**Jasme, Jasmin Grace L. BSIT 3-2A**

**PHP Forms**

**GET/POST**

The GET method appends form-data into the URL in name/value pairs. The length of a URL is limited to about 3000 characters, and it has limitations on the kinds of data it can send. It's often used for submitting non-sensitive data, and the resulting URL with the data can be bookmarked. However, because the data is visible in the URL, it's not a secure way of sending sensitive data like passwords.

The POST method, on the other hand, sends form-data as HTTP message body data. It does not have size limitations, and it can send any type of data. This method is a secure way of sending sensitive data. However, the resulting URL with the data can't be bookmarked as the data is not visible in the URL.

In PHP, you can use the superregionals $\_GET and $\_POST to collect the form-data sent with both methods. For example, if you have an HTML form field with the name "username", you can access its value in PHP with $\_GET['username'] (if the form was submitted with the GET method) or $\_POST['username'] (if the form was submitted with the POST method).

**Form Handling**

Form handling in PHP involves getting the data from the form, securing it, and then doing something with it, like storing it in a database or sending an email.

Form handling in PHP involves several steps:

* **Form Creation -** Create an HTML form with fields for user input. The form typically uses the POST method to send data.
* **Data Retrieval -** In your PHP script, use the $\_POST super global to retrieve the data submitted in the form.
* **Data Security -** Secure the retrieved data by trimming unnecessary whitespace, removing slashes, and converting special characters to their HTML entities.
* **Data Usage -** Use the secured data as needed, such as storing it in a database or sending an email.

**Form Validation -** An essential part of web development. It ensures that the data entered by the user is correct and safe to be processed.

**Form validation in PHP involves several steps:**

* Required Fields: Ensure no required fields are left empty by the user.
* Data Type Check: Verify the data is in the correct format. For instance, an email field should contain a valid email address.
* Data Length: Check the length of the data to ensure it's not too short or too long. For example, a username might need to be within a certain character range.
* Sanitizing Input: Even valid data needs to be safe. Always sanitize input by removing unnecessary characters and converting special characters to prevent security issues.
* Displaying Error Messages: If there are validation errors, these need to be displayed to the user.

**Form required -** In HTML, you can make form fields required by using the required attribute. This attribute ensures that the user must fill in the field before submitting the form.

**Example:**

html

<form action="submit\_form.php" method="post">

<label for="name">Name:</label>

<input type="text" id="name" name="name" required><br>

<label for="email">Email:</label>

<input type="email" id="email" name="email" required><br>

<input type="submit" value="Submit">

</form>

In the above example, the required attribute is added to the <input> elements for the "Name" and "Email" fields. This ensures that the user must fill in these fields before submitting the form.

When the form is submitted, the browser will automatically validate the required fields and prevent the form from being submitted if any of them are empty. If a required field is left empty, the browser will display an error message to the user.

**Form URL/Email**

To validate URLs and email addresses in PHP forms, you can use server-side validation. Here's a summary of the steps:

* **URL Validation:** Use the filter\_var() function with the FILTER\_VALIDATE\_URL filter to check if the URL is in a valid format. Store an error message if the URL is invalid.
* **Email Validation:** Use the filter\_var() function with the FILTER\_VALIDATE\_EMAIL filter to check if the email address is in a valid format. Store an error message if the email address is invalid.

Performing server-side validation is important to ensure data integrity and security. Remember to validate both on the client-side (using HTML input types) and server-side to provide a better user experience and prevent malicious data submission.

**Form Complete**

A "form complete" in PHP refers to the implementation of a fully functional form that allows users to input data and submit it to a server for processing.

**To create a complete form in PHP, follow these steps:**

* Define variables for form data and set them to empty values.
* Check if the form is submitted using $\_SERVER["REQUEST\_METHOD"].
* Retrieve and sanitize form data using the test\_input() function.
* Perform form validation by checking if required fields are filled in and if data is in the correct format.
* If there are no errors, process the form data (e.g., stored in a database) and redirect it to a success page.
* Display error messages if there are validation errors.
* Create an HTML form with appropriate input fields and include PHP code to populate form fields with data and handle form submission.
* Use the htmlspecialchars() function to prevent cross-site scripting (XSS) attacks when displaying form data.

**PHP include & require.**

The include statement is used to include and execute the contents of another PHP file within the current PHP script. It allows you to modularize your code by separating it into multiple files and reusing common code across different pages.

**The include statement has two main forms:**

* **include:** This form includes the specified file and continues the execution of the script even if the file is not found or there is an error during inclusion. If the file cannot be included, a warning is issued, but the script execution continues.
* **require:** This form is similar to include, but if the specified file cannot be included, a fatal error is issued, and the script execution is halted.

**PHP Cookies and Session**

Cookies and sessions are used to manage and store data related to user sessions on a website. They serve different purposes and have different characteristics:

**PHP Cookies:**

* Cookies are small pieces of data that are stored on the user's browser.
* Cookies are typically used to store user-specific information, such as login credentials, preferences, or shopping cart items.
* Cookies are sent to the server with each request, allowing the server to identify and personalize the user's experience.
* Cookies can have an expiration time, after which they are automatically deleted from the user's browser.
* Cookies can be set using the setcookie() function in PHP and accessed using the $\_COOKIE superglobal variable.

**PHP Sessions:**

* + Sessions are a way to store and manage user-specific data on the server.
  + When a user visits a website, a unique session ID is generated and stored as a cookie on the user's browser.
  + The session ID is used to associate the user's requests with their specific session data on the server.
  + Session data is stored on the server and can hold more complex information compared to cookies.
  + Sessions are typically used to store sensitive data, such as user authentication status or user-specific data.
  + Session data remains on the server and is not sent back and forth with each request.
  + Session data can be accessed and manipulated using the $\_SESSION superglobal variable in PHP.
  + Sessions are automatically destroyed after a certain period of inactivity or when the user closes their browser.

**PHP File**

**1. Opening a File:**

To open a file in PHP, you can use the `fopen()` function. It takes two parameters: the path to the file and the mode in which the file should be opened (e.g., read, write, append, etc.).

**Example:** `$file = fopen("example.txt", "r");`

**2. Reading a File:**

To read the contents of a file, you can use functions like `fread()`, `fgets()`, or `file\_get\_contents()`.

**Example:** using `file\_get\_contents()`: `$content = file\_get\_contents("example.txt");`

**3. Writing to a File:**

To write data to a file, you can use functions like `fwrite()` or `file\_put\_contents()`.

**Example**: using `file\_put\_contents()`: `file\_put\_contents("example.txt", "Hello, world!");`

**4. Appending to a File:**

To append data to an existing file, you can use functions like `fwrite()` with the "a" mode or `file\_put\_contents()` with the `FILE\_APPEND` flag.

**Example:** using `file\_put\_contents()` with `FILE\_APPEND`: `file\_put\_contents("example.txt", "New content", FILE\_APPEND);`

**5. Deleting a File:**

To delete a file, you can use the `unlink()` function.

**Example:** `unlink("example.txt");`

**6. File Upload:**

* File upload in PHP involves handling file uploads from HTML forms.
* The uploaded file can be accessed using the `$\_FILES` superglobal, which provides information about the uploaded file, such as its name, type, size, and temporary location.
* You can move the uploaded file to a desired location using the `move\_uploaded\_file()` function.

**Example:**

```php

$file = $\_FILES['file'];

$targetDir = "uploads/";

$targetFile = $targetDir . $file['name'];

move\_uploaded\_file($file['tmp\_name'], $targetFile);

```

**PHP MVC Architecture**

The PHP Model-View-Controller (MVC) architecture is a design pattern that separates the concerns of an application into three main components: Model, View, and Controller.

**Model:**

* The Model represents the data and business logic of the application.
* It encapsulates the data structures, database interactions, and business rules.
* The Model is responsible for retrieving and manipulating data, performing validation, and implementing the application's core logic.
* It interacts with the database or other data sources to fetch and update data.

**View:**

* The View is responsible for presenting the data to the user.
* It generates the user interface and displays the information provided by the Model.
* Views are typically implemented using HTML templates mixed with PHP code to render dynamic content.
* Views should not contain any business logic and should focus solely on displaying data.

**Controller:**

* The Controller acts as an intermediary between the Model and the View.
* It handles user input, processes requests, and coordinates the interaction between the Model and the View.
* Controllers receive user requests, invoke the appropriate actions in the Model, and determine which View should be displayed.
* They handle form submissions, user authentication, and other application-specific logic.

The PHP MVC architecture promotes separation of concerns, making the code more modular, maintainable, and reusable. It allows for better organization and separation of business logic from presentation logic. With MVC, changes in one component have minimal impact on the other components, making it easier to modify or extend the application.

**Top 10 PHP Architecture**

1. **Laravel -** A robust and expressive framework that emphasizes elegance and simplicity. It provides features like routing, ORM, caching, and authentication.
2. **Symfony -** A flexible and scalable framework that follows the MVC pattern. It offers a wide range of reusable components and tools for building complex web applications.
3. **CodeIgniter** - A lightweight and straightforward framework known for its speed and simplicity. It has a small footprint and requires minimal configuration.
4. **Yii -** A high-performance framework that promotes rapid development. It includes features like caching, security, and scaffolding for generating code automatically.
5. **CakePHP -** A powerful framework with a convention-over-configuration approach. It provides a robust set of features and follows the MVC pattern.
6. **Zend Framework -** A highly customizable framework that offers a collection of professional-grade PHP packages. It provides a modular architecture and follows best practices.
7. **Phalcon -** A full-stack framework known for its speed and low resource consumption. It is implemented as a C extension, resulting in exceptional performance.
8. **Slim -** A micro-framework designed for building simple yet powerful web applications and APIs. It has a minimalistic approach and is ideal for small-scale projects.
9. **FuelPHP -** A flexible and modular framework that focuses on security and performance. It provides features like ORM, caching, and authentication.
10. **PhPixie -** A lightweight and high-performance framework that follows the HMVC (Hierarchical Model-View-Controller) architectural pattern. It emphasizes code organization and reusability.