-:Introduction to Php:-

PHP (Hypertext Preprocessor) is a popular open-source server-side scripting language used primarily for web development. It allows developers to create dynamic and interactive web pages. PHP is especially known for its ability to integrate seamlessly with databases, particularly MySQL, and is widely used in content management systems (CMS) like WordPress.

Key Features of PHP:-

1. Server-Side Scripting:

PHP runs on the web server, and the output is sent to the user's browser as HTML.

2. Dynamic Content:

PHP is capable of generating dynamic page content, such as data from databases, user interactions, and more.

3. Database Integration:

PHP has strong support for interacting with databases, allowing for the storage, retrieval, and manipulation of data. It works well with MySQL, PostgreSQL, and others.

4. Cross-Platform:

PHP is platform-independent and works on various operating systems, including Windows, Linux, and macOS.

5. Open Source:

PHP is free to use, with a large and active community of developers.

6. Embedded into HTML:

PHP code can be embedded directly into HTML, making it easy to develop web pages and applications without a need for complex setups.

7. Extensive Library Support:

PHP provides a wide range of built-in functions and libraries for performing tasks like string manipulation, file handling, session management, and more.

How PHP Works:

1. Client Request:

A user sends a request via a web browser (such as accessing a URL).

2. Server Processing:

The request is sent to the web server where the PHP script is located.

3. PHP Execution:

The server processes the PHP code and executes it. Any dynamic content, such as fetching data from a database, is handled at this stage.

4. Generate Output:

The result of the PHP script (often HTML) is sent back to the client's browser.

5. Display:

The browser displays the HTML result.

Basic PHP Syntax:

```
?php
// This is a single-line comment
echo "Hello, World!"; // Outputs Hello, World! to the browser
?>
```

-: OOPs With PhP :-

Object-Oriented Programming (OOP) in PHP is a programming paradigm that organizes software design around objects rather than functions and logic. Objects can contain data in the form of fields (often known as properties) and code in the form of methods. OOP allows for the creation of reusable and modular code.

1. Classes and Objects

Class:-

A blueprint for creating objects. A class defines properties and methods that objects of that class will have.

Object:-

An instance of a class.

Example:-

```
// Properties
public $make;
public $model;
public $color;
// Constructor
public function __construct($make, $model, $color) {
$this->make = $make;
$this->model = $model;
$this->color = $color;
}
// Method
public function displayInfo() {
```

```
echo "This is a $this->color $this->make $this->model.";
}
}
// Creating an object
$car1 = new Car("Toyota", "Corolla", "blue");
$car1->displayInfo(); // Output: This is a blue Toyota Corolla.
?>
```

2. Encapsulation (Private and Getter/Setter Methods)

Encapsulation involves hiding the internal state of an object and providing methods to access and modify it. Here's how we can encapsulate properties using private and provide getter and setter methods.

Example:-

```
class BankAccount {
private $balance;
// Constructor
public function __construct($balance) {
$this->balance = $balance;
}
// Getter method
public function getBalance() {
return $this->balance;
}
// Setter method
public function deposit($amount) {
if ($amount > 0) {
$this->balance += $amount;
}
}
// Method to withdraw money
public function withdraw($amount) {
if ($amount > 0 && $this->balance >= $amount) {
$this->balance -= $amount;
}
}
}
// Creating an object and using getter/setter methods
$account = new BankAccount(100);
$account->deposit(50);
echo "Balance: " . $account->getBalance() . "
"; // Output: Balance: 150
$account->withdraw(30);
```

```
echo "Balance after withdrawal: " . $account->getBalance(); // Output: Balance after withdrawal: 120
?>
```

3. Inheritance

Inheritance allows a class to inherit methods and properties from another class. A subclass can override methods from a parent class to customize or extend functionality.

Example:-

```
class Animal {
public $name;
// Constructor
public function __construct($name) {
$this->name = $name;
}
// Method
public function speak() {
echo "This animal makes a sound.
";
}
}
class Dog extends Animal {
// Overriding the speak method
public function speak() {
echo "$this->name says Woof!
";
}
}
$dog = new Dog("Buddy");
$dog->speak(); // Output: Buddy says Woof!
?>
```

4. Polymorphism

Polymorphism allows objects of different classes to be treated as objects of a common base class. It is especially useful when dealing with objects of different types in a uniform way.

Example:-

```
class Animal {
public function speak() {
echo "This animal makes a sound.
```

```
";
}
}
class Dog extends Animal {
public function speak() {
echo "Woof!
";
}
}
class Cat extends Animal {
public function speak() {
echo "Meow!
";
}
}
// Polymorphism: Using the same method name for different classes
$animals = [new Dog(), new Cat()];
foreach ($animals as $animal) {
$animal->speak(); // Output: Woof! Meow!
}
?>
```

5. Abstract Classes

Abstract classes cannot be instantiated directly. They can have abstract methods (without implementation) that must be implemented in subclasses.

Example:-

```
abstract class Animal {
// Abstract method
abstract public function sound();
// Regular method
public function sleep() {
echo "This animal is sleeping.
";
}
}
class Dog extends Animal {
public function sound() {
echo "Woof!
";
}
}
class Cat extends Animal {
```

```
public function sound() {
echo "Meow!
";
}
}
$dog = new Dog();
$dog->sound(); // Output: Woof!
$dog->sleep(); // Output: This animal is sleeping.
$cat = new Cat();
$cat->sound(); // Output: Meow!
?>
```

6. Interfaces

Interfaces define a contract that classes must follow. A class that implements an interface must implement all methods declared in the interface.

Example:-

```
interface AnimalInterface {
public function sound();
}
class Dog implements AnimalInterface {
public function sound() {
echo "Woof!
";
}
}
class Cat implements AnimalInterface {
public function sound() {
echo "Meow!
";
}
}
$dog = new Dog();
$dog->sound(); // Output: Woof!
$cat = new Cat();
$cat->sound(); // Output: Meow!
?>
```

7. Static Methods and Properties

Static properties and methods are associated with the class rather than with instances of the class. They are accessed using the class name rather than the object.

Example:-

```
class Calculator {
public static $pi = 3.14159;
public static function areaOfCircle($radius) {
return self::$pi * $radius * $radius;
}
}
echo Calculator::$pi; // Output: 3.14159
echo "
";
echo Calculator::areaOfCircle(5); // Output: 78.53975
?>
```

1. Variables in PHP

A variable in PHP is a container used to store data, such as numbers, strings, or arrays. Variables are dynamically typed in PHP, meaning you don't need to declare the data type when you assign a value to a variable.

Syntax for Declaring Variables:-

```
$variable_name = value;
```

- **Variables in PHP start with a dollar sign ($).**
- **PHP variables are case-sensitive (i.e., $Var and $var are considered different).**

Example of Variables:-

```
// Integer variable
$age = 25;
// String variable
$name = "John";
// Float variable
$height = 5.9;
// Boolean variable
$isStudent = true;
// Array variable
$colors = array("Red", "Green", "Blue");
// Associative array variable
$person = array("name" => "John", "age" => 25);
echo "Name: $name
";
echo "Age: $age
```

```
";
echo "Height: $height meters
";
echo "Is Student: " . ($isStudent ? "Yes" : "No") . "
";
echo "Colors: " . implode(", ", $colors) . "
";
// Accessing array values
echo "Person's Name: " . $person['name'] . "
";
?>
```

Variables Scope

PHP variables can have different scopes, meaning they are accessible in different parts of a script. The common types of variable scope are:

Local Scope:

Variables declared inside a function are local to that function.

Global Scope:

Variables declared outside any function are globally accessible.

Static Variables:

Variables that maintain their state across function calls.

Example: Variable Scope:-

```
// Global variable
$globalVar = "I am global";
function myFunction() {
// Local variable
$localVar = "I am local";
global $globalVar; // Accessing global variable inside the function
echo $globalVar . "
";
// Local variable inside function
echo $localVar . "
";
}
myFunction();
// Uncommenting the line below will give an error because $localVar is local to the function
// echo $localVar; // Error: Undefined variable
?>
```

3. Constants in PHP

A constant is an identifier (name) for a simple value. Unlike variables, constants cannot be changed or redefined after they are declared.

Defining Constants in PHP

PHP provides two ways to define constants:

define() function:

It is used to define a constant.

const keyword:

This can be used to define constants, but it is only available in classes or at the global level.

Syntax for Defining Constants:-

```
define("CONSTANT_NAME", value);
```

Example: Constants:-

```
// Defining a constant using define()
define("SITE_NAME", "My Awesome Website");
define("MAX_USERS", 100);
// Defining a constant using const (valid only at global level or within classes)
const VERSION = "1.0.0";
// Displaying constant values
echo "Welcome to " . SITE_NAME . "
";
echo "Max users allowed: " . MAX_USERS . "
";
echo "Version: " . VERSION . "
";
// Constants cannot be changed or redefined
// Uncommenting the following line will cause an error
// define("SITE_NAME", "Another Website"); // Error: Cannot redefine constant
?>
```

Data Types and Operators

In programming, data types specify the kind of data that can be stored in a variable, while operators are symbols or keywords used to perform operations on data.

1. Data Types:

Different programming languages might have different data types, but here are the most common types:

1. Primitive Data Types:

- **Integer (int):**

Represents whole numbers (e.g., 1, -5, 100).

- **Floating Point (float, double):**

Represents decimal numbers (e.g., 3.14, -0.001, 1.5).

- **Character (char):**

Represents a single character (e.g., 'a', 'X', '9').

- **Boolean (bool):**

Represents a true or false value (e.g., true, false).

2. Non-Primitive Data Types (Complex Data Types):

- **String (str):**

Represents a sequence of characters (e.g., "Hello", "123").

- **Array:**

A collection of elements of the same type (e.g., [1, 2, 3], ["apple", "banana"]).

2. Operators:

Operators are symbols or keywords that perform operations on variables and values.

Arithmetic Operators:

Used to perform basic mathematical operations.

- **+ (Addition):**

Adds two operands (e.g., a + b).

- **- (Subtraction):**

Subtracts the second operand from the first (e.g., a - b).

- *** (Multiplication):**

Multiplies two operands (e.g., a * b).

- **/ (Division):**

Divides the first operand by the second (e.g., a / b).

- **% (Modulus):**

Returns the remainder of the division (e.g., a % b).

- **** (Exponentiation):**

Raises the first operand to the power of the second (e.g., a ** b).

Comparison Operators:

Used to compare values.

- **== (Equal to):**

Checks if two operands are equal (e.g., a == b).

- **!= (Not equal to):**

Checks if two operands are not equal (e.g., a != b).

- **> (Greater than):**

Checks if the left operand is greater than the right (e.g., a > b).

- **< (Less than):**

Checks if the left operand is less than the right (e.g., a < b).

- **>= (Greater than or equal to):**

Checks if the left operand is greater than or equal to the right (e.g., a >= b).

- **<= (Less than or equal to):**

Checks if the left operand is less than or equal to the right (e.g., a <= b).

Logical Operators:

Used to combine conditional statements.

- **&& (AND):**

Returns true if both operands are true (e.g., a && b).

- **|| (OR):**

Returns true if at least one operand is true (e.g., a || b).

- **! (NOT):**

Reverses the logical state of its operand (e.g., !a).

Assignment Operators:

Used to assign values to variables.

- **= (Assign):**

Assigns the value of the right operand to the left operand (e.g., a = 5).

- **+= (Add and assign):**

Adds the right operand to the left operand and assigns the result (e.g., a += b).

- **-= (Subtract and assign):**

Subtracts the right operand from the left operand and assigns the result (e.g., a -= b).

- ***= (Multiply and assign):**

Multiplies the left operand by the right operand and assigns the result (e.g., a *= b).

- **/= (Divide and assign):**

Divides the left operand by the right operand and assigns the result (e.g., a /= b).

Increment and Decrement Operators:

Used to increase or decrease a variable's value by 1.

- **++ (Increment):**

Increases the value of a variable by 1 (e.g., a++ or ++a).

- **-- (Decrement):**

Decreases the value of a variable by 1 (e.g., a-- or --a).

Bitwise Operators:

Used to perform bit-level operations.

- **& (AND):**

Performs a bitwise AND operation.

- **| (OR):**

Performs a bitwise OR operation.

- **^ (XOR):**

Performs a bitwise XOR operation.

- **~ (NOT):**

Inverts all the bits.

- **<< (Left Shift):**

Shifts bits to the left.

- **>> (Right Shift):**

Shifts bits to the right.

Ternary Operator:

A shorthand for an if-else statement.

- **condition ? expression1 :**

expression2 (e.g., x > y ? "x is greater" : "y is greater").

Special Operators:

- **typeof:**

Returns the type of a variable (e.g., typeof x).

- **instanceof:**

Tests whether an object is an instance of a specific class or type (e.g., x instanceof MyClass).

Flow Control Statements

In PHP, flow control statements are used to manage the flow of execution based on conditions, loops, and other logic. These are similar to flow control mechanisms in other programming languages like if, else, while, for, break, continue, and return. Below, I'll explain the various flow control statements in PHP with examples.

1. Conditional Statements:

If Statement:

The if statement is used to execute a block of code only if a specified condition is true.

```
$age = 18;
if ($age >= 18) {
echo "You are eligible to vote.";
}
?>
```

If-Else Statement:

The if-else statement allows you to define an alternative block of code to execute when the condition is false.

```
$age = 16;
if ($age >= 18) {
echo "You are eligible to vote.";
} else {
echo "You are not eligible to vote.";
}
?>
```

If-Elif-Else Statement:

The if-elseif-else statement allows checking multiple conditions. If the first condition is false, the program checks the next condition and so on.

```
$score = 75;
if ($score >= 90) {
echo "Grade A";
} elseif ($score >= 75) {
```

```
echo "Grade B";
} elseif ($score >= 50) {
echo "Grade C";
} else {
echo "Grade F";
}
?>
```

2. Looping Statements:

For Loop:

The for loop is used when you know in advance how many times you want to execute a statement or a block of code.

```
for ($i = 0; $i < 5; $i++) {
echo "The value of i is: " . $i . "
";
}
?>
```

While Loop:

The while loop executes as long as the specified condition evaluates to true.

```
$i = 0;
while ($i < 5) {
echo "The value of i is: " . $i . "
";
$i++;
}
?>
```

Do-While Loop:

The do-while loop is similar to the while loop, except that the condition is checked after the code block has been executed, ensuring that the block runs at least once.

```
$i = 0;
do {
echo "The value of i is: " . $i . "
";
$i++;
} while ($i < 5);
?>
```

3. Break and Continue Statements:

Break Statement:

The break statement is used to exit from a loop prematurely, regardless of the loop's condition.

```
for ($i = 0; $i < 10; $i++) {
if ($i == 5) {
break; // Exit the loop when $i equals 5
}
echo "The value of i is: " . $i . "
";
}
?>
```

Continue Statement:

The continue statement skips the current iteration of the loop and moves to the next iteration.

```
for ($i = 0; $i < 5; $i++) {
if ($i == 3) {
continue; // Skip the iteration when $i equals 3
}
echo "The value of i is: " . $i . "
";
}
?>
```

4. Return Statement:

The return statement is used in functions to return a value and exit the function.

```
function add($a, $b) {
return $a + $b; // Return the sum of $a and $b
}
$result = add(5, 3);
echo "The result is: " . $result;
?>
```

5. Switch Statement:

The switch statement is used to perform different actions based on multiple conditions. It's an alternative to using many if-else statements.

```
$day = 2;
switch ($day) {
case 1:
```

```
echo "Monday";
break;
case 2:
echo "Tuesday";
break;
case 3:
echo "Wednesday";
break;
case 4:
echo "Thursday";
break;
case 5:
echo "Friday";
break;
default:
echo "Invalid day";
}
?>
```

In this example, the output will be Tuesday, since $day is equal to 2.

6. Example Combining Flow Control Statements:

```
$score = 85;
$attendance = 75;
// Check if student passed based on both score and attendance
if ($score >= 50 && $attendance >= 80) {
echo "The student passed.";
} elseif ($score >= 50 && $attendance < 80) {
echo "The student passed, but attendance is low.";
} else {
echo "The student failed.";
}
// Loop to print numbers 1 to 10, skip number 5, and break when number 8 is reached
for ($i = 1; $i <= 10; $i++) {
if ($i == 5) {
continue; // Skip printing number 5
}
if ($i == 8) {
break; // Exit loop when $i is 8
}
echo $i . "
";
}
?>
```

Arrays in PHP

An array is a special variable in PHP that can hold multiple values. Rather than declaring multiple variables for each value, you can store all values in a single array variable. PHP supports two types of arrays:

1. Indexed Arrays (Numeric keys)
2. Associative Arrays (Custom keys)
3. Multidimensional Arrays (Arrays containing other arrays)

1. Indexed Arrays:

Indexed arrays are arrays where each element is assigned a numeric index, starting from 0.

Example of Indexed Array:

```
// Declaring an indexed array
$fruits = array("Apple", "Banana", "Orange", "Mango");
// Accessing array elements by index
echo $fruits[0]; // Outputs: Apple
echo "
";
echo $fruits[2]; // Outputs: Orange
?>
```

Modifying Indexed Array:

You can modify elements in an indexed array by referencing the index.

```
$fruits = array("Apple", "Banana", "Orange", "Mango");
$fruits[1] = "Grapes"; // Changing 'Banana' to 'Grapes'
echo $fruits[1]; // Outputs: Grapes
?>
```

Adding Elements to Indexed Array:

You can add elements to an indexed array using [].

```
$fruits = array("Apple", "Banana", "Orange");
$fruits[] = "Mango"; // Adds 'Mango' to the end of the array
echo $fruits[3]; // Outputs: Mango
?>
```

2. Associative Arrays:

Associative arrays are arrays where each element is associated with a custom key (usually a string). Instead of numeric indices, you can use meaningful keys for each element.

Example of Associative Array:

```
// Declaring an associative array
$person = array("name" => "John", "age" => 25, "city" => "New York");
// Accessing elements by key
echo $person["name"]; // Outputs: John
echo "
";
echo $person["age"]; // Outputs: 25
?>
```

Modifying Associative Array:

You can modify elements in an associative array by referencing the key.

```
$person = array("name" => "John", "age" => 25, "city" => "New York");
$person["age"] = 30; // Changing 'age' from 25 to 30
echo $person["age"]; // Outputs: 30
?>
```

Adding Elements to Associative Array:

You can add new key-value pairs to an associative array.

```
$person = array("name" => "John", "age" => 25);
$person["city"] = "New York"; // Adding a new key-value pair
echo $person["city"]; // Outputs: New York
?>
```

3. Multidimensional Arrays:

A multidimensional array is an array that contains one or more arrays. These arrays can represent more complex data structures like tables or matrices.

```
// Declaring a multidimensional array
$contacts = array(
array("name" => "John", "phone" => "1234567890", "email" => "john@example.com"),
array("name" => "Jane", "phone" => "9876543210", "email" => "jane@example.com"),
array("name" => "Tom", "phone" => "5555555555", "email" => "tom@example.com")
);
// Accessing elements of a multidimensional array
echo $contacts[0]["name"]; // Outputs: John
echo "
";
echo $contacts[1]["phone"]; // Outputs: 9876543210
?>
```

 Modifying Multidimensional Array:

You can modify elements in a multidimensional array by specifying the row and column.

```
$contacts = array(
array("name" => "John", "phone" => "1234567890", "email" => "john@example.com"),
array("name" => "Jane", "phone" => "9876543210", "email" => "jane@example.com")
);
$contacts[0]["email"] = "john_new@example.com"; // Changing John's email
echo $contacts[0]["email"]; // Outputs: john_new@example.com
?>
```

 Adding Elements to Multidimensional Array:

You can also add new rows or columns.

```
$contacts = array(
array("name" => "John", "phone" => "1234567890", "email" => "john@example.com"),
array("name" => "Jane", "phone" => "9876543210", "email" => "jane@example.com")
);
$contacts[] = array("name" => "Tom", "phone" => "5555555555", "email" => "tom@example.com");
// Adding a new row
echo $contacts[2]["name"]; // Outputs: Tom
?>
```

4. Array Functions in PHP:

 PHP offers several useful functions to manipulate arrays:

1. **count():**

Returns the number of elements in an array.

2. **array_push():**

Adds one or more elements to the end of an array.

3. **array_pop():**

Removes the last element of an array.

4. **array_merge():**

Merges two or more arrays.

5. **in_array():**

Checks if a value exists in an array.

6. **array_keys():**

Returns all the keys of an array.

7. **array_values():**

Returns all the values of an array.

Example of Using Array Functions:

```
$fruits = array("Apple", "Banana", "Orange");
// Count the number of elements in the array
echo count($fruits); // Outputs: 3
// Add a new element to the array
array_push($fruits, "Mango");
echo "
" . $fruits[3]; // Outputs: Mango
// Check if a value exists in the array
if (in_array("Banana", $fruits)) {
echo "
Banana is in the array.";
}
// Merge two arrays
$vegetables = array("Carrot", "Cabbage");
$all_items = array_merge($fruits, $vegetables);
print_r($all_items);
?>
```