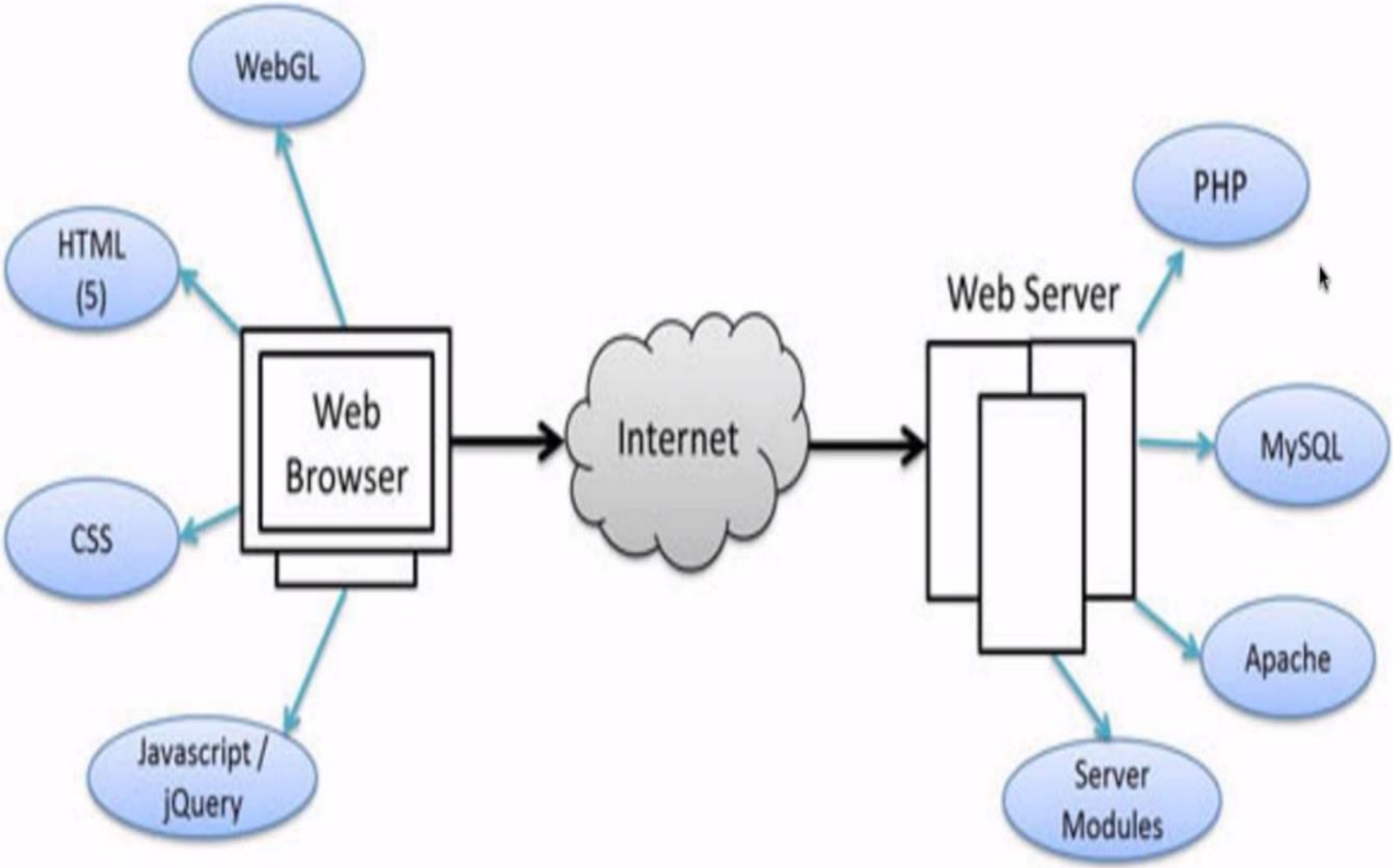
Unit 5:
PHP Basics

Server-Side Programming With PHP
(Hypertext Pre-processor)


WebGL
HTML
(5)
CSS
Javascript /
jQuery
Web
Browser
Internet
Web Server
PHP
MySQL
Apache
Server
Modules

Difference between Client-side Scripting and Server-side Scripting

Client side scripting	Server side scripting
Source code is visible to user.	Source code is not visible to user because it's output of server side is a HTML page.
It usually depends on browser and it's version.	In this any server side technology can be use and it does not depend on client.
It runs on user's computer.	It runs on web server.
There are many advantages link with this like faster. response times, a more interactive application.	The primary advantage is it's ability to highly customize, response requirements, access rights based on user.
It does not provide security for data.	It provides more security for data.
It is a technique use in web development in which scripts runs on clients browser.	It is a technique that uses scripts on web server to produce a response that is customized for each clients request.
HTML, CSS and javascript are used.	PHP, Python, Java, Ruby are used.

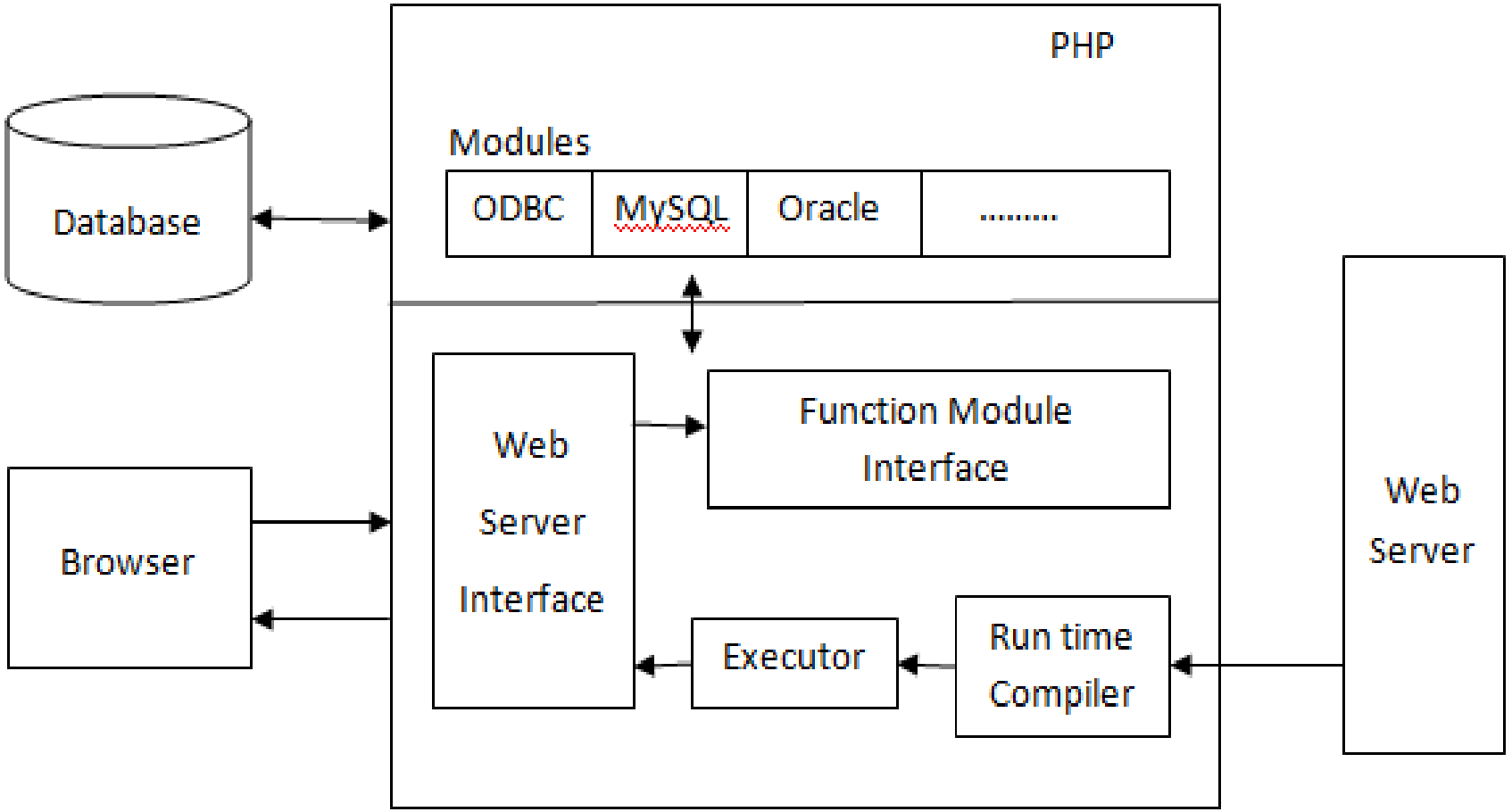
Introduction to PHP

- Rasmus Lerdorf unleashed the first version of PHP way back in 1994
- 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

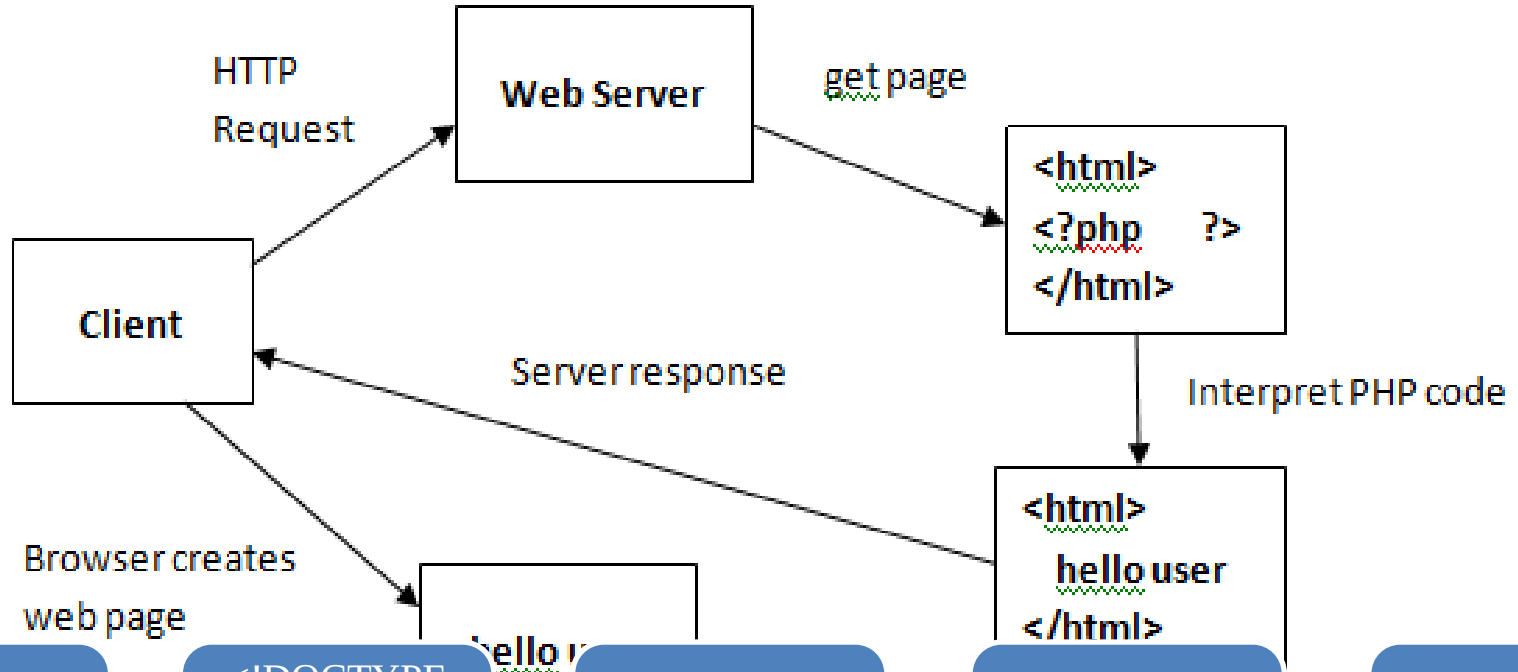
Features of PHP

- 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?



Web server retrieves the file from file system

<!DOCTYPE html>
<html>
<body>
<?php
echo "My first PHP script!";
?>
</body>
</html>

Parsed by PHP engine

<!DOCTYPE html>
<html>
<body>
My first PHP script!
</body>
</html>

Browser will receive the response through HTTP response message and page is rendered.

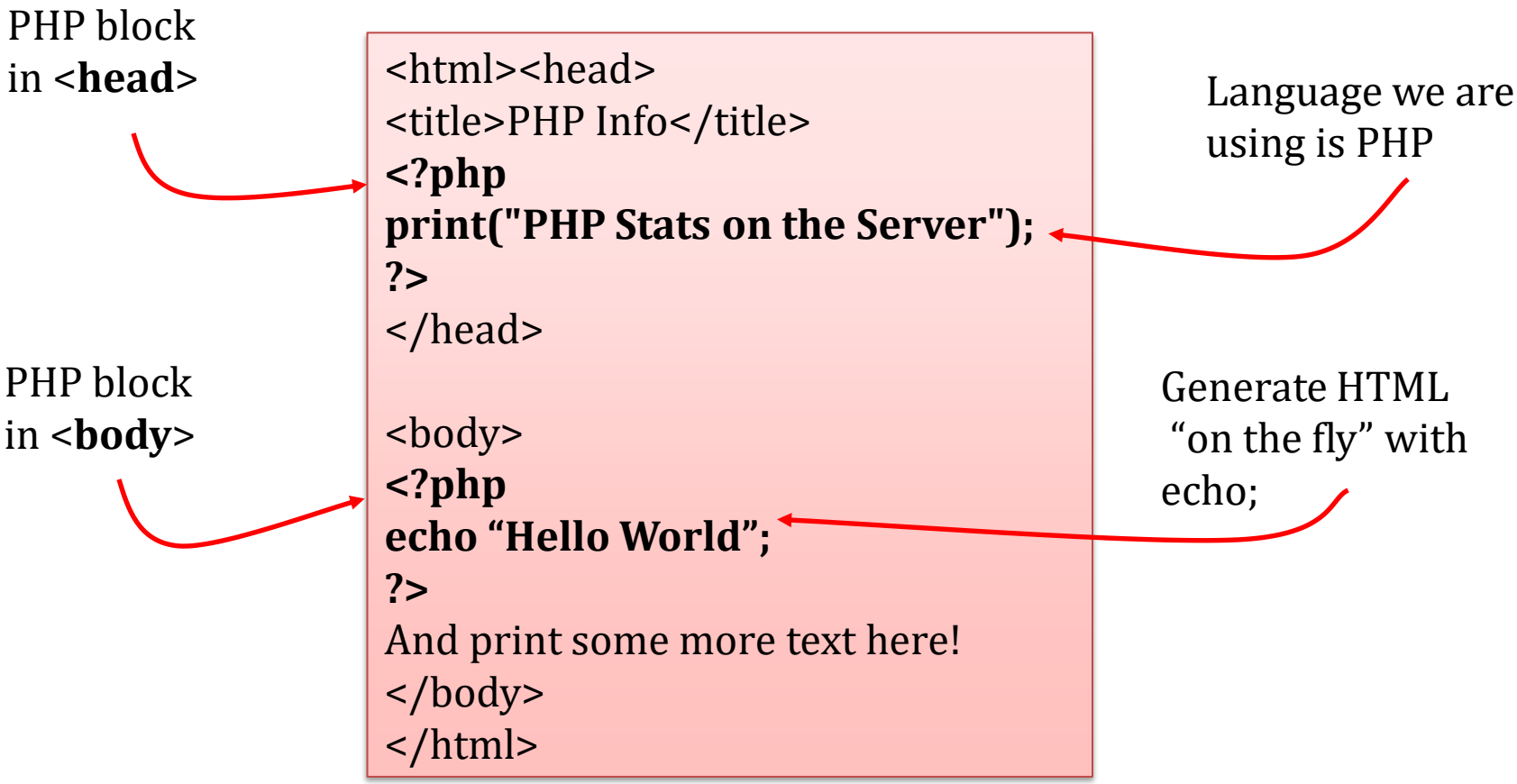
Basic Syntax of PHP

- 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 single line comment
                # This is single line 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

- **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['Tyre'] = 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, 'Tyre' => 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('GCET', 'BVM');
2. $college[] = 'ADIT';
```

- **Getting the Size of an Array**
  - The count( ) functions return the number of elements in the array.

```
1. $college= array('GCET', 'BVM');
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);
?>
```

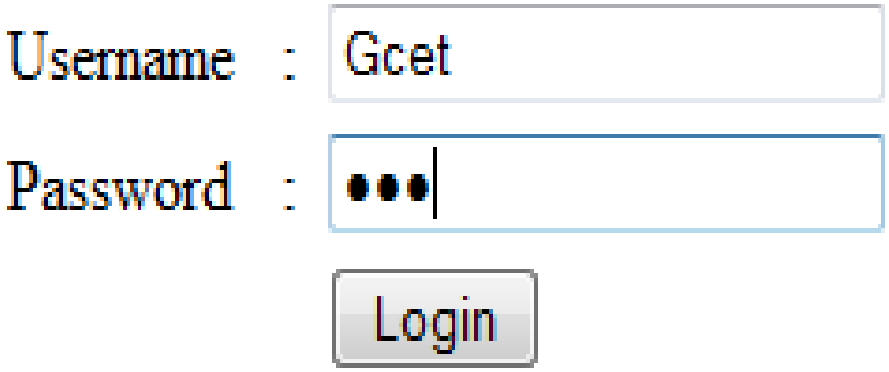
o/p : Array ([0] => Dog [1] => Cat [2] => Horse

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

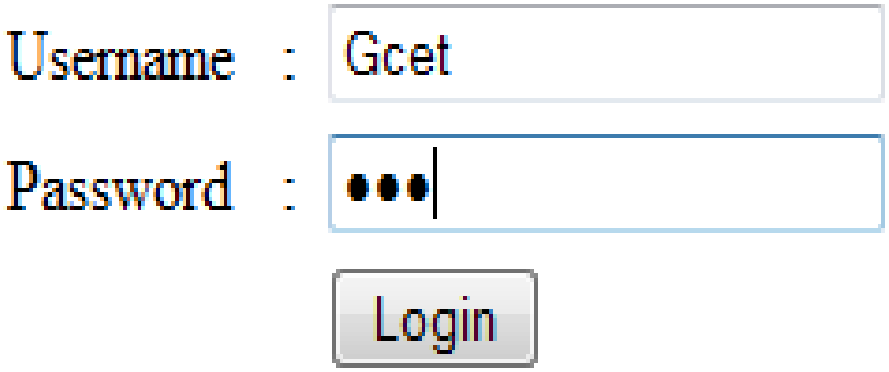
Password: abc

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:

2022-09-27
2022/09/27
Sep 27, 2022
September 27, 2022
Tue Sep 27, 2022
Tuesday September 27, 2022
Tuesday September 27, 2022, 14:55:17
Tuesday September 27, 2022, 14:55 PM

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 Form Processing

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

form1.html

```
<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>
```

PHP Form Processing

- 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.

welcome.php

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST")
{
   $name = $_POST['name'];
   if (empty($name)) {
      echo "Name is empty";
   } else {
      echo "Your Name is:"." ".$name;
   }
   echo "<br><br>";
   $email = $_POST['email'];
   if (empty($email)) {
      echo “E-mail is empty";
   } else {
      echo "Your E-mail is:"." ".$email;     } } ?>
```

PHP Form Processing

- PHP - A Simple HTML Form with GET method.

form1_get.html

```
<!DOCTYPE HTML>
<html>
   <head>
      <title>PHP Form Processing</title>
   </head>
   <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>
```

PHP Form Processing

- PHP - A Simple HTML Form with GET method.

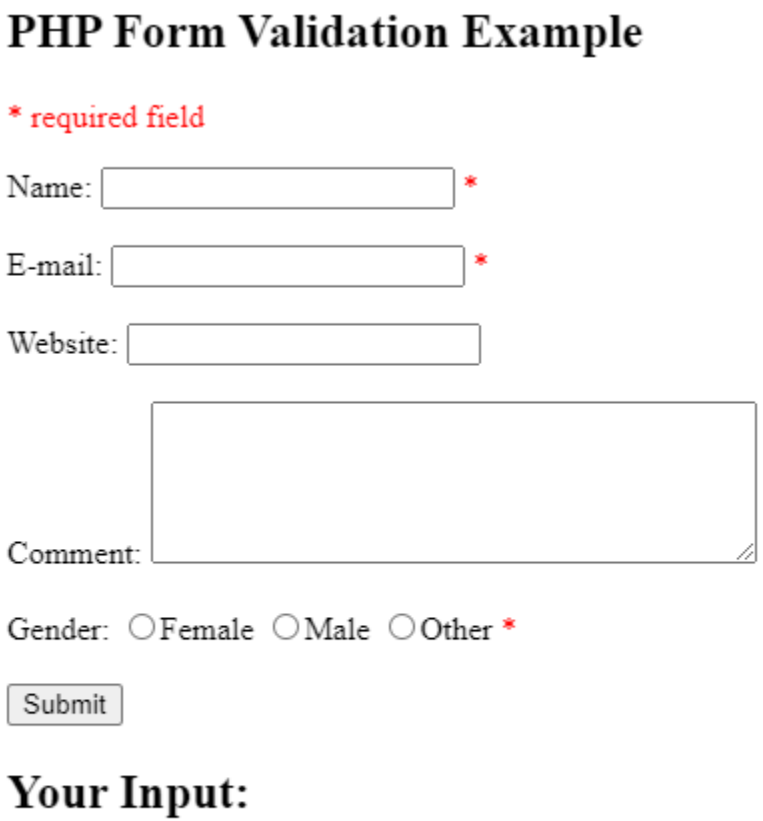
welcome_get.php

```
<html>
<body>

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

</body>
</html>
```

PHP Form Processing

PHP Form Validation

- Think SECURITY when processing PHP forms!
  - These 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!
  - Example



PHP Form Processing

PHP Form Validation

- The validation rules for the form above are as follows

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

PHP Form Processing

PHP Form Validation

- The Form Element
- The HTML code of the form looks like this:
- <form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
- When the form is submitted, the form data is sent with method="post".
- The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.
- So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

PHP Form Processing

PHP Form Validation

- What is the htmlspecialchars() function?
- The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

PHP Form Processing

PHP Form Validation

Note on PHP Form Security

- The $_SERVER["PHP_SELF"] variable can be used by hackers!
- If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

"Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users."

- Assume we have the following form in a page named "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

PHP Form Processing

PHP Form Validation

Big Note on PHP Form Security

- Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

  <form method="post" action="test_form.php"> So far, so good.

- However, consider that a user enters the following URL in the address bar: http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

- In this case, the above code will be translated to: <form method="post" action="test_form.php/"><script>alert('hacked')</script>

PHP Form Processing

PHP Form Validation

Big Note on PHP Form Security

- This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.
- Be aware of that **any JavaScript code can be added inside the <script> tag!** A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

PHP Form Processing

PHP Form Validation

Big Note on PHP Form Security

- How To Avoid $_SERVER["PHP_SELF"] Exploits?
- $_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.
- The form code should look like this:
  <form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
- The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:
  <form method="post" action="test_form.php/"><script>alert('hacked')</script>">
- The exploit attempt fails, and no harm is done!

PHP Form Processing

PHP Form Validation

Validate Form Data With PHP

- The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.
- When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

  <script>location.href('http://www.hacked.com')</script>

- this would not be executed, because it would be saved as HTML escaped code, like this:

  <script>location.href('http://www.hacked.com')</script>

- The code is now safe to be displayed on a page or inside an e-mail.

PHP Form Processing

PHP Form Validation

Validate Form Data With PHP

- We will also do two more things when the user submits the form:
- Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
- Remove backslashes (\) from the user input data (with the PHP stripslashes() function)
- The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).
- We will name the function test_input().
- Now, we can check each $_POST variable with the test_input() function

Example - form_validation.php

PHP Form Processing

PHP Form Validation

Validate Form Data With PHP

- Notice that at the start of the script, we check whether the form has been submitted using $_SERVER["REQUEST_METHOD"]. If the REQUEST_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.
- However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

PHP Form Processing

PHP Form Validation

PHP - Required Fields

- From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.
- In the code (form_validation_req_error.php) we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields.
- We have also added an if else statement for each $_POST variable. This checks if the $_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function:

PHP Form Processing

PHP Form Validation

PHP - Required Fields

- The next step is to validate the input data, that is "Does the Name field contain only letters and whitespace?", and "Does the E-mail field contain a valid e-mail address syntax?", and if filled out, "Does the Website field contain a valid URL?".

PHP - Validate Name

- The code below shows a simple way to check if the name field only contains letters, dashes, apostrophes and whitespaces. If the value of the name field is not valid, then store an error message:
  - $name = test_input($_POST["name"]);
  - if (!preg_match("/^[a-zA-Z-']*$/",$name)) {
  - $nameErr = "Only letters and white space allowed"; }
- The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP Form Processing

PHP Form Validation

PHP - Required Fields

- The next step is to validate the E-mail, "Does the E-mail field contain a valid e-mail address syntax?".

PHP - Validate E-mail

- The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function.
- In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

PHP Form Processing

PHP Form Validation

PHP - Required Fields

- The next step is to validate the website data, that is if filled out, "Does the Website field contain a valid URL?".

PHP - Validate URL

- The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if   (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

PHP Date and Time

The PHP Date() Function

- The PHP date() function formats a timestamp to a more readable date and time.
- Syntax

  date(format, timestamp)

Parameter	Description
format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

- A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

PHP Date and Time

The PHP Date() Function

Get a Date

- The required format parameter of the date() function specifies how to format the date (or time).
- Here are some characters that are commonly used for dates:
  - d - Represents the day of the month (01 to 31)
  - m - Represents a month (01 to 12)
  - Y - Represents a year (in four digits)
  - l (lowercase 'L') - Represents the day of the week
- Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

PHP Date and Time

The PHP Date() Function

PHP Tip - Automatic Copyright Year

- Use the date() function to automatically update the copyright year on your website:
- <!DOCTYPE html>
- <html>
- <body>
- © 2010-<?php echo date("Y");?>
- </body>
- </html>

PHP Date and Time

The PHP Date() Function

Get a Time

- Here are some characters that are commonly used for times:
  - H - 24-hour format of an hour (00 to 23)
  - h - 12-hour format of an hour with leading zeros (01 to 12)
  - i - Minutes with leading zeros (00 to 59)
  - s - Seconds with leading zeros (00 to 59)
  - a - Lowercase Ante meridiem and Post meridiem (am or pm)
- The example (time.php) outputs the current time in the specified format:
- Note that the PHP date() function will return the current date/time of the server!

PHP File Upload

- With PHP, it is easy to upload files to the server.
- However, with ease comes danger, so always be careful when allowing file uploads!
- Configure The "php.ini" File
- First, ensure that PHP is configured to allow file uploads.
- In your "php.ini" file, search for the file_uploads directive, and set it to On:

  file_uploads = On

PHP File Upload

Create The HTML Form

- Next, create an HTML form that allow users to choose the image file they want to upload:

```
<!DOCTYPE html>
<html>
  <body>
    <form action="file_upload.php" method="post" enctype="multipart/form-data">
      Select image to upload:
      <input type="file" name="fileToUpload" id="fileToUpload">
      <input type="submit" value="Upload Image" name="submit">
    </form>
  </body>
</html>
```

PHP File Upload

Create The HTML Form

- Some rules to follow for the HTML form above:
  - Make sure that the form uses method="post"
  - The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form
- Without the requirements above, the file upload will not work.
- Other things to notice:
  - The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control
- The form above sends data to a file called “file_upload.php",

PHP File Upload

Create The Upload File PHP Script

- The “file_upload.php" file contains the code for uploading a file:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
  $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
  if($check !== false) {
    echo "File is an image - " . $check["mime"] . ".";
    $uploadOk = 1;   }
 else {     echo "File is not an image.";     $uploadOk = 0;  } } ?>
```

PHP File Upload

Check if File Already Exists

- Now we can add some restrictions.
- First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and $uploadOk is set to 0:
  - // Check if file already exists
  - if (file_exists($target_file)) {
  - echo "Sorry, file already exists.";
  - $uploadOk = 0;
  - }

PHP File Upload

Limit File Size

- The file input field in our HTML form above is named "fileToUpload".
- Now, we want to check the size of the file. If the file is larger than 500KB, an error message is displayed, and $uploadOk is set to 0:

```
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
  echo "Sorry, your file is too large.";
  $uploadOk = 0;
}
```

PHP File Upload

Limit File Type

- The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting $uploadOk to 0:

```
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg“  && $imageFileType != "gif" )
{
  echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
  $uploadOk = 0;
}
```

PHP Working with Regular Expressions

What is a Regular Expression?

- A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for. A regular expression can be a single character, or a more complicated pattern. Regular expressions can be used to perform all types of text search and text replace operations.

Syntax

- In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers. $exp = "/GCET/i";
- / is the delimiter, GCET is the pattern that is being searched for, and i is a modifier that makes the search case-insensitive. The delimiter can be any character that is not a letter, number, backslash or space. The most common delimiter is the forward slash (/), but when your pattern contains forward slashes it is convenient to choose other delimiters such as # or ~.

PHP Working with Regular Expressions

Regular Expression Functions

- PHP provides a variety of functions that allow you to use regular expressions. The preg_match(), preg_match_all() and preg_replace() functions are some of the most commonly used ones:

Function	Description
preg_match()	Returns 1 if the pattern was found in the string and 0 if not
preg_match_all()	Returns the number of times the pattern was found in the string, which may also be 0
preg_replace()	Returns a new string where matched patterns have been replaced with another string

PHP Working with Regular Expressions

Regular Expression Functions

Using preg_match()

- The preg_match() function will tell you whether a string contains matches of a pattern.

```
<?php
$str = "Visit gcet.ac.in ";
$pattern = "/gcet.ac.in/i";
echo preg_match($pattern, $str); // Outputs 1
?>
```

PHP Working with Regular Expressions

Regular Expression Functions

Using preg_match_all()

- The preg_match_all() function will tell you how many matches were found for a pattern in a string.

```
<?php
  $str = "CE department in GCET";
  $pattern = "/CE/i";
  echo preg_match_all($pattern, $str);
?>
```

PHP Working with Regular Expressions

Regular Expression Functions

Using preg_replace()

- The preg_replace() function will replace all of the matches of the pattern in a string with another string.
- Use a case-insensitive regular expression to replace Microsoft with W3Schools in a string

```
<?php
$str = "Visit GCET!";
$pattern = "/gcet/i";
echo preg_replace($pattern, “CE GCET", $str); // Outputs "Visit CE GCET"
?>
```

PHP Working with Regular Expressions

Regular Expression Modifiers

- Modifiers can change how a search is performed.

Modifier	Description
i	Performs a case-insensitive search
m	Performs a multiline search (patterns that search for the beginning or end of a string will match the beginning or end of each line)
u	Enables correct matching of UTF-8 encoded patterns

PHP Working with Regular Expressions

Regular Expression Patterns

- Brackets are used to find a range of characters:

Expression	Description
[abc]	Find one character from the options between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find one character from the range 0 to 9

PHP Working with Regular Expressions

Quantifiers

- Quantifiers define quantities:

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{x}	Matches any string that contains a sequence of *X* *n*'s
n{x,y}	Matches any string that contains a sequence of X to Y *n*'s
n{x,}	Matches any string that contains a sequence of at least X *n*'s

- If your expression needs to search for one of the special characters you can use a backslash (\ ) to escape them. For example, to search for one or more question marks you can use the following expression: $pattern = '/\?+/';

PHP Working with Regular Expressions

Grouping

- You can use parentheses ( ) to apply quantifiers to entire patterns. They also can be used to select parts of the pattern to be used as a match.
- Example
- Use grouping to search for the word "banana" by looking for ba followed by two instances of na:

```
<?php
$str = "Apples and bananas.";
$pattern = "/ba(na){2}/i";
echo preg_match($pattern, $str); // Outputs 1
?>
```

PHP Exception Handling

- An exception is unexpected program result that can be handled by the program itself. Exception Handling in PHP is almost similar to exception handling in all programming languages.
- PHP provides following specialized keywords for this purpose.
  - **try**: It represent block of code in which exception can arise.
  - **catch**: It represent block of code that will be executed when a particular exception has been thrown.
  - **throw**: It is used to throw an exception. It is also used to list the exceptions that a function throws, but doesn't handle itself.
  - **finally**: It is used in place of catch block or after catch block basically it is put for cleanup activity in PHP code.

PHP Exception Handling

- Why Exception Handling in PHP ?
- Following are the main advantages of exception handling over error handling
  - **Separation of error handling code from normal code**: In traditional error handling code there is always if else block to handle errors. These conditions and code to handle errors got mixed so that becomes unreadable. With try Catch block code becomes readable.
  - **Grouping of error types**: In PHP both basic types and objects can be thrown as exception. It can create a hierarchy of exception objects, group exceptions in namespaces or classes, categorize them according to types.

PHP Exception Handling

- PHP Exception Object
- Exceptions are used by functions and methods to send information about errors and unexpected behavior.
- The Exception object has no public properties, but it has private and protected properties which can be written to or read from using the constructor and methods.
- The Exception object has the following methods:

PHP Exception Handling

Method	Description
Exception()	The contructor of the Exception object
getCode()	Returns the exception code
getFile()	Returns the full path of the file in which the exception was thrown
getMessage()	Returns a string describing why the exception was thrown
getLine()	Returns the line number of the line of code which threw the exception
getPrevious()	If this exception was triggered by another one, this method returns the previous exception. If not, then it returns *null*
getTrace()	Returns an array with information about all of the functions that were running at the time the exception was thrown
getTraceAsString()	Returns the same information as getTrace(), but in a string

PHP Exception Handling

Exception()

- The Exception() constructor is used to create an Exception object and set some of its properties.

Syntax: new Exception(message, code, previous)

Parameter	Description
message	Optional. A string describing why the exception was thrown
code	Optional. An integer that can be used used to easily distinguish this exception from others of the same type
previous	Optional. If this exception was thrown in a catch block of another exception, it is recommended to pass that exception into this parameter

PHP Exception Handling

- The getCode() method returns an integer which can be used to identify the exception.
- The getFile() method returns the absolute path to the file where an exception occurred.
- The getMessage() method returns a description of the error or behaviour that caused the exception to be thrown.
- The getLine() method returns the line number of the line of code which threw the exception.
- If the exception was triggered by another one, then the getPrevious() method returns the other exception. Otherwise it returns null.
- The getTrace() method returns a stack trace in the form of an array.

  Stack traces contain information about all of the functions that are running at a given moment. The stack trace provided by this method has information about the stack at the time that the exception was thrown.
- The getTraceAsString() method returns a stack trace in the form of a string.

Programs

PHP State Management

- HTTP is a **stateless** protocol which means every user request is processed independently and it has nothing to do with the requests processed before it. Hence there is no way to store or send any user specific details using HTTP protocol.
- But in modern applications, user accounts are created, and user specific information is shown to different users, for which we need to have knowledge about who the user(or what he/she wants to see etc) is on every webpage.
- PHP provides for two different techniques for state management of your web application, they are:
  - **Server-Side State Management**
  - **Client-Side Server Management**

PHP State Management

Server-Side State Management

- In **server-side state management** we store **user specific** information required to **identify** the user on the **server**. And this information is available on **every webpage**.

- In PHP we have **Sessions** for server-side state management. PHP session variable is used to store user session information like username, userid etc and the same can be retrieved by accessing the session variable on any webpage of the web application until the session variable is destroyed.

PHP State Management

Client-Side State Management

- In **client-side state management** the user specific information is **stored** at the **client side** i.e., in the **browser**. Again, this information is available on all the webpages of the web application.

- In PHP we have **Cookies** for client-side state management. Cookies are saved in the browser with some data and expiry date(till when the cookie is valid).

- One drawback of using cookie for state management is the user can easily access the cookie stored in their browser and can even delete it.

PHP State management using query string

Query string

- The information can be sent across the web pages. This information is called **query string**.
- This query string can be passed from one page to another by appending it to the address of the page. You can **pass more than one query string** by **inserting** the **&** sign **between** the **query strings**.
- A query string can contain **two** things: the query **string ID** and its **value**. The query string passed across the web pages is stored in $_REQUEST, $_GET, or $_POST variable.

PHP State management using query string

Query string

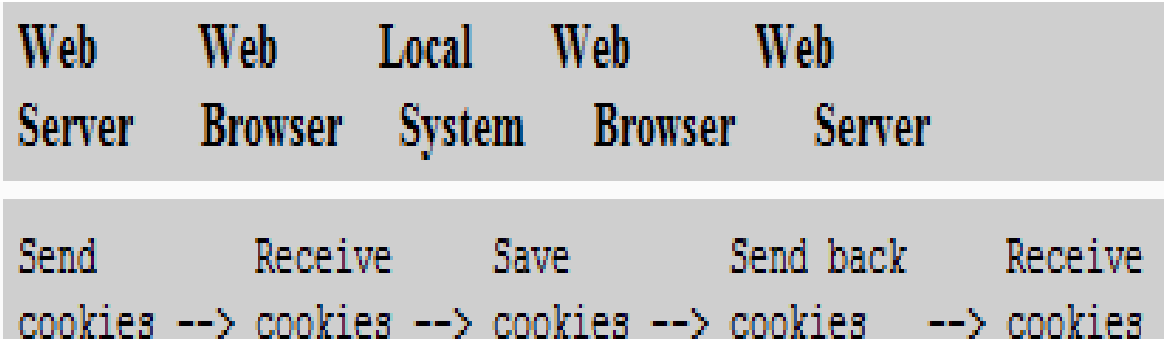
- If we want to access the query string, we can use these variables.
- We should note that whether the query string is passed by the GET or POST method it can be accessed by using the $_REQUEST variable.
- If We want to use $_GET variable to access the query string the form method need to be GET. Also, we can use $_POST variable to get the query string if the form method is POST.
- In the example (Querylogin.php), the query strings username and email(the names of texboxes) are passed from a page called Querylogin.php to another page called Querywelcome.php when you click the submit button.

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

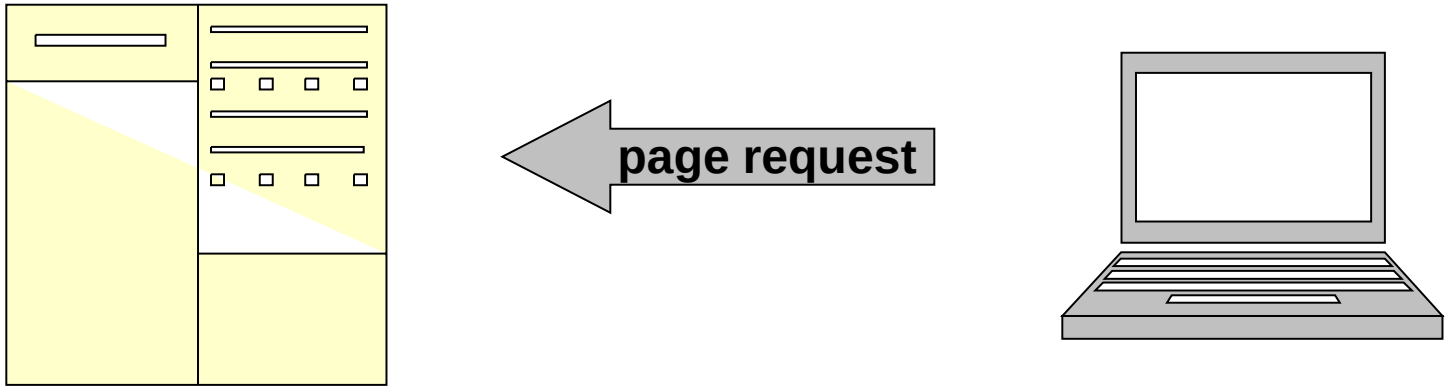
PHP –Cookies

- 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.



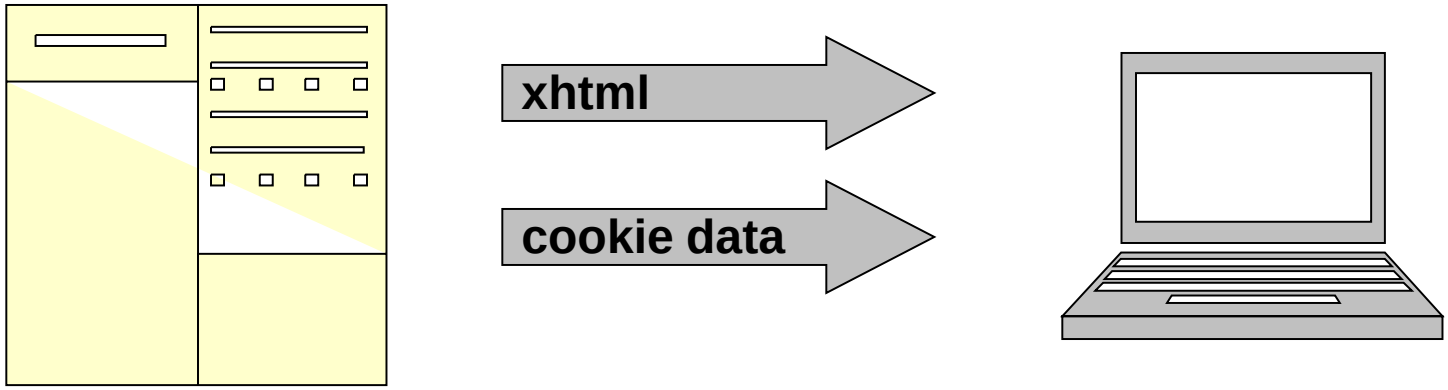
PHP –Cookies

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



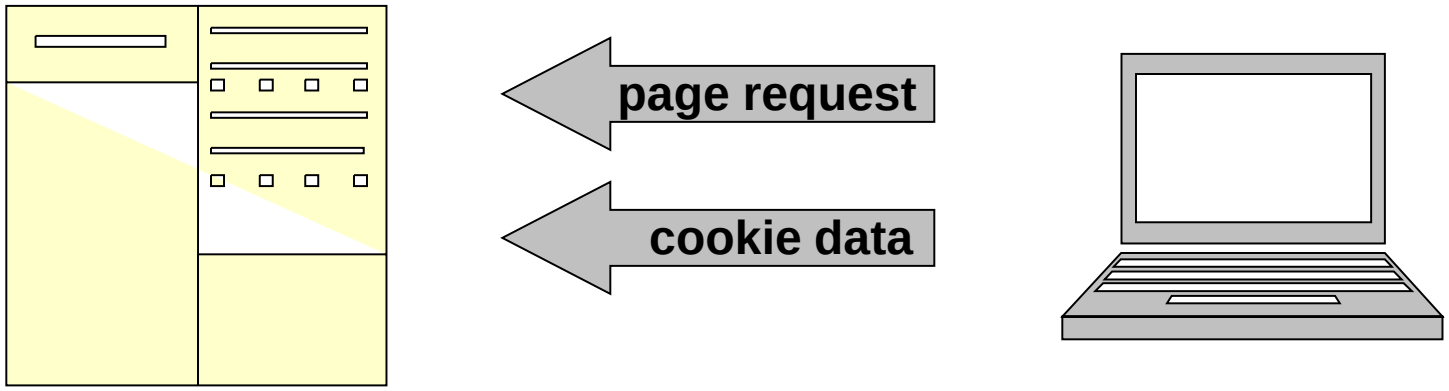
PHP –Cookies

- 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

Type of Cookies:

- **Session Cookie**: This is a type of cookie that expires when the session will destroy.
- **Persistent Cookie**: Persistent cookie is a kind of cookie that is stored permanently on browser's system and expires on some specific time.

- 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>
```

Example (unit - 4 session1.php)

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 –Auto start Session Variables

- Turning on Auto Session

- You don't need to call start_session() function to start a session when a user visits your site if you can set session.auto_start variable to 1 in php.ini file.

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.