

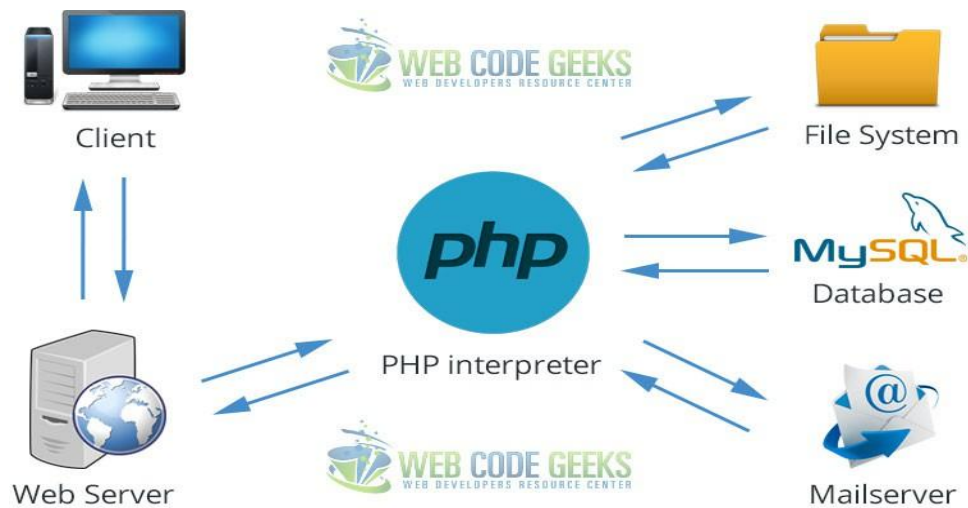
علوم الحاسب

الفرقة الثانية

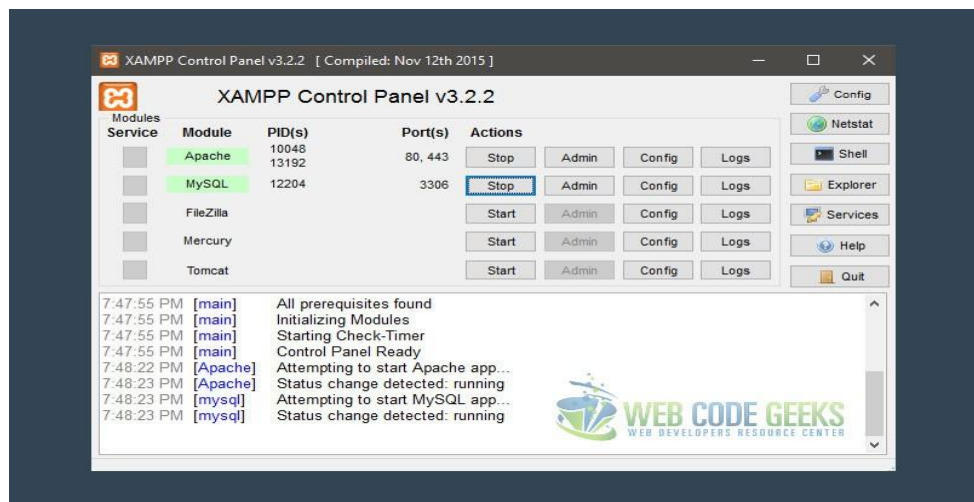
Lab #6

برمجة الويب

Web Programming



1.1 XAMPP Setup



File Explorer view of the XAMPP installation directory:

Path: This PC > Local Disk (C:) > xampp > htdocs

Name	Date modified	Type	Size
yourWebProject	5/20/2016 8:04 PM	File folder	
applications	8/27/2015 5:15 PM	Chrome HTML Do...	4 KB
bitnami	7/21/2015 11:08 PM	Cascading Style S...	1 KB
favicon	7/16/2015 5:32 PM	Icon	31 KB
index	7/16/2015 5:32 PM	PHP File	1 KB

1.2 PHP Language Basics

Canonical PHP Tags

```
<?php...?>
```

1.2.1 Commenting on PHP

```
<?php
    // this is also a comment in PHP, a single line comment
?>

<?php
    /* this is a multi-line comment
       Name: Web Code Geeks
       Purpose: Web Development
    */
?>
```

1.2.2 Hello World

The very basic example of outputting a text in PHP would be:

```
<?
    print("Hello World");
    echo "Hello World";
    printf("Hello World");
?>
```

- **print** returns a value. It always returns 1.
- **echo** can take a comma delimited list of arguments to output.
- **printf** is a direct analog of C's `printf()`.

1.2.3 Variables in PHP

- **Integers** - whole numbers like 23, 1254, 964 etc
- **Doubles** - floating-point numbers like 46.2, 733.21 etc
- **Booleans** - only two possible values, true or false
- **Strings** - set of characters, like *Web Code Geeks*
- **Arrays** - named and indexed collections of other values
- **Objects** - instances of predefined classes

The following snippet shows all these data types declared as variables:

```
<?
    $intNum = 472;
    $doubleNum = 29.3;
    $boolean = true;
    $string = 'Web Code Geeks';
    $array = array("Pineapple", "Grapefruit", "Banana");

    // creating a class before declaring an object variable
    class person {
        function agePrint() {
            $age = 5;
            echo "This person is $age years old!";
        }
    }

    // creating a new object of type person
    $object = new person;
?>
```

1.2.4 Conditional Statements in PHP

```
<?php
    $age = 18;

    if ($age < 20) {
        echo "You are a teenager";
    }
?>
```

Because the condition is true, the result would be:

```
You are a teenager
```

The If... Else statement

```
<?php
    $age = 25;

    if ($age < 20) {
        echo "You are a teenager";
    }
    else {
        echo "You are an adult";
    }
?>
```

The If... Elseif... Else statement

```
<?php
    $age = 3;

    if ($age < 10) {
        echo "You are a kid";
    } elseif ($age < 20) {
        echo "You are a teenager";
    } else {
        echo "You are an adult";
    }
?>
```

1.2.5 Loops in PHP

The for loop

```
for ($i=0; $i < 5; $i++) {
    echo "This is loop number $i";
}
```

The result of this code snippet would be:

```
This is loop number 0
    This is loop number 1
    This is loop number 2
    This is loop number 3
    This is loop number 4
```

The while loop

```
$i=0; // initialization
while ($i < 5) {
    echo "This is loop number $i";
    $i++; // step
}
```

The result of this code snippet would be just the same as before:

```
This is loop number 0
    This is loop number 1
    This is loop number 2
    This is loop number 3
    This is loop number 4
```

The do...while loop

```
$i = 0; // initialization
do {
    $i++; // step
    echo "This is loop number $i";
}
while ($i < 5); // condition
```

This time the first loop number would be 1, because the first echo was executed only after variable incrementation:

```
This is loop number 1
This is loop number 2
This is loop number 3
This is loop number 4
This is loop number 5
```

The foreach loop

```
$var = array('a','b','c','d','e'); // array declaration

foreach ($var as $key) {
    echo "Element is $key";
}
```

This time the first loop number would be 1, because the first echo was executed only after variable incrementation:

```
Element is a
Element is b
Element is c
Element is d
Element is e
```

1.3 PHP Arrays

Arrays are used to store multiple values in a single variable. A simple example of an array in PHP would be:

```
<?php
$languages = array("JS", "PHP", "ASP", "Java");
?>
```

Indexed Arrays

We can create these kind of arrays in two ways shown below:

```
<?php
$names = array("Fabio", "Klevi", "John");
?>
```

```
<?php
// this is a rather manual way of doing it
$names[0] = "Fabio";
$names[1] = "Klevi";
$names[2] = "John";
?>
```

An example where we print values from the array is:

```
<?php
$names = array("Fabio", "Klevi", "John");
echo "My friends are " . $names[0] . ", " . $names[1] . " and " . $names[2];
?>
```

```
// RESULT
My friends are Fabio, Klevi and John
```

Looping through an indexed array is done like so:

```
<?php
    $names = array("Fabio", "Klevi", "John");
    $arrayLength = count($names);

    for($i = 0; $i < $arrayLength; $i++) {
        echo $names[$i];
        echo " ";
    }
?>
```

This would just print the values of the array.

Associative Arrays

Associative arrays are arrays which use named keys that you assign. Again, there are two ways we can create them:

```
<?php
    $namesAge = array("Fabio"=>"20", "Klevi"=>"16", "John"=>"43");
?>
```

```
<?php
    // this is a rather manual way of doing it
    $namesAge['Fabio'] = "20";
    $namesAge['Klevi'] = "18";
    $namesAge['John'] = "43";
?>
```

An example where we print values from the array is:

```
<?php
    $namesAge = array("Fabio"=>"20", "Klevi"=>"16", "John"=>"43");
    echo "Fabio's age is " . $namesAge['Fabio'] . " years old.";
?>
```

```
// RESULT
Fabio's age is 20 years old.
```

Looping through an associative array is done like so:

```
<?php
    $namesAge = array("Fabio"=>"20", "Klevi"=>"16", "John"=>"43");

    foreach($namesAge as $i => $value) {
        echo "Key = " . $i . ", Value = " . $value;
        echo " ";
    }
?>
```

```
Key = Fabio, Value = 20
Key = Klevi, Value = 16
Key = John, Value = 43
```

Multidimensional Arrays

```
<?php
    $socialNetowrks = array (
        array("Facebook", "feb", 21),
        array("Twitter", "dec", 2),
        array("Instagram", "aug", 15));
?>
```

1.4 PHP Functions

The basic syntax of a function that we create is:

```
<?php
function functionName($argument1, $argument2...) {
    // code to be executed
}

functionName($argument1, $argument2...); // function call
?>
```

```
<?php
function showGreeting() { // function definition
    echo "Hello Chloe!";    // what this function does
}
```

```
showGreeting(); // function call
?>
```

This function would just print the message we wrote:

```
Hello Chloe!
```

Another example would be a function with arguments:

```
<?php
function personProfile($name, $city, $job) { // function definition with arguments
    echo "This person is ".$name." from ".$city.".";
    echo "";
    echo "His/Her job is ".$job.".";
}
personProfile("Fabio", "Tirana", "Web Dev");
echo "";
personProfile("Michael", "Athens", "Graphic Designer");
echo "";
personProfile("Xena", "London", "Tailor");
?>
```

The result would include the printed message together with the arguments:

```
This person is Fabio from Tirana.
His/Her job is Web Dev.
This person is Michael from Athens.
His/Her job is Graphic Designer.
This person is Xena from London.
His/Her job is Tailor.
```

In PHP functions to return a value:

```
<?php
function difference($a, $b) { // function definition with arguments
    $c = $a - $b;
    return $c;
}

echo "The difference of the given numbers is: ".difference(8, 3);
?>
```

The result would be:

```
The difference of the given numbers is: 5
```

1.5 Connecting to a Database

1.5.1 Connecting to MySQLi Databases

The MySQLi stands for MySQL improved. The syntax for connecting to a database using MySQLi extension is:

```
<?php
$username = "your_name";
$password = "your_password";
$hostname = "localhost";

//connection to the database
$dbConnect = mysqli_connect($hostname, $username, $password)

// another way of checking if the connection was successful
if (!$dbConnect) {
    die ("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";

//select a specific database
mysqli_select_db($dbConnect, "dbName")
?>
```

1.5.2 Connecting to MySQLi databases (Object-Oriented)

Although the functionality is basically the same, this is another way, the object-oriented way of connecting to a database using the MySQLi extension.

```
$username = "your_name";
$password = "your_password";
$hostname = "localhost";

// create connection
$dbConnect = new mysqli($hostname, $username, $password);

// check connection
if ($dbConnect->connect_error) {
    die("Connection failed: " . $dbConnect->connect_error);
}
echo "Connected successfully";

// select a specific database
$mysqli->select_db("dbName");
```

Even in this case, you can check to see if the database was successfully selected, but it is a matter of choice. Object-Oriented MySQLi is not a different MySQLi as far as code functionality is concerned, but rather a different way/logic of writing it.

1.6 PHP Form Handling

```
<form method="POST" action="wcg.php">
  <input type="text" name="name" placeholder="Name">
  <input type="email" name="email" placeholder="E-Mail">
  <input type="number" name="age" placeholder="Age">
  <input type="radio" name="gender" value="Male"> Male
  <input type="radio" name="gender" value="Female"> Female
  <input type="submit" name="submit" value="Submit">
</form>
```

SAMPLE FORM



Name

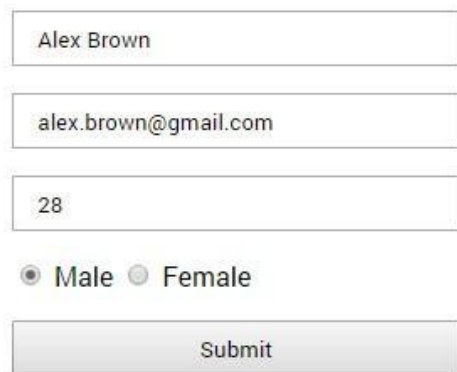
E-Mail

Age

☐ Male ☐ Female

Submit

FILLED FORM



Alex Brown

alex.brown@gmail.com

28

☒ Male ☐ Female

Submit

```
<?php
if (isset($_POST["name"]) && !empty($_POST["name"])) {
    $name = $_POST["name"];
    if (!preg_match("/^[a-zA-Z ]*$/",$name))
        echo "Name: Only letters and whitespace allowed";
    else
        echo "Name: " . $_POST["name"] . " ";
}
if (isset($_POST["email"]) && !empty($_POST["email"])) {
    echo "E-Mail: " . $_POST["email"] . " ";
}
if (isset($_POST["age"]) && !empty($_POST["age"])) {
    echo "Age: " . $_POST["age"] . " ";
}
if (isset($_POST["gender"]) && !empty($_POST["gender"])) {
    echo "Gender: " . $_POST["gender"];
}
?>
```

1.7 PHP Include & Require Statements

The include and require statements are the same, except upon failure of code execution where:

- **require** will produce a fatal error (E_COMPILE_ERROR) and stop the script from executing.
- **include** will only produce a warning (E_WARNING) and the script will continue.

header.php

```
<nav>
  <ul>
    <li>Home</li>
    <li>About</li>
    <li>Contact</li>
  </ul>
</nav>
```

index.php

```
<?php include 'header.php';?>

<body>
  <h3>Page content below</h3>
  <p>Lorem ipsum dolor</p>
</body>
```

index.php

```
<nav>
  <ul>
    <li>Home</li>
    <li>About</li>
    <li>Contact</li>
  </ul>
</nav>

<body>
  <h3>Page content below</h3>
  <p>Lorem ipsum dolor</p>
</body>
```



The syntax of these two statements is as follows:

```
<?php
include 'file.php'; // in case of include
require 'file.php'; // in case of require
?>
```

Let's now see a real-world example where we use a header and footer using include and require into our main file:

header.php

```
<header>
<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">Profile</a></li>
    <li><a href="#">About</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
</nav>
</header>
// styling to make the menu look right
<style type="text/css">
  ul, li {
    list-style-type: none;
    display: inline-block;
    margin-right: 1em;
    padding: 0;
  }
</style>
```

footer.php

```
<footer>All Rights Reserved. Do not reproduce this page.</footer>
```

In our main file, we'll use `require` for the header.php and `include` for the footer.php file:

index.php

```
<?php require 'header.php'; ?>

<body>
    <h2>Main Content Goes Here</h2>
</body>

<?php include 'footer.php' ?>
```

The result, as you might have an idea by now, would be the whole code being shown as one:

[Home](#) [Profile](#) [About](#) [Contact](#)

Main Content Goes Here

All Rights Reserved. Do not reproduce this page.

1.8 Object Oriented Concepts

Object oriented programming is a programming language model in the center of which are objects. Let's see some of the core concepts of classes and objects before we see actual examples of them:

- **Class**

A class is a predefined (by a programmer) data type, part of which are local data like variables, functions etc.

- **Object**

An object is an instance of a class. Objects can only be created after a class has been define. A programmer can create several objects.

- **Constructor**

A constructor refers to the concept where all data and member functions are encapsulated to form an object. A destructor, on the other hand, is called automatically whenever an object is deleted or goes out of scope.

1.8.1 PHP Classes

Let's now define a new clas in PHP.

```
<?php
class myClass {
    var $var1;           // use 'class' followed by the name of the class
    var $var2 = 5;       // declared an undefined variable
    var $var3 = "string"; // declared a number defined variable
                        // declared a string defined variable

    function myFunction ($argument1, $argument2) { // function definition
        // function code here
    }
}
?>
```

```

<?php
class Vehicle {
    var $brand;          // just a declared undefined variable
    var $speed = 80;     // a declared and defined variable

    function setSpeed($speedValue) { // a function to change speed
        $this->speed = $speedValue;  // this will replace speed with our value
    }

    function setBrand($brandName) { // a function to change brand
        $this->brand = $brandName;   // this will set a brand name
    }

    function printDetails(){ // a function to print details
        echo "Vehicle brand is: ".$this->brand;
        echo " ";
        echo "Vehicle speed is: ".$this->speed;
    }
}

$myCar = new Vehicle; // an instance of our Vehicle class (an object)
$myCar->setBrand("Audi"); // calling the function setBrand to define a brand
$myCar->setSpeed(120); // calling the function setSpeed to change speed
$myCar->printDetails(); // calling the printDetails function to see details
?>

```

The result of this code would be:

```

Vehicle brand is: Audi
Vehicle speed is: 120

```

As you can see, the speed is changed to the latest value set by us.

1.8.2 PHP Constructor Function

PHP provides a special function called `__construct()` to define a constructor, which can take as many arguments as we want. Constructors are called automatically whenever an object is created.

```

<?php
class Vehicle {
    function __construct ($brandName, $speedValue) {
        $this->brand = $brandName; // initialize brand
        $this->speed = $speedValue; // initialize speed
    }

    function printDetails(){
        echo "Vehicle brand is: ".$this->brand;
        echo " ";
        echo "Vehicle speed is: ".$this->speed;
        echo " ";
    }
}

$car1 = new Vehicle("Toyota", 130);
$car2 = new Vehicle ("Ferrari", 450);

$car1->printDetails();
$car2->printDetails();
?>

```

The result now is:

```

Vehicle brand is: Toyota
Vehicle speed is: 130

Vehicle brand is: Ferrari
Vehicle speed is: 450

```