


PHP

CPIT-405 Web Applications

Table of Contents

Server-side Scripting

- Server-side scripting is a technique used in web development that involves running a script on the web server.
- The server-side script enables you to make your web application more dynamic with a full access to the file system (e.g., store and retrieve from a database).
- Unlike client-side scripting, server-side scripting hides the source code from client applications.
- The script often results in HTML sent back to the client or produces a structured response (JSON) for each client's request.

PHP Programming

- PHP is a server-side or backend scripting language
- PHP Stands for the recursive acronym Hypertext Preprocessor
- PHP version 1.0 released in 1995
- PHP 8.x is the current stable release
- PHP code may be embedded into HTML and saved as a .php file
- Large-scale web applications are written in PHP such as Wikipedia, WordPress, Etsy, Tumblr, and Facebook
  - Facebook uses a dialect of PHP called Hack, which runs in a virtual machine called the HipHop Virtual Machine (HHVM).
- Many PHP web frameworks (e.g., Laravel, Symfony, CodeIgniter, Slim, and CakePHP)

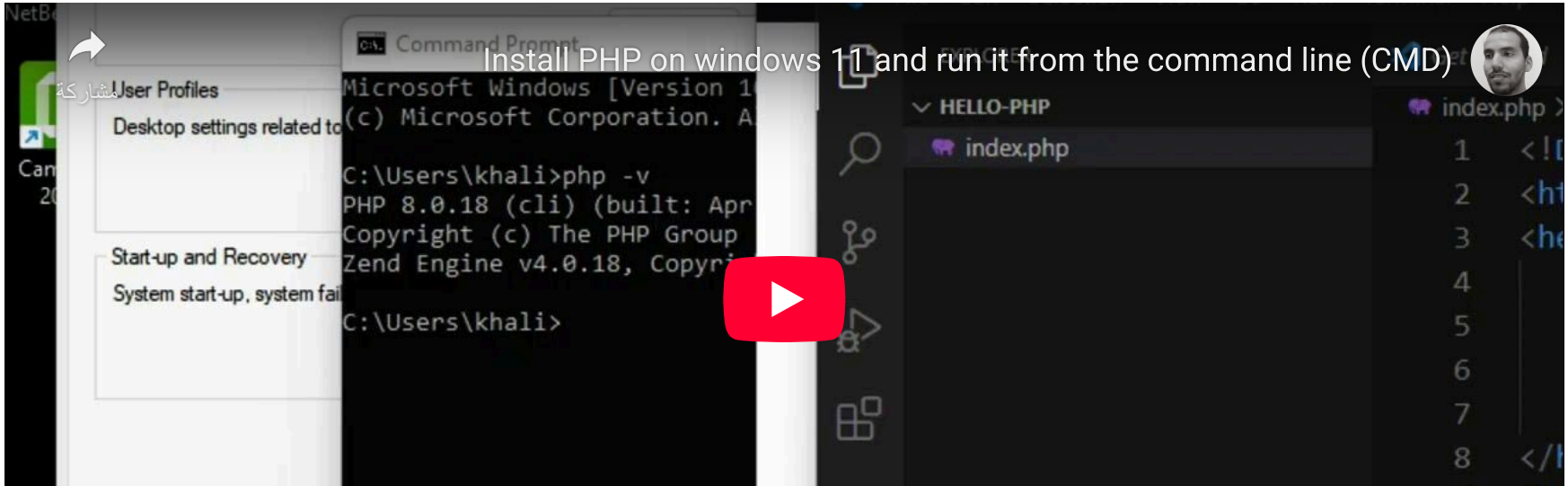
Getting Started: Installation

- To get started with PHP web development, you may need to install the following:
  - A web server (e.g., Apache HTTP Server)
    - https://httpd.apache.org
    - PHP comes with a built-in web server for development but it is not meant for production.
  - PHP, current stable version 8.2
    - http://php.net
  - MySQL Community Server or MariaDB
    - https://dev.mysql.com/downloads/mysql
    - https://mariadb.org
  - Alternatively, you can install a solution stack such as XAMPP.
    - https://www.apachefriends.org
- Next, we will look at how to install PHP on Windows, macOS, and Linux.

PHP Installation on Windows

- Download the zip file at https://windows.php.net/download.
- Extract the ZIP file.
- Add the extracted directory to the *PATH* environment variable.
- Start the *Command Prompt*/*CMD* application and run `php -v`

Below is a video tutorial on how to install PHP on Windows 11, configure the PATH environment variable, and run PHP from the *Command Prompt (CMD)/PowerShell*.



PHP Installation on macOS

The easiest way to install PHP on macOS is to use a package manager such as Homebrew or MacPorts

Using Homebrew

- Download and install from brew.sh
- Install PHP

```
brew install php
```

Using MacPorts

- Download and install MacPorts from macports.org
- Install PHP

```
sudo port install php
```

PHP Installation on Linux

- Installing PHP on Linux depends on the Linux distribution you are using.
- On a debian based Linux system, you may use `apt` to install PHP:

```
apt install php-common php-cli
```

- On an RPM-based distributions such as Red Hat Linux or Rocky Linux, you can use *yum*:

```
yum install php
```

Running PHP from the command line

1. Open your terminal app (e.g., Powershell on Windows or Terminal app on macOS)
2. Go to the directory/folder where your PHP project is stored at (e.g., `cd C:\path\tp\project/`).
3. Run PHP code from the command line:
   - Run the code and get the output in the console using: `php –f path/to/php-script.php`
   - Run the built-in web server using: `php -S localhost:4000 -t .` and open your browser and navigate to *http://localhost:4000*
     - `-S` is for running the built-in web server at the given host and port (e.g., http://localhost:4000).
     - `-t` specifies the document root or directory/folder for the built-in web server.
     - `.` refers to the current working directory you cd'ed to in the second step.

Your first PHP code

index.php

```
<!DOCTYPE html>
<html lang="en">

<head>
    <title>Hello PHP</title>
</head>

<body>

    <?php
        echo "<p>Hello PHP</p>";
    ?>
</body>

</html>
```

- Save the file as *index.php*
- run `php -S localhost:3000`
- navigate to `http://localhost:3000/index.php`

PHP Basic Syntax

- Files containing PHP source code typically have the **.php** extension.
- A PHP script begins with `<?php` and ends with `?>`
- Variable names start with the $ sign followed by the name of the variable (e.g., `$foo = 5;`)
- Variable names are case-sensitive.
  - Example: `$foo=10;` and `$Foo=1;` are two different variables.
- Single-line comments start with `//` or `#`. Multi-line comments start with `/*` and end with `*\`

```
// This is a single-line comment.
#  This is a single-line comment.
/* This is a multi-line comment.
   It can span multiple lines.
*/
```

- PHP Statements end with a semicolon `;`
- PHP uses a dynamic type checking system.
- `echo` is a language construct used to display data on the screen.

PHP `echo` and `var_dump`

- `echo` is a language construct in PHP used to output one or more strings.
  - It's not actually a function, so you can use it without parentheses

```php
<?php
echo "Hello, World!";
?>
```

```
Hello, World!
```

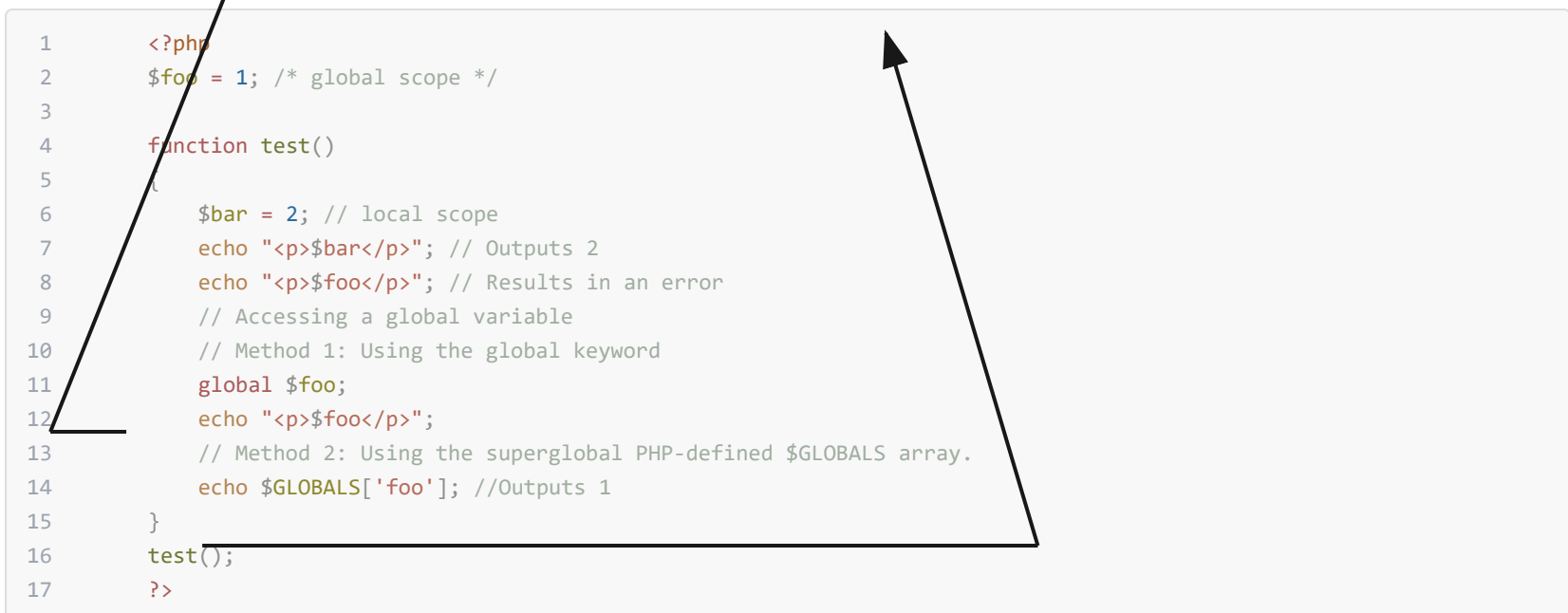
- `var_dump()` is a function that displays structured information about variables/expressions including its type and value
  - It's used mainly for debugging purposes

```php
<?php
$arr = array('a', 'b', 'c');
var_dump($arr);
?>
```

```
array(3) {
  [0]=>
  string(1) "a"
  [1]=>
  string(1) "b"
  [2]=>
  string(1) "c"
}
```

Variable Scopes

- In PHP, functions cannot access global variables without explicit declaration.
- We can access global variables in one of two ways:
  1. Use the `global` keyword inside the function.
  2. Use the special PHP-defined superglobal array `$GLOBALS`.



```php
<?php
$foo = 1; /* global scope */

function test()
{
    $bar = 2; // local scope
    echo "<p>$bar</p>"; // Outputs 2
    echo "<p>$foo</p>"; // Results in an error
    // Accessing a global variable
    // Method 1: Using the global keyword
    global $foo;
    echo "<p>$foo</p>";
    // Method 2: Using the superglobal PHP-defined $GLOBALS array.
    echo $GLOBALS['foo']; //Outputs 1
}
test();
?>
```

PHP Data Types (I)

Data Type	Description	Example
Integer	Whole numbers without a decimal point.	`$x = 405;`
Float	Numbers with a decimal point or in exponential form.	`$y = 3.14159;`
String	Sequence of characters.	`$s = 'This is a string';`
Boolean	true or false.	`$b = true;`
Array	Stores multiple values	`$a = array("JavaScript", "React", "PHP");`

PHP Data Types (II)

Data Type	Description	Example
Object	Instances of classes.	`$obj = new MyClass();`
Resource	Holds a reference to an external resource, such as a file or database connection.	`$f = fopen("test.txt", "r");`
NULL	Represents no value.	`$foo = NULL;`

Superglobals (I)

- Superglobals are built-in variables that are always available in all scopes.
- PHP has the the following superglobal variables:

Super Global	Description	Example
$GLOBALS	Contains all global variables declared in the script.	`$GLOBALS['VAR_NAME']`
$_GET	Contains the values of query string parameters passed in the URL.	`$_GET['param'];`
$_POST	Contains the values of form data submitted using the POST method.	`$_POST['username'];`
$_REQUEST	Contains the values of both GET and POST requests.	`$_REQUEST['username'];`

Superglobals (II)

Super Global	Description	Example
$_SERVER	Contains information about the server environment, such as the server name, IP address, script name, and request method.	`$_SERVER['SERVER_NAME'];`
$_FILES	Contains information about uploaded files.	`$_FILES['file']['name'];`
$_COOKIE	Contains the values of cookies set for the current browser.	`$_COOKIE['username'];`
$_SESSION	Contains data associated with the current user session.	`$_SESSION['username'];
$_ENV	Contains environment variables set in the server	`$_ENV['PATH'];`

String Functions

Function	Description	Example
`strlen()`	Returns the length of a string.	`$l = strlen("Learning PHP");`
`strrev()`	Reverses a string.	`$w= strrev("Learning PHP");`
`strpos()`	Finds the position of the first occurrence of a substring in a string.	`$p = strpos("Learning PHP", "PHP");`
`substr()`	Returns part of a string.	`$s = substr("Learning PHP", 10);`
`str_replace()`	Replaces all occurrences of the search string with the replacement string	`$r= str_replace("JavaScript", "Learn JavaScript", "PHP");`

Constants

- Constants are variables whose values can't be changed at runtime.
- Constants are automatically global and can be used across the entire script.
- Constant names are case-sensitive
- Constants are defined using the `define()` function:

```php
<?php
    define("COURSE", "CPIT 405");
    echo COURSE; // Outputs "CPIT 405"
?>
```

- We can also declare an array of constants using the `define` function:

```php
define("COURSES", [
  "CPIT-405",
  "CPIT-252",
  "CPIT-490"
]);
echo COURSES[0]; // Outputs "CPIT-405"
```

Operators (I): Arithmetic Operators

Operator	Name	Example	Output
`+`	Addition	`$x = 10; $y = 6; echo $x + $y;`	`16`
`-`	Subtraction	`$x = 10; $y = 6; echo $x - $y;`	`4`
`*`	Multiplication	`$x = 10; $y = 6; echo $x * $y;`	`60`
`/`	Division	`$x = 10; $y = 2; echo $x / $y;`	`5`
`%`	Modulus	`$x = 10; $y = 3; echo $x % $y;`	`1`
`++$x`	Pre-increment	`$x = 10; echo ++$x;`	`11`
`$x++`	Post-increment	`$x = 10; echo $x++;`	`10`

Operators (II): String & Equality Operators

Operator	Name	Example	Output
`.`	String Concatenation	`$txt1 = "CPIT"; $txt2 = "405"; echo $txt1 . "-" . $txt2;`	`CPIT-405`
`==`	Equal	`$x = 100; $y = "100"; var_dump($x == $y);`	`bool(true)`
`!=`	Not equal	`$x = 100; $y = "100"; var_dump($x != $y);`	`bool(false)`
`<>`	Not equal	`$x = 100; $y = "100"; var_dump($x <> $y);`	`bool(false)`

Operator (III): Identity and Comparison Operators

Operator	Name	Example	Output
`===`	Identical	`$x = 100; $y = "100"; var_dump($x === $y);`	`bool(false)`
`!==`	Not identical	`$x = 100; $y = "100"; var_dump($x !== $y);`	`bool(true)`
`<`	Less than	`$x = 10; $y = "20"; var_dump($x < $y);`	`bool(true)`
`<=`	Less than or equal to	`$x = 10; $y = "20"; var_dump($x <= $y);`	`bool(true)`

Operator (IV): Logical Operators

Operator	Name	Example	Output
`&&`	Logical AND	`$x = 6; $y = 3; var_dump($x > 0 && $y > 0);`	`bool(true)`
`and`	Logical AND	`$x = 6; $y = 3; var_dump($x > 0 and $y > 0);`	`bool(true)`
`||`	Logical OR	`$x = 6; $y = -3; var_dump($x > 0 || $y > 0);`	`bool(true)`
`or`	Logical OR	`$x = 6; $y = -3; var_dump($x > 0 or $y > 0);`	`bool(true)`
`!`	Logical NOT	`$x = 6; var_dump(!$x);`	`bool(false)`

String Concatenation Example

```php
<?php
  $x = "CPIT";
  $y = 405;
  echo "<p>Welcome to ".$x."-".$y."</p>"; // String concatenation
  $x.=$y; // Concatenation assignment. It appends $y to $x
  echo $x;
?>
```

Output: ``` Welcome to CPIT-405 CPIT405 ```

Control statements

- PHP has the following control statements:
  - if
  - else
  - elseif/else if
  - switch
  - while
  - do-while
  - for
  - foreach
  - break
  - continue

Control Statements: `if...else`

```php
<?php
function testNum($a) {
    $result = "";
    if ($a > 0) {
        $result = "positive";
    } else {
        $result = "NOT positive";
    }
    return $result;
}
echo(testNum(-5));
?>
```

```
NOT positive
```

Control Statements: `switch`

```php
<?php
/* The PHP_OS_FAMILY is a predefined constant that holds
* the operating system family name PHP was built for.
* One of 'Windows', 'BSD', 'Darwin', 'Solaris', 'Linux' or 'Unknown'.
*/
switch (PHP_OS_FAMILY) {
    case "Windows":
        echo ("You're running Windows");
        break;
    case "Darwin":
        echo ("You're running macOS");
        break;
    default:
        echo ("You're running " . PHP_OS_FAMILY);
}
```

On macOS, the output would be:

```
You're running macOS
```

Control Statements: `for` loop

```php
<?php
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
?>
```

```
12345678910
```

Control Statements: `while` loop

```php
<?php
$i = 1;
while ($i <= 5) {
    echo $i++;
}

?>
```

```
12345
```

Control Statements: `do...while` loop

```php
<?php
$i = 1;
do {
    echo $i++;
} while ($i <= 5);
?>
```

```
12345
```

Control Statements: `foreach` loop

```php
<?php
$teams = array("Lakers", "Warriors", "Bulls", "Celtics");

foreach ($teams as $team) {
  echo "$team <br>";
}
?>
```

```
Lakers <br>Warriors <br>Bulls <br>Celtics <br>
```

Control Statements: `break` and `continue`

```php
<?php
$teams = array("Lakers", "Warriors", "Bulls", "Celtics", "Heat", "Nets");

foreach ($teams as $team) {
    if ($team == "Bulls") {
        continue;
    }
    else if ($team == "Celtics") {
        break;
    }
    echo "$team <br>";
}
?>
```

```
Lakers <br>Warriors <br>
```

Equality check (==) vs Identity check (===)

In PHP, the `==` operator compares values with type conversion while `===` compares values and types with no type conversion

```php
<?php $i = 1;
/* == value comparison with type conversion. */
var_dump($i==1);    // bool(true)
var_dump($i=='1');  // bool(true)
var_dump($i==true); // bool(true)

/* === value and type comparison with no type conversion is done. */
var_dump($i===1);     // bool(true)
var_dump($i==='1');  // bool(false)
var_dump($i===true); // bool(false)
?>
```

Functions (I): Declaration and invocation

- A function is a block of statements that can be reused many times
- Executing a function involves calling it, which is also known as invoking it
  - Function invocation refers to calling or executing a function
- A function can return a value using the `return` statement in conjunction with a value or object

```
function add($num1, $num2) {
  $sum = $num1 + $num2;
  return $sum;
}
echo add(3, 4);
```

```
7
```

Functions (II): Default Parameters

- PHP functions may take default parameters.
- A default parameter is a value that a function uses as a parameter when no argument is provided.
- If we call the function without passing an argument, the default value will be used.

```php
<?php
  function set_department($dept= "IT"){
    return "<p>Department name: ".$dept."</p>";
  }

  echo set_department('CS');
  echo set_department();
?>
```

```
Department name: CS
Department name: IT
```

Arrays (I)

- An array is a data structure that stores one or more values in a single variable.
- There are three types of arrays in PHP:
  - Indexed arrays

```php
$teams = array("Lakers", "Warriors", "Bulls", "Celtics");
```

  - Associative arrays

```php
$ages = array("Ali" => "25", "Saad" => "37", "Badr" => "44");
```

  - Multidimensional arrays.

```php
$cars = array(array("Toyota", 2022, 10), array("BMW", 2021, 15), array("Mercedes", 2020, 12));
```

Arrays (I): Indexed Array

- An indexed array in PHP is an array with numeric indexes or keys.
- Indexed arrays in are zero-indexed, meaning the first element's index is 0.

```php
<?php
  $teams = array("Lakers", "Warriors", "Bulls", "Celtics");
  echo "NBA Teams: ".$teams[0].", ".$teams[1].", ".$teams[2].", and ".$teams[3].".";
?>
```

```
NBA Teams: Lakers, Warriors, Bulls, and Celtics.
```

Arrays (III): Associative array

- An associative array is an array with string keys rather than numeric keys.
- Each key in the associative array is associated with a value, hence the name.
- Associative arrays are more human-readable because their data access mechanism is intuitive and straightforward.

```php
<?php
  $players = array(
    "LeBron James" => "Lakers",
    "Stephen Curry" => "Warriors",
    "Kevin Durant" => "Suns",
    "Giannis Antetokounmpo" => "Bucks"
  );

  echo "LeBron James plays for the " . $players["LeBron James"] . "<br>";
  echo "Stephen Curry plays for the " . $players["Stephen Curry"] . "<br>";
?>
```

```
LeBron James plays for the Lakers<br>
Stephen Curry plays for the Warriors<br>
```

Arrays (IV): Multidimensional Arrays

- A multidimensional array is an array that contains one or more arrays within it.
- The contained arrays can also contain other arrays, and so on, allowing for multiple levels of depth.

```php
<?php
  $players = array (
    array("Kevin Durant", "Suns", 31.0),
    array("Stephen Curry", "Warriors", 30.0),
    array("LeBron James", "Lakers", 24.3),
    array("Giannis Antetokounmpo", "Bucks", 29.1)
  );

  echo "Player: " . $players[0][0] . ", Team: " . $players[0][1] . ", PPG: " . $players[0][2] . "<br>";
  echo "Player: " . $players[1][0] . ", Team: " . $players[1][1] . ", PPG: " . $players[1][2] . "<br>";
?>
```

```
Player: Kevin Durant, Team: Suns, PPG: 31<br>
Player: Stephen Curry, Team: Warriors, PPG: 30<br>
```

Importing PHP Script Files (I): `include` and `require`

- We can save multiple PHP scripts in external files and import them using the `include` or `require` statements

```php
<?php
  include './file1.php';
  require './file2.php';
?>
```

- `require` will produce an error and stop the script upon failure
- `include` will result in a warning and the script will continue running

Importing PHP Script Files (II): `include_once` and `require_once`

- We can also import a PHP script using the `include_once` or `require_once` statements:

```php
<?php
  include_once './file1.php';
  require_once './file2.php';
?>
```

- Recall that both `include` and `require` are similar, but they handle errors differently:
  - `include` will result in a warning while `require` in an error.
- `include_once` and `require_once` are also similar except PHP will check if the file has already been included/required, then it won't include/require it again.

Form Handling (I)

- PHP is often used to handle HTML forms submitted by the user
- This includes server-side form validation for security and application reasons
- There are two PHP superglobals used to collect HTML form data: `$_POST` and `$_GET`
- Both `$_POST` and `$_GET` contain an array of form data in name and value pairs
- `$_GET` is used when form data is passed to PHP via URL parameters
- `$_POST` is used when form data is passed to PHP via the HTTP POST method

Form Handling (II): GET vs POST

- Two HTTP methods are often used for submitting and receiving data between the client and the server: HTTP GET and HTTP POST
- GET
  - Often used to request data from the server
  - URL contains query string (e.g., ?name=CPIT)
  - Query string is sent in the URL of the HTTP GET request
  - URL example: `http://example.com/my_form.php?name=CPIT&number=405`
  - The URL of the request is often added to the browser's history
  - GET requests should be only used to retrieve data from the server and NOT submitting data especially sensitive data

Form Handling (II): GET vs POST (Cont.)

- POST
  - Often used to submit data from the client to the server.
  - Query string is not sent in the URL of the HTTP POST request
  - Form data is sent in the body of the POST request
  - Request example:

```
POST /courses/add_course.php  HTTP/1.1
HOST: example.com
course=cpit&number=405
```

  - The URL of the POST request is not added to the browser's history
  - Reloading the page will result in re-submitting the data to the server

Form Handling Example

index.php

```
<body>
 <form id="myForm" action="myform.php" method="post">
        <label for="userNameInput">User name:</label>
        <input type="text" id="userNameInput" name="uname">
        <label for="emailInput">E-mail:</label>
        <input type="email" id="emailInput" name="email">
        <label for="passwordInput">Password:</label>
        <input type="password" id="passwordInput" name="pw">
        <input type="submit" value="signup">
 </form>
</body>
```

myform.php

```
<body>
<p>Thank you, <?php echo $_POST["uname"]?>, for signing up!</p>
<p> Please check your email: <?php echo $_POST["email"] ?>
to complete registration and confirm your email</p>
</body>
```

Form Validation

- Input elements in HTML forms should be validated both on client and server sides.
- Client–side validation is done using JavaScript
  - Provide better user experience/feedback to the user
  - Users can see errors immediately as they enter the form
  - But you should not rely or trust client-side validation
  - Client-side validation can be easily bypassed and manipulated
- Server-side validation must be always done for all forms
  - Perform input validation and sanitization on the server to protect against malicious data (e.g., SQL injection).
  - More on SQL injection next.

Client-side Form Validation

- Using the HTML validation attributes: `type` and `required`

```
Email: <input type="text" type="email" name="email" required>
```

- Or using JavaScript

```
var uname = document.forms["myForm"]["username"].value;
  if (uname && uname.trim() == "") {
     document.getElementById("errorSection").innerText= "username is required";
     return false;
  }
```

Server-side Form Validation (I)

- Validation ensure the email is in a valid format.
- Sanitization ensures the email cleans the bad characters out of the email.
  - Sanitization can be done using the built in `FILTER_SANITIZE_EMAIL` constant or you write your own code to remove white spaces and strip any invalid characters.

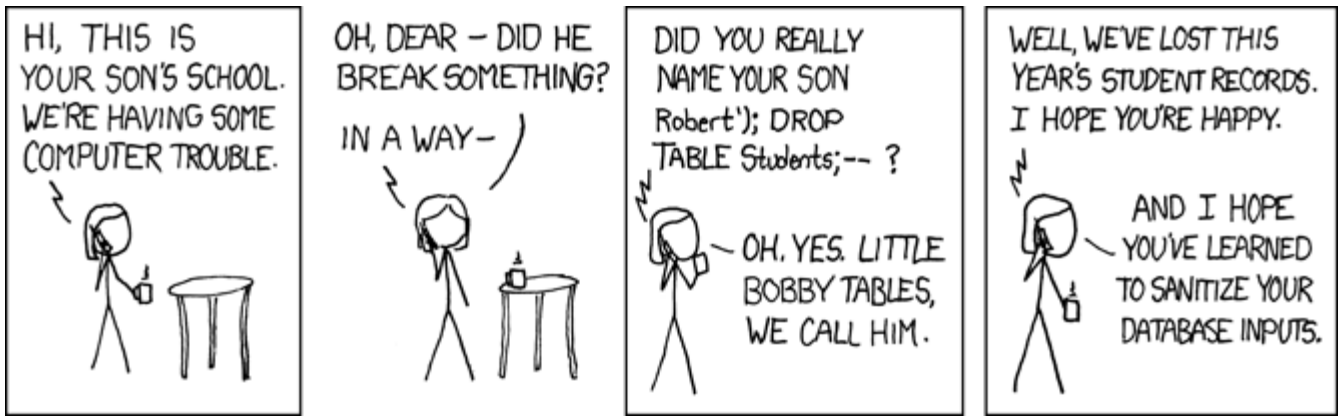
```
1
2   <?php
3   $clean_email = filter_var($email, FILTER_SANITIZE_EMAIL);
4
5   // Validate the input
6   if (filter_var($clean_email, FILTER_VALIDATE_EMAIL)) {
7     echo "Valid email address.";
8   } else{
9       echo "Invalid email address.";
10  }
```

Server-side Form Validation (II)

- Forms may be validated using PHP built-in functions:
  - `filter_var()` uses a set of predefined filters
  - `preg_match()` performs regular expression matching
- Third-party libraries are also available
  - Example: https://github.com/Respect/Validation
  - In most cases, it is better to use a library that has been rigorously tested in the wild than writing your own regular expressions.

What is SQL Injection

https://xkcd.com/327/



- SQL Injection is a code injection technique that attackers use to insert malicious SQL code into input fields to get the application to perform unintended actions.
- This can allow the attacker to view, modify, and delete data in the database.
- The attacker provides input that causes the application to generate an SQL query with unintended commands.
- This can occur when user input is incorrectly filtered or parameterized.

Prevention of SQL Injection

- Always sanitize and validate any user input before executing SQL code.
- Use parameterized queries or prepared statements to ensure that user input is not treated as part of the SQL command.
- Use automated tools to test for SQL injection vulnerabilities.

Cookies

- Cookies are small files that are stored on a user's computer.
- They are designed to store a small amount of data on a user's computer.
- In PHP, cookies can be set using the `setcookie()` function.
- The `setcookie()` function must appear BEFORE the `<html>` tag.
- Syntax: `setcookie(name, value, expire, path, domain, secure, httponly);`
- To retrieve a cookie value, you can use the `$_COOKIE` superglobal variable. For example, `$_COOKIE["name"]`.
- Cookies are part of the HTTP header, so `setcookie()` must be called before any output is sent to the browser.
- Cookies can be deleted by setting expiration date in the past.

Common Usages of Cookies in PHP

- **Session Management**: Cookies are often used to track user sessions. This allows the server to remember the state of a user's interaction with the website over multiple requests.
- **Personalization**: Cookies can be used to remember user preferences, such as language, theme, or other settings. This allows the website to provide a personalized experience for the user.
- **Tracking**: Cookies can be used to track user behavior on a website. This information can be used for analytics, advertising, or to provide recommendations.
- **Authentication**: Cookies can be used to remember a user's login information, allowing them to stay logged in over multiple sessions.

Cookies example

```php
<?php
// Setting a cookie in PHP secure (HTTPS) and httponly
setcookie("language", "arabic", time() + (86400 * 30), "/", true, true); // 86400 = 1 day

// Checking if the cookie is set and retrieving its value
if(isset($_COOKIE["language"])) {
    echo "Value of 'language' cookie is: " . $_COOKIE["language"];
} else {
    echo "'language' cookie is not set";
}
?>
```

Deleting a cookie

To delete a cookie, you can set the cookie with a past expiration date in seconds

```php
setcookie("language", "", time() - 3600, "/");
```

Session Management

- Sessions are a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the users' computer.
- Session variables hold information about a single user and are available to all pages in your application.
- To start a session in PHP, use the `session_start()` function.
  - This function must be the very line of your code before any HTML tags.
- PHP automatically generates a unique session ID for each visitor.
- We can store data in the session by setting the `$_SESSION` superglobal array.
- To destroy a session in PHP, use the `session_destroy()` function.
- Sessions are more secure than cookies as the data is stored on the server.
- However, session data is temporary and will be deleted after the user has left the website or after a certain period of inactivity or the server is rebooted.

Session Example

```php
<?php
// Start the session
session_start();

// Set session variables
$_SESSION["username"] = "Ali";
$_SESSION["email"] = "ali@example.com";

// Access session variables
echo "Username is: " . $_SESSION["username"] . "<br>";
echo "Email is: " . $_SESSION["email"];
?>
```

Destroying Session

```php
<?php
// Start or resume the session
session_start();

// Destroy the session
session_destroy();
?>
```

PHP and MySQL

CRUD Web Application

Create, Read, Update, and Delete (CRUD)

Creating a CRUD Web App in PHP

- CRUD stands for Create, Read, Update, and Delete
- We will create a simple “To Do” app in PHP that communicates with a MySQL database
- The server-side code will make a connection to the database server
- The client application sends requests to create, retrieve, update, and delete items
- The server-side code will handle all CRUD requests using PHP and communicates with our MySQL database and returns html to the client.

MariaDB Database installation

Windows

- Download the MariaDB MSI package from the official MariaDB website and follow the instructions in the installation wizard.
- If MariaDB was installed with the MSI package, it should have been configured to start automatically with Windows.

macOS

- Use Homebrew: Run `brew install mariadb` in the terminal.
- Start MariaDB server with `brew services start mariadb`.

Linux

- Use the package manager for your distribution. For example, on Ubuntu, you can use `sudo apt-get install mariadb-server`.
- Start the MariaDB service with `sudo systemctl start mariadb`.

Connect to MariaDB from Terminal

- Open command-line client (Command Prompt or PowerShell on Windows or Terminal application on macOS and Linux).
- Run the command: `mysql -u username -p`
- Replace `username` with your MariaDB username.
- You'll be prompted to enter your password.
- It supports prepared statements which can help prevent SQL injection attacks.

Create the Database

```
CREATE DATABASE todo_db;
USE todo_db;
CREATE TABLE tasks(
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    task VARCHAR(255) NOT NULL,
    date_added DATETIME NOT NULL,
    done BOOLEAN NOT NULL DEFAULT false,
    PRIMARY KEY (id)
    );
INSERT INTO tasks(task, date_added) VALUES ('Workout', NOW());
INSERT INTO tasks(task, date_added) VALUES ('Water the plants', NOW());
INSERT INTO tasks(task, date_added) VALUES ('Call Mom', NOW());
```

PHP PDO

- PDO (PHP Data Objects) is a database abstraction layer for PHP.
- The PDO class represents a connection between PHP and a database server.
- PDO has the following benefits:
  - **Flexibility:** PDO works with almost any database (MySQL, PostgreSQL, etc.) as long as we install the PDO driver for the chosen database.
    - Each database driver that implements the PDO interface can expose database-specific features.
  - **Consistency:** PDO provides a consistent interface in an object-oriented way to data access.
  - **Security:** PDO supports prepared statements, which provide protection against SQL injection attacks as well as better SQL execution performance.
  - **Error handling:** Improved error reporting and exception handling.

Connect to the database

- Connections are established by creating instances of the PDO class.
- The constructor accepts parameters for specifying the database source (known as the dsn) and optionally for the username and password.

```php
function db_connection($host, $port, $dbname, $username, $password)
{
    $dsn = "mysql:host=$host;port=$port;dbname=$dbname";
    try {
        // Create a PDO connection object and connect to the database
        $conn = new PDO($dsn, $username, $password);
        // throw exceptions whenever a database error occurs.
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        echo "Connected successfully";
        return $conn;
    } catch (PDOException $e) {
        echo "Connection failed: " . $e->getMessage();
    }
}
```

Insert Data

```php
<?php
function insert_todo($todo_task, $todo_date)
{
    $sql_query = "INSERT INTO tasks (task, date_added, done) VALUES(:task_value, :date_value, DEFAULT)";
    $todo_date = date("Y-m-d H:i:s", strtotime($todo_date));
    $stmt = $GLOBALS["conn"]->prepare($sql_query);
    if ($stmt) {
        // bind parameters in the prepared statement to the values
        $stmt->bindParam("task_value", $todo_task, PDO::PARAM_STR);
        $stmt->bindParam("date_value", $todo_date, PDO::PARAM_STR);

        // execute the prepared statement
        $stmt->execute();
        // check if it was successful (the affected rows should be 1)
        if ($stmt->rowCount() == 1) {
            echo "New record created successfully";
        } else {
            echo "Error: failed to insert the new record";
        }
    }
}
?>
```

Select Data

```php
function select_all_todos()
{
    $sql_query = "SELECT * FROM tasks";
    $stmt = $GLOBALS["conn"]->prepare($sql_query);
    if ($stmt) {
        // execute the prepared statement
        $stmt->execute();
        // fetch the result
        $result = $stmt->fetchAll(PDO::FETCH_ASSOC);
        // return the result
        return $result;
    }
}
```

Update Data

```php
function update_todo($todo_id, $todo_description, $todo_date, $todo_done){
    $sql_query = "UPDATE tasks SET task = :task_value, date_added = :date_value, done = :done_value WHERE id = :id_val
    $todo_date = date("Y-m-d H:i:s", strtotime($todo_date));
    $stmt = $GLOBALS["conn"]->prepare($sql_query);
    if ($stmt) {
        $stmt->bindParam("id_value", $todo_id, PDO::PARAM_INT);
        $stmt->bindParam("task_value", $todo_description, PDO::PARAM_STR);
        $stmt->bindParam("date_value", $todo_date, PDO::PARAM_STR);
        $stmt->bindParam("done_value", $todo_done, PDO::PARAM_INT);
        $stmt->execute();
        // check if the record was updated
        if ($stmt->rowCount() == 1) {
            echo "Record updated successfully";
        } else {
            echo "Error: failed to update the record";
        }
    }
}
```

Delete Data

```php

function delete_todo($todo_id)
{
    $sql_query = "DELETE FROM tasks WHERE id = :id_value";
    $stmt = $GLOBALS["conn"]->prepare($sql_query);
    if ($stmt) {
        $stmt->bindParam("id_value", $todo_id, PDO::PARAM_INT);
        $stmt->execute();
        // check if the record was deleted
        if ($stmt->rowCount() == 1) {
            echo "Record deleted successfully";
        } else {
            echo "Error: failed to delete the record";
        }
    }
}
```

Demo: Creating a To Do List Web App

https://gitlab.com/kalharbi/todo-php-mysql

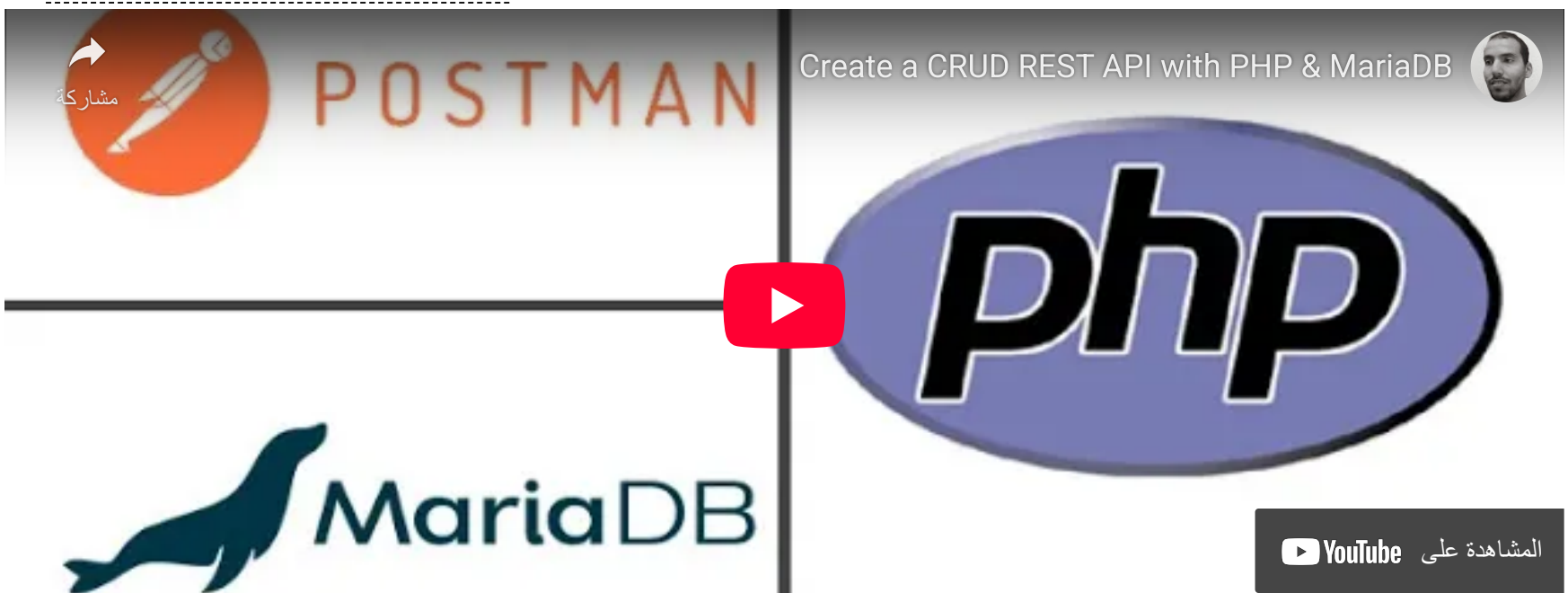
Embedding PHP in HTML is bad!!

- It breaks the fundamental concept of “separation of concerns”.
- Server logic should not be mixed with presentation code (HTML)
- This example is only for demonstration purposes.
- What we should do instead is have our PHP server code communicates with the database and return JSON to the client.
- We will see this right after this demo.

Working with APIs: Create a CRUD REST API

Create a CRUD REST API with PHP & MariaDB

- Below is a video tutorial on how to create a RESTful API in PHP for a simple ToDo application.
- Link to the complete source code



Wrapping up

- We have learned about the following topics:
  - Server-side scripting
  - PHP Syntax and data types
  - Importing PHP script files
  - Form handling and validation
  - Working with databases
  - Creating a CRUD web app
  - Creating a CRUD RESTful API