Lab # 05: Programming in PHP

OBJECTIVES OF THE LAB

This lab aims at the understanding of:

- *PHP Basics*
- *PHP Data Types*
- *PHP Expressions*
- *PHP Operators*
- *PHP Conditionals*
- *PHP Loops*

PHP BASICS

This section of the lab covers PHP syntax, comments, and variables.

Basic Syntax

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.

The $ symbol
In PHP, $ must be placed in front of all variables. This is required to make the PHP parser faster, as it instantly knows whenever it comes across a variable. Whether variables are numbers, strings, or arrays, they should all look something like those in Example 1.

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 characters	"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
Array	An ordered map (contains names or numbers mapped to values)
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;
2. echo gettype( $test_var ) . " < br / > ";        // Displays "NULL"
3. $test_var = 15;p
4. echo gettype( $test_var ) . " < br / > ";        // Displays "integer"
5. $test_var = 8.23;
6. echo gettype( $test_var ) . " < br / > ";        // Displays "double"
7. $test_var = "Hello, world!";
8. echo gettype( $test_var ) . " < br / > ";        // Displays "string"
```

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

Function	Description
is_int(*value*)	Returns **true** if *value* is an integer
is_float(*value*)	Returns **true** if *value* is a float
is_string(*value*)	Returns **true** if *value* is a string
is_bool(*value*)	Returns **true** if *value* is a Boolean
is_array(*value*)	Returns **true** if *value* is an array
is_object(*value*)	Returns **true** if *value* is an object
is_resource(*value*)	Returns **true** if *value* is a resource
is_null(*value*)	Returns **true** if *value* is 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"
```

----------Task 5.1----------

In contrast to settype(), there is another method that causes a variable's value to be treated as a specific type. It is known as type casting. Note that the variable itself remains unaffected during type casting. Consider the following variable:

- $test_var = 8.23;

Type cast this variable's value to integer, string, and Boolean and show result using echo.

----------Task 5.2----------

Use and Specify the purpose of following functions:

1) intval(value)
2) floatval(value)
3) strval(value)

PHP EXPRESSIONS

An expression in PHP is anything that evaluates to a value; this can be any combination of values, variables, operators, and functions. Here are some examples of expressions:

- $x + $y + $z
- $x
- 5

- gettype($test_var)

Literals

The simplest form of an expression is a literal, which simply means something that evaluates to itself, such as the number 73 or the string "Hello". The number 5 above is an example of literal.

PHP OPERATORS

Operators in PHP can be grouped into ten types, as show in Table 5.1.

TABLE 5.1 PHP OPERATORS

Type	Description
Arithmetic	Perform common arithmetic operations, such as addition and subtraction
Assignment	Assign values to variables
Bitwise	Perform operations on individual bits in an integer
Comparison	Compare values in a Boolean fashion
Error Control	Affect error handling
Execution	Cause execution of commands as though they are shell commands
Incrementing/Decrementing	Increment or decrement a variable's value
Logical	Boolean operators such as and, or, and not
String	Concatenates strings
Array	Perform operations on arrays

Arithmetic and Increment/Decrement Operators

Figure 5.1 and Figure 5.2 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 5.1 – Arithmetic Operators

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

Figure 5.2 – Increment/Decrement Operators

Assignment, Comparison, and Logical Operators

Figure 5.3, Figure 5.4 and Figure 5.5 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 5.3 – 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 5.4 – 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 5.5 – 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. It basic form is:

Syntax:

- if (*expression*) {
  // Run this code
  }
- // More code here

Example:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri") echo "Have a nice weekend!";
?>

</body>
</html>
```

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:

- if (*expression*) {
  // Run this code
  }
  else {
  // Run another code
  }

Example 1:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

Example 2:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

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

```
<?php
    if ($page == "Home") echo "You selected Home";
    elseif ($page == "About") echo "You selected About";
    elseif ($page == "News") echo "You selected News";
    elseif ($page == "Login") echo "You selected Login";
    elseif ($page == "Links") echo "You selected Links";
?>
```

Testing One Expressions 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

```
<?php
    switch ($page)
    {
        case "Home":
            echo "You selected Home";
            break;
        case "About":
            echo "You selected About";
            break;
        case "News":
            echo "You selected News";
            break;
        case "Login":
            echo "You selected Login";
            break;
        case "Links":
            echo "You selected Links";
            break;
    }
?>
```

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 “ < dt > $key < /dt > ”;
7.          echo “ < dd > $value < /dd > ”;
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 2 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.

----------Task 5.3----------

Write PHP script that shows the division table displayed as in Table 5.2 using different loops. For each number, display whether that number is an odd or even number, and also display a message if the number is a prime number. Display this information within an HTML table.

----------Task 5.4----------

Explore PHP Object Oriented using examples showing classes, objects, inheritance, and polymorphism.

----------Task 5.5----------

Build an image gallery website using PHP without database. Your website must show all the images in a given directory. Use bootstrap/CSS in your website. Please note that when a given image is clicked, it enlarges it and shows in higher resolution. Also, your website should be

flexible enough to show the images when number of images are increased or decreased from a given directory.

A division table

	1	2	3	4	5	6	7	8	9	10
1	1.000	0.500	0.333	0.250	0.200	0.167	0.143	0.125	0.111	0.100
2	2.000	1.000	0.667	0.500	0.400	0.333	0.286	0.250	0.222	0.200
3	3.000	1.500	1.000	0.750	0.600	0.500	0.429	0.375	0.333	0.300
4	4.000	2.000	1.333	1.000	0.800	0.667	0.571	0.500	0.444	0.400
5	5.000	2.500	1.667	1.250	1.000	0.833	0.714	0.625	0.556	0.500
6	6.000	3.000	2.000	1.500	1.200	1.000	0.857	0.750	0.667	0.600
7	7.000	3.500	2.333	1.750	1.400	1.167	1.000	0.875	0.778	0.700
8	8.000	4.000	2.667	2.000	1.600	1.333	1.143	1.000	0.889	0.800
9	9.000	4.500	3.000	2.250	1.800	1.500	1.286	1.125	1.000	0.900
10	10.000	5.000	3.333	2.500	2.000	1.667	1.429	1.250	1.111	1.000

Table 5.2 – Division Table for Task 5.3

----------Task 5.6----------

Tutorial on PHP MVC Framework: LARAVEL.

a) Install Composer: https://getcomposer.org/download/

b) composer create-project --prefer-dist laravel/laravel testProject

```
C:\wamp\www>composer create-project --prefer-dist laravel/laravel testProject
Installing laravel/laravel (v5.2.31)
  - Installing laravel/laravel (v5.2.31): Downloading (100%)
Created project in testProject
> php -r "copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 56 installs, 0 updates, 0 removals
  - Installing vlucas/phpdotenv (v2.4.0): Downloading (100%)
  - Installing symfony/polyfill-mbstring (v1.7.0): Downloading (100%)
  - Installing symfony/var-dumper (v3.0.9): Downloading (100%)
  - Installing symfony/translation (v3.0.9): Downloading (100%)
  - Installing symfony/routing (v3.0.9): Downloading (100%)
  - Installing symfony/process (v3.0.9): Downloading (100%)
  - Installing symfony/polyfill-util (v1.7.0): Downloading (100%)
  - Installing symfony/polyfill-php56 (v1.7.0): Downloading (100%)
  - Installing paragonie/random_compat (v1.4.3): Downloading (100%)
  - Installing symfony/http-foundation (v3.0.9): Downloading (100%)
  - Installing symfony/event-dispatcher (v3.4.8): Downloading (100%)
  - Installing psr/log (1.0.2): Downloading (100%)
  - Installing symfony/debug (v3.0.9): Downloading (100%)
  - Installing symfony/http-kernel (v3.0.9): Downloading (100%)
  - Installing symfony/finder (v3.0.9): Downloading (100%)
  - Installing symfony/console (v3.0.9): Downloading (100%)
  - Installing swiftmailer/swiftmailer (v5.4.9): Downloading (100%)
  - Installing jakub-onderka/php-console-color (0.1): Downloading (100%)
  - Installing jakub-onderka/php-console-highlighter (v0.3.2): Downloading (100%)
  - Installing dnoegel/php-xdg-base-dir (0.1): Downloading (100%)
  - Installing nikic/php-parser (v2.1.1): Downloading (100%)
  - Installing psy/psysh (v0.7.2): Downloading (100%)
```

```
  - Installing nesbot/carbon (1.27.0): Downloading (100%)
  - Installing mtdowling/cron-expression (v1.2.1): Downloading (100%)
  - Installing monolog/monolog (1.23.0): Downloading (100%)
  - Installing league/flysystem (1.0.44): Downloading (100%)
  - Installing jeremeamia/superclosure (2.4.0): Downloading (100%)
  - Installing doctrine/inflector (v1.1.0): Downloading (100%)
  - Installing classpreloader/classpreloader (3.2.0): Downloading (100%)
  - Installing laravel/framework (v5.2.45): Downloading (100%)
  - Installing fzaninotto/faker (v1.7.1): Downloading (100%)
  - Installing hamcrest/hamcrest-php (v1.2.2): Downloading (100%)
  - Installing mockery/mockery (0.9.9): Downloading (100%)
  - Installing symfony/yaml (v3.3.16): Downloading (100%)
  - Installing sebastian/version (1.0.6): Downloading (100%)
  - Installing sebastian/global-state (1.1.1): Downloading (100%)
  - Installing sebastian/recursion-context (1.0.5): Downloading (100%)
  - Installing sebastian/exporter (1.2.2): Downloading (100%)
  - Installing sebastian/environment (1.3.8): Downloading (100%)
  - Installing sebastian/diff (1.4.3): Downloading (100%)
  - Installing sebastian/comparator (1.2.4): Downloading (100%)
  - Installing doctrine/instantiator (1.0.5): Downloading (100%)
  - Installing phpunit/php-text-template (1.2.1): Downloading (100%)
  - Installing phpunit/phpunit-mock-objects (2.3.8): Downloading (100%)
  - Installing phpunit/php-timer (1.0.9): Downloading (100%)
  - Installing phpunit/php-file-iterator (1.4.5): Downloading (100%)
  - Installing phpunit/php-token-stream (1.4.12): Downloading (100%)
  - Installing phpunit/php-code-coverage (2.2.4): Downloading (100%)
  - Installing webmozart/assert (1.3.0): Downloading (100%)
  - Installing phpdocumentor/reflection-common (1.0.1): Downloading (100%)
  - Installing phpdocumentor/type-resolver (0.3.0): Downloading (100%)
  - Installing phpdocumentor/reflection-docblock (3.2.2): Downloading (100%)
  - Installing phpspec/prophecy (1.7.6): Downloading (100%)
  - Installing phpunit/phpunit (4.8.36): Downloading (100%)
  - Installing symfony/css-selector (v3.0.9): Downloading (100%)
  - Installing symfony/dom-crawler (v3.0.9): Downloading (100%)
symfony/var-dumper suggests installing ext-symfony_debug ()
symfony/translation suggests installing symfony/config ()
symfony/routing suggests installing doctrine/annotations (For using the annotation loader)
symfony/routing suggests installing symfony/config (For using the all-in-one router or any loader)
symfony/routing suggests installing symfony/dependency-injection (For loading routes from a service)
symfony/routing suggests installing symfony/expression-language (For using expression matching)
paragonie/random_compat suggests installing ext-libsodium (Provides a modern crypto API that can be used to ge
symfony/event-dispatcher suggests installing symfony/dependency-injection ()
symfony/http-kernel suggests installing symfony/browser-kit ()
symfony/http-kernel suggests installing symfony/class-loader ()
symfony/http-kernel suggests installing symfony/config ()
symfony/http-kernel suggests installing symfony/dependency-injection ()
psy/psysh suggests installing ext-pcntl (Enabling the PCNTL extension makes PsySH a lot happier :))
psy/psysh suggests installing ext-posix (If you have PCNTL, you'll want the POSIX extension as well.)
psy/psysh suggests installing ext-readline (Enables support for arrow-key history navigation, and showing and
psy/psysh suggests installing ext-pdo-sqlite (The doc command requires SQLite to work.)
monolog/monolog suggests installing aws/aws-sdk-php (Allow sending log messages to AWS services like DynamoDB)
monolog/monolog suggests installing doctrine/couchdb (Allow sending log messages to a CouchDB server)
monolog/monolog suggests installing ext-amqp (Allow sending log messages to an AMQP server (1.0+ required))
monolog/monolog suggests installing ext-mongo (Allow sending log messages to a MongoDB server)
monolog/monolog suggests installing graylog2/gelf-php (Allow sending log messages to a GrayLog2 server)
monolog/monolog suggests installing mongodb/mongodb (Allow sending log messages to a MongoDB server via PHP Dr
monolog/monolog suggests installing php-amqplib/php-amqplib (Allow sending log messages to an AMQP server usin
monolog/monolog suggests installing php-console/php-console (Allow sending log messages to Google Chrome)
monolog/monolog suggests installing rollbar/rollbar (Allow sending log messages to Rollbar)
monolog/monolog suggests installing ruflin/elastica (Allow sending log messages to an Elastic Search server)
monolog/monolog suggests installing sentry/sentry (Allow sending log messages to a Sentry server)
league/flysystem suggests installing league/flysystem-aws-s3-v2 (Allows you to use S3 storage with AWS SDK v2)
league/flysystem suggests installing league/flysystem-aws-s3-v3 (Allows you to use S3 storage with AWS SDK v3)
league/flysystem suggests installing league/flysystem-azure (Allows you to use Windows Azure Blob storage)
league/flysystem suggests installing league/flysystem-cached-adapter (Flysystem adapter decorator for metadata c
league/flysystem suggests installing league/flysystem-eventable-filesystem (Allows you to use EventableFilesyste
league/flysystem suggests installing league/flysystem-rackspace (Allows you to use Rackspace Cloud Files)
league/flysystem suggests installing league/flysystem-sftp (Allows you to use SFTP server storage via phpseclib)
league/flysystem suggests installing league/flysystem-webdav (Allows you to use WebDAV storage)
league/flysystem suggests installing league/flysystem-ziparchive (Allows you to use ZipArchive adapter)
league/flysystem suggests installing spatie/flysystem-dropbox (Allows you to use Dropbox storage)
league/flysystem suggests installing srmklive/flysystem-dropbox-v2 (Allows you to use Dropbox storage for PHP 5
laravel/framework suggests installing aws/aws-sdk-php (Required to use the SQS queue driver and SES mail driver
laravel/framework suggests installing doctrine/dbal (Required to rename columns and drop SQLite columns (~2.4).)
laravel/framework suggests installing guzzlehttp/guzzle (Required to use the Mailgun and Mandrill mail drivers a
|~6.0).)
```

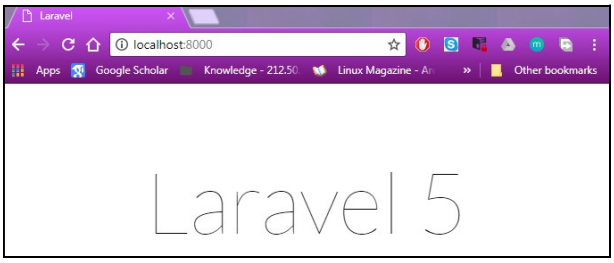
```
laravel/framework suggests installing league/flysystem-aws-s3-v3 (Required to use the Flysystem S3 driver (~1.0)
laravel/framework suggests installing league/flysystem-rackspace (Required to use the Flysystem Rackspace driver
laravel/framework suggests installing pda/pheanstalk (Required to use the beanstalk queue driver (~3.0).)
laravel/framework suggests installing predis/predis (Required to use the redis cache and queue drivers (~1.0).)
laravel/framework suggests installing pusher/pusher-php-server (Required to use the Pusher broadcast driver (~2.
laravel/framework suggests installing symfony/psr-http-message-bridge (Required to use psr7 bridging features (0
sebastian/global-state suggests installing ext-uopz (*)
phpunit/phpunit suggests installing phpunit/php-invoker (~1.1)
Writing lock file
Generating autoload files
> Illuminate\Foundation\ComposerScripts::postUpdate
> php artisan optimize
Generating optimized class loader
> php artisan key:generate
Application key [base64:mrlebVtD3XR2gRSt2ke7OgAsXkFBTRF5CkxS4OBjxkg=] set successfully.
```

c)

```
C:\wamp\www>cd testProject
```

d)

```
C:\wamp\www\testProject>php artisan serve
Laravel development server started on http://localhost:8000/
```



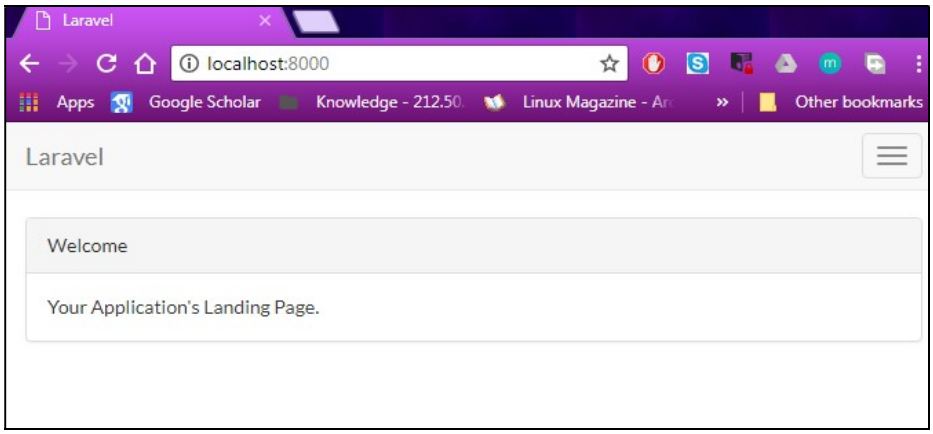
e) Now press CTRL+C to exit.

f)

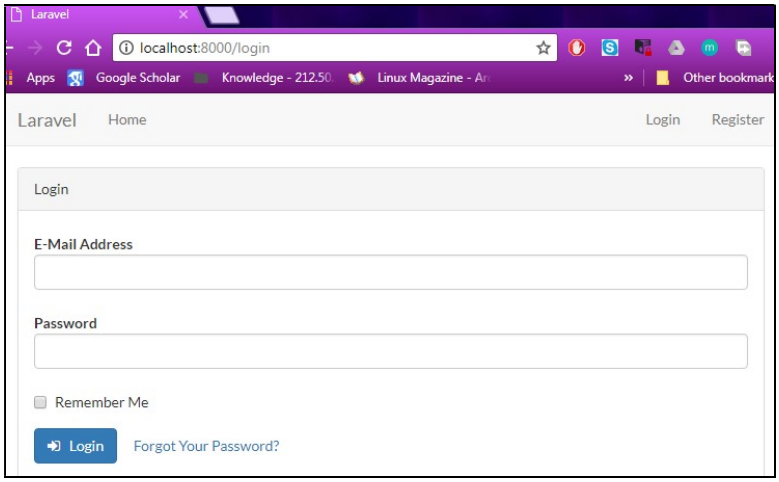
```
^C
C:\wamp\www\testProject>php artisan make:auth
Created View: C:\wamp\www\testProject\resources/views/auth/login.blade.php
Created View: C:\wamp\www\testProject\resources/views/auth/register.blade.php
Created View: C:\wamp\www\testProject\resources/views/auth/passwords/email.blade.php
Created View: C:\wamp\www\testProject\resources/views/auth/passwords/reset.blade.php
Created View: C:\wamp\www\testProject\resources/views/auth/emails/password.blade.php
Created View: C:\wamp\www\testProject\resources/views/layouts/app.blade.php
Created View: C:\wamp\www\testProject\resources/views/home.blade.php
Created View: C:\wamp\www\testProject\resources/views/welcome.blade.php
Installed HomeController.
Updated Routes File.
Authentication scaffolding generated successfully!
```

g)

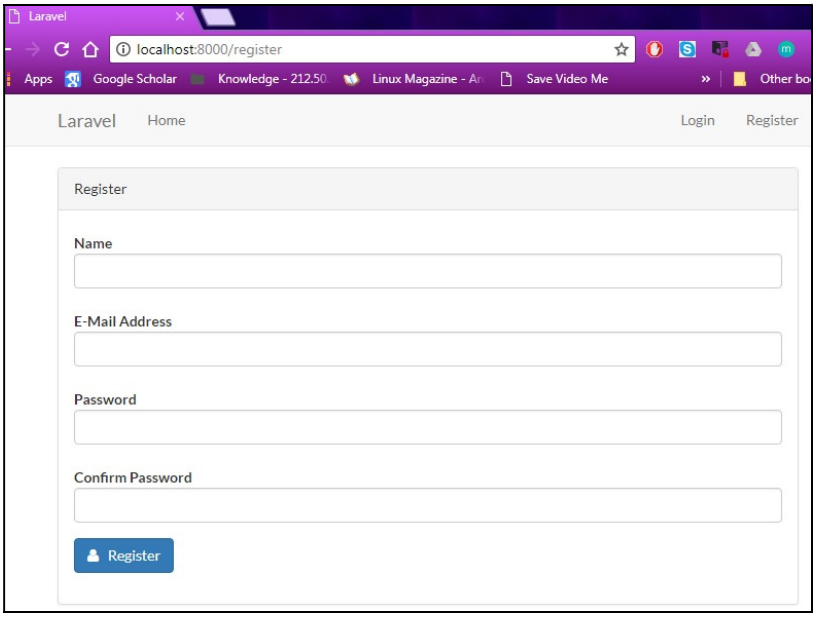
```
C:\wamp\www\testProject>php artisan serve
Laravel development server started on http://localhost:8000/
```



h) Press Home.



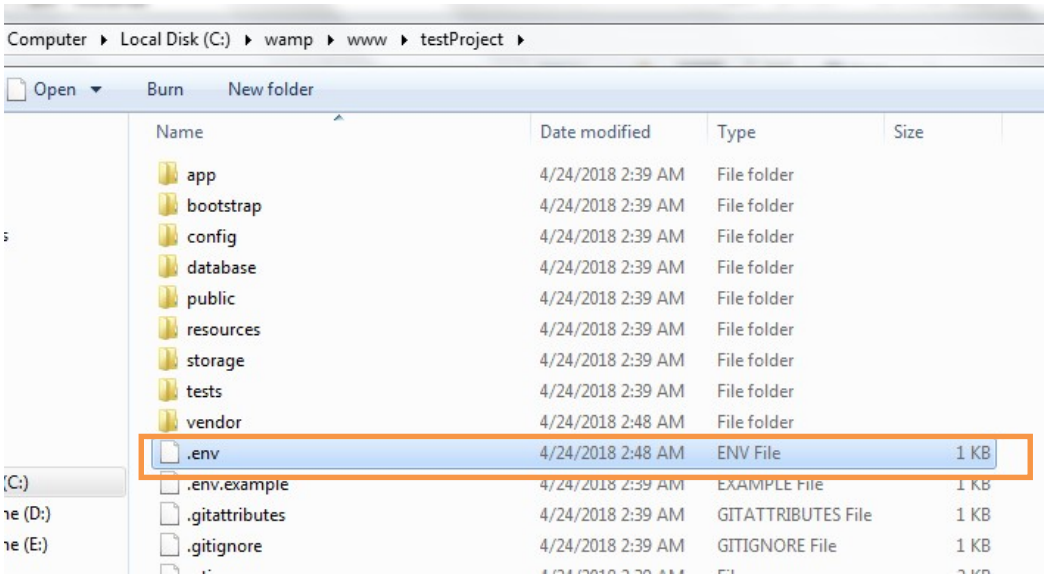
i) Press Register.



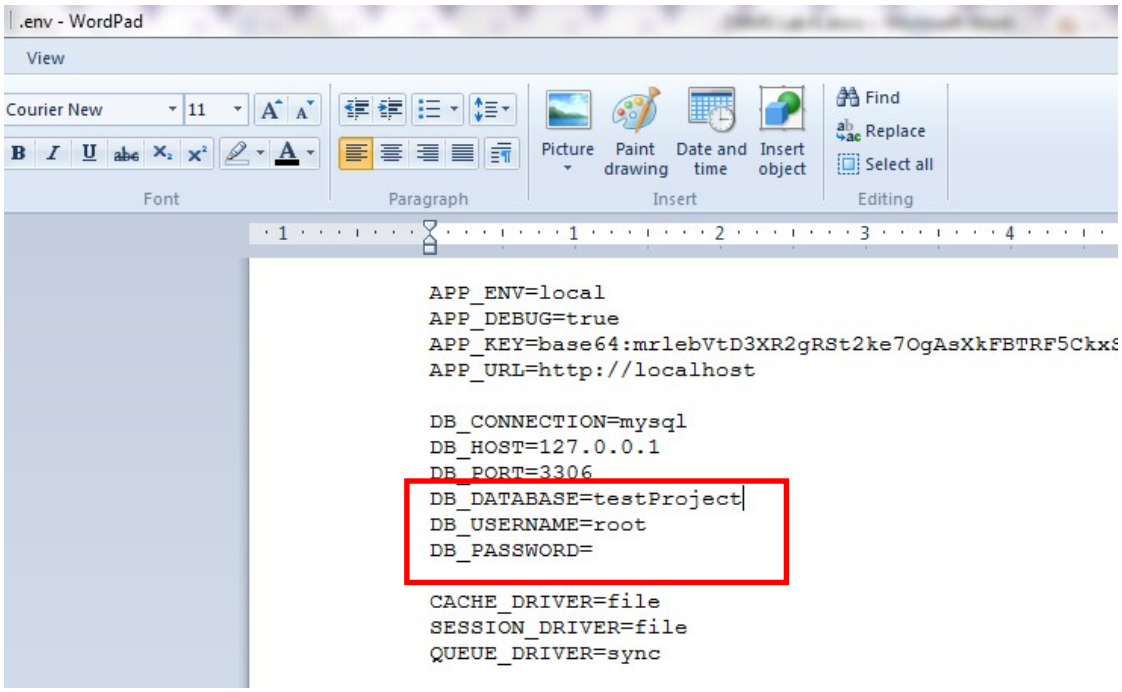
j) Next create new database in MySQL:

```
mysql> create database testProject;
Query OK, 1 row affected (0.04 sec)
```

k) Next open .env file.



l) Perform the following changes in .env file and save the file.



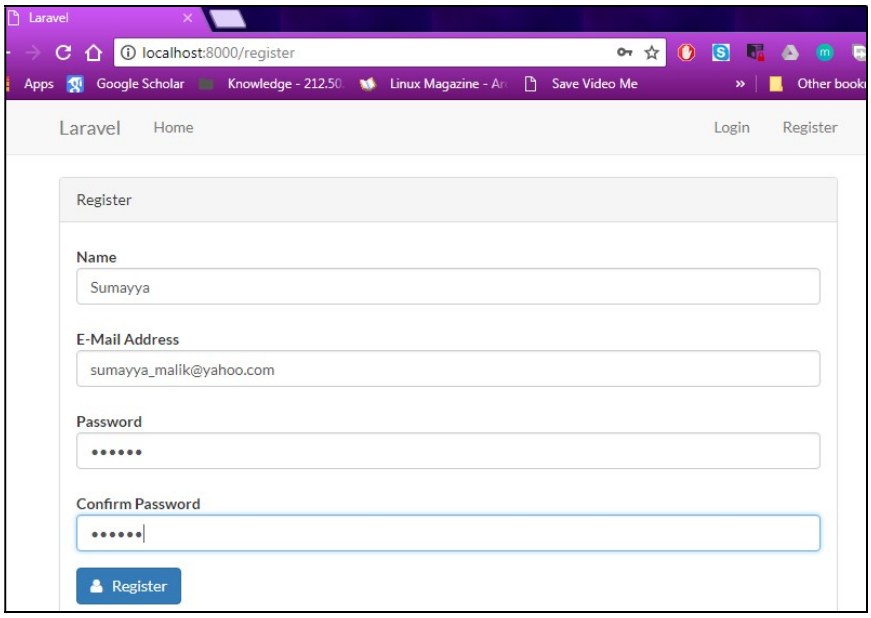
m) Next press CTRL+C in CMD and type **php artisan migrate**. Press Enter. This will create tables in testProject database created earlier.

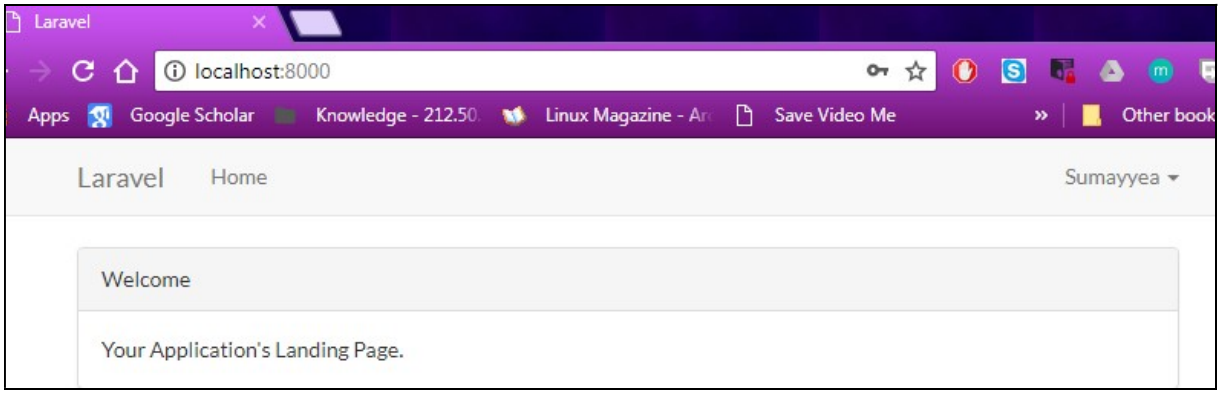
```
^C
C:\wamp\www\testProject>php artisan migrate
Migration table created successfully.
Migrated: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_100000_create_password_resets_table

C:\wamp\www\testProject>_
```

```
mysql> use testProject;
Database changed
mysql> show tables;
+-----------------------+
| Tables_in_testproject |
+-----------------------+
| migrations            |
| password_resets       |
| users                 |
+-----------------------+
3 rows in set (0.00 sec)
```

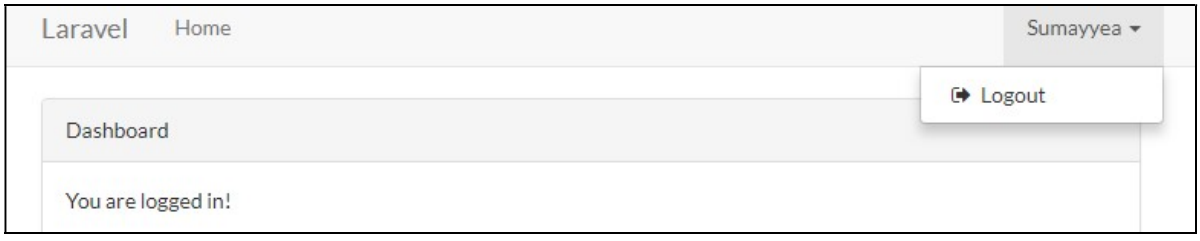
```
mysql> describe users;
+----------------+------------------+------+-----+---------+----------------+
| Field          | Type             | Null | Key | Default | Extra          |
+----------------+------------------+------+-----+---------+----------------+
| id             | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name           | varchar(255)     | NO   |     | NULL    |                |
| email          | varchar(255)     | NO   | UNI | NULL    |                |
| password       | varchar(255)     | NO   |     | NULL    |                |
| remember_token | varchar(100)     | YES  |     | NULL    |                |
| created_at     | timestamp        | YES  |     | NULL    |                |
| updated_at     | timestamp        | YES  |     | NULL    |                |
+----------------+------------------+------+-----+---------+----------------+
7 rows in set (0.10 sec)

mysql> select * from users;
Empty set (0.00 sec)
```

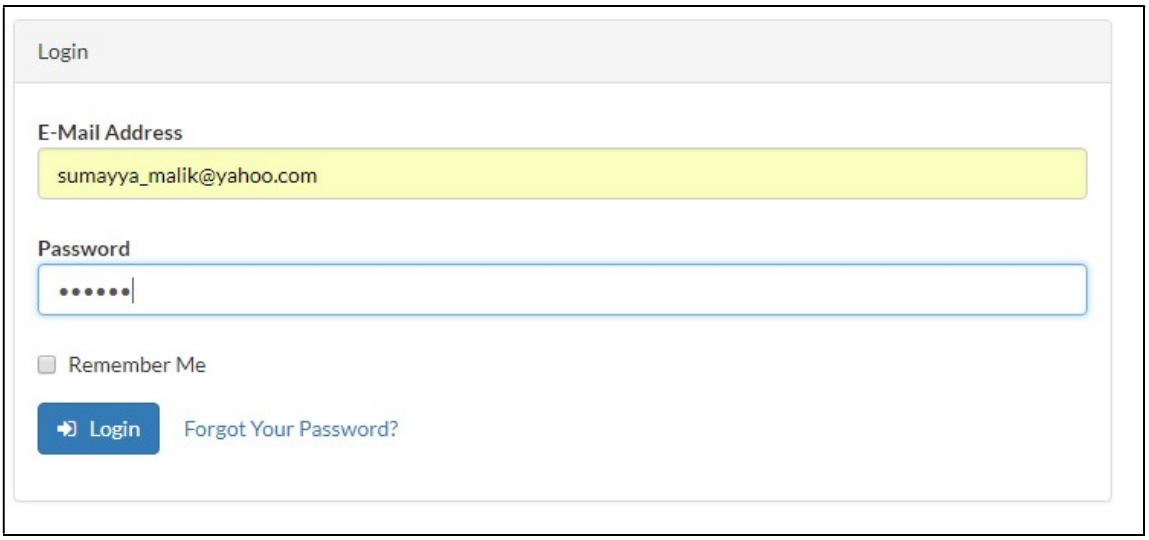
n) Run the server as shown in step g. Register a new user.





o) Click Sumayyea/Your name to logout. Now Login for Sumayyea/Your Name.





----------Task 5.7----------

Implement the following relationship in Laravel.

Department: depID, depName
Emplyee: empID, empName, empJob, dID

Note that the department has at least one or many employees working in it; while employee works exactly in one department. Also, feed atleast five records in the created tables.