Object-Oriented Programming.

In PHP.

What is OOP?

OOP stands for Object-Oriented Programming, which is a programming paradigm that uses objects – instances of classes – to organize and structure code.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Class: A class is user define datatype . A class is a blueprint or template for creating objects. It defines a set of attributes and methods that the objects of the class will have.

Purpose: Classes provide a way to model real-world entities and their behaviors in code. They encapsulate data and behavior into a single unit, allowing for code organization and reuse.

Example:

```php
<?php
class Fruit {
  // code goes here...
}
?>
```

Object: An object is an instance of a class. It is a concrete entity created from the blueprint defined by a class.

Purpose: Objects represent individual entities in your program, and they interact with each other by invoking each other's methods and accessing each other's attributes.

Objects of a class are created using the ‘new’ keyword.

Constructor:

In PHP, the **__construct** function is a special method that is automatically called when an object is created from a class. It is a constructor method, and its primary purpose is to initialize the properties (attributes) of the object or perform any setup operations needed for the object to be in a valid state.

<?php

```
class Car {
    public $color;
    public $model;

    // The __construct method
    public function __construct($color, $model) {
        $this->color = $color;
        $this->model = $model;
        echo "A new car instance has been created!\n";
    }
    // Other methods can be defined here
}
// Creating an object (instance) of the Car class
$myCar = new Car("Blue", "Sedan");
// Accessing properties of the object
echo "Color: " . $myCar->color . "\n";
echo "Model: " . $myCar->model . "\n";
?>
```

Explanation:

The **Car** class has two public properties (**$color** and **$model**).

The **__construct** method is used to initialize these properties when a new **Car** object is created.

When the **Car** object is instantiated with **new Car("Blue", "Sedan")**, the **__construct** method is automatically called.

The **echo** statement inside the constructor outputs a message indicating that a new car instance has been created.

Destructor:
In PHP, a destructor is a special method called **__destruct** that is automatically invoked when an object is destroyed or goes out of scope. The primary purpose of a destructor is to perform cleanup tasks, such as releasing resources or closing connections, before the object is removed from memory.

```
<?php
class Car {
    public $color;
    public $model;
    // The constructor method
    public function __construct($color, $model) {
        $this->color = $color;
        $this->model = $model;
        echo "A new car instance has been created!\n";
    }
    // The destructor method
    public function __destruct() {
        echo "The car instance is being destroyed. Cleanup tasks may be performed here.\n";
    }
    // Other methods can be defined here
}
// Creating an object (instance) of the Car class
$myCar = new Car("Blue", "Sedan");
// Accessing properties of the object
echo "Color: " . $myCar->color . "\n";
echo "Model: " . $myCar->model . "\n";
// The object goes out of scope or is explicitly destroyed
unset($myCar);
// Output: The car instance is being destroyed. Cleanup tasks may be performed here.
?>
```

The **Car** class has a destructor method named **__destruct**.

The constructor (**__construct**) is called when a new **Car** object is created, and it outputs a message.

The destructor (**__destruct**) is automatically called when the object goes out of scope or is explicitly destroyed using the **unset** function. The destructor's purpose is to perform cleanup tasks.

PHP - Access Modifiers: Properties and methods can have access modifiers which control where they can be accessed.

There are three access modifiers:

- public - the property or method can be accessed from everywhere. This is default.
- protected - the property or method can be accessed within the class and by classes derived from that class
- private - the property or method can ONLY be accessed within the class.

Code-Syntex:

```
<?php
class Fruit {
  public $name;
  public $color;
  public $weight;

  function set_name($n) {  // a public function (default)
    $this->name = $n;
  }
  protected function set_color($n) { // a protected function
    $this->color = $n;
  }
  private function set_weight($n) { // a private function
    $this->weight = $n;
  }
}
```

```
$mango = new Fruit();
$mango->set_name('Mango'); // OK
$mango->set_color('Yellow'); // ERROR
$mango->set_weight('300'); // ERROR
?>
```

Inheritance:

Inheritance is a fundamental concept in OOP that allows a new class (called a subclass or derived class) to inherit properties and behaviors from an existing class (called a superclass or base class). The subclass can then reuse and extend the functionalities of the superclass.

Types of Inheritance: There are different types of inheritance based on the relationships between the superclass and subclass:

1. **Single Inheritance:**
    - A subclass inherits from only one superclass.
    - Example: Class B inherits from Class A.
2. **Multiple Inheritance:**
    - A subclass inherits from more than one superclass.
    - Example: Class C inherits from both Class A and Class B.
3. **Multilevel Inheritance:**
    - A subclass becomes a superclass for another class, creating a chain of inheritance.
    - Example: Class B inherits from Class A, and Class C inherits from Class B.
4. **Hierarchical Inheritance:**
    - Multiple classes inherit from a single superclass.
    - Example: Both Class B and Class C inherit from Class A

PHP - Class Constants:

Class constants can be useful if you need to define some constant data within a class.

A class constant is declared inside a class with the const keyword.

A constant cannot be changed once it is declared.

Class constants are case-sensitive. However, it is recommended to name the constants in all uppercase letters.

We can access a constant from outside the class by using the class name followed by the scope resolution operator (::) followed by the constant name, like here:

```
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
}
echo Goodbye::LEAVING_MESSAGE;
?>
```

Abstract:

Abstract Class Definition:

- An abstract class is declared using the abstract keyword.
- An abstract class can contain both abstract and non-abstract (concrete) methods.
- Abstract classes cannot be instantiated on their own; they serve as a blueprint for other classes.

CODE:

```
class Circle extends Shape {
    private $radius;

    public function __construct($radius) {
        $this->radius = $radius;
    }

    // Implementing the abstract method
    public function calculateArea() {
        return pi() * $this->radius * $this->radius;
    }
```

```
    }
```

Polymorphism:

Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows objects of different types to be treated as objects of a common type. This concept provides a way to write code that can work with objects of various classes in a unified manner, promoting flexibility, code reuse, and abstraction.

Polymorphism in OOP typically manifests in two forms: compile-time polymorphism (method overloading) and runtime polymorphism (method overriding).

1. Compile-Time Polymorphism (Method Overloading):

- Definition: Method overloading occurs when a class has multiple methods with the same name but different parameters.
- Example in PHP:

```
class MathOperations {
    public function add($a, $b) {
        return $a + $b;
    }
    public function addThree($a, $b, $c) {
        return $a + $b + $c;
    }
}
$math = new MathOperations();
echo $math->add(2, 3) . "\n";          // Calls add($a, $b)
echo $math->addThree(2, 3, 5) . "\n";  // Calls addThree($a, $b, $c)
```

2. Runtime Polymorphism (Method Overriding):

- **Definition:** Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass.
- **Example in PHP:**

```
class Animal {
```

```
    public function makeSound() {
        return "Generic animal sound";
    }
}

class Dog extends Animal {
    public function makeSound() {
        return "Woof! Woof!";
    }
}

class Cat extends Animal {
    public function makeSound() {
        return "Meow!";
    }
}

$dog = new Dog();
$cat = new Cat();

echo "Dog says: " . $dog->makeSound() . "\n";
echo "Cat says: " . $cat->makeSound() . "\n";
```

Encapsulation:

Encapsulation is one of the fundamental principles of object-oriented programming (OOP) that involves bundling the data (attributes or properties) and the methods (functions or procedures) that operate on the data into a single unit known as a class. It restricts access to the internal details of an object and prevents the accidental modification of its state from outside the class. Encapsulation promotes modularity, code organization, and the concept of "data hiding," where the internal implementation details are hidden from the outside world.

Types of Encapsulation:

1. Public Access Modifier:
    - Description: All class members are accessible from outside the class.
    - Example in PHP:

```
class Car {
    public $model;

    public function startEngine() {
        echo "Engine started!\n";
    }
}

$myCar = new Car();
$myCar->model = "Sedan";
echo "Car model: " . $myCar->model . "\n";
$myCar->startEngine();
```

Protected Access Modifier:

- Description: Class members are accessible within the class and its subclasses.
- Example in PHP:

```
class BankAccount {
    protected $balance;

    public function __construct($initialBalance) {
        $this->balance = $initialBalance;
    }

    protected function deductFees() {
        // Deduct fees logic
```

```
    }
}

class SavingsAccount extends BankAccount {
    public function getBalance() {
        // Accessing protected property from the subclass
        return $this->balance;
    }
}

$savingsAccount = new SavingsAccount(1000);
echo "Balance: $" . $savingsAccount->getBalance() . "\n";
```

Private Access Modifier:

- Description: Class members are accessible only within the class.
- Example in PHP:

```
class User {
    private $username;
    private $password;

    public function __construct($username, $password) {
        $this->username = $username;
        $this->password = $password;
    }

    public function getUsername() {
        return $this->username;
    }
```

```
}

$user = new User("john_doe", "password123");
echo "Username: " . $user->getUsername() . "\n";
// Attempting to access private property results in an error
// echo "Password: " . $user->password . "\n";
```