**Module 2 – Advanced PHP Excercises**

OOPs Concepts

THEORY EXERCISE:

Define Object-Oriented Programming (OOP) and its four main principles: Encapsulation, Inheritance, Polymorphism, and Abstraction.

### -> ****Definition of OOP****

Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around objects. An object is a real-world entity that has **attributes (data)** and **methods (functions)**.  
OOP makes programs **modular, reusable, maintainable, and easier to understand**.

Four Main Principles of OOP

· **Encapsulation** → Wrapping data and methods together in a class. Data is hidden and accessed only through methods.

· **Inheritance** → One class (child) can reuse properties and methods of another class (parent).

· **Polymorphism** → Same method name can perform different tasks (overloading/overriding).

· **Abstraction** → Hiding implementation details and showing only essential features.

Practical Exercise:

Create a simple class in PHP that demonstrates encapsulation by using private and public properties and methods.

<?Php

Class BankAccount {

Private $balance;

Public function\_construct($amount) {

$this->balance =$amount;

}

Public function desposit($amount){

If ($amount>0) {

$this->balance+=$amount;

Echo”Deposite:$amount<br>”;

}

}

Public function withdrow($amount) {

If ($amount >0&& $amount <=$this->balance) {

$this->balance-=$amount;

Echo”withdrawn:$amount<br>”;

}

}

public function getBalance() {

return $this->balance;

}

}

// Object creation

$account = new BankAccount(1000);

$account->deposit(500);

$account->withdraw(200);

echo "Current Balance: " . $account->getBalance();

?>

**Deposited: 500  
Withdrawn: 200  
Current Balance: 1300 output**

Class

THEORY EXERCISE: 

Explain the structure of a class in PHP, including properties and methods

In php,a class is a blueprint for creating objects.it defines the properties (variables) and methods (functions) that describe the behavior and data of the objects.

**Structure of class in php**

1. Class keyword and Class Name

A class is defined using the class keyword followed by the class name.

1. Properties (Variables)

-> These are variables that hold data.

-> They can be declared as public, private, or protected.

1. Methods (Functions)

-> These are functions defined inside a class.

-> They define the behavior of the object.

1. Objects

An object is created from a class using the new keyword.

**Practical Exercise: **

Write a PHP script to create a class representing a "Car" with properties like make, model, and year, and a method to display the car details.

<?Php

Class car {

// Properties

public $make;

public $model;

public $year;

public function \_\_construct($make, $model, $year) {

$this->make = $make;

$this->model = $model;

$this->year = $year;

}

public function displayCarDetails() {

echo "Car Make: " . $this->make . "<br>";

echo "Car Model: " . $this->model . "<br>";

echo "Car Year: " . $this->year . "<br>";

}

}

// Object creation

$car1 = new Car();

$car1->displayCarDetails();

?>

**OUTPUT**

**Car Make: Toyota**

**Car Model: Corolla**

**Car Year: 2022**

Object

THEORY EXERCISE: 

What is an object in OOP? Discuss how objects are instantiated from classes in PHP.

An **object** is a real-world entity created from a **class**.

While a **class** is only a blueprint or template, an **object** is an actual instance of that blueprint.

Each object has its own **properties (data)** and can use the **methods (functions)** defined in the class.

### ****Object Instantiation in PHP****

In PHP, an object is created from a class using the new keyword.

This process is called **instantiation**.

Once an object is created, we can access class properties and methods using the -> operator.

**Practical Exercise: **

Instantiate multiple objects of the "Car" class and demonstrate how to access their properties and methods.

<?php

class Car {

public $make;

public $model;

public $year;

/ Method to display car details

public function displayCarDetails() {

echo "Car Make: " . $this->make . "<br>";

echo "Car Model: " . $this->model . "<br>";

echo "Car Year: " . $this->year . "<br><br>";

}

}

// First object

$car1 = new Car();

$car1->make = "Toyota";

$car1->model = "Corolla";

$car1->year = 2022;

$car1->displayCarDetails();

// second object

$car1 = new Car();

$car1->make = "Honda";

$car1->model = "Civic";

$car1->year = 2021;

$car1->displayCarDetails();

// Thired object

$car3 = new Car();

$car3->make = "Maruti";

$car3->model = "Swift";

$car3->year = 2023;

$car3->displayCarDetails();

?>

Extends

THEORY EXERCISE: 

Explain the concept of inheritance in OOP and how it is implemented in PHP.

**Inheritance** is an OOP concept where one class (child/derived class) acquires the **properties** and **methods** of another class (parent/base class).

**Types of inheritance**

· **Single Inheritance:** One child inherits from one parent.

· **Multilevel Inheritance:** A class inherits from a child class which is itself derived from another class.

· **Hierarchical Inheritance:** Multiple child classes inherit from the same parent.

**Practical Exercise:** 

Create a "Vehicle" class and extend it with a "Car" class. Include properties and methods inboth classes, demonstrating inherited behavior.

<?Php

// parent class

Class vehicle {

Public $brand;

Public $year;

// Method in parent class

public function start() {

echo "The vehicle is starting...<br>";

}

public function stop() {

echo "The vehicle is stopping...<br>";

}

}

// Child class

class Car extends Vehicle {

public $model;

// Method in child class

public function displayDetails() {

echo "Car Brand: " . $this->brand . "<br>";

echo "Car Model: " . $this->model . "<br>";

echo "Manufacturing Year: " . $this->year . "<br>";

}

}

// Object of Car class

$car1 = new Car();

$car1->brand = "Toyota";

$car1->year = 2022;

$car1->model = "Corolla";

// Calling methods

$car1->start();

$car1->displayDetails();

$car1->stop();

?>

**Ouput:**

**The vehicle is starting...**

**Car Brand: Toyota**

**Car Model: Corolla**

**Manufacturing Year: 2022**

**The vehicle is stopping...**

Overloading

THEORY EXERCISE:

Discuss method overloading and how it is implemented in PHP

**Method Overloading** in OOP means having multiple methods in the same class with the **same name but different parameters** (number or type).

It allows a class to handle different kinds of data inputs with the same method name.

In languages like **Java**, method overloading is directly supported by defining

multiple methods with the same name but different parameter lists.

**Method Overloading in PHP**

PHP **does not support traditional method overloading** (like Java or C++).

But PHP provides a way to achieve similar behavior using **magic methods**:

\_\_call($name, $arguments) → Handles calls to inaccessible or undefined methods.

\_\_callStatic($name, $arguments) → Handles calls to inaccessible or undefined static methods.

**Practical Exercise: **

Create a class that demonstrates method overloading by defining multiple methods withthesame name but different parameters.

<?php

class Calculator {

// Magic method for method overloading

public function \_\_call($name, $arguments) {

if ($name == "add") {

$count = count($arguments);

if ($count == 2) {

return $arguments[0] + $arguments[1];

} elseif ($count == 3) {

return $arguments[0] + $arguments[1] + $arguments[2];

} else {

return "Invalid number of arguments!";

}

}

}

}

// Object creation

$calc = new Calculator();

// Demonstrating method overloading

echo "Sum of 10 and 20: " . $calc->add(10, 20) . "<br>";

echo "Sum of 10, 20 and 30: " . $calc->add(10, 20, 30) . "<br>";

echo "Test with 4 numbers: " . $calc->add(10, 20, 30, 40);

?>

**Abstraction Interface**

THEORY EXERCISE: 

Explain the concept of abstraction and the use of interfaces in PHP

· **Definition:** Abstraction in OOP is the process of hiding the internal details of implementation and showing only the essential features to the user.

· It allows the programmer to **focus on what an object does** instead of how it does it.

· In PHP, abstraction can be achieved using **abstract classes** and **interfaces**.

### ****Abstract Classes****

An **abstract class** cannot be instantiated directly.

It can have both **abstract methods (without body)** and **regular methods (with body)**.

Child classes must **implement all abstract methods**.

**Practical Exercise: **

Define an interface named VehicleInterface with methods like start(), stop(), andimplement this interface in multiple classes.

<?php

// Interface definition

interface VehicleInterface {

public function start();

public function stop();

// Class Car implementing the interface

}

class Car implements VehicleInterface {

public function start() {

echo "Car is starting...<br>";

}

public function stop() {

echo "Car is stopping...<br>";

}

}

// Class Bike implementing the interface

class Bike implements VehicleInterface {

public function start() {

echo "Bike is starting...<br>";

}

public function stop() {

echo "Bike is stopping...<br>";

}

}

// Object of Car

$car = new Car();

$car->start();

$car->stop();

// Object of Bike

$bike = new Bike();

$bike->start();

$bike->stop();

?>

**Constructor**

THEORY EXERCISE:

 What is a constructor in PHP? Discuss its purpose and how it is used.

A **constructor** in PHP is a special type of method within a class that is automatically called when an object of that class is created. It is defined using the keyword \_\_construct().

Purpose of a Constructor

1. **Automatic Initialization** – It helps initialize object properties with default or user-defined values when the object is created.
2. **Avoids Repetition** – Instead of setting property values manually after creating an object, the constructor does it automatically.
3. **Encapsulation of Logic** – Allows setting up necessary configurations (like database connection, opening files, etc.) when the object is instantiated.
4. **Improves Readability** – Keeps code clean and makes object initialization easier.

How a Constructor is Used in PHP

**It is defined inside a class using function \_\_construct().**

**It can accept parameters for initializing object properties.**

**PHP automatically calls the constructor when the new keyword is used.**

**Destructor**

THEORY EXERCISE:

Explain the role of a destructor in PHP and when it is called.

A **destructor** is a special function of a class in PHP, just like a constructor.

It is automatically **called when an object is destroyed** or goes out of scope.

### ****Role of a Destrouctor****

To perform **cleanup operations** before an object is removed from memory.

To **release resources** such as database connections, files, or sockets.

To perform **final tasks** like saving logs or removing temporary data.

It works together with PHP’s **garbage collector** for memory management.

**When Destructor is Called**

At the end of script execution.

When an object goes out of scope (e.g., function ends).

When an object is explicitly destroyed using unset().

**Practical Exercise:**

Write a class that implements a destructor to perform cleanup tasks when an object is destroyed.

<?Php

class FileHandler {

private $file;

// Constructor opens a file

public function \_\_construct($filename) {

$this->file = fopen($filename, 'w'); // Open file in write mode

echo "File '$filename' opened.<br>";

}

// Method to write data to the file

public function writeData($data) {

fwrite($this->file, $data . PHP\_EOL); // Write data to file

echo "Data written: $data<br>";

}

// Destructor closes the file automatically

public function \_\_destruct() {

fclose($this->file); // Close the file

echo "File closed.<br>";

}

}

// Creating object of FileHandler class

$fileObj = new FileHandler("example.txt");

// Writing data into the file

$fileObj->writeData("Hello, world!");

$fileObj->writeData("This is a test.");

?>

**Magic Methods**

**Practical Exercise:**

-> Create a class that uses the \_\_set() and \_\_get() magic methods to

dynamically create and access properties based on user input.

<?php

class DynamicProperties {

private $data = array(); // To store dynamic properties

// Magic method to set property dynamically

public function \_\_set($name, $value) {

$this->data[$name] = $value;

echo "Property '$name' has been set to '$value'.<br>";

}

// Magic method to get property dynamically

public function \_\_get($name) {

if (array\_key\_exists($name, $this->data)) {

return $this->data[$name];

} else {

echo "Property '$name' does not exist.<br>";

return null;

}

}

}

// Create object

$obj = new DynamicProperties();

// Dynamically set properties

$obj->name = "Shruti";

$obj->course = "PHP Development";

$obj->city = "Ajmer";

// Access properties dynamically

echo "Name: " . $obj->name . "<br>";

echo "Course: " . $obj->course . "<br>";

echo "City: " . $obj->city . "<br>";

?>

**Scope Resolution**

**Practical Exercise:**

o Define a class with static properties and methods to keep track of the number

of instances created. Use the scope resolution operator to access these static

members.

<?php

class Counter {

// Static property to keep track of number of instances

public static $count = 0;

// Constructor increases count whenever a new object is created

public function \_\_construct() {

self::$count++; // Access static property using scope resolution operator

}

// Static method to display total instances

public static function displayCount() {

echo "Total objects created: " . self::$count . "<br>";

}

}

// Creating objects

$obj1 = new Counter();

$obj2 = new Counter();

$obj3 = new Counter();

// Access static method using class name and scope resolution operator

Counter::displayCount();

?>

**Traits**

1. **Practical Exercise:**

o Create two traits: Logger and Notifier. Use these traits in a class User to log

user activities and send notifications.

// Trait 1: Logger

trait Logger {

public function logActivity($activity) {

echo "Logging activity: $activity<br>";

}

}

// Trait 2: Notifier

trait Notifier {

public function sendNotification($message) {

echo "Sending notification: $message<br>";

}

}

// Class User using both traits

class User {

use Logger, Notifier; // Using traits

public $name;

public function \_\_construct($name) {

$this->name = $name;

}

public function performAction($action) {

echo $this->name . " performed: $action<br>";

$this->logActivity($action); // Call Logger trait method

$this->sendNotification($action); // Call Notifier trait method

}

}

// Testing

$user1 = new User("Shruti");

$user1->performAction("Login");

$user2 = new User("Ravi");

$user2->performAction("Update Profile");

**Visibility**

1. **Practical Exercise:**

o Develop a class Account with properties for username (public), password

(private), and accountBalance (protected). Demonstrate how to access these

properties in a derived class.

<?php

// Base class

class Account {

public $username;

private $password;

protected $accountBalance;

// Constructor

public function \_\_construct($username, $password, $balance) {

$this->username = $username;

$this->password = $password;

$this->accountBalance = $balance;

}

// Public method to get password safely

public function getPassword() {

return $this->password;

}

}

// Derived class

class SavingsAccount extends Account {

public function showBalance() {

// Can access protected property

echo "Account Balance for {$this->username}: {$this->accountBalance}<br>";

}

public function showPassword() {

// Cannot access private property directly

// echo $this->password; // This will give error

echo "Password (using getter): " . $this->getPassword() . "<br>";

}

}

// Testing

$user1 = new SavingsAccount("Shruti", "12345", 5000);

// Access public property directly

echo "Username: " . $user1->username . "<br>";

// Access protected and private via derived class methods

$user1->showBalance();

$user1->showPassword();

?>

**Type Hinting**

13. **Practical Exercise:**

o Write a method in a class Order that accepts an array of products (type-hinted)

and calculates the total order amount

class Order {

public function calculateTotal(array $products) {

$total = 0;

foreach ($products as $product) {

$total += $product['price'];

}

return $total;

}

}

// Testing

$order = new Order();

$products = [

['name' => 'Lipstick', 'price' => 500],

['name' => 'Perfume', 'price' => 1500],

['name' => 'Face Cream', 'price' => 800]

];

$totalAmount = $order->calculateTotal($products);

echo "Total Order Amount: ₹" . $totalAmount;

**Final Keyword**

1. **Practical Exercise:**

o Create a base class Animal and a final class Dog. Attempt to extend Dog and

demonstrate the restriction imposed by the final keyword.

<?php

// Base class

class Animal {

public function eat() {

echo "Animal is eating.<br>";

}

}

// Final class

final class Dog extends Animal {

public function bark() {

echo "Dog is barking.<br>";

}

}

// Testing Dog class

$dog = new Dog();

$dog->eat(); // Inherited method from Animal

$dog->bark(); // Dog's own method

// Attempt to extend final class

/\*

class Puppy extends Dog {

// This will cause an error: Cannot extend final class Dog

}

\*/

?>

**Email Security Function**

15. **Practical Exercise:**

o Write a function that sanitizes user input for an email address, validates it, and

throws an exception if it fails validation.

<?php

Function validateEmail($email){

$sanitizedEmail =filter\_var($email,FILTER\_SANITIZE\_EMAIL);

If(!filter\_var($anitizedEmail,FILTER\_SANITIZE\_EMAIL);

Throw new EXception(“invalid email address:$email);

}

return $sanitizedEmail;

}

Try{

$email=”shruti@@example.com”;//try valid email like shruti@gmail.com

$validEmail=validateEmail($email);

echo”valid email:”.$validEmail;

}catch(Exception $e){

Echo”Error.”.$e->getMessage();

}

?>

**File Handling**

16. **Practical Exercise:**

o Create a script that uploads a file and reads its content. Implement error

handling to manage any file-related exceptions.

<!DOCTYPE html>

<html>

<body>

<form action="" method="post" enctype="multipart/form-data">

<label>Select a text file:</label>

<input type="file" name="myfile">

<input type="submit" name="upload" value="Upload">

</form>

<?php

if (isset($\_POST['upload'])) {

try {

if (!isset($\_FILES['myfile']) || $\_FILES['myfile']['error'] != 0) {

throw new Exception("File upload failed!");

}

$fileName = $\_FILES['myfile']['name'];

$fileTmp = $\_FILES['myfile']['tmp\_name'];

// folder create if not exists

if (!file\_exists("uploads")) {

mkdir("uploads");

}

$target = "uploads/" . $fileName;

// move file

move\_uploaded\_file($fileTmp, $target);

// read content

$content = file\_get\_contents($target);

echo "<h3>File Uploaded Successfully!</h3>";

echo "<p>File Name: $fileName</p>";

echo "<h4>File Content:</h4><pre>$content</pre>";

} catch (Exception $e) {

echo "<p style='color:red;'>Error: " . $e->getMessage() . "</p>";

}

}

?>

</body>

</html>

**Handling Emails**

17. **Practical Exercise:**

o Develop a function to send a welcome email to a user upon registration,

ensuring the email format is validated first

<?php

function sendWelcomeEmail($email) {

// Step 1: Validate email format

if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {

throw new Exception("Invalid email format: $email");

}

// Step 2: Email details

$to = $email;

$subject = "Welcome to Our Website!";

$message = "Hello,\n\nThank you for registering with us!\nWe’re excited to have you on board.\n\nBest regards,\nTeam Shruti 😊";

$headers = "From: noreply@yourwebsite.com";

// Step 3: Send email

if (mail($to, $subject, $message, $headers)) {

echo "✅ Welcome email sent successfully to $email";

} else {

throw new Exception("Failed to send email to $email");

}

}

// Example usage

try {

$userEmail = "shruti@example.com"; // change as needed

sendWelcomeEmail($userEmail);

} catch (Exception $e) {

echo "Error: " . $e->getMessage();

}

?>

**MVC Architecture**

18. **Practical Exercise:**

o Extend the simple MVC application to include a model for managing user

profiles, a view for displaying user details, and a controller for handling user

actions.