Text

Description automatically generated

**CENTRE FOR DIPLOMA STUDIES**

**UNIVERSITI TUN HUSSEIN ONN MALAYSIA (UTHM)**

**LAB 5**

**ADVANCED PHP**

**COURSE CODE         DAT21303**

**COURSE NAME        WEB DEVELOPMENT**

**FACULTY                  CENTER OF DIPLOMA STUDY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**STUDENT NAME     WAN MUHAMMAD ALIF FIRDAUS BIN WAN HANAPI**

**NUM MATRIX AA210292**

**SECTION                   6**

**LECTURE NAME    NOR NADHIA BINTI NOR AZMAN SUBRAMANIAM**

**SUBMIT DATE         26 NOVEMBER 2022**

**LABORATORY 4: ADVANCED PHP**

|  |  |
| --- | --- |
| **Objective** | In advanced PHP student would be exposed on more advanced features available in PHP such as Multidimensional Array, Date Time, file handling, session, cookies and many more. Furthermore, it is just the beginning of next stages of rookies to intermediate. |
| **Required**  **Resources** | Macromedia Dreamweaver, Sublime Text, Bracket or if you are good enough just use text editor notepad. Xampp (used to dedicate your machine as server). |
| **References** | <https://www.w3schools.com/php/php_arrays_multi.asp>  https://www.smashingmagazine.com/2009/03/10-useful-php-tips-revisited/ |
| **Date**  **Released** | 15/10/2017 |
| **Date**  **Submission** | 22/10/2017 |

Note: Do this Lab in a group of two people

**i. Why advanced PHP?**

Because of PHP’s huge popularity, it has become almost impossible for Web developers not to have at least a working knowledge of PHP. This tutorial is aimed at people who are just past the beginning stages of learning PHP and are ready to roll up their sleeves and get their hands dirty with the language

# Why we need advanced PHP

In advances PHP there are many features help us to improve the PHP programming skills and at the same time would help student to set up their system in more secure and well organized. In order to upgrade the knowledge student must able to understand and try to do the question given in this lab.

# Sessions and cookies

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

# Sessions and Cookies differentiation

The main difference between a **session** and a **cookie** is that session data is stored on the server, whereas cookies store data in the visitor's browser. Sessions are more secure than cookies as it is stored in server. Cookie can be turn off from browser. Below in figure 1.0 is cookie setup.

|  |
| --- |
| <!DOCTYPE html>  <?php  $cookie\_name = "user"; $cookie\_value = "John Doe";  setcookie($cookie\_name, $cookie\_value, time() + (86400 \* 30), "/"); // 86400 = 1 day  ?>  <html><body> <?php  if(!isset($\_COOKIE[$cookie\_name])) {  echo "Cookie named '" . $cookie\_name . "' is not set!";  } else {  echo "Cookie '" . $cookie\_name . "' is set!<br>"; echo "Value is: " . $\_COOKIE[$cookie\_name];  }  ?> <p><strong>Note:</strong> You might have to reload the page to see the value of the cookie.</p>  </body>  </html> |

**Figure 1.0: Code Segmentation of Setting the Cookie**

# Session setup

<?php

// Start the session session\_start();

?>

<!DOCTYPE html>

<html>

<body>

<?php

// Set session variables

$\_SESSION["favcolor"] = "green"; $\_SESSION["favanimal"] = "cat"; echo "Session variables are set.";

?>

</body>

</html>

**Figure 1.1: Code Segmentation of Setting the Session**

Furthermore in advanced PHP there are a lot of directories in PHP manual (visit PHP manual site). The function such as Date, Multidimensional Array, file manipulation and many more. In order to master those PHP features you must keep trying and do more experimental task. Don’t hesitate to write down PHP code with advanced function and try to troubleshoot the issues that rise upon your experimental testing.

**Question 1:** Based on Table 1.0 is about a tabulation of personal data. Provide the output such shows in Table 1.0 with logical explanation. Display 4 rows and 3 columns by using PHP codes.

The method must be involving nested loop and declaration multidimensional array. **Table 1.0: Tabulation of Personal Data**

|  |  |  |
| --- | --- | --- |
| **Id** | **Name** | **Ages** |
|  |  |  |
| 1 | Anip | 20 |
| 2 | Hatta | 12 |
|  |  |  |
| 3 | Nazri | 13 |
| 4 | Mamat | 22 |

<!DOCTYPE html>

<html>

<style>

table, th, td {

border:1px solid black;

text-align: center;

}

tr:nth-child(even) {

background-color: #D6EEEE;

}

tr:nth-child(odd) {

background-color: #D6EEEE;

}

</style>

<body>

<?php

$id = 1;

$name=array(

"anip"=>"20",

"hatta"=>"12",

"nazri"=>"13",

"mamat"=> "22"

)

/\*foreach($age as $x => $xValue)\*/

?>

<table style="width:50%">

<tr>

<th>ID</th>

<th>NAME</th>

<th>AGE</th>

</tr>

<?php $id = 1; ?>

<tr>

<?php foreach($name as $x => $xValue){?>

<td><?php echo $id; ?></td>

<td><?php echo $x; ?></td>

<td><?php echo $xValue; ?></td>

<?php $id++; ?>

</tr>

<?php } ?>

</table>

Table

Description automatically generated

**Question 2:** Related to Figure 1.2 below create a txt file and name it as test.txt. The content of test.txt must be same as figure 1.2. So you need to open the test.txt and change the “Khan” to “Muhammad” and delete the line “I love to play football”. All tasks must be done by using PHP file handling.

<?php

// Open the file in write mode

$file = fopen('test.txt', 'w');

// Read the contents of the file into a string

$contents = file\_get\_contents('test.txt');

// Replace the string "Khan" with "Muhammad"

$contents = str\_replace('Khan', 'Muhammad', $contents);

// Delete the line "I love to play football"

$contents = str\_replace("I love to play football\n", '', $contents);

// Write the modified contents back to the file

fwrite($file, $contents);

// Close the file

fclose($file);

echo 'The file has been modified successfully!';

In this example, the fopen() function is used to open the test.txt file in write mode, which allows the script to modify the contents of the file. The file\_get\_contents() function is used to read the contents of the file into a string.

The str\_replace() function is then used to replace the string "Khan" with "Muhammad" and to delete the line "I love to play football" by replacing it with an empty string.

Finally, the fwrite() function is used to write the modified contents of the string back to the file, and the fclose() function is used to close the file.

It is important to note that the fopen() function requires that the script has permission to write to the file. You may need to adjust the file permissions or the ownership of the file to allow the script to modify it.

My name is Khan

I am ten years old

I love to play football

**Figure 1.2: A content of test.txt file**

**Question 3:** Create 2 PHP files. *Login.php, successPage.php*. Once both files created use sessions and cookies to recognize the user login. For example Ali log in the *Login.php* page and the *successPage.php* will print “welcome ali”.

(lab5question3.php)

<!DOCTYPE html>

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

Name:<label for="name">Your name:</label><br>

<input type="text" id="name" name="name" required/><br><br>

<input type="submit">

</form>

<?php

$name="null";

$cookie\_name = $name;

$cookie\_value = $name;

setcookie($cookie\_name, $cookie\_value, time() + (86400 \* 30), "/"); // 86400 = 1 day

?>

<html>

<body>

<?php

if(!isset($\_COOKIE[$cookie\_name])) {

echo "Cookie named '" . $cookie\_name . "' is not set!";

} else {

echo "Cookie '" . $cookie\_name . "' is set!<br>";

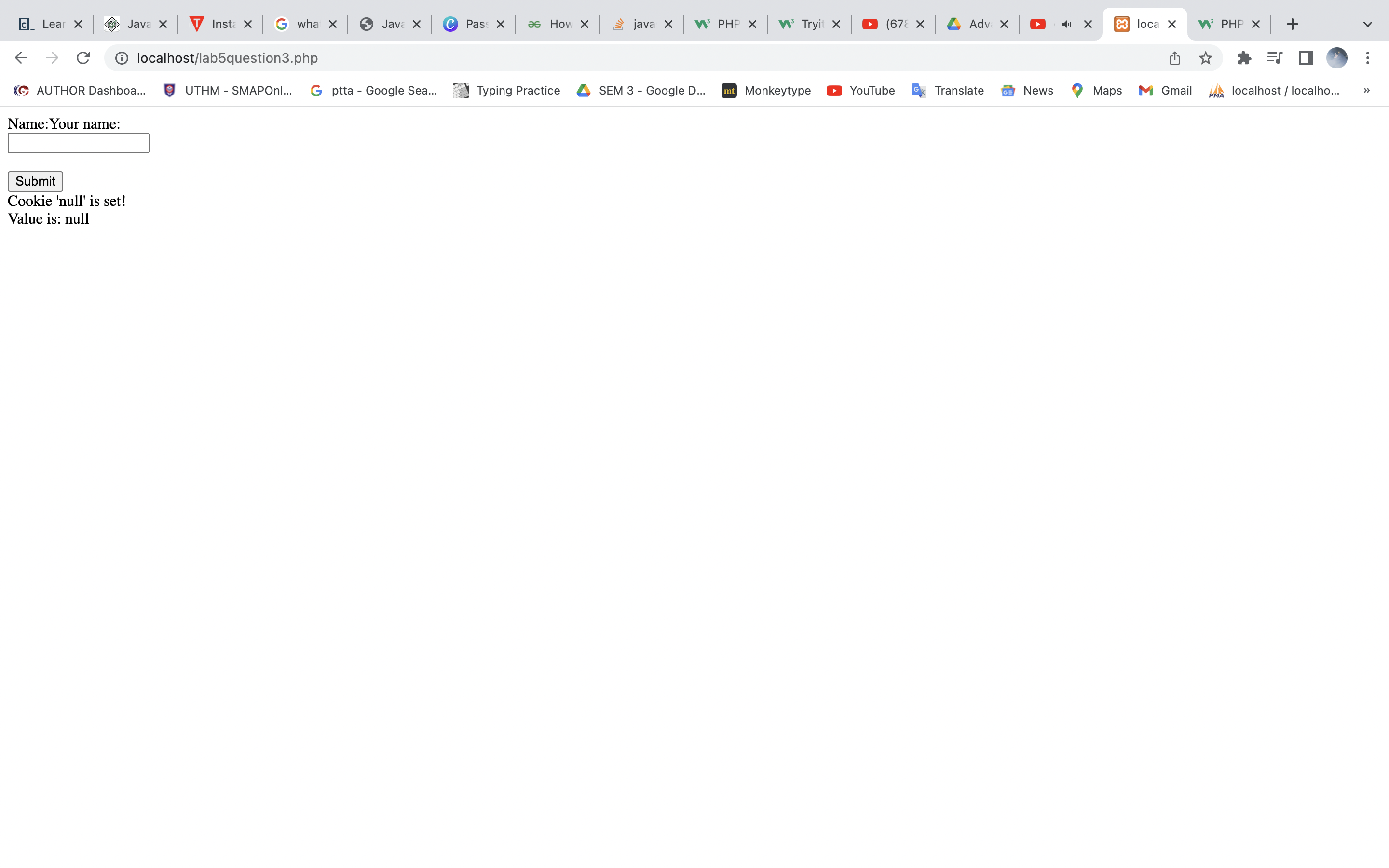
echo "Value is: " . $\_COOKIE[$cookie\_name];

}

?>

</body>

</html>



(success.php)

<!DOCTYPE html>

<body>

<?php

// Start the session

session\_start();

?>

<!DOCTYPE html>

<html>

<body>

<?php

// Set session variables

$\_SESSION["favcolor"] = "green";

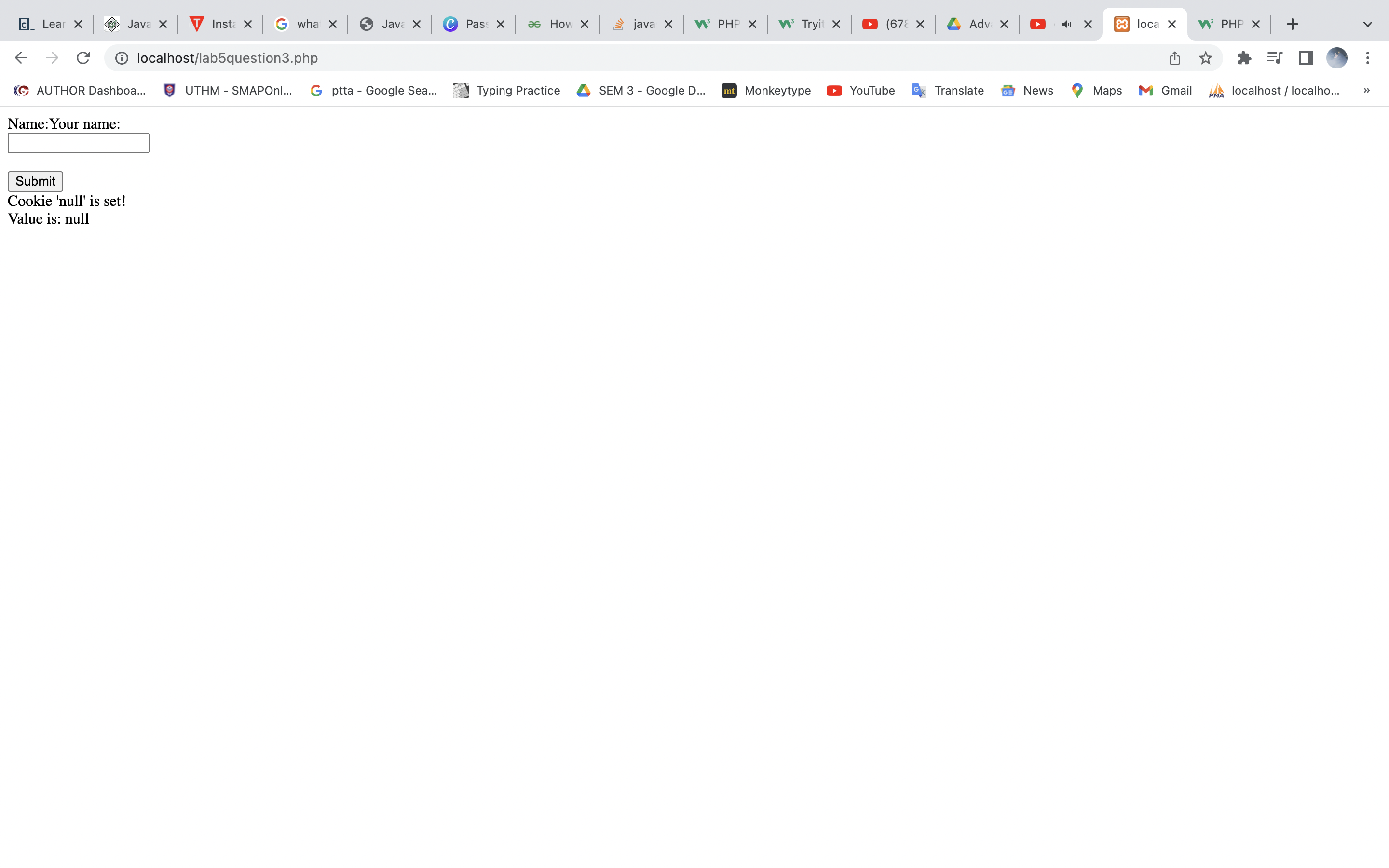
$\_SESSION["favanimal"] = "cat";

?>

Welcome <?php echo $\_POST["name"]; ?><br>

</body>

</html>



**Question 4:** Provide example and explanation based on Table 1.1 below:

|  |  |  |
| --- | --- | --- |
| **Predefined**  **Variables** | **Coding Segmentation (example)** | **Simple Explanation** |
| $GLOBALS | <?php  $var1 = 'Hello';  $var2 = 123;  $var3 = true;  function printVars() {  echo $GLOBALS['var1'] . '<br>';  echo $GLOBALS['var2'] . '<br>';  echo $GLOBALS['var3'] . '<br>';  }  printVars();  // Output:  // Hello  // 123  // 1 | In PHP, the $GLOBALS superglobal array is an associative array that contains a list of all global variables available in the current script. The keys of the array are the names of the global variables, and the values are the current values of the variables.  The $GLOBALS array is automatically available in all PHP scripts, and you can use it to access global variables from anywhere in your code. For example, to access the value of a global variable $var  In this example, the script defines three global variables: $var1, $var2, and $var3. The printVars function uses the $GLOBALS array to access the values of these variables and print them to the screen. |
| $\_SERVER | <?php  echo 'Host: ' . $\_SERVER['HTTP\_HOST'] . '<br>';  echo 'URI: ' . $\_SERVER['REQUEST\_URI'] . '<br>';  echo 'Method: ' . $\_SERVER['REQUEST\_METHOD'] . '<br>';  echo 'Server name: ' . $\_SERVER['SERVER\_NAME'] . '<br>';  echo 'Server port: ' . $\_SERVER['SERVER\_PORT'] . '<br>';  echo 'Client IP: ' . $\_SERVER['REMOTE\_ADDR'] . '<br>';  echo 'Script path: ' . $\_SERVER['SCRIPT\_FILENAME'] . '<br>';  // Output:  // Host: www.example.com  // URI: /index.php  // Method: GET  // Server name: localhost  // Server port: 80  // Client IP: 192.168.1.100  // Script path: /var/www/html/index.php | In PHP, the $\_SERVER superglobal array is an associative array that contains a list of server and execution environment information. The keys of the array are the names of the variables, and the values are the current values of the variables.  The $\_SERVER array is automatically available in all PHP scripts, and it contains a variety of information about the server, the current script, and the execution environment. Some common elements of the $\_SERVER array include:   * $\_SERVER['HTTP\_HOST']: the hostname of the server (e.g., www.example.com) * $\_SERVER['REQUEST\_URI']: the URI of the current request (e.g., /index.php) * $\_SERVER['REQUEST\_METHOD']: the HTTP method of the current request (e.g., GET, POST, etc.) * $\_SERVER['SERVER\_NAME']: the name of the server (e.g., localhost) * $\_SERVER['SERVER\_PORT']: the port number on which the server is running (e.g., 80) * $\_SERVER['REMOTE\_ADDR']: the IP address of the client making the request * $\_SERVER['SCRIPT\_FILENAME']: the full path of the current   In this example, the script uses the $\_SERVER array to access several pieces of information about the server, the current request, and the execution environment. It then prints this information to the screen using the echo statement.  The output of the script will depend on the specific values of the variables in the $\_SERVER array, which will vary depending on the server, the script, and the execution environment. The example output provided is just one possible example. |
| $\_GET | <?php  if ($\_SERVER['REQUEST\_METHOD'] == 'GET') {  // Display the form  echo '  <form action="process-form.php" method="get">  <label for="name">Name:</label><br>  <input type="text" id="name" name="name"><br>  <label for="age">Age:</label><br>  <input type="number" id="age" name="age"><br><br>  <input type="submit" value="Submit">  </form>  ';  } elseif ($\_SERVER['REQUEST\_METHOD'] == 'GET') {  // Process the form submission  $name = $\_GET['name'];  $age = $\_GET['age'];  echo 'Name: ' . $name . '<br>';  echo 'Age: ' . $age;  } | In PHP, the $\_GET superglobal array is an associative array that contains a list of variables passed to the current script via the HTTP GET method. The keys of the array are the names of the variables, and the values are the current values of the variables.  The $\_GET array is automatically available in all PHP scripts, and it is used to access variables that are passed to the script via the URL query string  In this example, the script uses the $\_SERVER['REQUEST\_METHOD'] variable to determine whether the request is a GET request (i.e., the form is being displayed) or a POST request (i.e., the form has been submitted).  If the request is a GET request, the script displays the form. If the request is a POST request, the script processes the form submission by accessing the $\_GET array and extracting the name and age variables. It then displays the values of these variables to the screen.  To test this script, you can open it in your web browser and fill out the form. When you submit the form, the script will process the submission and display the values of the name and age variables. |
| $\_POST | <?php  if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {  // Process the form submission  $name = $\_POST['name'];  $age = $\_POST['age'];  echo 'Name: ' . $name . '<br>';  echo 'Age: ' . $age;  } else {  // Display the form  echo '  <form action="process-form.php" method="post">  <label for="name">Name:</label><br>  <input type="text" id="name" name="name"><br>  <label for="age">Age:</label><br>  <input type="number" id="age" name="age"><br><br>  <input type="submit" value="Submit">  </form>  ';  } | In PHP, the $\_POST superglobal array is an associative array that contains a list of variables passed to the current script via the HTTP POST method. The keys of the array are the names of the variables, and the values are the current values of the variables.  The $\_POST array is automatically available in all PHP scripts, and it is used to access variables that are passed to the script via an HTTP POST request. POST requests are often used to submit forms or upload files, and the $\_POST array is used to access the data that is submitted with the request.  In this example, the script uses the $\_SERVER['REQUEST\_METHOD'] variable to determine whether the request is a POST request (i.e., the form has been submitted) or a GET request (i.e., the form is being displayed).  If the request is a POST request, the script processes the form submission by accessing the $\_POST array and extracting the name and age variables. It then displays the values of these variables to the screen.  If the request is a GET request, the script displays the form.  To test this script, you can open it in your web browser and fill out the form. When you submit the form, the script will process the submission and display the values of the name and age variables. |
| $\_FILES | <?php  if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {  // Process the uploaded file  $file = $\_FILES['file'];  // Check for errors  if ($file['error'] > 0) {  echo 'Error: ' . $file['error'] . '<br>';  } else {  // Move the uploaded file to the uploads directory  $target\_dir = 'uploads/';  $target\_file = $target\_dir . basename($file['name']);  move\_uploaded\_file($file['tmp\_name'], $target\_file);  // Confirm that the file was uploaded  echo 'File uploaded successfully: ' . $target\_file;  }  } else {  // Display the form  echo '  <form action="upload.php" method="post" enctype="multipart/form-data">  <label for="file">Select a file:</label><br>  <input type="file" id="file" name="file"><br><br>  <input type="submit" value="Upload">  </form>  ';  } | In PHP, the $\_FILES superglobal array is an associative array that contains a list of variables that represent uploaded files. The keys of the array are the names of the input fields in the HTML form that was used to upload the files, and the values are arrays containing information about the uploaded files.  The $\_FILES array is automatically available in all PHP scripts, and it is used to access files that are uploaded to the server via an HTML form. To use the $\_FILES array, you must first create an HTML form that includes an input element with the type attribute set to "file".  In this example, the script uses the $\_SERVER['REQUEST\_METHOD'] variable to determine whether the request is a POST request (i.e., the form has been submitted) or a GET request (i.e., the form is being displayed).  If the request is a POST request, the script processes the uploaded file by accessing the $\_FILES array and extracting the file element. It then checks for any errors (e.g., if the file is too large or if there was an issue with the upload process). If there are no errors, the script moves the uploaded file to the uploads/ directory on the server and confirms that the file was uploaded successfully.  If the request is a GET request, the script displays the form.  To test this script, you can open it in your web browser and select a file to upload. When you submit the form, the script will process the file upload and either display an |
| $\_REQUEST | <?php  if ($\_SERVER['REQUEST\_METHOD'] == 'POST') {  // Process the form submission  $name = $\_REQUEST['name'];  $age = $\_REQUEST['age'];  echo 'Name: ' . $name . '<br>';  echo 'Age: ' . $age;  } else {  // Display the form  echo '  <form action="process-form.php" method="post">  <label for="name">Name:</label><br>  <input type="text" id="name" name="name"><br>  <label for="age">Age:</label><br>  <input type="number" id="age" name="age"><br><br>  <input type="submit" value="Submit">  </form>  ';  } | In PHP, the $\_REQUEST superglobal array is an associative array that contains a list of variables passed to the current script via the HTTP request method (GET, POST, or COOKIE). The keys of the array are the names of the variables, and the values are the current values of the variables.  The $\_REQUEST array is automatically available in all PHP scripts, and it is used to access variables that are passed to the script via an HTTP request (either GET, POST, or COOKIE).  In this example, the script uses the $\_SERVER['REQUEST\_METHOD'] variable to determine whether the request is a POST request (i.e., the form has been submitted) or a GET request (i.e., the form is being displayed).  If the request is a POST request, the script processes the form submission by accessing the $\_REQUEST array and extracting the name and age variables. It then displays the values of these variables to the screen.  If the request is a GET request, the script displays the form.  To test this script, you can open it in your web browser and fill out the form. When you submit the form, the script will process the submission and display the values of the name and age variables.  It is important to note that the $\_REQUEST array can potentially |
| $\_SESSION | <?php  session\_start();  // Set a session variable  $\_SESSION['name'] = 'John';  echo 'Welcome, ' . $\_SESSION['name'] . '!<br>';  echo '<a href="page2.php">Go to page 2</a>';  // Output:  // Welcome, John!  // Go to page 2 | In PHP, the $\_SESSION superglobal array is an associative array that is used to store variables across multiple pages of a website. The values of the variables are stored on the server, and the variables are accessible to all pages of the website that are linked to the same session.  To use the $\_SESSION array, you must first start a session at the beginning of your PHP script using the session\_start() function. This function initializes a session and creates a session ID that is stored on the server and sent to the client as a cookie.  In this example, the page1.php script starts a session and sets a session variable called name to 'John'. It then displays a welcome message and a link to page2.php.  When the user clicks the link, they are redirected to page2.php, which retrieves the name session variable and displays a welcome back message and a link to page1.php.  This allows the user to navigate between the two pages and have the name session variable preserved across the pages.  It is important to note that the session\_start() function must be called at the beginning of each script that uses the $\_SESSION array. This is necessary to initialize the session and to create the session ID that is stored on the server and sent to the client as a cookie. |
| <?php  session\_start();  // Retrieve the session variable  echo 'Welcome back, ' . $\_SESSION['name'] . '!<br>';  echo '<a href="page1.php">Go to page 1</a>';  // Output:  // Welcome back, John!  // Go to page 1 |
| $\_COOKIE | <?php  // Set a cookie  setcookie('name', 'John', time() + 86400);  echo 'Welcome, ' . $\_COOKIE['name'] . '!<br>';  echo '<a href="page2.php">Go to page 2</a>';  // Output:  // Welcome, John!  // Go to page 2 | In PHP, the $\_COOKIE superglobal array is an associative array that contains a list of variables sent to the current script via HTTP Cookies. The keys of the array are the names of the cookies, and the values are the current values of the cookies.  HTTP Cookies are small pieces of data that are stored on the client's computer by the web browser. They are used to store information about the user and their preferences, and they can be accessed by the server when the user visits the website again.  To use the $\_COOKIE array, you must first set a cookie using the setcookie() function. This function sends a cookie to the client's web browser, and the cookie is stored on the client's computer.  In this example, the page1.php script sets a cookie called name with a value of 'John' and an expiration time of 24 hours. It then displays a welcome message and a link to page2.php.  When the user clicks the link, they are redirected to page2.php, which retrieves the name cookie and displays a welcome back message and a link to page1.php.  This allows the user to navigate between the two pages and have the name cookie preserved across the pages.  It is important to note that the setcookie() function must be called before any output is sent to the browser. This is because the cookie is sent to the client via the HTTP response headers, and the headers must be sent before any other output. |
| <?php  // Retrieve the cookie  echo 'Welcome back, ' . $\_COOKIE['name'] . '!<br>';  echo '<a href="page1.php">Go to page 1</a>';  // Output:  // Welcome back, John!  // Go to page 1 |
| $\_ENV | <?php  // Check if the NAME environment variable is set  if (isset($\_ENV['NAME'])) {  // The NAME environment variable is set  $name = $\_ENV['NAME'];  echo 'Welcome, ' . $name . '!';  } else {  // The NAME environment variable is not set  echo 'Welcome, guest!';  }  // Output: | In PHP, the $\_ENV superglobal array is an associative array that contains a list of variables passed to the current script via the environment. The keys of the array are the names of the variables, and the values are the current values of the variables.  The $\_ENV array is automatically available in all PHP scripts, and it is used to access variables that are passed to the script via the environment. These variables can be set in a number of ways, including via the server's configuration file (e.g., php.ini, .htaccess), the operating system's environment variables, or the export command in a shell script.  In this example, the script checks if the NAME environment variable is set using the isset() function. If the variable is set, the script retrieves the value of the variable using the $\_ENV array and displays a welcome message. If the variable is not set, the script displays a default welcome message.  To test this script, you can set the NAME environment variable in your operating system's environment variables or in a shell script using the export command, and then run the PHP script. The script will use the value of the NAME environment variable to display the welcome message.  It is important to note that the $\_ENV array is specific to the current script and is not shared across different scripts or processes. Additionally, the $\_ENV array is not persistent across requests, and the variables are reset for each new request. |

**Question 5:** Explains why Cookies and Session is so important. Your words must not exceed 400 words.

Cookies and sessions are both mechanisms for storing data on the client side in a web application. They are both useful for maintaining state between HTTP requests, which is important because HTTP is a stateless protocol, meaning that each request is treated as an independent unit of work and does not carry any information about previous requests.

Cookies are small pieces of data that are stored on the client's computer by the web browser. They can be used to store small amounts of data, such as a user's preferences or login information. One of the main advantages of cookies is that they are persistent, meaning that they remain on the client's computer until they expire or are deleted by the user. This allows the web application to recognize the user and maintain state across multiple sessions.

Sessions are similar to cookies, but they are stored on the server side instead of the client side. When a user visits a web application that uses sessions, the server generates a unique session ID and sends it to the client as a cookie. The client's web browser stores the session ID and sends it back to the server with each subsequent request. The server uses the session ID to look up the corresponding session data and use it to maintain state for the user.

There are a few key differences between cookies and sessions,cookies are stored on the client side, while sessions are stored on the server side. This means that cookies are more vulnerable to tampering, while sessions are more secure.Cookies have a limited size (typically around 4KB), while sessions have no size limit. This means that cookies are suitable for storing small amounts of data, while sessions are better for storing large amounts of data.Cookies have an expiration date, while sessions typically do not. This means that cookies will expire and be deleted automatically after a certain period of time, while sessions will remain active until the user closes their web browser or the session is explicitly destroyed by the server.

Both cookies and sessions have their own advantages and disadvantages, and the best choice for a particular application will depend on the specific requirements of the application. In general, cookies are more suitable for storing small amounts of data that need to persist across multiple sessions, while sessions are better for storing larger amounts of data that need to be accessed more frequently.

In summary, cookies and sessions are important in PHP (and other web development languages) because they allow web applications to maintain state between HTTP requests and recognize returning users. They are both useful tools for storing data on the client side and are widely used in modern web applications. Thanks and Good Luck

# Psychomotor Rubric for Laboratory 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Criteria | Beginner (1) | Moderate (2) | Good (3) | Excellent  (4) | Scores |
| Recognize and able to demonstrate multidimensional array and nested loop | Ambiguously presented but  available | Able to  accomplish correctly but not really clear | Presented in correctly and  clear | Perfectly  clear and proper |  |
| Able to differentiate between Sessions and Cookies approach while dealing with dynamic web development | Ambiguously presented but  available | Able to  accomplish correctly but not really clear | Presented in correctly and clear | Perfectly  clear and proper |  |
| Able to manipulate and control file objects using PHP | Ambiguously presented but  available | Able to  accomplish correctly but not really clear | Presented in correctly and clear | Perfectly  clear and proper |  |
| Combination of looping and array, session and cookie in proper way | Ambiguously presented but  available | Able to  accomplish correctly but not really clear | Presented in correctly and clear | Perfectly  clear and proper |  |
| Theory and Practical understanding and proper  coding structure | Ambiguously presented but  available | Able to  accomplish correctly but not really clear | Presented in correctly and clear | Perfectly  clear and proper |  |
|  |  |  | Total Scores (20 Marks) | |  |