive cookies, thereby limiting the use of cookies.
  2. Cookies can be tampered and thus creating a security hole.
  3. Cookies can expire thus leading to inconsistency.

PHP –Session

- 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.
- 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.

PHP –Session

- 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.
- Sessions allow you to maintain state information even when clients disable cookies in their Web browsers.

PHP –Start PHP Session

- A session is started with the **session_start()** function.
- The **session_start()** function starts a new session or continues an existing one.
- The **session_start()** function generates a unique **session ID** to identify the session.
- A session ID is a random alphanumeric string that looks something like:

  7f39d7dd020773f115d753c71290e11f

- The **session_start()** function creates a text file on the Web server that is the same name as the session ID, preceded by **sess_**

PHP –Start PHP Session

- Session ID text files are stored in the Web server directory specified by the **session.save_path** directive in your **php.ini** configuration file.
- The **session_start()** function does not accept any functions, nor does it return a value that you can use in your script.

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

- The **session_start()** function must be the very first thing in your document. Before any HTML tags.

PHP –Start PHP Session

- If a client's Web browser is configured to accept cookies, the session ID is assigned to a temporary cookie named PHPSESSID.
- Pass the session ID as a query string or hidden form field to any Web pages that are called as part of the current session.

```
<?php
        session_start();
?>
<a href='<?php echo "sentmsg.php?PHPSESSID=".session_id() ?>'>Sent Message</a></p>
```

PHP –Working with Session Variables

- Session state information is stored in the **$_SESSION** autoglobal.
- When the session_start() function is called, PHP either initializes a new $_SESSION autoglobal or retrieves any variables for the current session (based on the session ID) into the $_SESSION autoglobal.

```
$_SESSION['uname'] = $uname;
$_SESSION['age'] = $age;
```

PHP –Working with Session Variables

- Data is simply read back from the $_SESSION autoglobal array.

  e.g.

```
$un  = $_SESSION['uname'];
$a = $_SESSION['age'];
```

PHP –Session Example

Main.php:

```
<?php
        session_start();
?>
<html>
<body>
<?php
        $un=$_REQUEST[‘username’];
        // Set session variables
        $_SESSION[“uname"] = $un;
        echo "Session variables are set.";
?>
</body></html>
```

PHP –Session Example cont…

Next.php:

```
<?php
        session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
        // Echo session variables that were set on previous page
        echo “Current User is  . $_SESSION[‘uname’] ";
?>
</body> </html>
```

PHP –Session Example cont…

- Use the **isset()** function to ensure that a session variable is set before you attempt to use it

```
<?php
session_start();
if (isset($_SESSION[‘uname']) )
{
        echo $_SESSION[‘uname'];
}
?>
```

PHP –Destroying the Session

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

```
<?php
session_start();
?>
<html><body>
<?php
        // remove all session variables
                session_unset();
        // destroy the session
                session_destroy();
?> </body></html>
```

PHP –Cookie Vs Session

Cookies	Sessions
Limited storage space	Practically unlimited space
Insecure storage client-side	Reasonably securely stored server-side
User controlled	No user control

PHP –Browser Control

- Php can control various features of a browser
- It can reload a page or redirect page to another page.
- Header function is used to redirect page to another page.

header('Location: index.html');

PHP –Browser Control

- The Browser that the server is dealing with can be identified using

  $browser_ID = $_SERVER[‘HTTP_USER_AGENT’];

  $Ipaddress = $_SERVER[‘REMOTE_ADDR’]

- $_SERVER is a global array with useful information stored in it about server’s current status.
- HTTP_USER_AGENT is an environment variable in the table that contains the information about browser like browser name , version , on which OS browser run.