University of Global Village (UGV), Barishal

Web Page Design and Development (Level-3)

Amartya Kundu Durjoy

Lecturer

Department Of CSE

University Of Global Village, Barishal

Course Code	--
Name of Course Title	Web page design and development (Level-1)
Course Type	Skill Course
Level	3rd Semester
Academic Session	Winter 2025
Name(s) of Academic Course teacher(s)	Amartya Kundu Durjoy, Lecturer, CSE. Mobile: 0176638973 E-mail:amartyakundudurjoy@gmail.com
Consultation Hours:	

Course Code:--	Credits: 1
Credit Hours: 85	CIE Marks:20
Course for 1stSemester, Bachelor of Science in Computer Science Engineering (CSE)	SEE Marks:30

Course Learning Outcome (CLO) at the end of the course, the students will be able to-

Describe	CLO 1: Describe the core concepts and principles of web development, including client-server architecture, HTTP/HTTPS protocols, web standards, and the role of PHP, ASP, and front-end technologies in full-stack development.
Analyze and design	CLO 2: Analyze and design the structure of web applications using front-end technologies (HTML, CSS, JavaScript) and back-end scripting languages (PHP, ASP), ensuring effective user interfaces and robust server-side logic.
Create and develop	CLO 3: Create and develop dynamic and interactive web applications that utilize a range of web development technologies and frameworks and meet specified functional and non-functional requirements.
Apply	CLO 4: Apply knowledge of web development best practices and principles to optimize the performance, usability, accessibility, and security of web applications.
Identify and evaluate	CLO 5: Identify and evaluate emerging trends and technologies in web development and demonstrate an ability to learn and adapt to new tools, frameworks, and programming paradigms.

Lab topics

Week	Topics	Teaching-Learning Strategy(s)	Class Hour	Practice Hour	Assessment Strategy(s)	Mapping with CLO
01	Introduction to PHP/ASP on Web Development: Understanding the process of web development, from planning, designing, coding, testing, to deployment.	Lecture, Demonstration, Interactive Q&A	5h	2h	Participation, Short Quiz	CLO 1 & CLO 2
02	Data Types, Operators, and Control Structures: Understanding PHP Data Types, Strings, Numbers, and Type Casting	Lecture, Hands-on Exercises with PHP/ASP	5h	3h	Lab Performance, Practical Exercise	CLO 2
03 & 04	Functions, Arrays, and Superglobals: Understanding PHP Functions, Arrays, and Superglobals	Lecture, Hands-on Exercises	5h	4h	Lab Participation, Practical Assignment	CLO 2 & CLO 3
05	PHP Forms and Advanced: Understanding form handling, validation, PHP date & time function	Hands-on sessions with , Coding tutorials	5h	5h	Lab Performance, Coding Test	CLO 2, CLO 3 & CLO 4
06	Project Presentation: Ability to effectively communicate web design project designs and	Presentation preparation and peer review	5h	2h	Lab Performance Assessment	CLO 3 & CLO 5

Lab topics

Week	Topics	Teaching-Learning Strategy(s)	Class Hour	Practice Hour	Assessment Strategy(s)	Mapping with CLO
07, 08, 09 & 10	Filters, Cookies, and Sessions: Cookies and Sessions (User Authentication) & Understanding PHP advanced concept	Lecture, Hands-on Exercises with PHP	15h	12h	Lab Performance, Coding Test	CLO 3 & CLO 4
11 & 12	Database Integration with MySQL: Understanding Database Connection	Lecture, Hands-on Exercises with HTML and CSS	10h	8h	Lab Performance	CLO 3
13, 14 & 15	Laravel: Understanding PHP framework	Lecture, Hands-on sessions with , Coding tutorials	10h	10h	Lab Performance, Coding Test	CLO 3, CLO 4
16	Project Submission: Evaluation of knowledge and practical skills in Web development for both back and front end	Practical assessment covering all components	5h		Lab Performance Assessment	CLO 5
17	Final Assessment: Evaluation of knowledge and practical skills in Web development.	Written test, Practical assessment covering all components	5h		Written Exam, Lab Performance Assessment	CLO1 – CLO 5

Week 1 & 2
Lecture 1 & 2

Introduction to PHP on Web Development & PHP Basics - Structure and Elements

Introduction

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP scripts are executed on the server
- PHP files have extension ".php"
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database.

Check php version

```
<!DOCTYPE html>
<html>
<body>
<?php
echo phpversion();
?>
</body>
</html>
```

PHP Syntax

- A PHP script can be placed anywhere in the document.
- A PHP script starts with <?php and ends with ?>

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

Example


```
<!DOCTYPE html>
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

OUTPUT

My first PHP page

Hello World!

PHP Case sensitivity

OUTPUT
Hello World!
Hello World!
Hello World!

```
<!DOCTYPE
html>
<html>
<body>
<?php
ECHO "Hello
World!<br>";
echo "Hello
World!<br>";
EcHo "Hello
World!<br>";
?>
</body>
</html>
```

PHP Comments

```
<?php
// This is a single-line comment
# This is also a single-line comment
/* This is a multi-line comment */
?>
```

Single Comment

```
<?php
// Outputs a welcome message:
echo "Welcome Home!";
?>
```

OUTPUT

Welcome Home!

Multi-Line Comment


```
<!DOCTYPE html>
<html>
<body>
<p>Using comments to ignore parts of a code line:</p>
<?php
$x = 5 /* + 15 */ + 5;
echo $x;
?>
</body>
</html>
```

OUTPUT

Using comments to ignore parts of a code line:

10

PHP Variables

- a variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive

Example of Variables


```
<!DOCTYPE html>
<html>
<body>
<?php
$txt = "W3Schools.com";
echo "I love $txt!";
?>
</body>
</html>
```

OUTPUT

I love W3Schools.com!

Reference:
https://www.w3schools.com/php/php_variables.asp

PHP Data Types

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

Data type

- Get the data type of any object by using the var_dump() function.

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5;
var_dump($x);
?>
</body>
</html>
```

OUTPUT

int(5)

Reference:

https://www.w3schools.com/php/php_datatypes.asp

PHP Object


```
<?php
class Car {
  public $color;
  public $model;
  public function __construct($color, $model) {
    $this->color = $color;
    $this->model = $model;
  }
  public function message() {
    return "My car is a " . $this->color . " " . $this->model .
"!";
  }
}
$myCar = new Car("red", "Volvo");
var_dump($myCar);
?>
```

OUTPUT

object(Car)#1 (2) { ["color"]=> string(3) "red" ["model"]=> string(5) "Volvo" }

Change Data Type


```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5;
var_dump($x);
echo "<br>";
$x = "Hello";
var_dump($x);
?>
</body>
</html>
```

OUTPUT

```
int(5)
string(5) "Hello"
```

PHP Math


```
<!DOCTYPE html>
<html>
<body>
<?php
echo(pi());
?>
</body>
</html>
```

OUTPUT

3.141592653589

PHP Math


```
<!DOCTYPE html>
<html>
<body>
<?php
echo(min(0, 150, 30, 20, -8, -200) . "<br>");
echo(max(0, 150, 30, 20, -8, -200));
?>
</body>
</html>
```

OUTPUT

-200

150

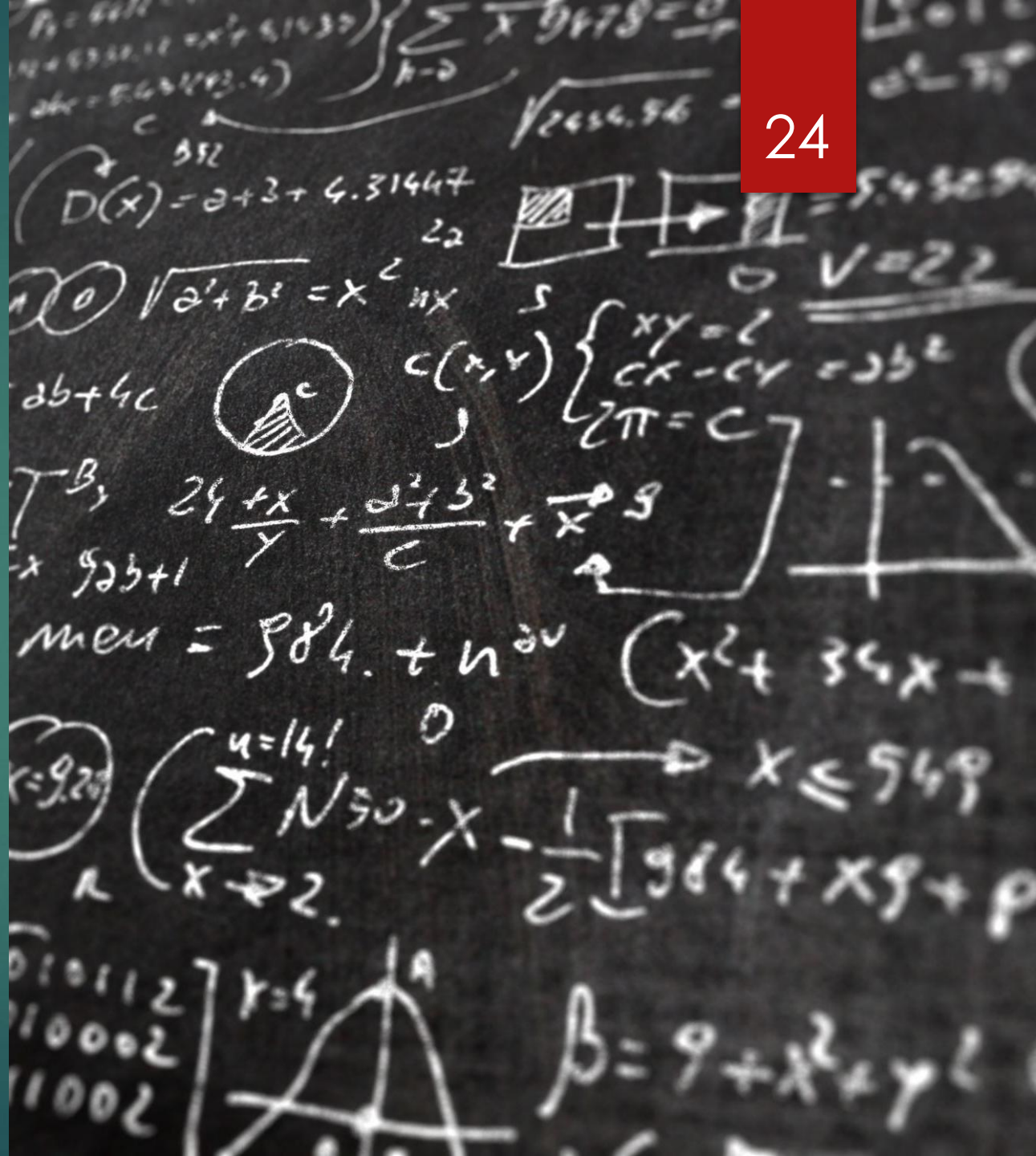
PHP Math

Reference:

https://www.w3schools.com/php/php_math.asp

Student's Working

Tasks include writing scripts to check PHP versions, create simple PHP pages, and experiment with PHP comments and variables.



Thank you

Week 3 & 4
Lecture 3 & 4

Functions, Arrays, and Superglobals

PHP Arrays


```
<!DOCTYPE html>
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
</body>
</html>
```

OUTPUT

3

PHP Indexed Arrays


```
<!DOCTYPE html>
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo $cars[0];
?>
</body>
</html>
```

OUTPUT
Volvo

Loop Through an Indexed Array


```
<!DOCTYPE html>
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
foreach ($cars as $x) {
  echo "$x <br>";
}
?>
</body>
</html>
```

OUTPUT
Volvo
BMW
Toyota

Loop Through an Indexed Array


```
<!DOCTYPE html>
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
foreach ($cars as $x) {
  echo "$x <br>";
}
?>
</body>
</html>
```

OUTPUT

Volvo
BMW
Toyota

PHP Functions


```
<!DOCTYPE html>
<html>
<body>
<?php
function myMessage() {
  echo "Hello world!";
}
myMessage();
?>
</body>
</html>
```

OUTPUT
Hello world!

PHP Function Arguments


```
 <body>
<?php
function familyName($fname)
{
  echo "$fname Refsnes.<br>";
}
familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
</body>
```

OUTPUT

Jani Refsnes.
Hege Refsnes.
Stale Refsnes.
Kai Jim Refsnes.
Borge Refsnes.

PHP Functions - Returning values


```
<body>
<?php
function sum($x, $y) {
  $z = $x + $y;
  return $z;
}
echo "5 + 10 = " . sum(5,10) . "<br>";
echo "7 + 13 = " . sum(7,13) . "<br>";
echo "2 + 4 = " . sum(2,4);
?>
</body>
```

OUTPUT

5 + 10 = 15
7 + 13 = 20
2 + 4 = 6

Passing Arguments by Reference


```
<body>
<?php
function add_five(&$value) {
  $value += 5;
}
$num = 2;
add_five($num);
echo $num;
?>
</body>
```

OUTPUT

7

Superglobals

- The PHP superglobal variables are:
- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

PHP $GLOBALS


```
<body>
<?php
$x = 75;
function myfunction() {
  echo $GLOBALS['x'];
}
myfunction()
?>
</body>
```

OUTPUT

75

PHP $GLOBALS

- Refer to global variables inside functions by defining them as global with the global keyword .

```
<body>
<?php
$x = 75;
function myfunction() {
  global $x;
  echo $x;
}
myfunction()
?>
</body>
```

OUTPUT

75

PHP - $_REQUEST

- $_REQUEST is an array containing data from $_GET, $_POST, and $_COOKIE

```
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = htmlspecialchars($_REQUEST['fname']);
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>
```

OUTPUT

Name: Submit

$_REQUEST on $_GET Requests


```
<!DOCTYPE html>
<html>
<body>
<a href="demo_phpfile.php?subject=PHP
&web=W3schools.com">Test $GET</a>
</body>
</html>
```

OUTPUT

Test $GET

$_POST in HTML Forms

- $_POST contains an array of variables received via the HTTP POST method.
- There are two main ways to send variables via the HTTP Post method:
  - HTML forms
  - JavaScript HTTP requests

$_POST in HTML Forms


```
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>
</body>
```

OUTPUT

Name: Submit

PHP $_GET

- $_GET contains an array of variables received via the HTTP GET method.
- There are two main ways to send variables via the HTTP GET method:
- Query strings in the URL
- HTML Forms

Query string in the URL


```
<!DOCTYPE html>
<html>
<body>
<a href="demo_phpfile.php?subject=PHP&web=W3schools.com">Test
$GET</a>
</body>
</html>
```

OUTPUT

Test $GET

$_GET in HTML Forms


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

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

OUTPUT

Name:

E-mail:

Submit

Student's Working

Focus on functions, arrays, and superglobals in PHP.

Students work on creating and manipulating arrays, looping through array elements, and using PHP functions with and without arguments.

They are also tasked with using superglobals like $_GET, $_POST, and $GLOBALS.

Thank you

Week 5
Lecture 5

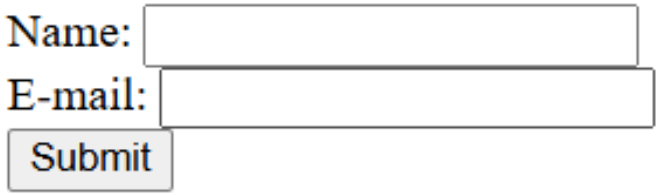
PHP Forms and Advanced: Understanding form handling, validation, PHP date & time function

PHP Form Handling

- The PHP superglobals $_GET and $_POST are used to collect form-data.

```
<!DOCTYPE HTML>
<html>
<body>
<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```

OUTPUT



GET vs. POST

$_GET is an array of variables passed to the current script via the URL parameters.

$_POST is an array of variables passed to the current script via the HTTP POST method.

PHP Form Validation

* required field

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male Other *

Submit

Your Input:

PHP Forms - Validate E-mail and URL

PHP - Validate Name:

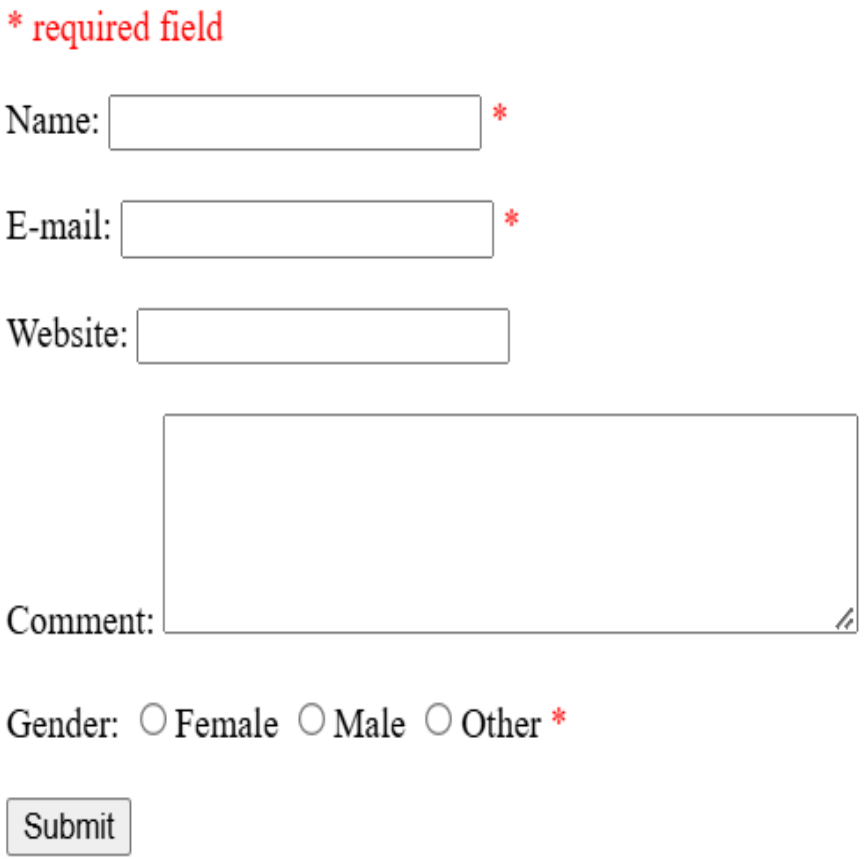
```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z-' ]*$/",$name))
{ $nameErr = "Only letters and white space allowed"; }
```

PHP - Validate E-mail:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL))
{ $emailErr = "Invalid email format"; }
```

PHP - Validate URL:

```
$website = test_input($_POST["website"]);
 if(!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website))
{ $websiteErr = "Invalid URL"; }
```



PHP Calendar Functions

- **PHP cal_days_in_month() Function**:

```
<?php
$d=cal_days_in_month(CAL_GREGORIAN,10,2
005);
echo "There was $d days in October 2005";
?>
```

OUTPUT

There was 28 days in February 1965.
There was 29 days in February 2004.

PHP Calendar Functions

- **PHP cal_from_jd() Function**:

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

<?php
$d=unixtojd(mktime(0,0,0,6,20,2007));
print_r(cal_from_jd($d,CAL_GREGORIAN));
?>

</body>
</html>
```

OUTPUT

Array ([date] => 6/20/2007 [month] => 6 [day] => 20 [year] => 2007 [dow] => 3 [abbrevdayname] => Wed [dayname] => Wednesday [abbrevmonth] => Jun [monthname] => June)

PHP Calendar Functions

- **PHP easter_date() Function:**

```
<?php
echo easter_date() . "<br />";
echo date("M-d-Y",easter_date()) . "<br />";
echo date("M-d-Y",easter_date(1975)) . "<br />";
echo date("M-d-Y",easter_date(1998)) . "<br />";
echo date("M-d-Y",easter_date(2007));
?>
```

OUTPUT

```
1745107200
Apr-20-2025
Mar-30-1975
Apr-12-1998
Apr-08-2007
```

PHP Calendar Functions

- PHP jdtogregorian() Function:

```
<body>

<?php
$jd=gregoriantojd(6,20,2007);
echo $jd . "<br>";
echo jdtogregorian($jd);
?>

</body>
```

OUTPUT

2454272
6/20/2007

PHP Date Functions

- PHP date_add() Function:

```
<body>

 <?php
$date=date_create("2013-03-15");
date_add($date,date_interval_create_from_date_s
tring("40 days"));
echo date_format($date,"Y-m-d");
?>

</body>
```

OUTPUT

2013-04-24

PHP Date Functions

- **PHP date_create_from_format() Function:**

```
<body>
 <?php
$date=date_create_from_format("j-M-Y","15-Mar-2013");
?>
</body>
```

OUTPUT

2013-04-24

PHP Date Functions

- PHP date_add() Function:

```
<body>

 <?php
$date=date_create("2013-03-15");
date_add($date,date_interval_create_from_date_s
tring("40 days"));
echo date_format($date,"Y-m-d");
?>

</body>
```

OUTPUT

2013-04-24

PHP Date Functions

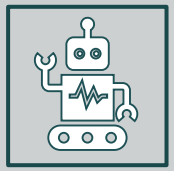
- **date_default_timezone_get() Function:**

```
<body>
<?php
echo date_default_timezone_get();
?>
</body>
```

OUTPUT

UTC

Student's Working



Emphasis on form handling and validation.

Students handle HTML forms with PHP, validate input data (e.g., names, emails, and URLs), and use PHP date and time functions.

Thank you

Week 7, 8, 9 & 10
Lecture 7, 8, 9 & 10

Filters, Cookies, and Sessions: Cookies and Sessions (User Authentication) & Understanding PHP advanced concept

PHP Cookies

01

A cookie is often used to identify a user.

02

A cookie is a small file that the server embeds on the user's computer.

03

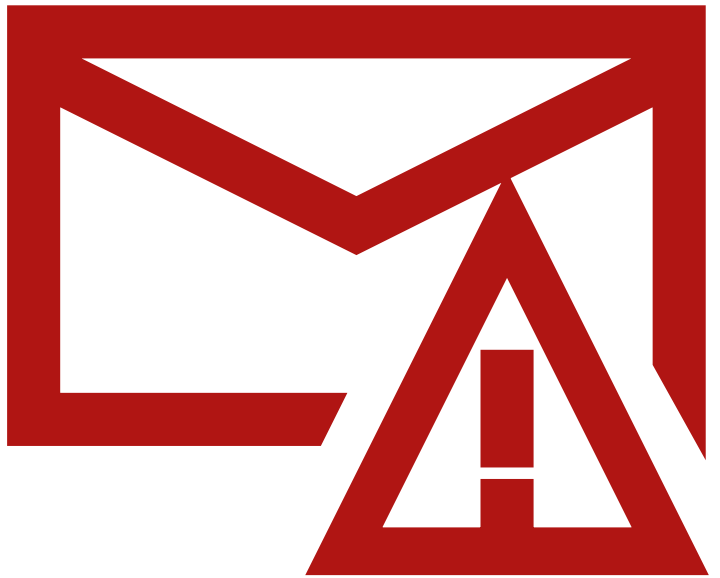
A cookie is created with the setcookie() funct ion.

04

Syntax:
setcookie(name, value, expire, path, domain, secure, httponly);

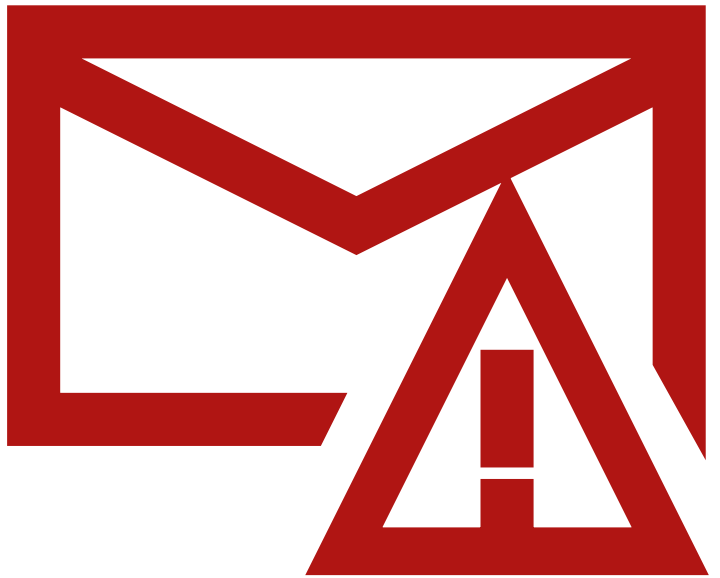
PHP Create/Retrieve a Cookie


```
<?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];
}
?>
</body>
</html>
```



Modifying a Cookie Value

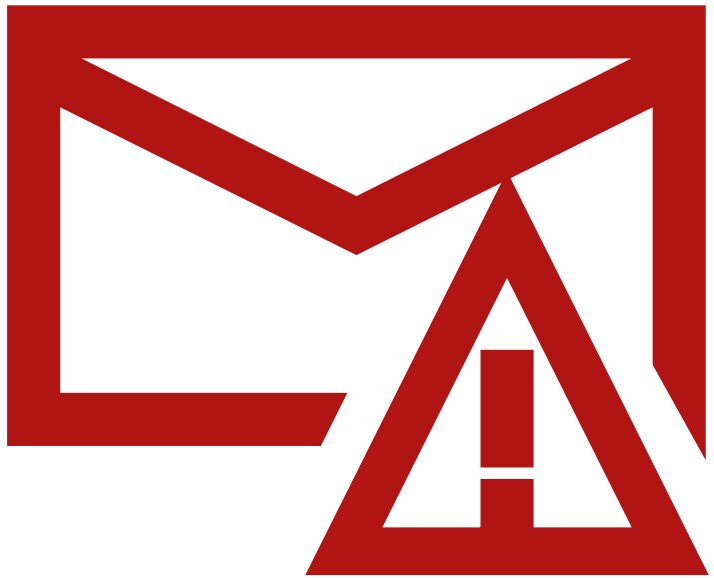

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() +
(86400 * 30), "/");
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
  echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
  echo "Cookie '" . $cookie_name . "' is set!<br>";
  echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```



Delete a Cookie

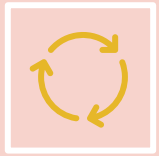
- To delete a cookie, use the setcookie() function with an expiration date in the past:

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```



PHP Sessions

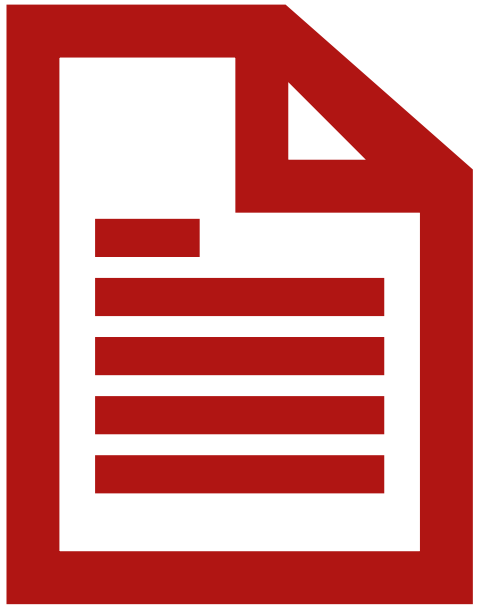
A session is a way to store information (in variables) to be used across multiple pages.



A session is started with the session_start() function.

Start a PHP Session

```
<?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>
```

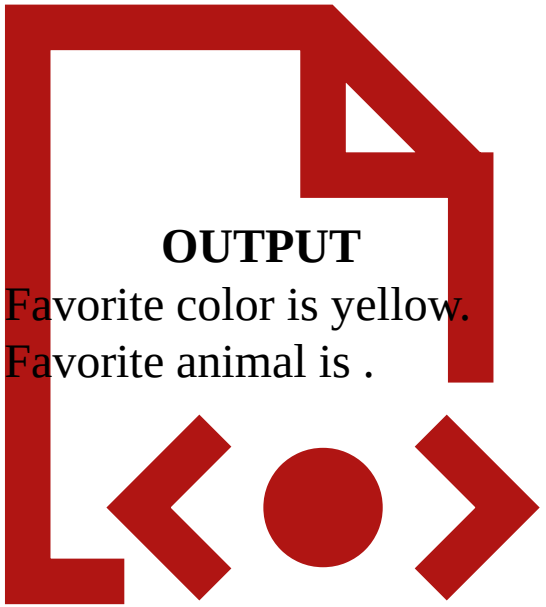


OUTPUT

Session variables are set.

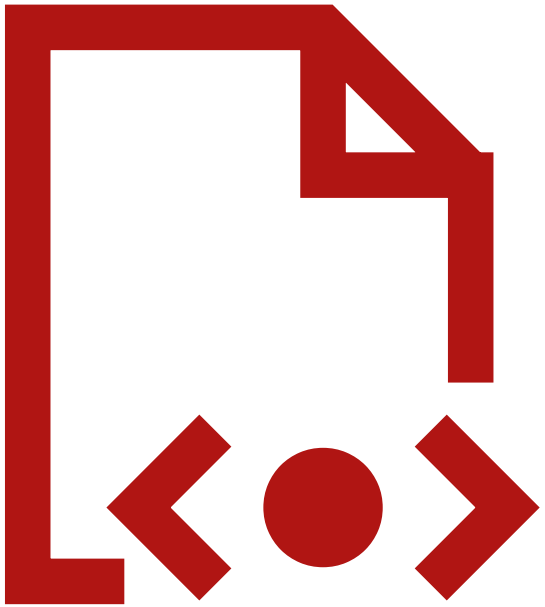
Getting PHP Session Variable Values


```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```



Destroy a PHP Session

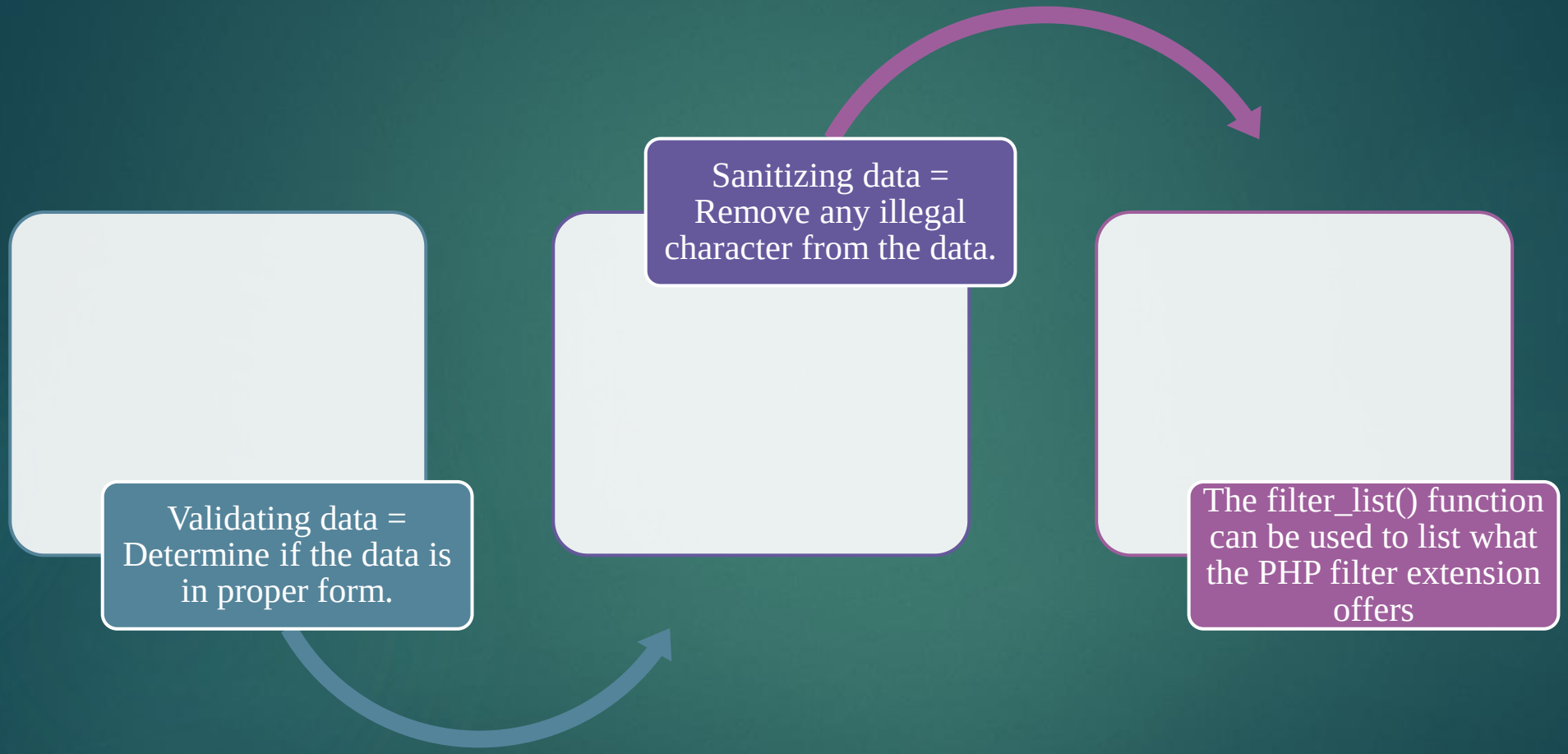
```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```



OUTPUT

All session variables are now removed, and the session is destroyed.

PHP Filters


Validating data = Determine if the data is in proper form.
Sanitizing data = Remove any illegal character from the data.
The filter_list() function can be used to list what the PHP filter extension offers

PHP filter_var() Function


```
<?php
$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str,
FILTER_SANITIZE_STRING);
echo $newstr;
?>
```

OUTPUT

Hello World!

Validate an Integer

```
<?php
$int = 100;

if (!filter_var($int, FILTER_VALIDATE_INT) === false) {
  echo("Integer is valid");
} else {
  echo("Integer is not valid");
}
?>
```

OUTPUT

Integer is valid

Validate an IP Address

```
<?php
$ip = "127.0.0.1";

if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {
  echo("$ip is a valid IP address");
} else {
  echo("$ip is not a valid IP address");
}
?>
```

OUTPUT

127.0.0.1 is a valid IP address

Validate an Integer Within a Range

```
<?php
$int = 122;
$min = 1;
$max = 200;

if (filter_var($int,
FILTER_VALIDATE_INT, array("options" => array("min_range"=>
$min, "max_range"=>$max))) === false) {
  echo("Variable value is not within the legal range");
} else {
  echo("Variable value is within the legal range");
}
?>
```

OUTPUT

Variable value is within the legal range

PHP and JSON

JSON stands for JavaScript Object Notation, and is a syntax for storing and exchanging data.

PHP has some built-in functions to handle JSON.

First, we will look at the following two functions:

json_encode()

json_decode()

PHP - json_encode()

- The json_encode() function is used to encode a value to JSON format.

```
<?php
$age
= array("Peter"=>35, "Ben"=>37, "Joe"=>43);

echo json_encode($age);
?>
```

OUTPUT

{"Peter":35,"Ben":37,"Joe":43}

PHP - json_decode()

- The json_decode() function is used to decode a JSON object into a PHP object or an associative array.

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

var_dump(json_decode($jsonobj));
?>
```

OUTPUT

```
object(stdClass)#1 (3) { ["Peter"]=> int(35) ["Ben"]=>
int(37) ["Joe"]=> int(43) }
```

PHP OOP - Classes and Objects


```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;}
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color; }
  function get_color() {
     return $this->color;
   }
}
$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple-
>get_name();
echo "<br>";
echo "Color: " .  $apple-
>get_color();
?>
```

OUTPUT

Name: Apple
Color: Red

PHP OOP

HTTPS://WWW.W3SCHOOLS.COM/PHP/PHP_OOP_WHAT_IS.ASP

Student's Working

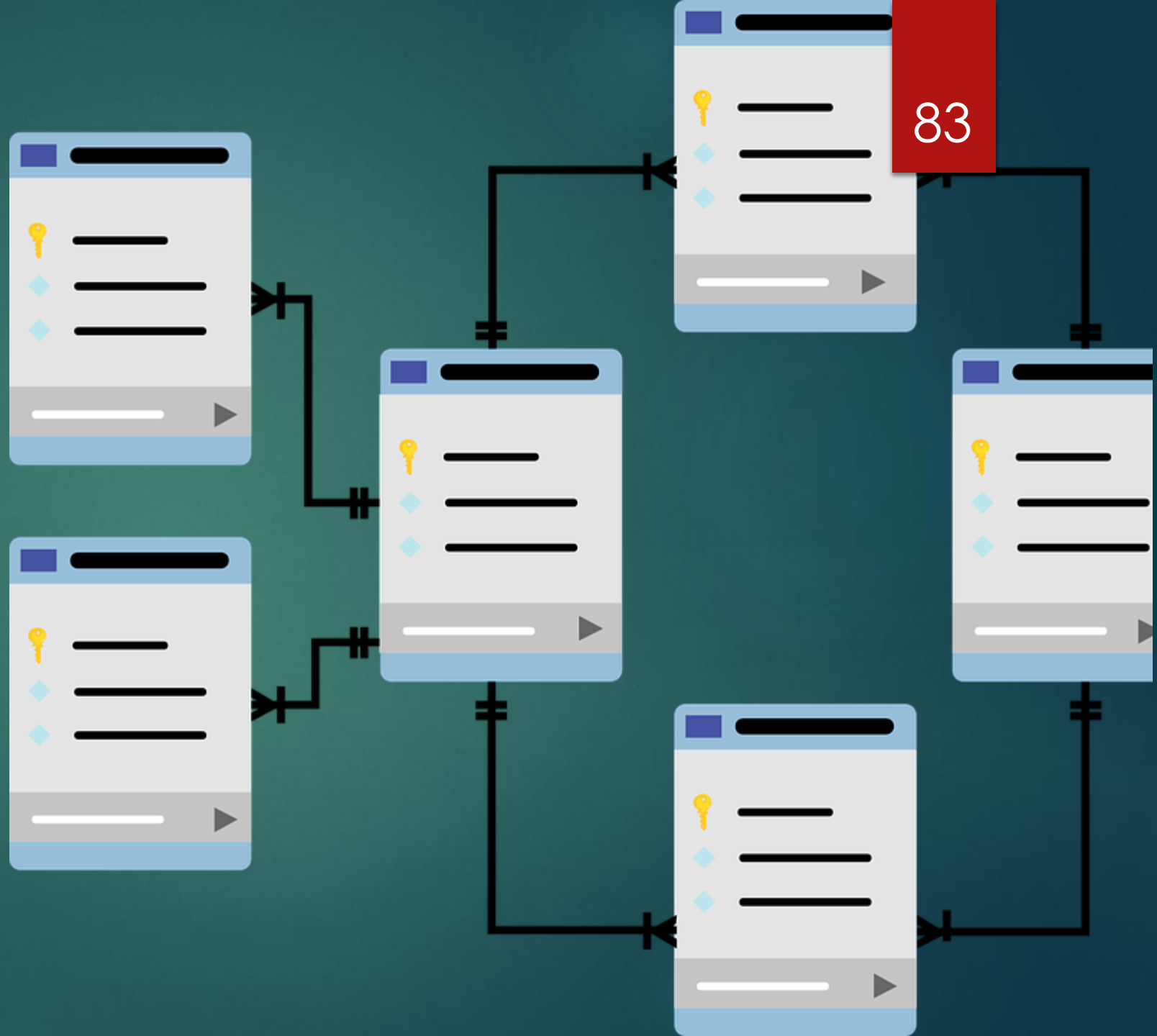
Exploration of cookies, sessions, and advanced PHP concepts.

Tasks include creating, modifying, and deleting cookies; managing sessions; and using PHP filters to sanitize and validate data.

Thank you

Week 11 & 12
Lecture 11 & 12

DATABASE INTEGRATION WITH MYSQL



Introduction

MySQL is a relational database management system

RDBMS stands for Relational Database Management System.

RDBMS is the basis for all modern database systems such as MySQL, Microsoft SQL Server, Oracle, and Microsoft Access

RDBMS uses SQL queries to access the data in the database.

SQL Statements

Tablenames	Records
Customers	91
Categories	8
Employees	10
OrderDetails	518
Orders	196
Products	77
Shippers	3
Suppliers	29

Result:
Number of Records: 91

SELECT * FROM Customers;

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table

The SQL SELECT Statement

Demo Database

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

SELECT CustomerName,City FROM Customers;

Result:
Number of Records: 91

CustomerName	City
Alfreds Futterkiste	Berlin
Ana Trujillo Emparedados y helados	México D.F.
Antonio Moreno Taquería	México D.F.
Around the Horn	London

The SQL WHERE Clause

Demo Database

CustomerID	CustomerName	Contact Name	Address	City	Postal Code	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

Result

Number of Records: 5

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
13	Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.	05022	Mexico

The SQL ORDER BY

Demo Database

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

```
SELECT * FROM Products
ORDER BY Price;
```

Result

ProductID	Product Name	Supplier ID	CategoryID	Unit	Price
33	Geitost	15	4	500 g	2.5
24	Guaraná Fantástica	10	1	12 - 355 ml cans	4.5
13	Konbu	6	8	2 kg box	6

The SQL AND Operator

Demo Database

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
29	Galería del gastrónomo	Eduardo Saavedra	Rambla de Cataluña, 23	Barcelona	08022	Spain
30	Godos Cocina Típica	José Pedro Freyre	C/ Romero, 33	Sevilla	41101	Spain
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

SELECT * FROM Customers
WHERE Country = 'Spain' AND
CustomerName LIKE 'G%';

Result

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
29	Galería del gastrónomo	Eduardo Saavedra	Rambla de Cataluña, 23	Barcelona	08022	Spain
30	Godos Cocina Típica	José Pedro Freyre	C/ Romero, 33	Sevilla	41101	Spain

The SQL UPDATE Statement

Demo Database

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt',
City= 'Frankfurt'
WHERE CustomerID = 1;
```

Result

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany

The SQL DELETE Statement

Demo Database

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Result

CustomerID	CustomerName	Contact Name	Address	City	Postal Code	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico

SQL Aggregate Functions

An aggregate function is a function that performs a calculation on a set of values and returns a single value.

The most commonly used SQL aggregate functions are:

MIN() - returns the smallest value within the selected column

MAX() - returns the largest value within the selected column

COUNT() - returns the number of rows in a set

SUM() - returns the total sum of a numerical column

AVG() - returns the average value of a numerical column

Aggregate functions ignore null values (except for COUNT()).

The SQL MIN() and MAX() Functions

Demo Database

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	2.5
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	263.5

```
SELECT MIN(Price)
FROM Products;
```

Expr1000
2.5

```
SELECT MAX(Price)
FROM Products;
```

Expr1000
263.5

SQL JOIN

```
SELECT Customers.customer_id,
Customers.first_name, Orders.amount
FROM Customers
JOIN Orders
ON Customers.customer_id =
Orders.customer;
```

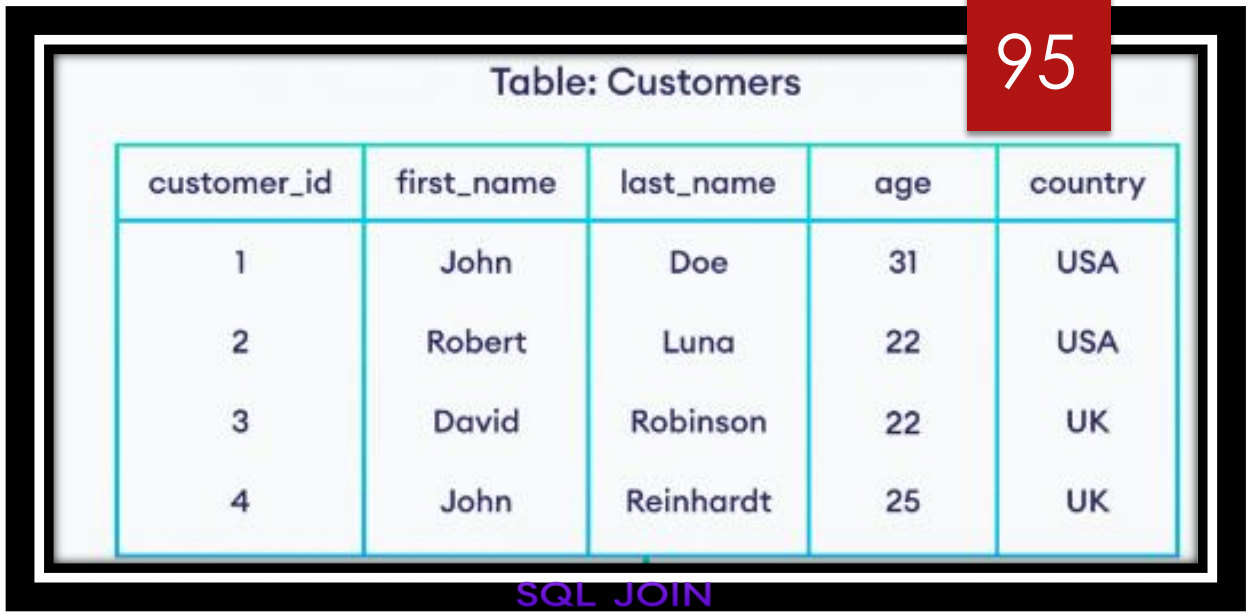


Table: Customers

customer_id	first_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

order_id	amount	customer
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

customer_id	first_name	amount
3	David	500
5	Betty	800

Sql Database Operation

https://www.w3schools.com/sql/default.asp

Database Connection


```
<?php
$dbServername = "localhost";
$dbUsername = "root";
$dbPassword = "";
$dbnName = “labDatabase";
$conn = mysqli_connect($dbServername, $dbUsername, $dbPassword, $dbnName);
?>
<!DOCTYPE html>
<html>
<body>
<?php
If(!$conn){
die(“failed to connect … …”.mysqli_connect_error());
}
Echo “connection successful!!!!”;
?>
</body>
</html>
```

Database Connection(Contn'd)


```
<?php
$dbServername = "localhost";
$dbUsername = "root";
$dbPassword = "";
$dbnName = "labDatabase";
$conn = mysqli_connect($dbServername, $dbUsername, $dbPassword, $dbnName);
?>
<!DOCTYPE html>
<html>
<body>
<?php
$sql = "SELECT * FROM users;";
$result = mysqli_query($conn, $sql);
$resultCheck = mysqli_num_rows($result);
    if( $resultCheck >0){
          while($row = mysqli_fetch_assoc( $result)){
                echo $row['name']."<br>";
          } }
?>  </body>    </html>
```

Student's Work

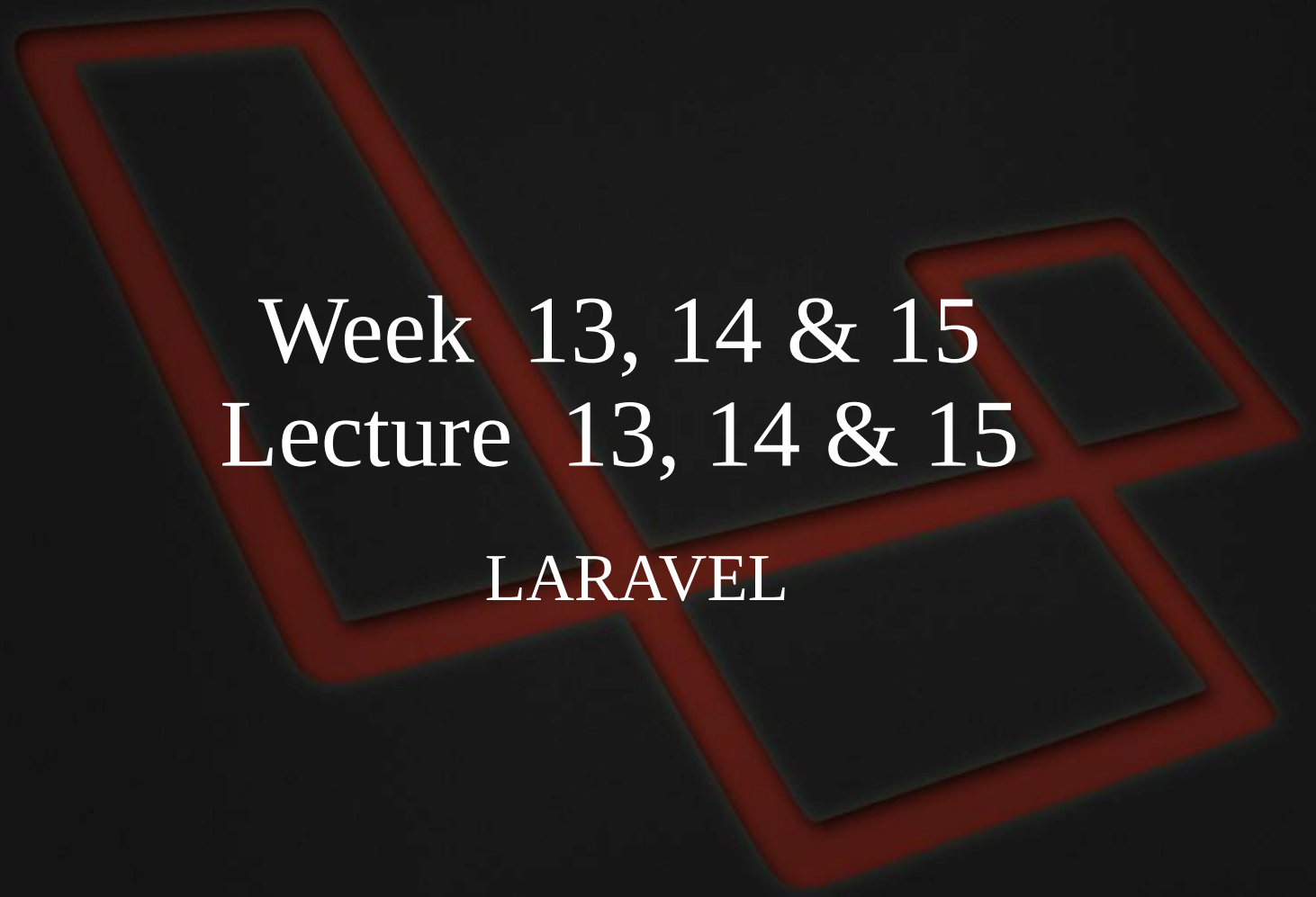
Integration of PHP with MySQL databases.

Students perform database operations such as connecting to a MySQL database, retrieving data using SELECT, updating, and deleting records.

They practice SQL commands like WHERE, ORDER BY, and JOIN.

References

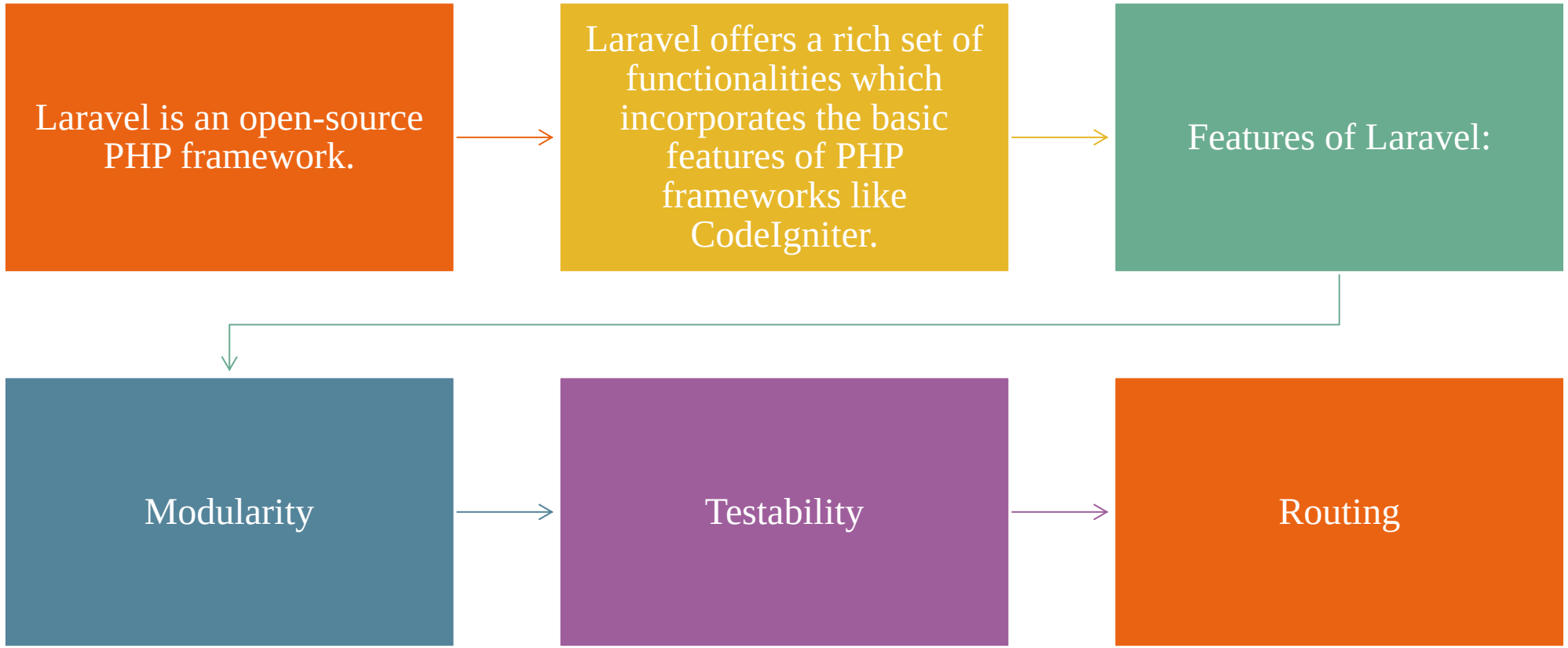
- https://www.w3schools.com/sql/
- https://www.programiz.com/sql



Week 13, 14 & 15
Lecture 13, 14 & 15

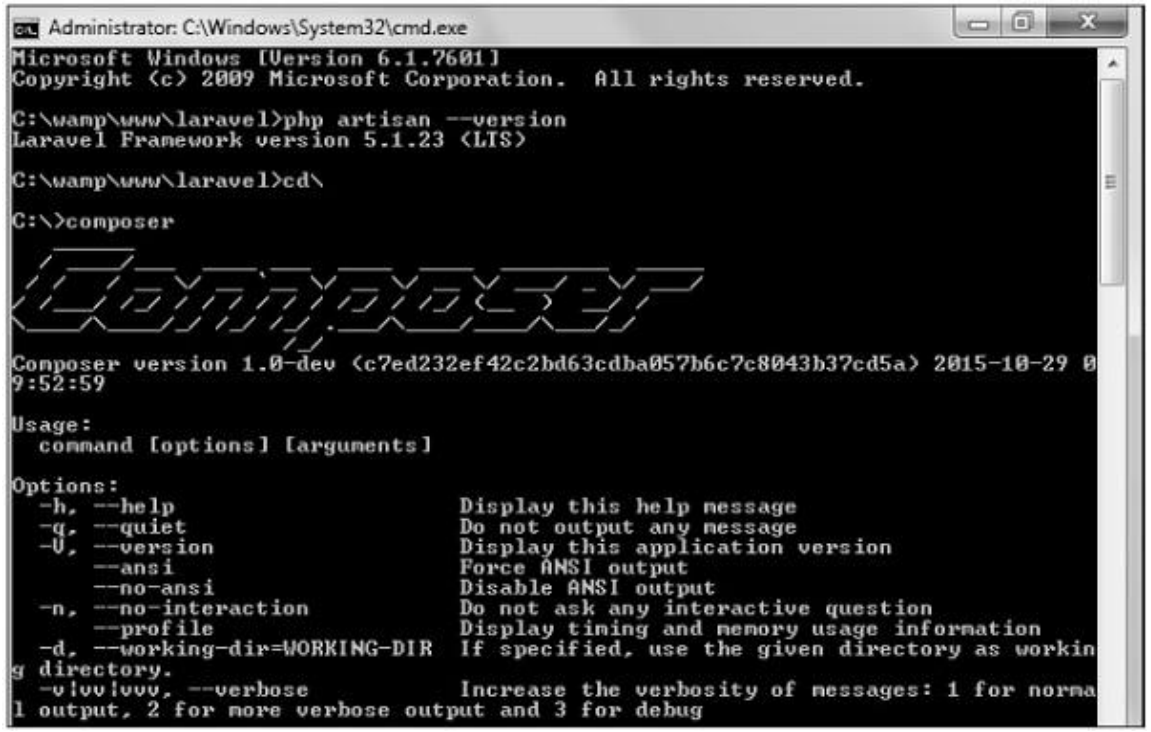
LARAVEL

Introduction



Laravel Installation

- **Step 1** – Visit the following URL and download composer to install it on your system.
  https://getcomposer.org/download/
- **Step 2** – After the Composer is installed, check the installation by typing the Composer command in the command prompt as shown in the following screenshot.



Laravel Installation

Step 3 − Create a new directory anywhere in your system for your new Laravel project. After that, move to path where you have created the new directory and type the following command there to install Laravel.

- composer create-project laravel/laravel –-prefer-dist

For version 5.7 –

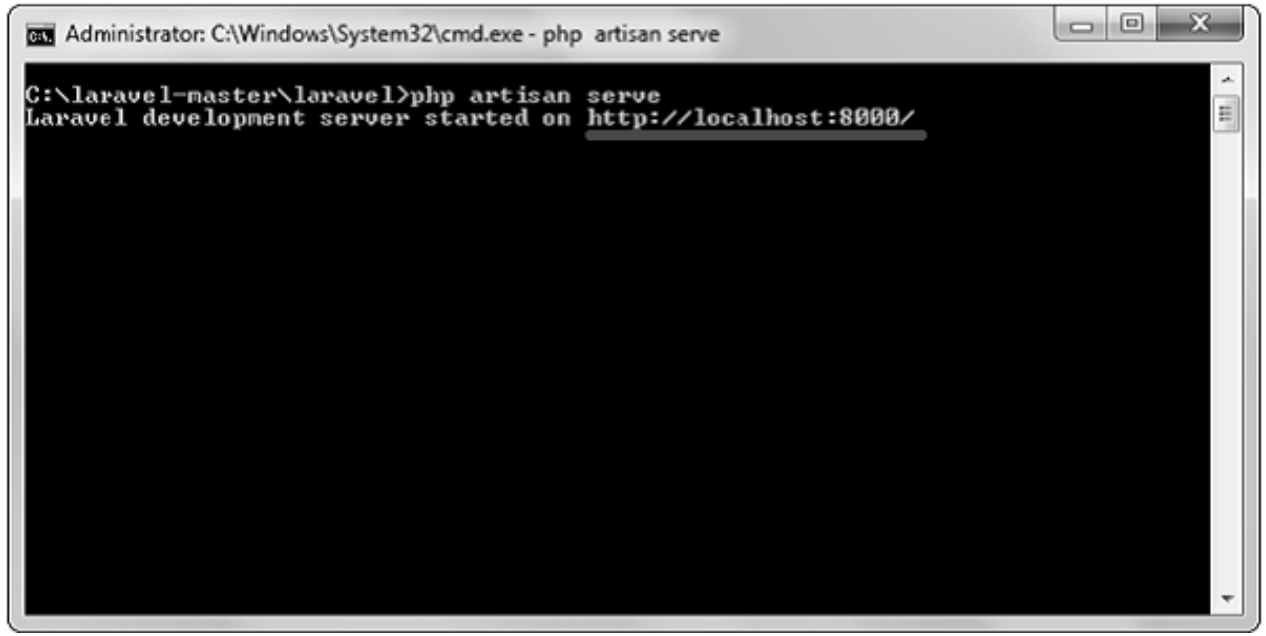
composer create-project laravel/laravel test dev-develop

Step 4 − The above command will install Laravel in the current directory. Start the Laravel service by executing the following command.

- php artisan serve

Laravel Installation

- **Step 5** − After executing the above command, you will see a screen as shown below −

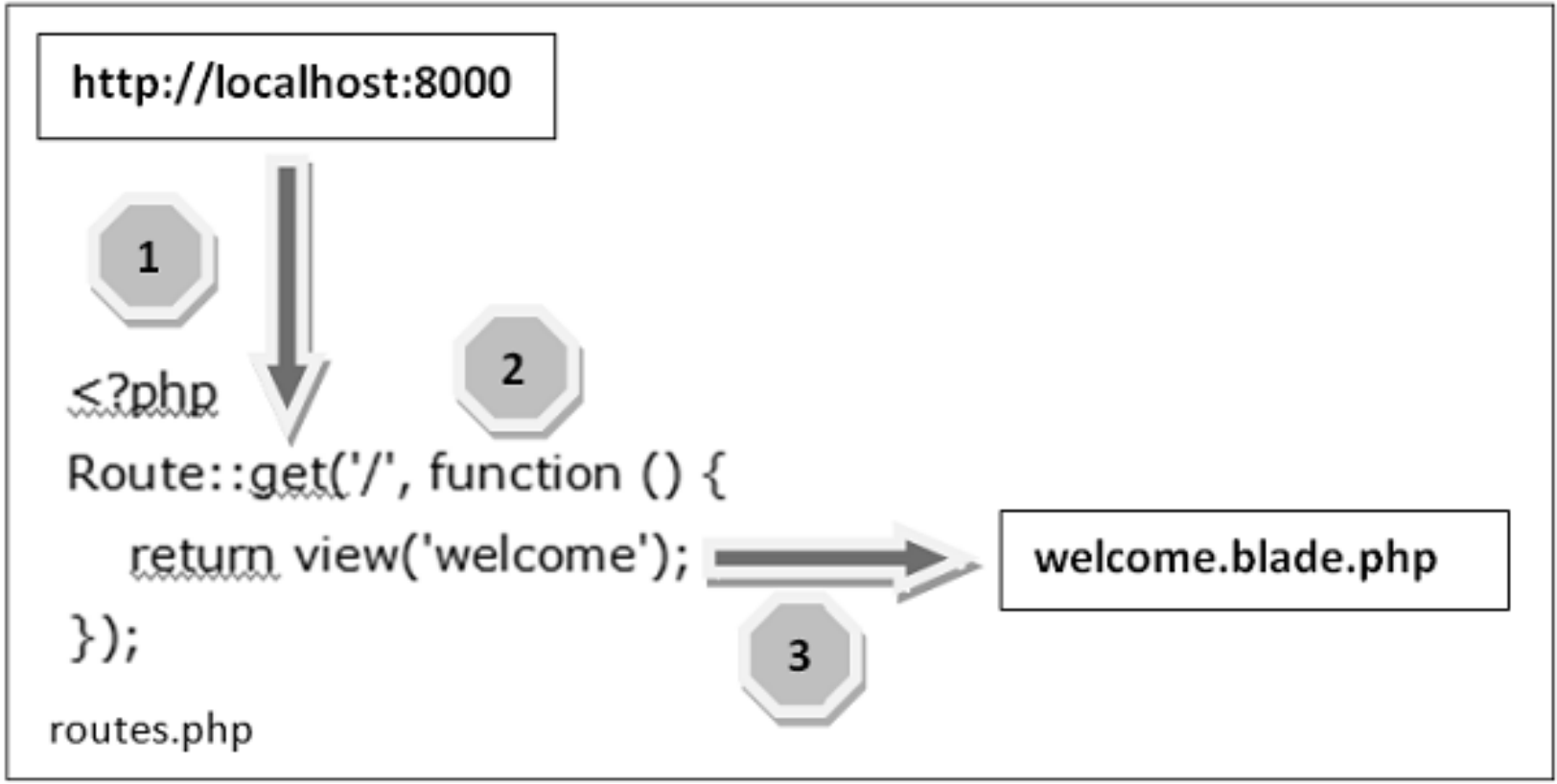


Step 6 − Copy the URL underlined in gray in the above screenshot and open that URL in the browser.

Laravel – Routing Mechanism

Required Parameters:

```
Route::get('ID/{id}',function($id) { echo 'ID: '.$id; });
```



Middleware

Define Middleware

As the name suggest, Middleware acts as a middle man between request and response. It is a type of filtering mechanism. For example, Laravel includes a middleware that verifies whether user of the application is authenticated or not. If the user is authenticated, he will be redirected to the home page otherwise, he will be redirected to the login page.

Middleware can be created by executing the following command:

```
php artisan make:middleware <middleware-name>
```

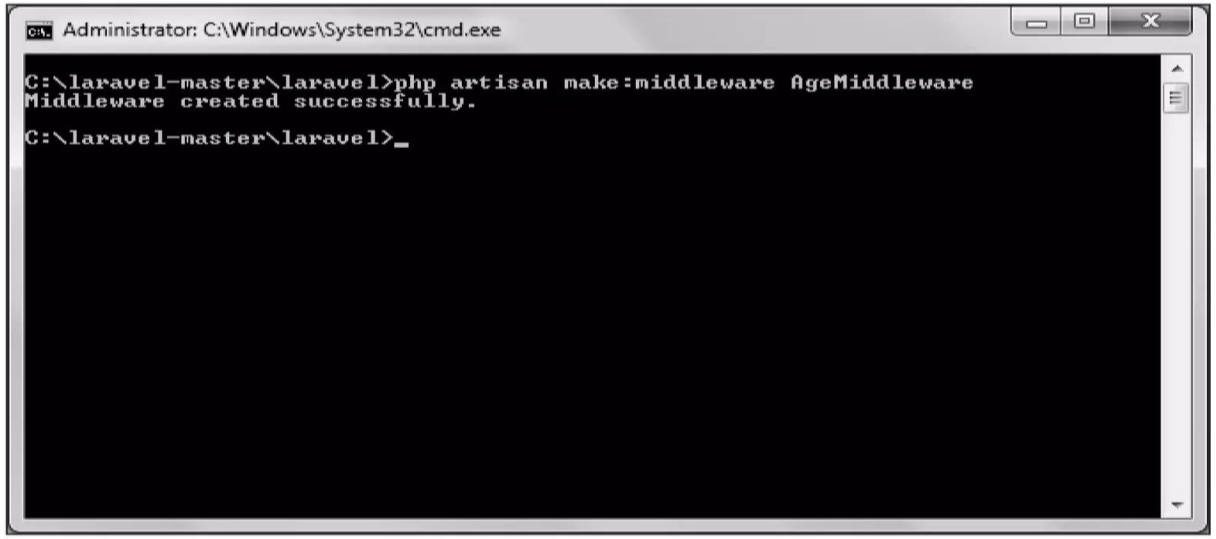
Replace the <middleware-name> with the name of your middleware. The middleware that you create can be seen at **app/Http/Middleware** directory.

Example

Step 1: Let us now create AgeMiddleware. To create that, we need to execute the following command:

```
php artisan make:middleware AgeMiddleware
```

Step 2: After successful execution of the command, you will receive the following output:



Step 3: AgeMiddlware will be created at **app/Http/Middleware**. The newly created file will have the following code already created for you.

```
<?php

namespace App\Http\Middleware;

use Closure;

class AgeMiddleware
{
    public function handle($request, Closure $next)
    {
        return $next($request);
    }
}
```

Register Middleware

We need to register each and every middleware before using it. There are two types of Middleware in Laravel.

- Global Middleware
- Route Middleware

The **Global Middleware** will run on every HTTP request of the application, whereas the **Route Middleware** will be assigned to a specific route. The middleware can be registered at **app/Http/Kernel.php.** This file contains two properties **$middleware** and **$routeMiddleware**. **$middleware** property is used to register Global Middleware and **$routeMiddleware** property is used to register route specific middleware.

To register the global middleware, list the class at the end of $middleware property.

```
protected $middleware = [
        \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
    ];
```

Register Middleware

Register Middleware

To register the route specific middleware, add the key and value to $routeMiddleware property.

```
protected $routeMiddleware = [
        'auth' => \App\Http\Middleware\Authenticate::class,
        'auth.basic' =>
\Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    ];
```

Example

We have created **AgeMiddleware** in the previous example. We can now register it in route specific middleware property. The code for that registration is shown below.

The following is the code for **app/Http/Kernel.php:**

```
<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel
{
    protected $middleware = [
        \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
    ];

    protected $routeMiddleware = [
        'auth' => \App\Http\Middleware\Authenticate::class,
        'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
        'Age' => \App\Http\Middlware\AgeMiddleware::class,
    ];
}
```

Middleware Parameters

We can also pass parameters with the Middleware. For example, if your application has different roles like user, admin, super admin etc. and you want to authenticate the action based on role, this can be achieved by passing parameters with middleware. The middleware that we create contains the following function and we can pass our custom argument after the **$next** argument.

```
public function handle($request, Closure $next)
{
        return $next($request);
}
```

Example

Step 1: Create RoleMiddleware by executing the following command:

```
php artisan make:middleware RoleMiddleware
```

Step 2: After successful execution, you will receive the following output:

Middleware Parameters

Middleware Parameters

Step 3: Add the following code in the handle method of the newly created RoleMiddleware at **app/Http/Middleware/RoleMiddleware.php**.

```
<?php

namespace App\Http\Middleware;

use Closure;

class RoleMiddleware
{
    public function handle($request, Closure $next, $role)
    {
echo "Role: ".$role;
            return $next($request);
    }
}
```

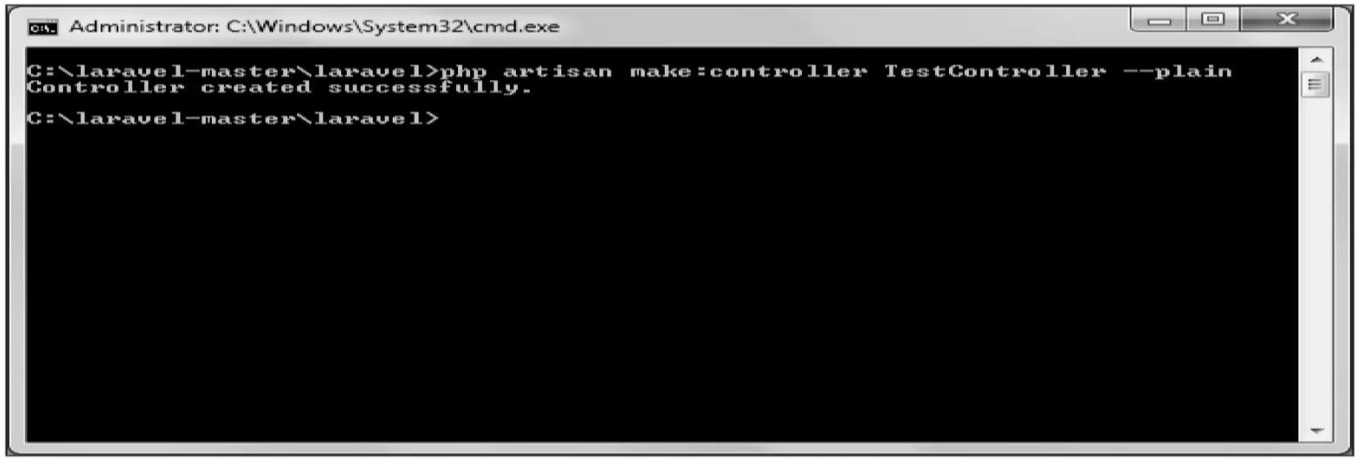
Step 4: Register the RoleMiddleware in **app\Http\Kernel.php** file. Add the line highlighted in gray color in that file to register RoleMiddleware.

```
/**
 * The application's route middleware.
 *
 * @var array
 */
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'Age' => \App\Http\Middleware\AgeMiddleware::class,
    'After' => \App\Http\Middleware\AfterMiddleware::class,
    'Before' => \App\Http\Middleware\BeforeMiddleware::class,
    'First' => \App\Http\Middleware\FirstMiddleware::class,
    'Second' => \App\Http\Middleware\SecondMiddleware::class,
    'Role' => \App\Http\Middleware\RoleMiddleware::class,
];
```

Step 5: Execute the following command to create **TestController**:

```
php artisan make:controller TestController --plain
```

Step 6: After successful execution, you will receive the following output:



Step 7: Copy the following code to **app/Http/TestController.php** file.

app/Http/TestController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class TestController extends Controller
{
    public function index(){
            echo "<br>Test Controller.";
     }
}
```

Step 8: Add the following line of code in **app/Http/routes.php** file.

Middleware Parameters

Middleware Parameters

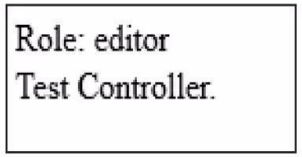
app/Http/routes.php

```
Route::get('role',[
     'middleware' => 'Role:editor',
     'uses'        => 'TestController@index',
]);
```

Step 9: Visit the following URL to test the Middleware with parameters

```
http://localhost:8000/role
```

Step 10: The output will appear as shown in the following image.



Creating Controllers

Basic Controllers

In MVC framework, the letter **'C'** stands for Controller. It acts as a directing traffic between Views and Models.

Creating a Controller

Open the command prompt or terminal based on the operating system you are using and type the following command to create controller using the Artisan CLI (Command Line Interface).

```
php artisan make:controller <controller-name> --plain
```

Replace the <controller-name> with the name of your controller. This will create a plain constructor as we are passing the argument **— plain**. If you don't want to create a plain constructor, you can simply ignore the argument. The created constructor can be seen at **app/Http/Controllers.** You will see that some basic coding has already been done for you and you can add your custom coding. The created controller can be called from routes.php by the following syntax.

```
Route::get('base URI','controller@method');
```

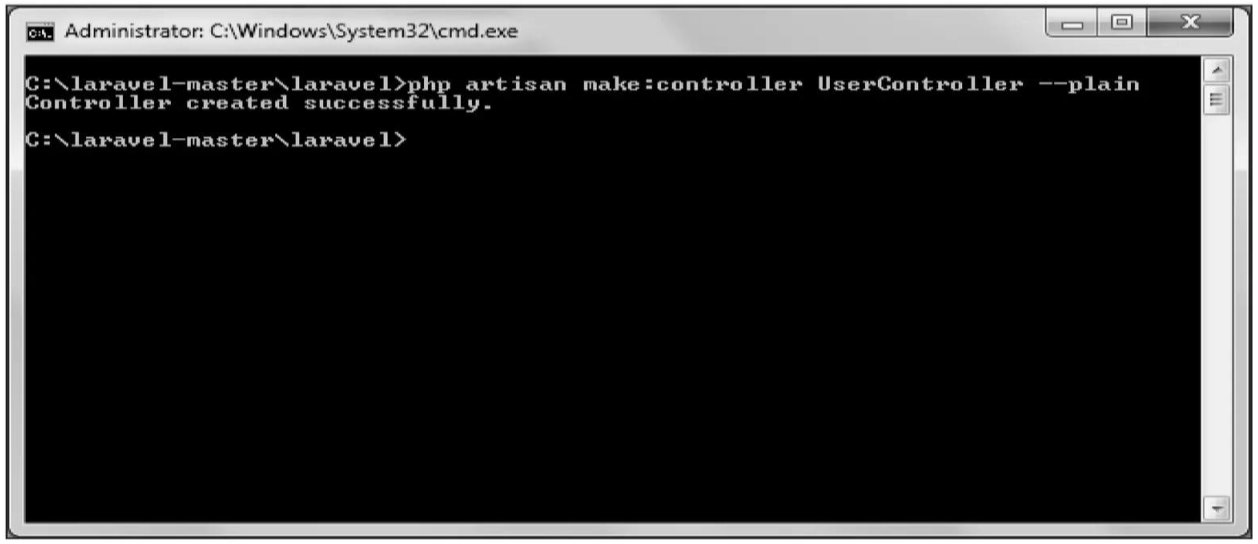
Example

Step 1: Execute the following command to create **UserController**.

```
php artisan make:controller UserController --plain
```

Creating Controllers

Step 2: After successful execution, you will receive the following output.



Step 3: You can see the created controller at **app/Http/Controller/UserController.php** with some basic coding already written for you and you can add your own coding based on your need.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    //
}
```

Creating Controllers

Step 3: Add the following line of code in **app/Http/routes.php** file.

app/Http/routes.php

```
Route::resource('my','MyController');
```

Step 4: We are now registering all the methods of MyController by registering a controller with resource. Below is the table of actions handled by resource controller.

Verb	Path	Action	Route Name
GET	/my	index	my.index
GET	/my/create	create	my.create
POST	/my	store	my.store
GET	/my/{my}	show	my.show
GET	/my/{my}/edit	edit	my.edit
PUT/PATCH	/my/{my}	update	my.update
DELETE	/my/{my}	destroy	my.destroy

Step 5: Try executing the URLs shown in the following table.

URL	Description	Output Image
http://localhost:8000/my	Executes index method of MyController.php	index
http://localhost:8000/my/create	Executes create method of MyController.php	create
http://localhost:8000/my/1	Executes show method of MyController.php	show
http://localhost:8000/my/1/edit	Executes edit method of MyController.php	edit

Laravel Request

Retrieving the Request URI

The "**path**" method is used to retrieve the requested URI. The "**is**" method is used to retrieve the requested URI which matches the particular pattern specified in the argument of the method. To get the full URL, we can use the "**url**" method.

Example

Step 1: Execute the below command to create a new controller called **UriController**.

```
php artisan make:controller UriController -plain
```

Step 2: After successful execution of the URL, you will receive the following output:

Step 3: After creating a controller, add the following code in that file.

app/Http/Controllers/UriController.php

```
<?php

namespace App\Http\Controllers;
```

Laravel Request

```
use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class UriController extends Controller
{
    public function index(Request $request){

            // Usage of path method
            $path = $request->path();
            echo 'Path Method: '.$path;
            echo '<br>';

            // Usage of is method
            $pattern = $request->is('foo/*');
            echo 'is Method: '.$pattern;
            echo '<br>';

            // Usage of url method
            $url = $request->url();
            echo 'URL method: '.$url;
     }
}
```

Step 4: Add the following line in the **app/Http/route.php** file.

app/Http/route.php

```
Route::get('/foo/bar','UriController@index');
```

Step 5: Visit the following URL.

```
http://localhost:8000/foo/bar
```

Step 6: The output will appear as shown in the following image.

```
Path Method: foo/bar
is Method: 1
URL method: http://localhost:8000/foo/bar
```

Retrieving Input

The input values can be easily retrieved in Laravel. No matter what method was used "**get**" or "**post**", the Laravel method will retrieve input values for both the methods the same way. There are two ways we can retrieve the input values.

- Using the input() method
- Using the properties of Request instance

Using the input() method

The input() method takes one argument, the name of the field in form. For example, if the form contains username field then we can access it by the following way.

```
$name = $request->input('username');
```

Using the properties of Request instance

Like the input() method, we can get the username property directly from the request instance.

```
$request->username
```

Example

Step 1: Create a Registration form, where user can register himself and store the form at **resources/views/register.php**

```
<html>
<head>
      <title>Form Example</title>
</head>
<body>
      <form action="/user/register" method="post">
      <input type="hidden" name="_token" value="<?php echo csrf_token() ?>">
            <table>
                  <tr>
                        <td>Name</td>
```

Retrieving Input

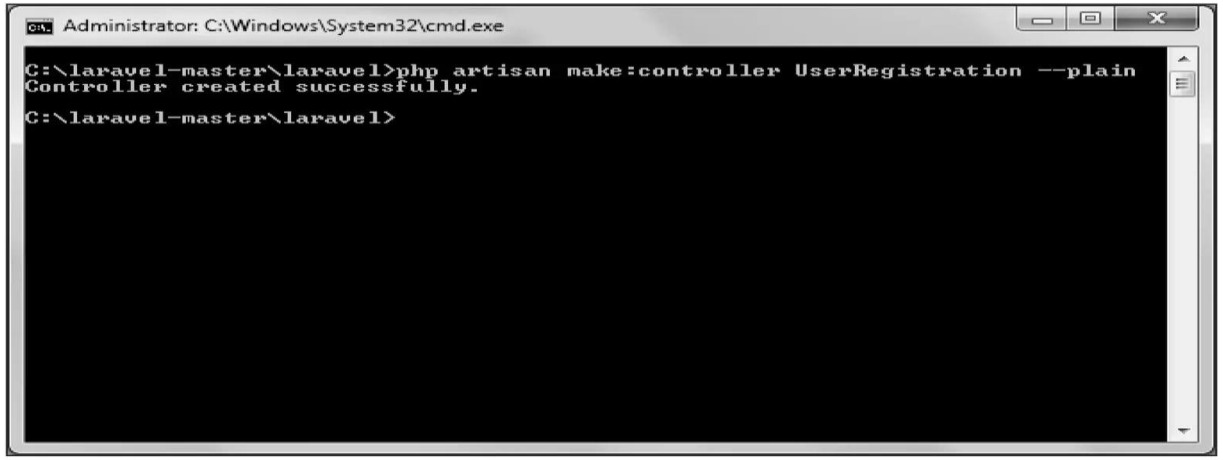
Request example

```
                        <td><input type="text" name="name" /></td>
                    </tr>
                    <tr>
                        <td>Username</td>
                        <td><input type="text" name="username" /></td>
                    </tr>
                    <tr>
                        <td>Password</td>
                        <td><input type="text" name="password" /></td>
                    </tr>
                    <tr>
                        <td colspan="2" align="center"><input type="submit"
value="Register" /></td>
                    </tr>
            </table>
     </form>
</body>
</html>
```

Step 2: Execute the below command to create a **UserRegistration** controller.

```
php artisan make:controller UserRegistration --plain
```

Step 3: After successful execution, you will receive the following output:



Step 4: Copy the following code in **app/Http/Controllers/UserRegistration.php** controller.

app/Http/Controllers/UserRegistration.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class UserRegistration extends Controller
{
    public function postRegister(Request $request){
            //Retrieve the name input field
            $name = $request->input('name');
            echo 'Name: '.$name;
            echo '<br>';

            //Retrieve the username input field
            $username = $request->username;
```

Request example (Contn'd)

```
            echo 'Username: '.$username;
            echo '<br>';

            //Retrieve the password input field
            $password = $request->password;
            echo 'Password: '.$password;

      }
}
```

Step 5: Add the following line in **app/Http/routes.php** file.

app/Http/routes.php

```
Route::get('/register',function(){
      return view('register');
});
Route::post('/user/register',array('uses'=>'UserRegistration@postRegister'));
```

Step 6: Visit the following URL and you will see the registration form as shown in the below figure. Type the registration details and click Register and you will see on the second page that we have retrieved and displayed the user registration details.

```
http://localhost:8000/register
```

Step 7: The output will look something like as shown in below the following images.

Request example (Contn'd)

Creating & Retrieving Cookie

Creating Cookie

Cookie can be created by global cookie helper of Laravel. It is an instance of **Symfony\Component\HttpFoundation\Cookie**. The cookie can be attached to the response using the withCookie() method. Create a response instance of **Illuminate\Http\Response** class to call the withCookie() method. Cookie generated by the Laravel are encrypted and signed and it can't be modified or read by the client.

Here is a sample code with explanation.

```
//Create a response instance
$response = new Illuminate\Http\Response('Hello World');


//Call the withCookie() method with the response method
$response->withCookie(cookie('name', 'value', $minutes));


//return the response
return $response;
```

Cookie() method will take 3 arguments. First argument is the name of the cookie, second argument is the value of the cookie and the third argument is the duration of the cookie after which the cookie will get deleted automatically.

Cookie can be set forever by using the forever method as shown in the below code.

```
$response->withCookie(cookie()->forever('name', 'value'));
```

Retrieving Cookie

Once we set the cookie, we can retrieve the cookie by cookie() method. This cookie() method will take only one argument which will be the name of the cookie. The cookie method can be called by using the instance of **Illuminate\Http\Request**.

Here is a sample code.

```
//'name' is the name of the cookie to retrieve the value of
$value = $request->cookie('name');
```

Response

Basic Response

Each request has a response. Laravel provides several different ways to return response. Response can be sent either from route or from controller. The basic response that can be sent is simple string as shown in the below sample code. This string will be automatically converted to appropriate HTTP response.

Example

Step 1: Add the following code to **app/Http/routes.php** file.

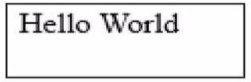
app/Http/routes.php

```
Route::get('/basic_response', function () {
    return 'Hello World';
});
```

Step 2: Visit the following URL to test the basic response.

```
http://localhost:8000/basic_response
```

Step 3: The output will appear as shown in the following image.



Attaching Headers

The response can be attached to headers using the header() method. We can also attach the series of headers as shown in the below sample code.

```
return response($content,$status)
            ->header('Content-Type', $type)
            ->header('X-Header-One', 'Header Value')
            ->header('X-Header-Two', 'Header Value');
```

JSON Response

JSON response can be sent using the json method. This method will automatically set the Content-Type header to application/json. The json method will automatically convert the array into appropriate json response.

Example

Step 1: Add the following line in **app/Http/routes.php** file.

app/Http/routes.php

```
Route::get('json',function(){
     return response()->json(['name' => 'Virat Gandhi', 'state' => 'Gujarat']);
});
```

Step 2: Visit the following URL to test the json response.

```
http://localhost:8000/json
```

Step 3: The output will appear as shown in the following image.

```
{"name":"Virat Gandhi","state":"Gujarat"}
```

Json Response

Views

the presentation logic. Views are stored in **resources/views** directory. Generally, the view contains the HTML which will be served by the application.

Example

Step 1: Copy the following code and save it at **resources/views/test.php**

```
<html>
    <body>
        <h1>Hello, World</h1>
    </body>
</html>
```

Step 2: Add the following line in **app/Http/routes.php** file to set the route for the above view.

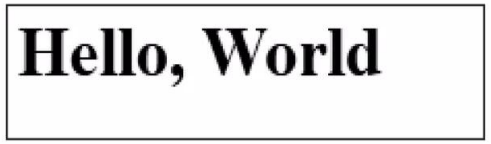
app/Http/routes.php

```
Route::get('/test', function(){
     return view('test');
});
```

Step 3: Visit the following URL to see the output of the view.

```
http://localhost:8000/test
```

Step 4: The output will appear as shown in the following image.



Passing Data to Views

While building application it may be required to pass data to the views. Pass an array to view helper function. After passing an array, we can use the key to get the value of that key in the HTML file.

Example

Step 1: Copy the following code and save it at **resources/views/test.php**

```
<html>
    <body>
        <h1><?php echo $name; ?></h1>
    </body>
</html>
```

Step 2: Add the following line in **app/Http/routes.php** file to set the route for the above view.

app/Http/routes.php

```
Route::get('/test', function(){
     return view('test',['name'=>'Virat Gandhi']);
});
```

Step 3: The value of the key name will be passed to test.php file and $name will be replaced by that value.

Step 4: Visit the following URL to see the output of the view.

```
http://localhost:8000/test
```

Step 5: The output will appear as shown in the following image.

Sharing Data with all Views

We have seen how we can pass data to views but at times, there is a need to pass data to all the views. Laravel makes this simpler. There is a method called **"share()"** which can be used for this purpose. The share() method will take two arguments, key and value. Typically **share()** method can be called from boot method of service provider. We can use any service provider, AppServiceProvider or our own service provider.

Passing Data to Views

Redirections

Redirecting to Named Routes

Named route is used to give specific name to a route. The name can be assigned using the **"as"** array key.

```
Route::get('user/profile', ['as' => 'profile', function () {
    //
}]);
```

Note: Here, we have given the name **"profile"** to a route **"user/profile"**.

Example

Step 1: Create a view called test.php and save it at **resources/views/test.php**.

```
<html>
    <body>
        <h1>Example of Redirecting to Named Routes</h1>
    </body>
</html>
```

Step 2: In routes.php, we have set up the route for test.php file. We have renamed it to **"testing"**. We have also set up another route **"redirect"** which will redirect the request to the named route **"testing"**.

app/Http/routes.php

```
Route::get('/test', ['as'=>'testing',function(){
     return view('test2');
}]);
Route::get('redirect',function(){
     return redirect()->route('testing');
});
```

Step 3: Visit the following URL to test the named route example.

```
http://localhost:8000/redirect
```

Step 4: After execution of the above URL, you will be redirected to http://localhost:8000/test as we are redirecting to the named route **"testing"**.

Step 5: After successful execution of the URL, you will receive the following output:

Redirecting to Controller Actions

Not only named route but we can also redirect to controller actions. We need to simply pass the controller and name of the action to the **action** method as shown in the following example. If you want to pass a parameter, you can pass it as second argument of action method.

```
return redirect()->action('NameOfController@methodName',[parameters]);
```

Example:

Redirections

Working with Database

Connecting to Database

Laravel has made processing with database very easy. Laravel currently supports following 4 databases:

- MySQL
- Postgres
- SQLite
- SQL Server

The query to the database can be fired using raw SQL, the fluent query builder, and the Eloquent ORM. To understand the all CRUD (Create, Read, Update, Delete) operations with Laravel, we will use simple student management system.

Configure the database in **config/database.php** file and create the college database with structure in MySQL as shown in the following table.

Database: College

Table: student

Column Name	Column Datatype	Extra
Id	int(11)	Primary key | Auto increment
Name	varchar(25)	

We will see how to add, delete, update and retrieve records from database using Laravel in student table.

Insert Records

We can insert the record using the **DB** facade with **insert** method. The syntax of insert method is as shown in the following table.

Syntax	bool insert(string $query, array $bindings = array())
Parameters	• $query(string) – query to execute in database • $bindings(array) – values to bind with queries
Returns	bool
Description	Run an insert statement against the database.

Errors & Logging

Errors

A project while underway, is borne to have a few errors. Errors and exception handling is already configured for you when you start a new Laravel project. Normally, in a local environment we need to see errors for debugging purposes. We need to hide these errors from users in production environment. This can be achieved with the variable **APP_DEBUG** set in the environment file **.env** stored at the root of the application.

For local environment the value of **APP_DEBUG** should be **true** but for production it needs to be set to **false** to hide errors.

Note: After changing the **APP_DEBUG** variable, restart the Laravel server.

Logging

Logging is an important mechanism by which system can log errors that are generated. It is useful to improve the reliability of the system. Laravel supports different logging modes like single, daily, syslog, and errorlog modes. You can set these modes in **config/app.php** file.

```
'log' => 'daily'
```

You can see the generated log entries in **storage/logs/laravel.log** file.

Laravel Localization

Localization feature of Laravel supports different language to be used in application. You need to store all the strings of different language in a file and these files are stored at **resources/views** directory. You should create a separate directory for each supported language. All the language files should return an array of keyed strings as shown below.

```
<?php
return [
    'welcome' => 'Welcome to the application'
];
```

Example

Step 1: Create 3 files for languages — **English**, **French**, and **German**. Save English file at **resources/lang/en/lang.php**

```
<?php
      return [
            'msg' => 'Laravel Internationalization example.'
      ];
?>
```

Step 2: Save French file at **resources/lang/fr/lang.php.**

```
<?php
      return [
            'msg' => 'Exemple Laravel internationalisation.'
      ];
?>
```

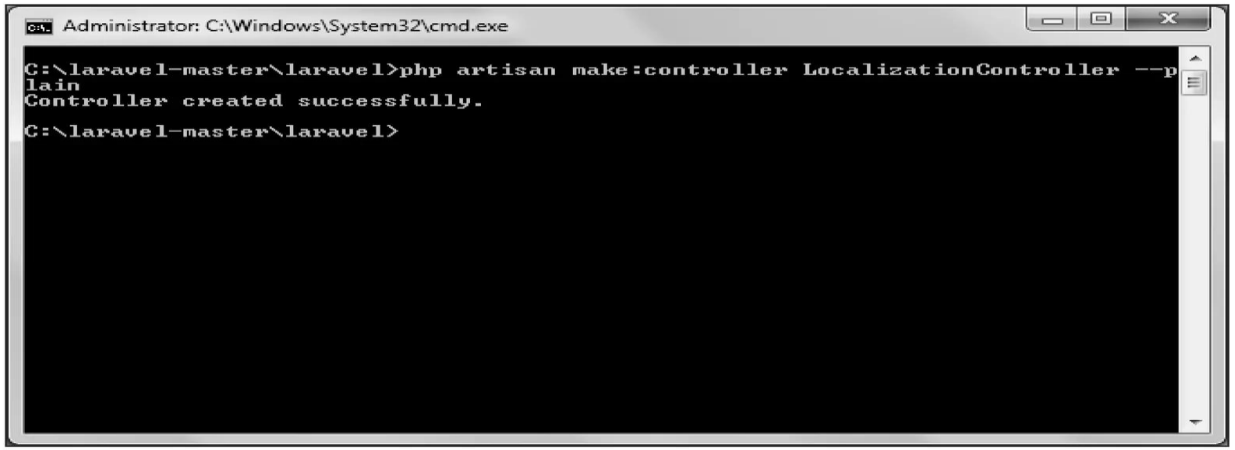
Step 3: Save German file at **resources/lang/de/lang.php.**

```
<?php
      return [
            'msg' => 'Laravel Internationalisierung Beispiel.'
      ];
?>
```

Step 4: Create a controller called **LocalizationController** by executing the following command.

```
php artisan make:controller LocalizationController --plain
```

Step 5: After successful execution, you will receive the following output:



Step 6: Copy the following code to file **app/Http/Controllers/LocalizationController.php**

app/Http/Controllers/LocalizationController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class LocalizationController extends Controller
{
    public function index(Request $request,$locale){
```

Laravel Localization

Laravel Localization

```
            //set's application's locale
            app()->setLocale($locale);

            //Gets the translated message and displays it
            echo trans('lang.msg');
    }
}
```

Step 7: Add a route for LocalizationController in **app/Http/routes.php** file. Notice that we are passing {locale} argument after localization/ which we will use to see output in different language.

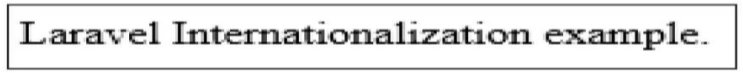
app/Http/routes.php

```
Route::get('localization/{locale}','LocalizationController@index');
```

Step 8: Now, let us visit the different URLs to see all different languages. Execute the below URL to see output in English language.

```
http://localhost:8000/localization/en
```

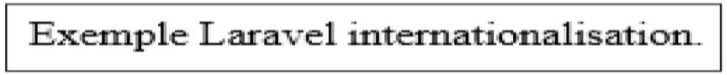
Step 9: The output will appear as shown in the following image.



Step 10: Execute the below URL to see output in French language.

```
http://localhost:8000/localization/fr
```

Step 11: The output will appear as shown in the following image.

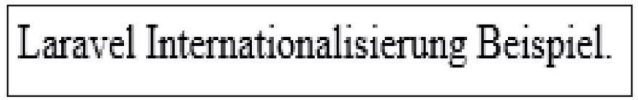


Step 12: Execute the below URL to see output in German language.

Laravel

```
http://localhost:8000/localization/de
```

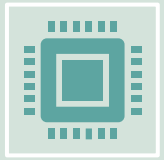
Step 13: The output will appear as shown in the following image.



Laravel Localization

Student's Working

Introduction to Laravel framework.



Students learn Laravel installation, routing mechanisms, middleware, controllers, views, and database operations.

Tasks include creating routes, handling requests, managing cookies, and logging errors.

Thank you

Week 16 & 17

Project submission and final assessment