Advanced Higher
Computing Science

Web Development

PHP

Name:_____

IMPLEMENTATION: PHP

PHP is a server-side scripting language, and a powerful tool for making dynamic and interactive Web pages.

A PHP script is executed on the server, and the **plain HTML** result is sent back to the browser.

PHP is used to **generate HTML code** that is then sent back to the browser**.**

INSERTING PHP CODE

A PHP script can be placed anywhere in the document.

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

Example:

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

<h1>My first PHP page</h1>

<?php

     echo "Hello World!";  //write ‘hello world’ to the
     document

?>

</body>
</html>
```

PHP COMMENTS

// is used to comment out sections within the php to allow entry of internal commentary

/* multi line comments are written using like this ***/***

GET AND POST

As you learned in the HTML Forms section, both GET and POST create an array containing **key** and **value** pairs.

- **Keys** are the *names* of the **form controls** (e.g text box called *username*)
- **Values** are the *input data* from the user (e.g. *Fred123* entered by user)

Data can be passed from the HTML form to the PHP script using either GET or POST.

GET

$_GET is an array of key/value pairs passed via the URL (data is visible in the address bar)

```
http://127.0.0.1/L1.php?userName=Fred123&Age=16
```

? in the address bar indicates the start of passed data with & used to separate key/value pairs.

In this case **$_GET** will contain the following:

Key	Value
userName	Fred
Age	16

POST

$_POST is an array of key/value pairs passed vie the HTTP POST method (data is not visible as it is passed)

This method achieves the same result as GET but should be used when the data being passed should not be visible.

Accessing GET and POST Data

The data transferred to $_GET or $_POST can now be accessed and assigned to appropriate server-side variables within the PHP script.

```
<?php

$userFirstname = $_POST['userName']

$userAge = 34 = $_POST['Age']

?>
```

Note:

GET can be used in the same way by replacing POST with GET.

PHP ECHO

Echo is used to write data to the HTML document that the PHP code is generating.

Whatever is entered between the quotes of the echo statement will be written directly to the HTML file.

This includes writing HTML tags contained in the echo statement.

Example 1:

```
echo "Hello World!";
```

This will write plain unformatted text

Example 2:

```
echo "<h1>Hello World! </h1>";
```

This will write text with tags to format as a main heading

PHP VARIABLES

Variables in PHP do not have to be declared specifically. A variable simply has to be assigned a value in order to be used.

Variables always start with a **$** symbol followed by the name.

Example:

```
<?php
      $userFirstname = "Fred";
      $userAge = 34;
?>
```

Apart from that, the rules for variable names are the same as for other programming languages you have learned.

Scope – Local v Global

The following code block shows a PHP function. We have declared a variable $count inside this function. This variable is said to be a **local variable** of this function and it is in the local scope of the function block.

```
<?php
      function calculate_count() {
            $count = 5;
            //will print 5; the value of local variable
            echo $count++;
      }
?>
```

Local variables will be destroyed once the end of the code block is reached. Hence the same named variables can be declared within different local scopes.

PHP **global variables** can be defined by using global keyword. If we want to use global variables inside a function, we have to prefix the global keyword with the variable. The following code shows a code block to learn how to use the global keyword with a PHP variable to declared it as a global variable.

```
<?php
	$count = 0;
	function calculate_count() {
		global $count;
		// will print 0 and increment global variable
		echo $count++ . "<br/>";
	}
	calculate_count();
	echo $count;
?>
```

PHP has a predefined **superglobal variable** called **$GLOBALS**. It is an associative array with the name of the variable as key and value as the array element. We can use this array variable to add an array of PHP variables in a global scope.

Let us change the above example with the global keyword by using $GLOBALS superglobal to access the variable in global scope.

```
<?php
	$count = 0;
	function calculate_count() {
		// will print 0 and increment global variable
		declared outside function
		echo $GLOBALS["count"]++ . "<br/>";
	}
	calculate_count();
	echo $count;
?>
```

A **static variable** is again a variable with local scope. But the difference with the regular local variable is that it is not destroyed outside the scope boundary.

A static variable does not lose its value when the program execution goes past the scope boundary. But it can be accessed only within that boundary.

```
<?php
      function counter()
      {
           static $count = 0;
           echo $count;
           $count++;
      }
?>
```

The above counter function has the static variable ‘count’ declared within its local scope.

When the function execution is complete, the static count variable still retains its value for further computation.

PHP CONSTRUCTS

Assignment v Comparison

To assign values to a variable, a single equals symbol is used

```
$userAge = 34;
```

When testing the value of a variable in a condition, a double equals must be used.

```
if($userAge == 34){
```

Selection: IF statement

The IF statement in PHP is written with the following syntax:

```
if (condition){
      true statements
}
else{
      false statements
}
```

Example:

```
<?php
$userAge = 16;
if ($userAge == 16){
      echo "<h1>Welcome!<Center></h1>";
      echo "<p>You are exactly 16 years old</p>";
}
else{
      echo "<h1><Center>Welcome! </h1>";
      echo "<p>You are not 16 years old</p>";
}
?>
```

Repetition: Conditional Loop (While)

The While Loop in PHP is written with the following syntax:

```
while (condition){
      true statements to be repeated
}
```

Example:

```
<?php
$userNum = 1;
while($userNum <= 5) {
      echo "The number is: $userNum <br>";
      $userNum++;
}
```

Repetition: Fixed Loop (For)

The For Loop in PHP is written with the following syntax:

```
for (initialize counter; test counter; increment counter){
      statements to be repeated
}
```

Example:

```
<?php
for($userNum = 0; userNum <=10; userNum++) {
      echo "The number is: $userNum <br>";
}
```

IMPLEMENTATION: PHP & MYSQL

Before a MySQL database can be used, a connection must first be made to the database using its name.

Things you will need:

- The name of your web server (this is *localhost* is our classroom)
- A valid MySQL username and password
- The name of an existing MySQL database (accessible by the above account)
- At least one table containing data (within the database above)

OPEN A CONNECTION TO MYSQL

The connection is written in PHP using the **mysqli_connect** function with the following syntax:

```
$conn = mysqli_connect(servername, username, password, dbname);
```

It is often more convenient to assign these values to variables to be used as parameters with mysqli_connect:

```
$servername = "localhost";

$username = "myusername";

$password = "mypassword";

$dbname="mydatabase";
```

$conn is a variable that stores the result of the attempt to connect to the specified database.

CHECKING THE CONNECTION

If !$conn (not $conn) is true, then the connection failed and a mysqli_connect_error is generated which can be used to inform the user as follows:

```
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
```

If !$conn is false, then the connection has been established and the database can now be used.

Die

Die is used to in the connection check to prevent the php script from progressing further if it fails to establish a connection.

Die can, however, be used for any circumstance in which you want the current script to **halt**.

```
    if(answer != correct){
                die("answer incorrect");
            }
```

In the above example, the script will stop execution if the condition is true.

EXECUTING AN SQL QUERY

SQL can now be used within the PHP script to retrieve a set of results from the connected database.

```
$sql = "SELECT Field1, Field2  FROM `tablename` WHERE criteria";

$result = mysqli_query($conn, $sql);
```

Notice the advantage of using $conn and $sql variables instead of having to key in connection and sql commands repeatedly as parameters.

The **$result** variable will contain the returned data from the mysqli_query function.

SQL criteria can contain the values transferred via POST or GET which should have been stored in an appropriately name variable:

```
$sql = "SELECT Fname, Sname  FROM `people` WHERE Fname = $userName";

$result = mysqli_query($conn, $sql);
```

Note, **"$result ="** is only required when using SQL SELECT statements

CHECKING FOR RESULTS (MYSQLI_NUM_ROWS)

Once the query has been executed and any results are stored in a variable ($result), the data can be checked to ensure at least one record was returned.

```
if (mysqli_num_rows($result) > 0) {

      ***display records***

}

else{
     echo "There were no results matching your search";
     }
```

RETRIEVING RESULTS (MYSQLI_FETCH_ARRAY)

When there are results stored in the $result variable, they can be retrieved using the mysqli_fetch_array command. Each time the mysqli_fetch_array is used, the next row of results (record) is returned.

Mysqli_fetch_array has two parameters. The variable containing the query results and the method to refer to the columns - by column number or associative (field name).

Column Number v Field Name

Using column number means that fields must be referred to using a number value only.

```
$row = mysqli_fetch_array($result, MYSQLI_NUM);

echo $row[0];
echo $row[1];
echo $row[2];
```

Using the associative method, fields can be referred to using their actual field names.

```
$row = mysqli_fetch_array($result, MYSQLI_ASSOC);

echo $row["StudentNo"];
echo $row["StudentName"];
echo $row["Course"];
```

Using a Loop

Using a conditional loop allows for all of the records (rows) returned by the query to be displayed until there are no more records.

```
while($row = mysqli_fetch_array($result)) {

            echo $row["StudentNo"];
            echo $row["StudentName"];
            echo $row["Course"];
            }
```

FORMATTING SQL QUERY RESULTS

It is likely to be desirable that results from a database query are displayed within a table structure.

The database fields can be inserted into rows and columns using the PHP **echo** statement.

```
if (mysqli_num_rows($result) > 0) {

        echo "<table border='1'>";
        echo "<tr>";
        echo "<th>First Name</th>";
        echo "<th>Surname</th>";

        while($row = mysqli_fetch_array($result)) {
          echo "<tr>
                <td>".$row["Fname"]."</td>
                <td>".$row["Sname"]."</td>
                </tr>";
      }
      echo "</table>";
}

else {
      echo "There were no results matching your search";
}
```

Start new table

Add headings to first row

Conditional loop repeats through each record returned by the SQL query.

Echo adds the field contents for each record to a new row.

End table

CLOSING THE CONNECTION

Finally, the connection to the database should be closed using the command below:

```
mysqli_close($conn);
```

IMPLEMENTATION: PHP SESSIONS

When a browser loads a new web page, it forgets all the information from the previous page. A PHP session is a way of storing information within a website, so that it can be retained and used across multiple pages.

Sessions work in a similar way to a program. The website code opens (starts) the session. Information is generated, stored and sometimes changed. The website code then closes (destroys) the session to end it.

Examples of session use are:

- retaining selected items in a shopping cart, as the user navigates from page to page
- displaying a user's id on multiple pages, following a successful login
- retaining values, such as a user's quiz Score, when each new question page loads

STARTING A SESSION

The following PHP function is used to start a session. This should be placed at the **top of a page**, before any HTML code.

If data is being passed between multiple pages, **each page** that requires access to the session should contain the PHP code below.

```
<?php
      // Start a new session
      session_start();
?>
<!DOCTYPE html>
<html>
<head>
```

When a new session starts, a user key is stored on the user's computer. The session_start() function looks to see if a user key exists. If it does, the current session is continued. If no user key exists, a new session is started.

SESSION VARIABLES

Session variables are assigned values, as shown below.

```
$_SESSION['staffLogin']="False";
```

A PHP file that contains session_start() has access to any session variables previously created.

ENDING A SESSION

The PHP function session_destroy() is used to end a session. The code below destroys the session and removes the session variables.

```
session_destroy();
```