Introduction

IMY 220 ● Lecture 4

What is PHP?

PHP: Hypertext Preprocessor.

PHP is a server-side scripting language: PHP scripts are executed on the server.

PHP files are returned to the client (browser) as plain HTML.

- If you want to view the source in the browser, you will not see PHP.

Why PHP?

PHP runs on different platforms (Windows, Linux, Unix, etc.).

PHP is compatible with almost all servers used today (Apache, IIS, etc.).

PHP supports a wide range of databases. Writing a database-enabled web page is incredibly simple using one of the database specific extensions (e.g., for mysql).

Why PHP?

PHP is FREE to download from the official PHP resource: www.php.net

PHP is an open source software (OSS).

What is a PHP File?

PHP-enabled web pages are treated just like regular HTML pages and you can create and edit them the same way you normally create regular HTML pages.

PHP files may contain text, HTML tags, and scripts.

The default file extension is".php".

Take note

File naming conventions:

- using-dashes
- usingCamelCase
- using_underscores

Most important is to be consistent with your naming rules

What is a PHP File?

hello.php

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8" />
		<title> Example </title>
	</head>
	<body>
		<?php
			echo "Hi, I'm a PHP script!";
		?>
	</body>
</html>
```

What do you need?

You need a server that supports PHP.

If your server supports PHP, then you do not need to do anything. Just create your .php files, put them in your web directory and the server will automatically parse them for you.

What do you need?

If you want to develop locally, you will need to install a web server, such as **Apache**, and of course **PHP**. As well as a database, such as **MySQL**.

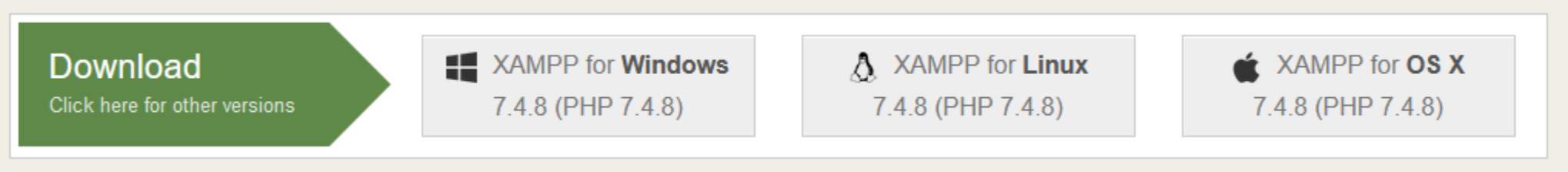
The simplest way is to use a pre-configured package.

I recommend XAMPP because it is cross-platform

What is XAMPP?

XAMPP is the most popular PHP development environment.

XAMPP is a completely free, easy to install Apache distribution containing MySQL, PHP, and Perl. The XAMPP open source package has been set up to be incredibly easy to install and to use.



https://www.apachefriends.org/index.html

XAMPP

Store all your files in a folder in the HTDOCS folder for XAMPP.

Open a browser and type in either:

http://localhost/(your folder name)/(filename)

or

http://127.0.0.1/(your folder name)/(filename)

Leave the filename out to automatically load index.php or index.html

XAMPP – Change Port

Open XAMPP -> Click on **Config** next to Apache -> Apache (httpd.conf)

Change the following two lines:

- `Listen 80`
- `ServerName localhost:80`

Replace “80” with whatever port you want to use instead

XAMPP – Set Error Handling

Open XAMPP -> Click on **Config** next to Apache -> PHP (php.ini)

Look for the line that starts with…

- `error_reporting = … (your error handling settings)`

And make sure it’s set to the Development value:

- `error_reporting = E_ALL`

XAMPP – Set Error Handling

Also include the following PHP to show all errors and warnings

```php
<?php
	error_reporting(E_ALL);
	ini_set('error_reporting', E_ALL);
?>
```

It is highly recommended that you do this while developing the project, since these errors and warnings will display on the project server

(Also, you should not ignore warnings when developing. They are there for a reason)

Assigning values by reference

```php
<?php
	$foo = 'Bob';                 // Assign the value 'Bob' to $foo
	$bar = &$foo;                 // Reference $foo via $bar.
	$bar = "My name is $bar";     // Alter $bar...
	echo $bar . "<br />";
	echo $foo;                    // $foo is altered too.
?>
```

Output: ?

Assigning values by reference

```php
<?php
	$foo = 'Bob';               // Assign the value 'Bob' to $foo
	$bar = &$foo;               // Reference $foo via $bar.
	$bar = "My name is $bar";   // Alter $bar...
	echo $bar . "<br />";
	echo $foo;                  // $foo is altered too.
?>
```

Output:

My name is Bob

My name is Bob

Assigning values by reference

This means that the new variable simply references (in other words, "becomes an alias for" or "points to") the original variable.

Changes to the new variable affect the original, and vice versa.

To assign by reference, simply prepend an ampersand (&) to the beginning of the variable which is being assigned (the source variable).

Types: *arrays*

An array can be created using the array() language construct. It takes any number of comma-separated key => *value* pairs as arguments.

The key can either be an *integer* or a *string*. The value can be of any type.

```
array(
    key  => value,
    key2 => value2,
    key3 => value3,
    ...
)
```

http://php.net/manual/en/language.types.array.php

Types: *arrays*

As of PHP 5.4 you can also use the short array syntax, which replaces array() with [ ].

```php
<?php
	$array = array(
		"foo" => "bar",
		"bar" => "foo",
	);
?>
```

```php
<?php
	// as of PHP 5.4
	$array = [
		"foo" => "bar",
		"bar" => "foo",
	];
?>
```

Note: The comma after the last array element is optional and can be omitted.

PHP Arrays

There are three different kind of arrays in PHP:

- **Numeric** array - An array with a numeric ID key
- **Associative** array - An array where each ID key is associated with a value
- **Multidimensional** array - An array containing one or more arrays

Accessing elements with square bracket syntax

Array elements can be accessed using the *array[key]* syntax.

```php
<?php
        $array = array(
                "foo" => "bar",
                42    => 24,
                "multi" => array(
                        "dimensional" => array(
                                "array" => "foo"
                        )
                )
        );
        var_dump($array["foo"]);
        var_dump($array[42]);
        var_dump($array["multi"]["dimensional"]["array"]);
?>
```

```
string(3) "bar"
int(24)
string(3) "foo"
```

Control structures: *foreach*

The foreach construct provides an easy way to iterate over arrays. foreach works only on arrays and objects, and will issue an error when you try to use it on a variable with a different data type or an uninitialised variable. There are two syntaxes:

```
foreach (array_expression as $value)
    statement
foreach (array_expression as $key => $value)
    statement
```

Control structures: *foreach*

```php
<?php
	$arr = array(1, 2, 3, 4);
	foreach ($arr as &$value) {
		$value = $value * 2;
	}
	// $arr is now array(2, 4, 6, 8)
	unset($value); // break the reference with the last element
?>
```

PHP Functions

The real power of PHP comes from its functions.

In PHP there are more than 700 built-in functions available.

For all PHP functions go to:

http://www.php.net/manual

PHP Functions

All functions and classes in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa.

Making arguments be passed by reference

By default, function arguments are passed by value (so that if the value of the argument within the function is changed, it does not get changed outside of the function). To allow a function to modify its arguments, they must be passed by reference.

```php
<?php
	function add_some_extra(&$string){
		$string .= 'and something extra.';
	}
	$str = 'This is a string, ';

	add_some_extra($str);

	echo $str;    // outputs 'This is a string, and something extra.'
```

PHP Form Handling

The most important thing to notice when dealing with HTML forms and PHP is that **any form element in an HTML page** will automatically be available to your PHP scripts.

HTML form control names

Input **names** are used to access the values entered by the user.

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

HTML form control IDs

Form control IDs are not required (for the page to be valid).

Use them for labels, CSS, jQuery, etc.

```
<label for="username"> Click me </label>
<input type="text" id="username" name="username" />
```

HTML Forms

HTML forms are used to collect user input.

The form-handler is specified in the form's **action** attribute

Specifies the HTTP method (**GET** or **POST**) to be used when submitting the forms

```
<form action="welcome.php" method="GET">
    Name: <input type="text" name="name" /><br/>
    E-mail: <input type="text" name="email" /><br/>
    <input type="hidden" name="hiddenData" /><br/>
    <input type="submit" />
</form>
```

Defines a button for **submitting** a form to a **form-handler**

A hidden input sends through data, but isn't visible in the form

Predefined variables: $_GET

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

The $_GET variable is used to collect values from a form with **method="get"**.

Predefined variables: $_GET

Information sent from a form with the GET method is **visible in the URL** and it has limits on the amount of information to send.

Maximum URL length is 2048 characters.

```
welcome.php?name=Diffie+&email=isak.bosman%40up.ac.za
```

Predefined variables: $_GET

index.html:

```
<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>
```

welcome.php:

```
Welcome <?php echo $_GET["name"]; ?><br />
Your email address is: <?php echo $_GET["email"]; ?>
```

```
Welcome John
Your email address is john.doe@example.com
```

Predefined variables: $_POST

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

The $_POST variable is used to collect values from a form with **method="post"**.

Information sent from a form with the POST method is **invisible to others** and has no limits on the amount of information to send.

Predefined variables: $_POST

index.html:

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

welcome.php:

```
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
```

```
Welcome John
Your email address is john.doe@example.com
```

GET & POST

Both GET and POST create an array

array(key => value, key2 => value2, key3 => value3, ...)

This array holds key/value pairs:

keys are the names of the form controls

values are the input data from the user

These are **superglobals**, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

GET & POST

GET vs. POST?

When + why?

Why Use $_GET?

When using the $_GET variable all variable names and values are displayed in the URL. So this method should NOT be used when sending passwords or other sensitive information!

However, because the variables are displayed in the URL, **it is possible to bookmark the page**. This can be useful in some cases.

The HTTP GET method is **not suitable on large variable values**.

GET values can also be cached by the browser

Why Use $_POST?

Variables sent with HTTP POST are **not shown** in the URL.

Variables have **no length limit**.

However, because the variables are not displayed in the URL, it is **not possible to bookmark the page**.

POST values are also not cacheable by the browser.

GET & POST

Generally speaking: GET should be used when retrieving data…

…while POST should be used for inserting/updating data

The $_REQUEST Variable

The PHP $_REQUEST variable is an associative array that contains the contents of:

- $_GET
- $_POST
- $_COOKIE

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

```
$name = $_REQUEST['fname'];
```

The $_REQUEST Variable

Using $_REQUEST should be unnecessary with proper site design.

$_REQUEST is not often recommended, because it includes extra, potentially unrelated, data in the form of $_COOKIE

Connecting to a MySQL Database

Before you can access and work with data in a database, you must create a connection to the database.

Previously we used mysql_connect to open a connection to a MySQL Server.

This extension is deprecated as of PHP 5.5.0, and have been removed as of PHP 7 in favour of MySQLi

http://php.net/manual/en/function.mysql-connect.php

Connecting to a MySQL Database

The MySQLi extension ("i" = improved).

The mysqli extension allows you to access the functionality provided by MySQL 4.1 and above.

What is MySQL?

MySQL is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL).

SQL is the most popular language for adding, accessing and managing content in a database.

Connecting to a MySQL Database

MySQLi Procedural

Store the connection in a variable ($conn) for later use in the script.

The "die" part will be executed if the connection fails:

```
$servername = "localhost";
$username = "username";
$password = "password";
$db = "database_name";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $db);
// Check connection
if (!$conn)
    die("Connection failed: " . mysqli_connect_error());
```

Connecting to a MySQL Database

MySQLi Object Oriented (OO)

Store the connection in a mysqli object

```php
$servername = "localhost";
$username = "username";
$password = "password";
$db = "database_name";

// Create connection
$mysqli = new mysqli($servername, $username, $password, $db);
// Check connection
if ($mysqli->connect_errno)
    die("Connection failed: " . $mysqli->connect_error);
```

Closing a Connection

The connection will be closed as soon as the script ends.

To close the connection before, use the mysqli_close() function.

Procedural

```
mysqli_close($conn);
```

OO

```
$mysqli->close();
```

Create a Database

The CREATE DATABASE statement is used to create a database in MySQL.

```
CREATE DATABASE myDB
```

Create a Database

To get PHP to execute the statement above we must use the mysqli_query() function (procedural)

…or call the query() function from the mysqli object (OO).

This function is used to send a query or command to a MySQL connection.

Create a Database

Create a database called “myDB” (procedural):

```
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . mysqli_error($conn);
}
```

Create a database called “myDB” (OO):

```
$sql = "CREATE DATABASE myDB";
if (mysqli->query($sql)) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . mysqli_error($conn);
}
```

Select database

You can select the default database with 4th parameter in mysqli_connect().

You can use the mysqli_select_db function to change the default database.

Create a Table

The **CREATE TABLE** statement is used to create a database table in MySQL.

```
CREATE TABLE tblName
```

Create a Table

We will create a table named MyGuests, with five columns: id, firstname, lastname, email and reg_date:

```
// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";
```

Create a Table

We must add the CREATE TABLE statement to the mysqli_query() function to execute the command.

```
if (mysqli->query($conn, $sql)) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . mysqli_error($conn);
}
```

Insert data from a form into a database

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)";
```

Insert data from a form into a database

When a user clicks the submit button in an HTML form, the form data is sent to the php file.

The php file connects to a database, and retrieves the values from the form with the PHP $_POST/$_GET variables.

Then, the mysqli_query() function executes the INSERT INTO statement, and a new record will be added to the database table.

Insert data from a form into a database

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('$_POST[firstname]',
        '$_POST[lastname]',
        '$_POST[email]')";

if (mysqli->query($sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $mysqli->error;
}

$mysqli->close();
```

SELECT

The SELECT statement is used to select data from one or more tables:

```
SELECT column_name(s) FROM table_name
```

or we can use the * character to select ALL columns from a table:

```
SELECT * FROM table_name
```

SELECT

```
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"] .
             " - Name: " .
             $row["firstname"]. " " . $row["lastname"].
             "<br>";
    }
} else {
    echo "0 results";
}

mysqli_close($conn);
```

SELECT

Set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table.

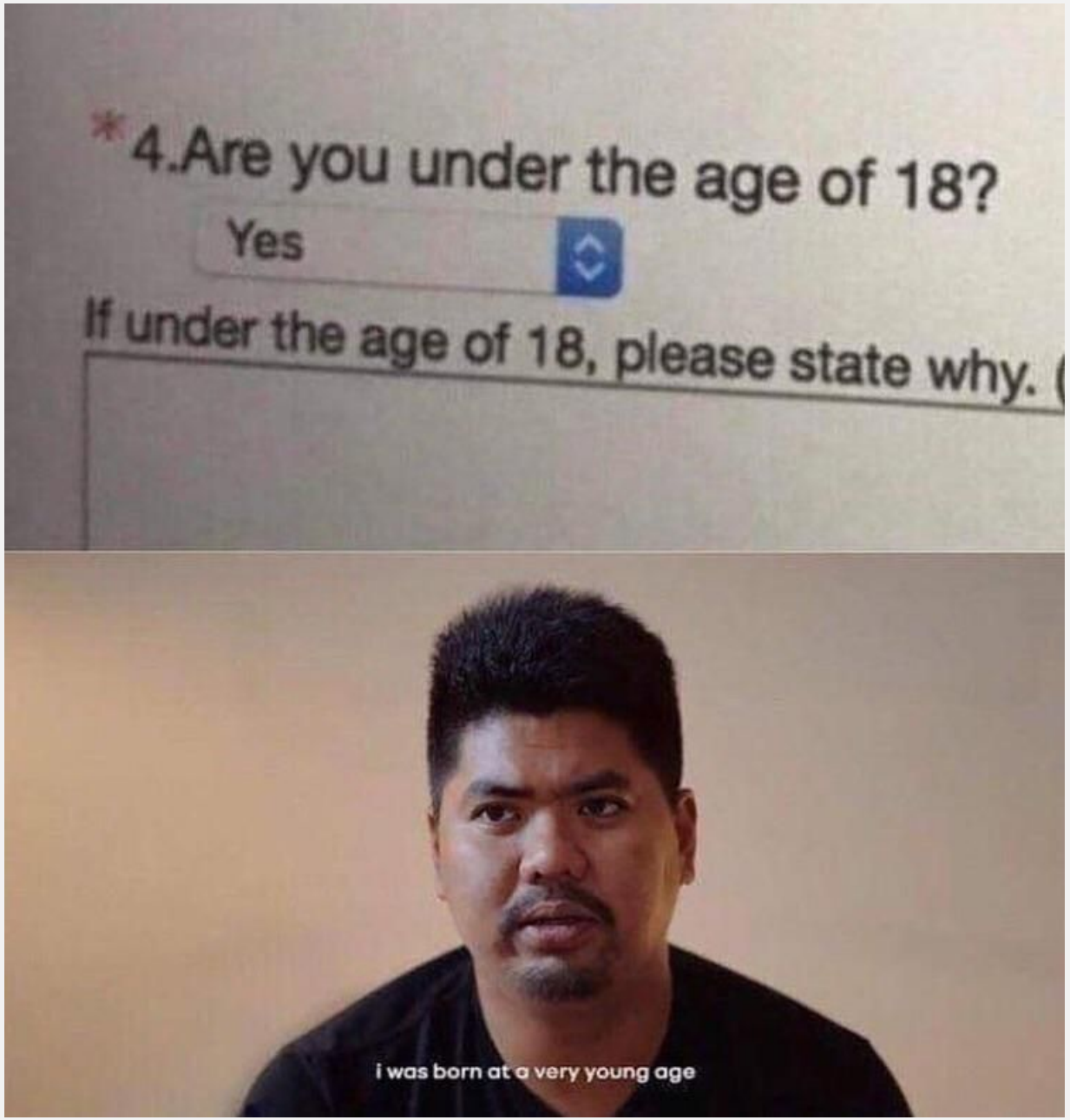
The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the function num_rows() checks if there are more than zero rows returned.

SELECT

If there are more than zero rows returned, the function fetch_assoc() puts all the results into an associative array that we can loop through.

The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.


* 4.Are you under the age of 18?
Yes
If under the age of 18, please state why.
i was born at a very young age