FULLSTACK COHORT 3

PHP Fundamentals

JOAQUIN ANTONIO

Introduction to Web Development & PHP

What is web development?

1. Web development refers to the creation of websites and web applications that are accessed over the internet.

2. It involves various aspects, including web design, front-end development, back-end development, and server management.

3. Web developers use programming languages such as HTML, CSS, and JavaScript for front-end development, and languages like PHP, Python, and Ruby for back-end development.

4. They also work with databases like MySQL, PostgreSQL, and MongoDB to store and manage data for dynamic websites and applications.

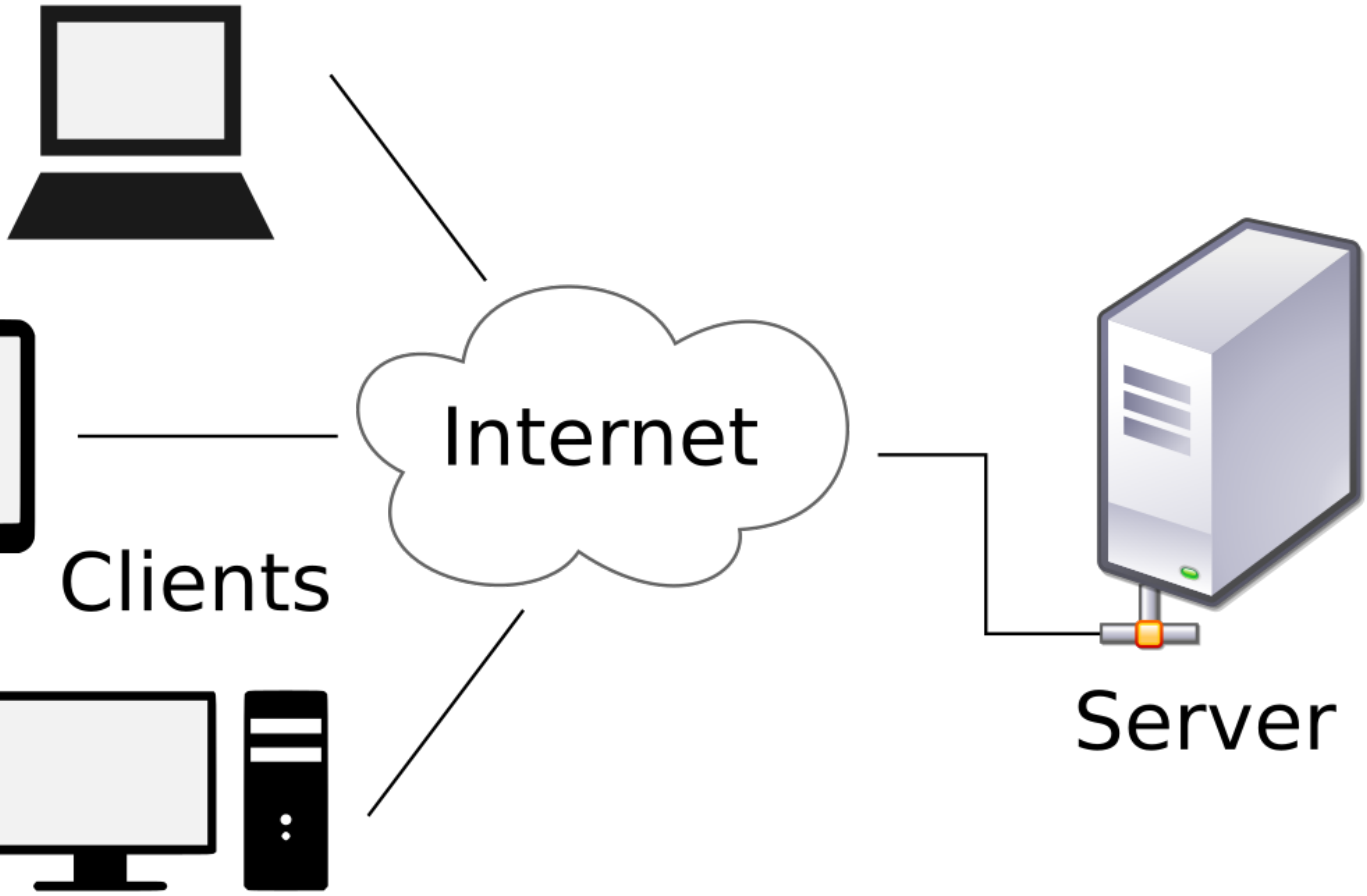
5. Web development encompasses a range of tasks, from designing the layout and appearance of a website to implementing functionality and ensuring its security and performance.

Understanding the server-client model

1. **Client** - The client is a computer or device (e.g., web browser) that requests services or resources from a server. It is typically used in web development to access websites or web applications.

2. **Server** - The server is a computer or device that provides services or resources to clients over a network. In web development, it hosts websites, web applications, or web services and responds to client requests.

3. **Request** - Response Cycle: When a client wants to access a web page or resource, it sends a request to the server. The server processes the request and sends back a response containing the requested resource. This cycle forms the basis of communication between clients and servers in web development.


Internet
Clients
Server

Introduction to server-side scripting and PHP

1. Server-side scripting is a technique used in web development to generate dynamic web pages or manage server-side processes.

2. Unlike client-side scripting, which runs in the user's browser, server-side scripting runs on the web server and generates HTML, CSS, and JavaScript code that is sent to the client's browser.

3. PHP (Hypertext Preprocessor) is a popular server-side scripting language used for web development.

4. It is especially well-suited for creating dynamic web pages and interacting with databases.

Basic PHP syntax

1. PHP code is embedded within HTML pages and is enclosed in special delimiters, usually `<?php` and `?>`.
2. For example, to display "Hello, World!" in PHP, you would use:

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

PHP Basics

Basics of PHP syntax: Comments

```
// This is a single-line comment

/*

This is a

multi-line comment

*/
```

Adding External php to your files

1. `include`: includes a file and continues executing the script if the file is not found or an error occurs. It generates a warning but does not halt the script execution.

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

2. `require`: also includes a file but halts the script execution and generates a fatal error if the file is not found or an error occurs. It is more strict than include.

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

Basics of PHP syntax: Keywords

1. Keywords are reserved words in PHP that have special meaning and cannot be used as identifiers (such as variable names or function names).

2. Some examples of keywords are:

3. `if, else, while, for, function, class, namespace, use, echo, return, try, catch, finally, include, require, include_once, require_once, global, static, new, abstract, final, extends, implements, interface, public, private, protected, const, self, parent, true, false, null`

Basics of PHP syntax: Identifiers

1. dentifiers in PHP are names used for variables, functions, classes, etc.

2. They must start with a letter or underscore, followed by letters, numbers, or underscores. Identifiers are case-sensitive.

```
1. $variableName = "value"; // variable identifier

2. function functionName() {} // function identifier

3. class ClassName {} // class identifier
```

Data types: Integers

1. Whole numbers without a decimal point, such as -5, 0, 42.
2. `$integerVar = 42;`

Data types: Floats

1. Floats (or doubles): Numbers with a decimal point, such as 3.14, -0.001, 2.0.
2. `$floatVar = 3.14;`

Data types: Strings

1. Strings: Sequence of characters, such as "Hello, World!", 'PHP is fun’.
2. `$stringVar = "Hello, World!";`

Data types: Booleans

1. Booleans: Represents true or false.
2. `$boolVar = true;`

Data types: Arrays

1. Arrays: Ordered map that holds key-value pairs.

2. It's worth noting that PHP is loosely typed, meaning you don't need to explicitly declare the data type of a variable. PHP determines the data type based on the context in which the variable is used.

3. `$arrayVar = array("apple", "banana", "cherry");`

Variables: Declaring, assigning, and manipulating variables

1. In PHP, you can declare variables using the dollar sign ($) followed by the variable name.
2. Variable names must start with a letter or underscore, followed by letters, numbers, or underscores.
3. Variables in PHP are dynamic, meaning their data type can change based on the value assigned to them.

```
1.$name = "John";
2.$age = 30;
3.$height = 5.11;
4.$isStudent = true;
```

Variables: Declaring, assigning, and manipulating variables

1. You can assign values to variables using the assignment operator (=). Variables can hold different types of values, such as strings, integers, floats, booleans, and arrays.
2. To manipulate variables, you can perform operations based on their data types. For example, with integers and floats, you can perform arithmetic operations

```
$a = 10;
$b = 20;
$sum = $a + $b; // Sum of $a and $b
$diff = $a - $b; // Difference between $a and $b
$product = $a * $b; // Product of $a and $b
$quotient = $a / $b; // Quotient of $a divided by $b
```

Variables: Declaring, assigning, and manipulating variables

1. With strings, you can concatenate them using the dot (.) operator:

```php
$firstName = "John";
$lastName = "Doe";
$fullName = $firstName . " " . $lastName; // Concatenate first name and last name
```

Variables: Declaring, assigning, and manipulating variables

1. With booleans, you can use logical operators such as && (and), || (or), and ! (not) to manipulate their values:

```
$hasPermission = true;
$isAuthenticated = false;
$canAccessResource = $hasPermission && $isAuthenticated; // Check if both conditions are true
```

Variables: Declaring, assigning, and manipulating variables

1. Arrays can store multiple values in a single variable. You can manipulate arrays by adding, removing, or modifying elements:

```php
$fruits = array("apple", "banana", "cherry");
// Add an element to the array
$fruits[] = "orange";
// Remove an element from the array
unset($fruits[1]);
// Modify an element in the array
$fruits[0] = "pear";
```

Operators: Arithmetic

1. Arithmetic Operators: Used to perform arithmetic operations like addition, subtraction, multiplication, division, and modulus.

```
$a = 10;
$b = 5;
$sum = $a + $b; // Addition: $sum is 15
$diff = $a - $b; // Subtraction: $diff is 5
$product = $a * $b; // Multiplication: $product is 50
$quotient = $a / $b; // Division: $quotient is 2
$remainder = $a % $b; // Modulus: $remainder is 0
```

Operators: Comparison

1. Comparison Operators: Used to compare values and return true or false.

```
$a = 10;
$b = 5;
$result1 = ($a == $b); // Equal to. Result: false
$result2 = ($a != $b); // Not equal to. Result: true
$result3 = ($a > $b); //Greater than. Result: true
$result4 = ($a < $b); // Less than. Result: false
$result5 = ($a >= $b); // Greater than or equal to. Result: true
$result6 = ($a <= $b); // Less than or equal to. Result: false
```

Operators: Logical

1. Logical Operators: Used to combine conditional statements.

```php
$hasPermission = true;
$isAuthenticated = false;
// AND operator
$canAccessResource = $hasPermission && $isAuthenticated; // false
// OR operator
$canAccessResource = $hasPermission || $isAuthenticated; // true
// NOT operator
$canAccessResource = !$isAuthenticated; // true
```

Operators: Assignment operators

1. Assignment Operators: Used to assign values to variables.

```
1.$a = 10;
2.$b = 5;
3.$c = $a + $b; // Simple assignment: $c is 15
4.$a += $b; // Add and assign: $a is now 15
5.$a -= $b; // Subtract and assign: $a is now 10
6.$a *= $b; // Multiply and assign: $a is now 50
7.$a /= $b; // Divide and assign: $a is now 10
8.$a %= $b; // Modulus and assign: $a is now 0
```

EXERCISE

Create a simple PHP script that displays your name, a greeting message, and the current date using variables and string manipulation

Control Flow Statements

Conditional statements: if, else, elseif

1. Conditional statements in PHP allow you to execute different blocks of code based on different conditions.

```php
<?php
$temperature = 20;
if ($temperature < 0) {
    echo "It's freezing!";
} elseif ($temperature >= 0 && $temperature < 15) {
    echo "It's cold.";
} elseif ($temperature >= 15 && $temperature < 25) {
    echo "It's nice outside.";
} else {
    echo "It's warm!";
}
?>
```

Looping statements: for

1. Executes a block of code a specified number of times.

```php
<?php
for ($i = 0; $i < 5; $i++) {
    echo "The number is: $i <br>";
}
?>
```

Looping statements: while

1. Executes a block of code as long as a specified condition is true.

```php
<?php
$i = 0;
while ($i < 5) {
    echo "The number is: $i <br>";
    $i++;
}
?>
```

Looping statements: do-while

1. Similar to a while loop, but the block of code is executed at least once, even if the condition is false.

```php
<?php
$i = 0;
do {
    echo "The number is: $i <br>";
    $i++;
} while ($i < 5);
?>
```

Switch statements for making multi-way decisions

1. Switch statements in PHP allow you to make multi-way decisions based on the value of an expression.

```
switch ($day) {
    case "Monday":
        echo "Today is Monday";
        break;
    case "Tuesday":
        echo "Today is Tuesday";
        break;
    default:
        echo "Not a valid day";
```

EXERCISE

1. Write a PHP script that asks the user for their age and displays a message based on their age category (e.g., child, teenager, adult).

2. Use a combination of control flow statements and user input.

Functions

Defining and calling functions for code reusability

1. In PHP, you can define functions to encapsulate a block of code for reuse.
2. In this example, the greet() function simply outputs "Hello, World!". You can call this function anywhere in your code to reuse this behaviour.

```php
<?php
// Define a function
function greet() {
    echo "Hello, World!";
}
// Call the function
greet();
?>
```

Defining and calling functions for code reusability

1. You can also define functions with parameters:

```php
<?php
// Define a function with parameters
function greetByName($name) {
    echo "Hello, $name!";
}

// Call the function with an argument
greetByName("Alice"); // Outputs "Hello, Alice!"
?>
```

Defining and calling functions for code reusability

1. Functions can also return values:

```php
<?php
// Define a function that returns a value
function add($a, $b) {
    return $a + $b;
}
// Call the function and store the result in a variable
$result = add(3, 5);
echo "The result is: $result"; // Outputs "The result is: 8"
?>
```

Passing arguments to functions

1. In PHP, you can pass arguments to functions and return values from functions. In this example, the add() function takes two arguments ($a and $b) and returns their sum.

```php
<?php
// Define a function that takes two arguments
function add($a, $b) {
    return $a + $b;
}
// Call the function with arguments
$result = add(3, 5);
echo "The sum is: $result"; // Outputs "The sum is: 8"
?>
```

Returning values

1. Returning Values from Functions. In this example, the multiply() function takes two arguments ($a and $b) and returns their product:

```php
<?php
// Define a function that returns a value
function multiply($a, $b) {
    return $a * $b;
}
// Call the function and store the result in a variable
$result = multiply(3, 5);
echo "The product is: $result"; // Outputs "The product is: 15"
?>
```

Void functions

1. You can also have functions with no return value. In this example, the displayMessage() function takes a message as an argument and echoes it to the output without returning a value:

```php
<?php
// Define a function that doesn't return a value
function displayMessage($message) {
    echo $message;
}
// Call the function
displayMessage("Hello, World!"); // Outputs "Hello, World!"
?>
```

Understanding scope and variable visibility

1. In PHP, scope refers to the visibility of variables within different parts of your code.

Function scope

1. Variables declared inside a function are in the function scope and can only be accessed within that function

```
function myFunction() {
    $functionVar = "I'm a function variable";
    echo $functionVar; // Accessing function variable inside the function
}
myFunction(); // Outputs: I'm a function variable
echo $functionVar; // This will cause an error since $functionVar is not defined in this scope
```

Global scope

1. Variables declared outside of any function are in the global scope and can be accessed from anywhere in the script.

```php
$globalVar = "I'm a global variable";
function myFunction() {
    echo $globalVar; // Accessing global variable inside a function
}
myFunction(); // Outputs: I'm a global variable
```

Static scope

1. Variables declared as static inside a function retain their value between function calls and are only accessible within that function.

```php
function increment() {
    static $count = 0;
    $count++;
    echo $count;
}
increment(); // Outputs: 1
increment(); // Outputs: 2
increment(); // Outputs: 3
```

Class scope

1. Variables declared within a class but outside of any method (or declared using the static keyword inside a method) are in the class scope and can be accessed using the scope resolution operator ::

```php
class MyClass {
    public static $classVar = "I'm a class variable";
    public static function myMethod() {
        echo self::$classVar; // Accessing class variable inside a method
    }
}
echo MyClass::$classVar; // Outputs: I'm a class variable
MyClass::myMethod(); // Outputs: I'm a class variable
```

EXERCISE

1. Create a function that calculates the area of a rectangle based on its width and height provided as arguments.
2. Use the function in a script to calculate the area of two different rectangles.

Forms and User Input

Understanding HTML forms and POST method for submitting data

1. HTML forms are used to collect user input on a web page.
2. They consist of various form elements like text fields, checkboxes, radio buttons, and submit buttons.
3. The POST method is one of the HTTP methods used to submit form data to a web server.

Accessing form data using the $_POST superglobal

1. To access form data submitted using the POST method in PHP, you can use the $_POST superglobal.

Validating user input to prevent errors and security vulnerabilities

1. Validating user input is crucial to prevent errors and security vulnerabilities in your application. Here are some common validation techniques in PHP:

Required Fields: Ensure that required fields are not empty

```
if (empty($_POST['name'])) {
    $errors[] = 'Name is required.';
}
```

String Length: Limit the length of input strings

```
if (strlen($_POST['username']) > 20) {
    $errors[] = 'Username must be less than 20 characters.';
}
```

Numeric Values: Ensure numeric values are within a certain range

```
if ($_POST['age'] < 18 || $_POST['age'] > 65) {
    $errors[] = 'Age must be between 18 and 65.';
}
```

Email Validation: Validate email addresses

```
if (!filter_var($_POST['email'], FILTER_VALIDATE_EMAIL)) {
    $errors[] = 'Invalid email address.';
}
```

Password Strength: Enforce strong passwords

```
if (strlen($_POST['password']) < 8) {
    $errors[] = 'Password must be at least 8 characters long.';
}
```

File Uploads: Validate file uploads

```
if ($_FILES['file']['size'] > 1000000) {
    $errors[] = 'File size must be less than 1MB.';
}
```

SQL Injection Prevention

1. Use prepared statements or parameterized queries to prevent SQL injection attacks.

```
$stmt = $pdo->prepare('SELECT * FROM users WHERE username = ?');
$stmt->execute([$_POST['username']]);
$user = $stmt->fetch();
```

EXERCISE

1. Create a simple HTML form that asks for the user's name and email.
2. Write a PHP script that processes the form submission, validates the data (e.g., email format), and displays a confirmation message with the user's information.