

Lab 7: Programming in PHP and Interfacing PHP with MySQL

OBJECTIVES OF THE LAB

This lab aims at the understanding of:

- PHP Basics
 - ✓ PHP Data Types
 - ✓ PHP Loops
 - ✓ PHP Conditionals
 - ✓ PHP Operators
 - Types of MySQL Drivers in PHP
 - Interfacing PHP with MySQL
 - PHP Registration Form using GET,POST Methods
-

7.1 What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

Basic PHP Syntax

A PHP script can be placed anywhere in the document.

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

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

The default file extension for PHP files is `".php"`.

A PHP file normally contains HTML tags, and some PHP scripting code.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the `$` sign, followed by the name of the variable:

```
<?php
$txt = "W3Schools.com";
echo "Hello World"; ?>
```

Semicolons

PHP commands end with a semicolon, like this:

```
$x += 10;
```

One of the most common causes of errors in PHP is forgetting this semicolon. This makes PHP to treat multiple statements like one statement, which it is unable to understand, prompting it to produce a Parse error message.

Example 1 Three Different Types of Variable Assignment

```
1. <?php
2.     $username = "Ali Ahmad";
3.     echo $username;
4.     $mycount = 1;
5.     echo "<br />".$mycount;
6.     $team = array('Maqsood', 'Maryam', 'Adam', 'Naila');
7.     echo "<br />".$team[0]." ".$team[1]." ".$team[2]." ".$team[3];
8. ?>
```

Comments

There are two ways to comment PHP code. The first turns a single line into a comment by preceding it with a pair of forward slashes, like this:

```
// This is a comment
```

This version of the comment feature is a great way to temporarily remove a line of code from a program that is giving you errors. For example, you could use such a comment to hide a debugging line of code until you need it, like this:

```
// echo "X equals $x";
```

You can also use this type of comment directly after a line of code to describe its action, like this:

```
$x += 10; // Increment $x by 10
```

When you need multiple-line comments, there's a second type of comment, which looks like Example 2.

Example 2: A Multiline Comment

1. `<?php`
2. `/* This is a section`
3. `of multiline comments`
4. `which will not be`
5. `interpreted */`
6. `?>`

The `/*` and `*/` pairs of characters can be used to open and close comments almost anywhere inside the code. Most, if not all, programmers use this construct to temporarily comment out entire sections of code that do not work or that, for one reason or another, they do not wish to be interpreted.

Variables

Variables are used for storing values, like text strings, numbers or arrays. When a variable is declared, it can be used over and over again in your script. All variables in PHP start with a `$` sign symbol.

In PHP, a variable does not need to be declared before adding a value to it. In Example 1, you see that you do not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value.

String Variables

String value can be assigned to a variable like this:

```
$username = "Ali Ahmad";
```

The quotation marks indicate that "Ali Ahmad" is a string of characters. You must enclose each string in either quotation marks or apostrophes (single quotes), although there is a subtle difference between the two types of quote, which is explained later. To see the contents, `echo` command can be used:

```
echo $username;
```

Or you can assign it to another variable, like this:

```
$current_user = $username;
```

Numeric Variables

Numeric variables contain numbers. For instance:

```
$mycount = 17;
```

Similarly, a floating-point number (containing a decimal point) can be used; the syntax is the same:

```
$mycount = 17.5;
```

The value of \$mycount can be assigned to another variable or can be echoed to the web browser.

Arrays

Arrays can be created using array() construct. For instance:

```
$team = array('Maqsood', 'Maryam', 'Adam', 'Naila');
```

\$team array contains five strings inside. Each string is enclosed in apostrophes. To know who player 4 is, following command is used:

```
echo $team[3];           // Displays the name Naila
```

The reason the previous statement has the number 3, not 4, is because the first element of a PHP array is actually the zero element, so the player numbers will therefore be 0 through 3.

Variable Naming Conventions

Following rules must be practiced when naming variables:

- 1) A variable name must start with a letter or an underscore "_" -- not a number
- 2) A variable name can only contain alpha-numeric characters, underscores (a-z, A-Z, 0-9, and _)
- 3) 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 DATA TYPES

All data stored in PHP variables fall into one of eight basic categories, known as data types. A variable's data type determines what operations can be carried out on the variable's data, as well as the amount of memory needed to hold the data.

PHP supports four scalar data types. Scalar data means data that contains only a single value. Here's a list of them, including examples:

Scalar Data Type	Description	Example
Integer	A whole number	15
Float	A floating-point number	8.23
String	A series of character	"Hello World"
Boolean	Represents either true or false	true

PHP supports two compound types. Compound data is data that can contain more than one value. The following table describes PHP's compound types:

Compound Data Type	Description
	An ordered map (contains names or numbers mapped to values)
Array	
Object	A type that may contain properties and methods

Finally, PHP supports two special data types, so called because they don't contain scalar or compound data as such, but have a specific meaning:

Special Data Type	Description
Resource	Contains a reference to an external resource, such as a file or database
Null	May only contain null as a values, meaning the variable explicitly does not contain any value

Testing Variable Type

You can determine the type of a variable at any time by using PHP's `gettype()` function. To use `gettype()`, pass in the variable whose type you want to test. The function then returns the variable's type as a string. Example 3 shows `gettype()` in action. A variable is declared, and its type is tested with `gettype()`. Then, four different types of data are assigned to the variable, and the variable's type is retested with `gettype()` each time:

Example 3: Testing Variable Type Using `gettype()`

```
1. $test_var;  
                                     // Displays  
2. echo gettype( $test_var ) . " < br / > ";  "NULL"  
3. $test_var = 15;  
                                     // Displays  
4. echo gettype( $test_var ) . " < br / > ";  "integer"  
5. $test_var = 8.23;  
                                     // Displays  
6. echo gettype( $test_var ) . " < br / > ";  "double"  
7. $test_var = "Hello, world!";  
                                     // Displays  
8. echo gettype( $test_var ) . " < br / > ";  "string"
```

You can also test a variable for a specific data type using PHP's type testing functions:

Function	Description
	Returns true if <i>value</i> is an
is_int(<i>value</i>)	integer
	Returns true if <i>value</i> is a
is_float(<i>value</i>)	float
	Returns true if <i>value</i> is a
is_string(<i>value</i>)	string
	Returns true if <i>value</i> is a
is_bool(<i>value</i>)	Boolean
	Returns true if <i>value</i> is an
is_array(<i>value</i>)	array
	Returns true if <i>value</i> is an
is_object(<i>value</i>)	object
is_resource(<i>value</i>)	Returns true if <i>value</i> is a resource
	Returns true if <i>value</i> is
is_null(<i>value</i>)	null

Setting Variable Type

PHP's `settype()` function is used to change the type of a variable while preserving the variable's value as much as possible. To use `settype()`, pass in the name of the variable you want to alter, followed by the type to change the variable to (in quotation marks).

Example 4 shows `settype()` in action. A variable is declared, and its value is shown. Then, it is converted into four different types of data including string, integer, float, and Boolean using `settype()`. Output of each conversion is also shown.

Example 4: Setting Variable to Different Types

1. `$test_var = 8.23;`
2. `echo $test_var . " < br / > ";` // Displays "8.23"
3. `settype($test_var, "string");`
4. `echo $test_var . " < br / > ";` // Displays "8.23"
5. `settype($test_var, "integer");`
6. `echo $test_var . " < br / > ";` // Displays "8"
7. `settype($test_var, "float");`
8. `echo $test_var . " < br / > ";` // Displays "8"
9. `settype($test_var, "boolean");`
10. `echo $test_var . " < br / > ";` // Displays "1"

Arithmetic and Increment/Decrement Operators

Figure 7.1 and Figure 7.2 shows the arithmetic and increment/decrement operators along with examples and results.

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0

Figure 7.1 – Arithmetic Operators

Operator	Description	Example	Result
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Figure 7.2 – Increment/Decrement Operators

Assignment, Comparison, and Logical Operators

Figure 7.3 , Figure 7.4 and Figure 7.5. shows the assignment, comparison, and logical operators along with examples.

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
=	x=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

Figure 7.3 – Assignment Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
<>	is not equal	5<>8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Figure 7.4 – Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
<>	is not equal	5<>8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

Figure 7.5 – Logical Operators

PHP CONDITIONALS

Conditionals alter program flow. These enables you to ask questions about certain things and respond to the answers you get in different ways. Conditionals are central to dynamic web pages—the goal of using PHP in the first place— because they make it easy to create different output each time a page is viewed. There are three types of non-looping conditionals: the if statement, the switch statement, and the ? Operator By non-looping, it means that the actions initiated by the statement take place and program flow then moves on, whereas looping conditionals execute code over and over until a condition has been met.

Simple Decisions with if Statement

The easiest decision making statement to understand is the if statement. Its basic form is:

Syntax: <pre> if (<i>expression</i>) { / Run this code } / More code here </pre>	Example: <pre> <html> <body> <?php \$d=date("D"); if (\$d=="Fri") echo "Have a nice weekend!"; ?> </body> </html> </pre>
------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------

If the expression inside the parentheses evaluates to true, the code between the braces is run. If the expression evaluates to false, the code between the braces is skipped.

Providing Alternate Decisions with if- else Statement

The if statement allows you to run a block of code if an expression evaluates to true. If the expression evaluates to false, the code is skipped. This process can be enhanced by adding an else statement to an if construction. This lets you run one block of code if an expression is true, and a different block of code if the expression is false.

Syntax: <pre> if (<i>expression</i>) { / Run this code } else { / Run another code } </pre>	Example 1: <pre> <html> <body> <?php \$d=date("D"); if (\$d=="Fri") echo "Have a nice weekend!"; else echo "Have a nice day!"; ?> </body> </html> </pre>	Example 2: <pre> <html> <body> <?php \$d=date("D"); if (\$d=="Fri") echo "Have a nice weekend!"; elseif (\$d=="Sun") </pre>
---------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------

Example 5: Use of Multiple-line if ... elseif... Statements

1. <?php
2. if (\$page == "Home") echo "You selected Home";
3. elseif (\$page == "About") echo "You selected About";
4. elseif (\$page == "News") echo "You selected News";
5. elseif (\$page == "Log in") echo "You selected Login";
6. elseif (\$page == "Links") echo "You selected Links";
7. ?>

Testing One Expression Many Times with Switch Statements

Sometimes you want to test an expression against a range of different values, carrying out a different task depending on the value that is matched. This can be achieved using the switch statements.

Example 6: Use of Switch Statement

```
1. <?php
2.     switch ($page)
3.     {
4.         case "Home":
5.             echo "You selected Home";
6.             break;
7.         case "About":
8.             echo "You selected About";
9.             break;
10.        case "News":
11.            echo "You selected News";
12.            break;
13.        case "Login":
14.            echo "You selected Login";
15.            break;
16.        case "Links":
17.            echo "You selected Links";
18.            break;
19.    }
20. ?>
```

The ? Operator

The ? operator is passed an expression that it must evaluate, along with two statements to execute: one for when the expression evaluates to TRUE, the other for when it is FALSE. Example 7 shows sample use of ? operator.

Example 7: Use of ? Operator

```
1. <?php
2.     echo $fuel <= 1 ? "Fill tank now" : "There's enough fuel";
3. ?>
```

PHP LOOPS

Looping makes scripts more powerful and useful. The basic idea of a loop is to run the same block of code again and again, until a certain condition is met. As with decisions, that condition

must take the form of an expression. If the expression evaluates to true, the loop continues running. If the expression evaluates to false, the loop exits, and execution continues on the first line following the loop's code block.

while Loop

The simplest type of loop to understand uses the while statement. Its syntax is as follows:

```
initialization;
while ( expression ) {
    / increment/decrement
    / Run this code
}
// More code here
```

Here's how it works. The expression inside the parentheses is tested; if it evaluates to true, the code block inside the braces is run. Then the expression is tested again; if it's still true, the code block is run again, and so on. However, if at any point the expression is false, the loop exits and execution continues with the line after the closing brace. Here's a simple, practical example of a while loop:

Example 8: while Loop

```
1. < ?php
2.     $widgetsLeft = 10;
3.     while ( $widgetsLeft > 0 ) {
4.         echo "Selling a widget... ";
5.         $widgetsLeft - - ;
6.         echo "done. There are $widgetsLeft widgets left. < br / > ";
7.     }
8.     echo "We ' re right out of widgets!";
9. ? >
```

In this example, first a variable \$widgetsLeft is created to record the number of widgets in stock (10). Then the while loop works through the widgets, "selling" them one at a time (represented by decrementing the \$widgetsLeft counter) and displaying the number of widgets remaining. Once \$widgetsLeft reaches 0, the expression inside the parentheses (\$widgetsLeft > 0) becomes false , and the loop exits. Control is then passed to the echo() statement outside the loop, and the message " We're right out of widgets! " is displayed.

foreach Loop

foreach is a special kind of looping statement that works only on arrays (and objects). It is used in two ways: One to either retrieve just the value of each element, or to retrieve the element's key and value.

The simplest way to use foreach is to retrieve each element's value, as follows:

```
foreach ( $array as $value ) {

    // (do something with $value here)
}
// (rest of script here)
```

As you might imagine, the foreach loop continues to iterate until it has retrieved all the values in the array, from the first element to the last. On each pass through the loop, the \$value variable gets set to the value of the current element. You can then do whatever you need to do with the value within the loop's code block. Then, the loop repeats again, getting the next value in the array, and so on. Here's an example:

Example 9: Using foreach to Loop through Values

```
1. < ?php
2.     $authors = array( "Steinbeck", "Kafka", "Tolkien", "Dickens" );
3.     foreach ( $authors as $val ) {
4.         echo $val . " < br / > ";
5.     }
6. ? >
```

To use foreach to retrieve both keys and values, use the following syntax:

```
foreach ( $array as $key => $value ) {
    // (do something with $key and/or $value here)
}
// (rest of script here)
```

This behaves exactly like the previous foreach construct; the only difference is that the element's key is also stored in the \$key variable. (Again, you can use any variable names you like; they don't have to be \$key and \$value.)

Example 10: Using foreach to Loop through Keys and Values

```
1. < ?php
2.     $myBook = array( "title" => "The Grapes of Wrath",
3.                     "author" => "John Steinbeck",
4.                     "pubYear" => 1939 );
5.     foreach ( $myBook as $key => $value ) {
6.         echo "Key ".$key." Value ".$value;
7.         echo "<br>";
8.     }
9. ? >
```

do-while Loop

A slight variation to the while loop is the do ... while loop. It is used when you want a block of code to be executed at least once before checking the expression. Its syntax is as follows:

```
initialization;
do {
    / increment/decrement
    / Run this code
} while
( expression ); //
More code here
```

Here's a simple, practical example of a do-while loop:

Example 11: do-while Loop

```
1. < ?php
2.     $width = 1;
3.     $length = 1;
4.     do {
5.         $width++;
6.         $length++;
7.         $area = $width * $length;
8.     } while ( $area < 1000 );
9.     echo "The smallest square over 1000 sq ft in area is $width ft x $length ft.";
10. ? >
```

This example computes the width and height (in whole feet) of the smallest square over 1000 square feet in area (which happens to be 32 feet by 32 feet). It initializes two variables, \$width and \$height, then applies a do...while loop to increment these variables and compute the area of the resulting square, which it stores in \$area variable. Because the loop is always run at least once, you can be sure that \$area will have a value by the time it's checked at the end of the loop. If the area is still less than 1000, the expression evaluates to true and the loop repeats.

for Loop

The final kind of loop statement, the for loop, is also the most powerful, as it combines the abilities to set up variables as you enter the loop, test for conditions while iterating loops, and modify variables after each iteration.

Typically, you use a for loop when you know how many times you want to repeat the loop. You use a counter variable within the for loop to keep track of how many times you've looped. The general syntax of a for loop is as follows:

```
for ( initialization expressions; condition expression; modification
expressions ) {
// Run this code
}
// More code here
```

Here's a simple, practical example of a for loop:

Example 12: for Loop

```
1. <?php
2.     for ($count = 1 ; $count <= 12 ; ++$count)
3.         echo "$count times 12 is " . $count * 12 . "<br>";
4. ?>
```

This example prints the table of 12 using for loop. At the start of the first iteration of the loop, the initialization expression is executed. \$count is initialized to the value 1. Then, each time around the loop, the condition expression (in this case, \$count <= 12) is tested, and the loop is entered only if the condition is TRUE. Finally, at the end of each iteration, the modification expression is executed. Here, the variable \$count is incremented.

Connector

Connector refers to a piece of software that allows your application to connect to the MySQL database server. MySQL provides connectors for a variety of languages, including PHP. If your PHP application needs to communicate with a database server you will need to write PHP code to perform such activities as **connecting to the database server, querying the database and other database-related functions**. Software is required to provide the API that your PHP application will use, and also handle the communication between your application and the database server, possibly using other intermediate libraries where necessary. This software is known generically as a connector, as it allows your application to connect to a database server.

What are the main PHP API offerings for using MySQL?

There are three main API options when considering connecting to a MySQL database server:

- PHP's MySQL Extension
- PHP's MySQLi Extension
- PHP Data Objects (PDO)

mysql() is now obsolete because of security issues like SQL injection etc, but the other two are being actively used.

PHP Data Objects (PDO):

The PDO functions allows you to access MySQL database servers.

```
<?php
$username = "username";
$password = "password";

try {
$conn = new PDO("mysql:host=localhost;dbname=myDB", $username,
$password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>
```

try block: This block contains the code that may potentially throw an exception.

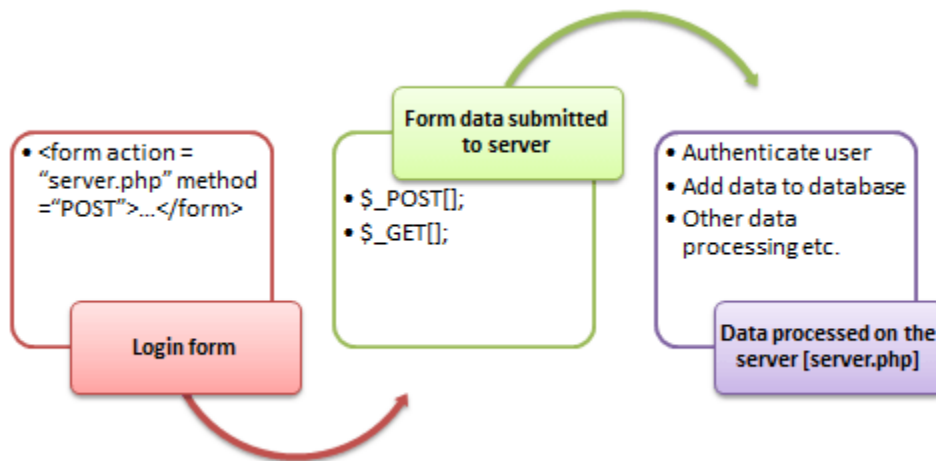
- `new PDO(. . .)` line: This line creates a new PDO object, establishing a connection to the MySQL database. The arguments passed to the PDO constructor are the Data Source Name (DSN), username, and password.
- `setAttribute(. . .)` line: This line sets the error handling mode for the PDO connection object. `PDO::ATTR_ERRMODE` is set to `PDO::ERRMODE_EXCEPTION`, which means that PDO will throw exceptions if errors occur.
- `catch(PDOException $e)` block: This block catches any exceptions thrown by the code inside the try block. If an exception occurs, it prints an error message containing the exception message using `$e->getMessage()`.

PHP Registration Form using GET, POST Methods

When you login into a website or into your mail box, you are interacting with a form.

Forms are used to get input from the user and submit it to the web server for processing.

The diagram below illustrates the form handling process.



A form is an HTML tag that contains graphical user interface items such as input box, check boxes radio buttons etc.

The form is defined using the `<form>...</form>` tags and GUI items are defined using form elements such as input.

When and why we are using forms?

- Forms come in handy when developing flexible and dynamic applications that accept user input.
- Forms can be used to edit already existing data from the database

Create a form

We will use HTML tags to create a form. Below is the minimal list of things you need to create a form.

- Opening and closing form tags `<form>...</form>`
- Form submission type POST or GET
- Submission URL that will process the submitted data
- Input fields such as input boxes, text areas, buttons, checkboxes etc.

```
<html>
<head>
  <title>Registration Form</title>
</head>
<body>

  <h2>Registration Form</h2>

  <form action="registration_form.php" method="POST"> First name:
```



```

        <input type="text" name="firstname"> <br> Last name:

        <input type="text" name="lastname">

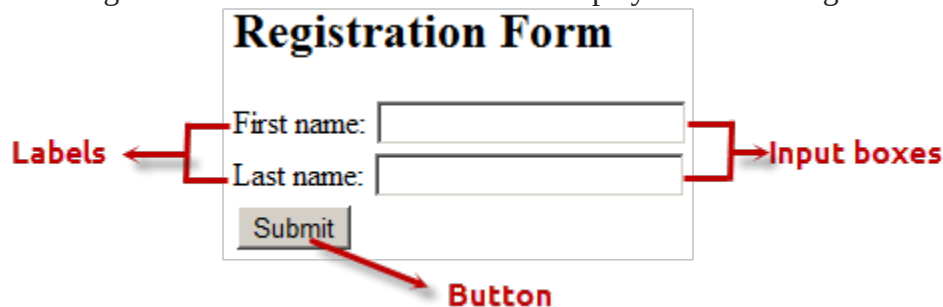
        <input type="hidden" name="form_submitted" value="1" />

        <input type="submit" value="Submit">

    </form>
</body>
</html>

```

Viewing the above code in a web browser displays the following form.



HERE,

- `<form...>...</form>` are the opening and closing form tags
- `action="registration_form.php" method="POST">` specifies the destination URL and the submission type.
- First/Last name: are labels for the input boxes
- `<input type="text"...>` are input box tags
- `
` is the new line tag
- `<input type="hidden" name="form_submitted" value="1"/>` is a hidden value that is used to check whether the form has been submitted or not
- `<input type="submit" value="Submit">` is the button that when clicked submits the form to the server for processing

Submitting the form data to the server

The action attribute of the form specifies the submission URL that processes the data. The method attribute specifies the submission type.

PHP POST method

- This is the built in PHP super global array variable that is used to get values submitted via HTTP POST method.
- The array variable can be accessed from any script in the program; it has a global scope.
- This method is ideal when you do not want to display the form post values in the URL.
- A good example of using post method is when submitting login details to the server.

It has the following syntax.

```
<?php
$_POST['variable_name'];
?>
```

HERE,

- “\$_POST[...]” is the PHP array
- “variable_name” is the URL variable name.

PHP GET method

- This is the built in PHP super global array variable that is used to get values submitted via HTTP GET method.
- The array variable can be accessed from any script in the program; it has a global scope.
- This method displays the form values in the URL.
- It's ideal for search engine forms as it allows the users to bookmark the results.

It has the following syntax.

```
<?php
$_GET['variable_name'];
?>
```

HERE,

- “\$_GET[...]” is the PHP array
- “variable_name” is the URL variable name.

GET vs POST Methods

POST	GET
Values not visible in the URL	Values visible in the URL
Has not limitation of the length of the values since they are submitted via the body of HTTP	Has limitation on the length of the values usually 255 characters. This is because the values are displayed in the URL. Note the upper limit of the characters is dependent on the browser.
Has lower performance compared to Php_GET method due to time spent encapsulation the Php_POST values in the HTTP body	Has high performance compared to POST method dues to the simple nature of appending the values in the URL.
Supports many different data types such as string, numeric, binary etc.	Supports only string data types because the values are displayed in the URL
Results cannot be book marked	Results can be book marked due to the visibility of the values in the URL

The below diagram shows the difference between get and post

FORM SUBMISSION POST METHOD

```
<form action="registration_form.php" method="POST">
  First name: <input type="text" name="firstname"><br>
  Last name: <input type="text" name="lastname">
  <br>
  <input type="hidden" name="form_submitted" value="1"/>
  <input type="submit" value="Submit">
</form>
```

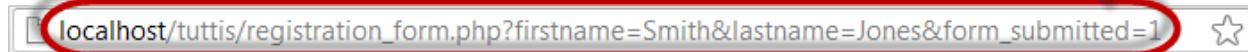
Submission URL does not show form values

localhost/tuttis/registration_form.php

FORM SUBMISSION GET METHOD

```
<form action="registration_form.php" method="GET">
  First name: <input type="text" name="firstname"><br>
  Last name: <input type="text" name="lastname">
  <br>
  <input type="hidden" name="form_submitted" value="1"/>
  <input type="submit" value="Submit">
</form>
```

SUBMISSION URL SHOWS FORM VALUES



Processing the registration form data

The registration form submits data to itself as specified in the action attribute of the form.

When a form has been submitted, the values are populated in the `$_POST` super global array.

We will use the PHP `isset` function to check if the form values have been filled in the `$_POST` array and process the data.

We will modify the registration form to include the PHP code that processes the data. Below is the modified code

```
<html>
<head>
  <title>Registration Form</title>
</head>
<body>

  <?php if (isset($_POST['form_submitted'])) {
    //this code is executed when the form is submitted
    ?>

    <h2>Thank You <?php echo $_POST['firstname']; ?> </h2>

    <p>You have been registered as
      <?php echo $_POST['firstname'] . ' ' . $_POST['lastname']; ?>
    </p>

    <p>Go <a href="/registration_form.php">back</a> to the form</p>

    <?php }
    else { ?>

    <h2>Registration Form</h2>
```

```

        <form action="registration_form.php" method="POST">

            First name:
            <input type="text" name="firstname">

            <br> Last name:
            <input type="text" name="lastname">

        <input type="hidden" name="form_submitted" value="1" />

        <input type="submit" value="Submit">

    </form>

    <?php } ?>
</body>
</html>

```

HERE,

- <?php if (isset(\$_POST['form_submitted'])): ?> checks if the form_submitted hidden field has been filled in the \$_POST[] array and display a thank you and first name message.

If the form_fobmitted field hasn't been filled in the \$_POST[] array, the form is displayed.

Exercises

7.1 Write a script to find sum of the numbers given as a string.

\$input = "1,2,3,4,5,6,7";

7.2 Remove the last occurrence of "89" from the following string.

\$input = "A89C89";

7.3 Create a form which contains the following field as shown in below figure.

User should be able to submit the form. (using GET method). Upon submitting the form these submitted fields must be inserted in the themepark (database) that we have used throughout the labs.

Theme Park Registration

Reference:

<https://www.guru99.com/php-forms-handling.html>