2...

Web Techniques

Learning Objectives...

- To study the HTTP Basics, Variables
- To understand the Server Information and Processing Forms.
- To learn about File uploading.
- To study the Setting Response Headers and Maintaining State.

2.1 HTTP BASICS

- HTTP stands for **HyperText Transfer Protocol**. It's the foundational protocol used on the web to transfer hypertext documents, like HTML pages, between a client (usually your web browser) and a web server.

Key points about HTTP:

- **Purpose:** It enables communication between web browsers and servers, allowing users to access websites.
- **Request-Response Model:** The client sends an HTTP request (like asking for a webpage), and the server responds with the requested resource or an error message.
- **Stateless Protocol:** Each request is independent; the server does not remember previous requests.
- **Methods:** Common HTTP methods include:
  - **GET:** Request data from a server (e.g., get a web page).
  - **POST:** Send data to a server (e.g., form submission).
  - **PUT:** Update data on the server.
  - **DELETE:** Remove data on the server.

2.2 VARIABLES

- Variables are used to create space in computer memory. How much space that depends on data types. Variables in web development are used to store and manipulate data dynamically.

- A variable starts with the $ sign, followed by the name of the variable.
- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and_)
- Variable names are case-sensitive ($age and $AGE are two different variables).

For Example,

```
$name = "mohsin";
$age = 25;
```

2.3 SERVER INFORMATION

- The $_SERVER is a superglobal in PHP. It includes information about HTTP headers, path, script location, and other things. It is an associative array that contains information about the execution environment and server.
- The majority of these details are filled in by the web server and each server may have different entries. When running PHP scripts from the command line, some of these entries might not be available.

Sr. No.	Server Variables & Description
1.	**PHP_SELF** Stores filename of currently executing script.
2.	**SERVER_ADDR** This property of array returns the IP address of the server under which the current script is executing.
3.	**SERVER_NAME** Name of server host under which the current script is executing. In case of a server running locally, localhost is returned.
4.	**REQUEST_METHOD** HTTP request method used for accessing a URL, such as POST, GET, PUT or DELETE. In the above query string example, a URL attached to query string with the "?" symbol requests the page with GET method.
5.	**REMOTE_ADDR** IP address of the machine from where the user is viewing the current page.
6.	**SERVER_PORT** Port number on which the web server is listening to the incoming request. Default is 80.
7.	**SCRIPT_FILENAME** The absolute path to the currently executing script.
8.	**HTTP_HOST** The contents of the Host header from the current request.

Program 2.1: Write a PHP program to display server information

```
<html>
<head>
<title>Server Info
</title>
</head>
<body>
  <h1>Server and Client Info</h1>
  PHP Self: <?php echo $_SERVER['PHP_SELF'];?>
  Server Name: <?php echo $_SERVER['SERVER_NAME']; ?>
  Host:
<?php echo $_SERVER['HTTP_HOST']; ?>
  User Agent:
<?php echo $_SERVER['HTTP_USER_AGENT']; ?>
  Script Filename:<?php echo $_SERVER['SCRIPT_FILENAME']; ?>
  Client IP:
<?php echo $_SERVER['REMOTE_ADDR']; ?>
  Request Method: <?php echo $_SERVER['REQUEST_METHOD']; ?>
</body>
</html>
```

Output:

Server and Client Info

PHP Self: /serverinfo.php

Server Name: localhost

Host: localhost

User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36

Script Filename: C:/xampp/htdocs/serverinfo.php

Client IP: 127.0.0.1

Request Method: GET

2.4 PROCESSING FORMS

HTML Forms play an important role in PHP web applications. Although a web page composed purely with HTML is a static web page, the HTML form component is an important feature that helps in bringing interactivity and rendering dynamic content. PHP's form handling functionality can validate data collected from the user, before processing.

What is Form Processing in php?

- An HTML Form is a collection of various form controls such as text fields, checkboxes, radio buttons, etc., with which the user can interact, enter or choose certain data that may be either locally processed by JavaScript (client-side processing), or sent to a remote server for processing with the help of server-side programming scripts such as PHP.
- For example, suppose a user enters their name and email, clicks submit and you can use PHP to handle this data.
- Web forms allow users to submit data to the server. This data must be processed on the server side.

Program 2.2:

```
<!DOCTYPE html>
<html>
<head>
    <title>Form Example</title>
</head>
<body>
    <h2>User Form</h2>
    <form action="submit.php" method="POST">
        Name: <input type="text" name="username">
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

Processing in PHP (submit.php):

```
<?php
// Capture the data sent from form
$name = $_POST['username'];
// Display a response
echo "Welcome, " . htmlspecialchars($name);
?>
```

Steps:

The user fills the form and submits it.

Data is sent via GET or POST method.

Server-side script captures and processes the data.

Output:

Welcome, Madhuri

2.5 FILE UPLOADING

- Users can upload files (images, documents) via forms. This needs special handling on both client and server sides.

Program 2.3:

HTML Form:

```
<!DOCTYPE html>
<html>
<head>
   <title>File Upload Example</title>
</head>
<body>
   <h2>Upload a File</h2>
   <form action="upload.php" method="POST" enctype="multipart/form-data">
      Select file: <input type="file" name="myfile">
      <input type="submit" value="Upload">
   </form>
</body>
</html>
```

PHP Server-Side (php code):

```
<?php
  // Get file details from the $_FILES superglobal
  $myfile   = $_FILES['myfile']['name'];
  $temp_add = $_FILES['myfile']['tmp_name'];
  // Define target path (make sure "upload" folder exists and is writable)
  $path = "upload/" . basename($myfile);
  // Move file from temporary directory to the target directory
  if (move_uploaded_file($temp_add, $path)) {
     echo "File uploaded successfully.";
     echo "<br>Saved as: " . $path;
  } else {
     echo "File upload failed.";
  }
?>
```

Output:

File uploaded successfully.
Saved as: upload/resume.pdf

Explanation:

- **<form>:** Creates a form for user input.
- **action="#":** Submits the form to the same page (or you can set it to a file like upload.php).
- **method="POST":** Uses the POST method to send form data (required for file uploads).
- **enctype="multipart/form-data":** Important for file upload. Without it, file data won't be sent.
- **<input type="file" name="myfile">:** Lets the user browse and select a file. name="myfile" is the key to access this file on the server side.
- **<input type="submit" value="Upload">:** Submits the form to the server.
- **$_FILES['myfile']:** A superglobal array that stores information about the uploaded file. 'myfile' matches the name attribute in the <input type="file">. PHP stores uploaded files in a temp directory before you move them. $path="uploads/". $myfile; Destination path + original file name. Stores the file in uploads/folder (make sure this folder exists and has write permissions).
- **move_uploaded_file(...):** A built-in PHP function to move the uploaded file from temp to a permanent location. Returns true if successful, false if something fails.
- **if statement:** If the file is moved successfully, it prints: "File uploaded successfully." Otherwise: "File upload failed."

2.6 SETTING RESPONSE HEADERS

- The `header()` function is an inbuilt function in PHP which is used to send a raw HTTP header. The HTTP functions are those functions which manipulate information sent to the client or browser by the Web server, before any other output has been sent. The PHP header() function send a HTTP header to a client or browser in raw form. Before HTML, XML, JSON or other output has been sent to a browser or client, raw data is sent with a request (especially HTTP Request) made by the server as header information. HTTP header provides required information about the object sent in the message body more precisely about the request and response.
- In PHP, the `header()` function is used to send raw HTTP headers to the browser before any output is sent. It's commonly used to:
  - Change page location
  - Set timezone.
  - Set caching control
  - Initiate force download
  - Send HTTP Status.

Syntax of header():

```
header(string $header, bool $replace = true, int $http_response_code = 0)
```

$header: The header string to send (e.g., "Location: page.php")

$replace (optional): Whether to replace a previous header of the same type (default true)

$http_response_code (optional): Force a specific HTTP status code (e.g., 404, 200)

For Example,

```
<?php header("Location:demo.php");
  exit(); // Always call exit after header redirect
?>
```

2.7 MAINTAINING STATE

- HTTP is a stateless protocol due to which it is also known to be connectionless. The server and client are aware of each other only when the current request after that client and server forgot about each other, so that browsers cannot get information between different requests across the web pages.
- 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 web page.
- PHP provides for two different techniques for state management of your web application, they are:
  1. Server Side State Management
  2. Client Side Server Management

1. 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 web page.
- 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 web page of the web application until the session variable is destroyed.

2. 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 web pages 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).

2.7.1 Cookies

- Cookies are used for client-side state management system.
- Cookies are data by the browser, cookies are sent to the web server as a piece of header information with every HTTP request.
- Cookies can contain 1KB (1024B) size of data.

Uses of Cookies:

- To store information about visitors regarding the accessed website's page.
- Number of visit and views.
- Store first visit information and update it on every visit that pointed towards better experience for the user.

Create Cookie:

- The setcookie() function is used to create a cookie. The setcookie() function defines a cookie to be sent along with other HTTP headers. The setcookie() function should be appeared before the <html> and <head> tag.

Syntax:

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

Parameters:

- **name:** It is required. It specifies the name of the cookie to be sent.
- **value:** It is optional. It specifies the value of the cookie to be sent.
- **expire:** It is optional. It specifies when the cookie will expire. It has a default value of 0, which determines that the cookie will expire on the closing session (closing the browser).
- **path:** It is optional. It specifies the server path of the cookie. Its default value is the current directory that the cookie is being set in.
- **domain:** It is optional. It specifies the domain name of the cookie. For making the cookie available on all subdomains of "example.com", set it to "example.com".
- **secure:** It is optional. It specifies whether cookies should be only transmitted over a secure HTTPS connection. The default value is "false" (cookie will be set on any connection).

For Example,

```
<?php
Setcookie("username", "abc", time() + 60 * 60);
?>
```

Note:

- To retrieve cookies data in PHP use `$_COOKIES`.
- To check if it is set or not, use `isset()` function.

For Example,

```
<?php
   Setcookie("username", "abc", time() + 60 * 60);
?>
```

Program 2.4:

```
<?php
  if(isset($_COOKIE["username"]))
  {
    echo "Cookie is set";
  } else {
    echo "Cookie is not set";
  }
?>
```

Output:

Cookie is not set

- For updating cookies only, we need to change the argument by calling setcookie() function. We change the name "abc" to "xyz".

For Example,

```
<?php
  setcookie("username", "xyz", time() + 60 * 60);
?>
```

- For deleting cookies we need to set expiry time in negative.

For Example,

```
<?php
  Setcookie("username", "xyz", time() - 3600);
?>
```

2.7.2 Session

Creation of Session:

- In PHP, session starts from `session_start()` function and data can be set and get by using global variables `$_SESSION`.

How to Use PHP Sessions?

- Using PHP sessions involves several key steps: starting a session, storing data in session variables, retrieving data, and eventually destroying the session when no longer needed.

1. Starting a Session

- To begin using sessions in PHP, you need to start the session with `session_start()` at the very beginning of the PHP script. This function ensures that the session is available and creates a unique session ID if it doesn't already exist.

For Example,

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

2. Storing Data in Sessions

- Once the session is started, you can store any information in the $_SESSION superglobal array. This allows you to carry data across different pages on the website.

For Example,

```
<?php
    session_start();
    $_SESSION['username'] = 'Mohsin'; // Store session data
    $_SESSION['user_id'] = 123;
?>
```

- The username and user ID are stored in the session for use on other pages

3. Retrieving Session Data

- Once data is stored in a session, it can be accessed on any page where the session is started.

For Example,

```
<?php
    session_start();
    echo $_SESSION['username']; // Output: Moshin
?>
```

- You can use the session variables to display user-specific information, check login statuses, and perform various operations.

4. Checking if Session Variables Exist

- Before using session data, it's a good practice to check if the session variable exists to avoid errors.

Program 2.5:

```
<?php
    session_start();  // starts a session or resumes the existing one
    if (isset($_SESSION['username'])) {
        echo "Welcome, " . $_SESSION['username'];
    } else {
        echo "Please log in.";
    }
?>
```

Output:

Welcome, Madhuri
Please log in.

5. Destroying Sessions

- When a session is no longer needed, you can terminate it by using session_destroy(). This function removes all session data from the server. However, it does not automatically unset session variables; you need to manually clear them using unset() if needed.

For Example,

```
<?php
    session_start();
    unset($_SESSION['username']); // Remove specific session variable
    session_destroy(); // Destroy the session
?>
```

Output:

You have been logged out.

Sample programs

Program 2.6: Write a PHP program where a form is submitted to the same PHP file (self-processing). The form should take a username and age, and display a message after submission.

```
<?php
   $message = "";
   if ($_SERVER["REQUEST_METHOD"] == "POST") {
      $username = $_POST['username'];
      $age = $_POST['age'];
    if (!empty($username) && !empty($age)) {
        $message = "Hello $username, you are $age years old!";
    } else {
        $message = "Please fill all fields.";
    }
   }
?>
<!DOCTYPE html>
<html>
<head>
   <title>Self Processing Form</title>
</head>
<body>
<h2>Self Processing Form</h2>
  <form method="post" action="<?php echo
htmlspecialchars($_SERVER['PHP_SELF']); ?>">
     Username: <input type="text" name="username"><br><br>
     Age: <input type="number" name="age"><br><br>
     <input type="submit" value="Submit">
  </form>
<p style="color:blue;"><?php echo $message; ?></p>
</body>
</html>
```

Output:

Self Processing Form

Username: [_____]

Age: [___]

[Submit]

- After submitting form with:

  Username: **Madhuri**

  Age: **25**

Hello Madhuri, you are 25 years old!

Program 2.7: Create a PHP program with a form where the user can enter their name and email. The form should keep the entered data even after submission (sticky form).

```
<?php
$name = "";
$email = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST['name'];
    $email = $_POST['email'];
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>Sticky Form Example</title>
</head>
<body>
<h2>Sticky Form</h2>
<form method="post" action="">
    Name: <input type="text" name="name" value="<?php echo
htmlspecialchars($name); ?>"><br><br>
    Email: <input type="email" name="email" value="<?php echo
htmlspecialchars($email); ?>"><br><br>
    <input type="submit" value="Submit">
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    echo "<h3>Form Submitted Data:</h3>";
    echo "Name: $name <br>";
    echo "Email: $email";
}
?>
</body>
</html>
```

Output:

Before Submission

Name: [ ]
Email: [ ]
[Submit]

After Submitting:

Input → Name = Madhuri, Email = madhuri@example.com

Output on the same page:

Name: [Madhuri]

Email: [madhuri@example.com]

[Submit]

And below the form:

Form Submitted Data:

Name: Madhuri

Email: madhuri@example.com

Check Your Understanding

1. Which of the following is NOT a valid PHP variable name?
   (a) `$user_name` (b) `$1username`
   (c) `$_user1` (d) `$userName`
2. Which superglobal contains information about headers, paths, and script locations?
   (a) `$_POST` (b) `$_GET`
   (c) `$_SESSION` (d) `$_SERVER`
3. What does the isset($_POST['submit']) check for?
   (a) If the submit button is hidden (b) If the submit button is clicked
   (c) If the form action is set (d) None of the above
4. To upload a file in PHP, which superglobal is used?
   (a) `$_GET` (b) `$_REQUEST`
   (c) `$_FILES` (d) `$_SESSION`
5. Which PHP function is used to send raw HTTP headers?
   (a) `send_header()` (b) `raw_header()`
   (c) `header()` (d) `http_send()`

Answers

1. (b)	2. (d)	3. (b)	4. (c)	5. (c)

Practice Questions

Q.I Answer the following questions in short:

1. What is the difference between GET and POST methods?
2. Define `$_SERVER['HTTP_USER_AGENT']`.
3. How do you define a variable in PHP?

4. What is the role of enctype="multipart/form-data" in a file upload form?
5. How can you redirect a user to another page in PHP?
6. What does the `header()` function do in PHP?
7. Name any two `$_SERVER` superglobal elements.
8. Write a short code to create a session variable.
9. How can you check if a form has been submitted using POST?
10. What is cookie?

Q.II Answer the following questions in detail:

1. Explain the basic structure of an HTTP request and response. Provide examples.
2. What are variables in PHP and how are they used in web applications? Illustrate with examples.
3. Describe how server information can be accessed using $_SERVER in PHP. List at least 5 useful elements.
4. Explain the process of handling HTML forms using PHP. Include both GET and POST methods.
5. Describe the steps involved in uploading a file using an HTML form and processing it using PHP.
6. What are HTTP headers? Explain how to set response headers in PHP with appropriate examples.
7. Define sessions and cookies. Compare and contrast their usage with code examples.
8. Discuss the importance of maintaining state in a stateless HTTP protocol.
9. Explain file uploading in php with the help of an example.
10. Explain the process of setting, accessing, and deleting cookies in PHP.
11. Explain the process of the session in detail.

***