PHP 14 (Advanced PHP)

oops

**What Is Object Oriented Programming?**

Object-oriented programming (OOP) is a programming paradigm that uses objects - instances of classes - to represent and manipulate data and the operations that can be performed on that data. OOP focuses on modeling real-world objects and their interactions, using encapsulation, inheritance, and polymorphism to create modular, reusable code.

In OOP, data and behavior are combined into objects, which can communicate with one another by sending messages. A class is a blueprint for creating objects that share the same properties and methods. A class can inherit properties and methods from a parent class, and can also override or extend those properties and methods.

**What Are Properties Of Object Oriented Systems?**

Abstraction: The ability to represent complex systems and ideas using simplified models that capture the most important features of the system, while ignoring irrelevant details.

Encapsulation: The idea that the internal workings of an object should be hidden from external code, and that access to an object's data should be controlled through its public interface (methods)

Inheritance: The ability to create new classes by extending or modifying existing ones. Inheritance allows for code reuse and helps to reduce duplication, as common attributes and behaviors can be defined in a base class and inherited by derived classes.

Polymorphism: The ability of objects to take on multiple forms, by responding to the same message in different ways

**What Is Difference Between Class And Interface?**

The main difference between a class and an interface is that a class can provide both implementation and definition of methods, while an interface only provides a

definition of methods. In other words, a class can provide both the "what" and the "how" of a method, while an interface only defines the "what".

In object-oriented programming, a class is a blueprint for creating objects that share the same properties and methods. An interface, on the other hand, is a collection of abstract methods that defines a contract for a class to implement.

The main difference between a class and an interface is that a class can provide both implementation and definition of methods, while an interface only provides a definition of methods. In other words, a class can provide both the "what" and the "how" of a method, while an interface only defines the "what".

A class can also have data members, properties, constructors, and other features, while an interface only defines a set of abstract methods. A class can be instantiated, while an interface cannot.

Another key difference is that a class can only inherit from a single class, while it can implement multiple interfaces. This allows a class to define multiple contracts with other classes, and to share behavior across different interfaces.

In general, classes are used to define objects with state and behavior, while interfaces are used to define contracts between objects that need to interact with each other. Classes are typically used for more complex, domain-specific concepts, while interfaces are used for more general, reusable abstractions.

**What Is Overloading?**

Overloading is a feature in object-oriented programming that allows multiple methods or operators with the same name to be defined in the same class or interface. Overloading enables a method to perform different operations based on the arguments that are passed to it.

**What Is T\_PAAMAYIM\_NEKUDOTAYIM (Scope Resolution Operator (::) with Example**

T\_PAAMAYIM\_NEKUDOTAYIM is a Hebrew term meaning "double colon", and it is the name of the scope resolution operator in PHP. The scope resolution operator (::) is used to access static methods, constants, and properties of a class.

class Example {

public static function greet() { echo "Hello, world!";

}

}

Example::greet(); // Outputs "Hello, world!"

**What are the differences between abstract classes and interfaces?**

Abstract classes and interfaces are two fundamental concepts in object-oriented programming, but they serve different purposes and have some key differences.

Implementation: One of the main differences between abstract classes and interfaces is that an abstract class can provide an implementation for some of its methods, while an interface cannot provide any implementation

1. Instantiation: An abstract class cannot be instantiated directly, it must be subclassed, whereas an interface cannot be instantiated at all. Instead, a class can implement one or more interfaces, and provide an implementation for all of the methods declared in the interface.

Single inheritance vs. Multiple implementation: A class can only inherit from one abstract class, but can implement multiple interfaces

ole: Abstract classes are often used to define common behavior for a group of related classes, while interfaces are used to define a common set of methods that unrelated classes can implement,

Extensibility: Abstract classes can be extended by adding new methods and fields, whereas adding new methods to an interface can break any classes that implement that interface

**Define Constructor and Destructor?**

The constructor method has the same name as the class, and can accept parameters if needed. Here is an example of a constructor in PHP:

class Example { public $name;

public function construct($name) {

$this->name = $name;

}

}

$example = new Example("John"); echo $example->name; // Output: John

Destructor?

class Example {

public function destruct() { echo "Object destroyed";

}

}

$example = new Example(); unset($example); // Output: Object destroyed

**How to Call Parent Constructor?**

class ParentClass { protected $name;

public function construct($name) {

$this->name = $name;

}

}

class ChildClass extends ParentClass { protected $age;

public function construct($name, $age) { parent:: construct($name);

$this->age = $age;

}

public function getAge() { return $this->age;

}

}

$child = new ChildClass("John", 25); echo $child->getAge(); // Output: 25

**Are Parent Constructor Called Implicitly When Create An ObjectOf Class?**

class ParentClass {

protected $name;

public function construct($name) {

$this->name = $name;

echo "Parent constructor called\n";

}

}

class ChildClass extends ParentClass { protected $age;

public function construct($name, $age) {

$this->age = $age;

echo "Child constructor called\n";

}

public function getAge() { return $this->age;

}

}

$child = new ChildClass("John", 25); echo $child->getAge(); // Output: 25

**What Happen, If Constructor Is Defined As Private Or Protected?**

class MyClass {

protected function construct() { echo "Constructor called\n";

}

public static function createObject() { return new MyClass();

}

}

class ChildClass extends MyClass { public function construct() {

echo "Child constructor called\n";

}

}

$obj = MyClass::createObject(); // Output: Constructor called

$obj2 = new ChildClass(); // Output

**What are PHP Magic Methods/Functions? List them Write program for Static**

PHP magic methods, also known as magic functions, are special functions that allow you to implement certain behaviors in PHP classes.

1.

**construct()**

created.

2.

**destruct()**

destroyed.

- This is the constructor method that is called when an object is

- This is the destructor method that is called when an object is

3.

**call()**

exist.

4.

**callStatic()**

not exist.

- This method is called when an object method is called that does not

- This method is called when a static method is called that does

5. - This method is called when a class property is accessed that is not defined or is not accessible

**get()**

**Create multiple Traits and use it in to a single class?**

trait Trait1 {

public function method1() { echo "Method 1 from Trait 1";

}

}

trait Trait2 {

public function method2() { echo "Method 2 from Trait 2";

}

}

class MyClass { use Trait1, Trait2;

public function method3() { echo "Method 3 from MyClass";

}

}

$obj = new MyClass();

$obj->method1(); // Output: Method 1 from Trait 1

$obj->method2(); // Output: Method 2 from Trait 2

$obj->method3(); // Output: Method 3 from MyClass

**Write PHP Script of Object Iteration?**

class Person {

public $name; public $age;

public function construct($name, $age) {

$this->name = $name;

$this->age = $age;

}

}

$person = new Person("John", 30);

$person->email = "[john@example.com](mailto:john@example.com)";

foreach ($person as $key => $value) { echo "$key: $value\n";

}

name: John age: 30

email: xyz[@.com](mailto:john@example.com)

**Use of The $this keyword**

When you define a method in a class, you can use

**$this**

to access the properties and methods of the current instance

class Person { private $name;

public function construct($name) {

$this->name = $name;

}

public function greet() {

echo "Hello, my name is " . $this->name . "\n";

}

}

$person = new Person("John");

$person->greet();

**Jquery**

**What is jQuery?**

jQuery is a lightweight, "write less, do more", JavaScript library. The purpose of jQuery is **to make it much easier to use JavaScript on your website**. jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

**How are JavaScript and jQuery different?**

Though JavaScript is the basic language from which jQuery has evolved, **jQuery makes event handling, DOM manipulation, Ajax calls much easier than JavaScript**. jQuery also allows us to add animated effects on our web page which takes a lot of pain and lines of code with JavaScript.

**Which is the starting point of code execution in jQuery?**

The jQuery starts its code execution from the **$(document).** **ready() function** which is executed whenever the whole HTML DOM is loaded and is totally rendered by the browser, so that the event handlers work correctly without any errors.

**Document Load Vs Window. Load() jQuery**

The key difference between $(document). ready() and $(window). load() event is that the code included inside onload function will run once the entire page(images, iframes, stylesheets,etc) are loaded whereas the $(document). ready() event fires before all images,iframes etc.

**What is the difference between prop and attr?**

**jQuery .prop() :**This method is used to get the value of a property for the first element in the set of matched elements. This method returns the current value.

**$(selector).prop(property)**

**jQuery .attr() method:**This method is used to either fetch the value of an attribute from the first element in the matched set or set attribute values onto all matched elements. This method returns the default value.

**$(selector).attr(attribute)**

**How We Can Implement Animation Effects In Jquery?**

The jQuery animate() method is used to create custom animations.

**$(*selector*).animate({*params*}*,speed,callback*);**

The required params parameter defines the CSS properties to be animated.

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the animation completes.

**Example:**

$("button").click(function(){  
  $("div").animate({  
    left: '250px',  
    opacity: '0.5',  
    height: '150px',  
    width: '150px'  
  });  
});

**Apply jQuery validation using library.**

<html>

<head>

    <meta charset="utf-8">

    <title>Comment Form</title>

    <!-- below we are including the jQuery and jQuery plugin .js files -->

    <script src="<https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js>"></script>

    <script src="<https://cdn.jsdelivr.net/jquery.validation/1.16.0/jquery.validate.min.js>"></script>

    <script src="<https://cdn.jsdelivr.net/jquery.validation/1.16.0/additional-methods.min.js>"></script>

    <script>

        $().ready(function () {

            $("#signupForm").validate({

                // in 'rules' user have to specify all the constraints for respective fields

                rules: {

                    firstname: "required",

                    lastname: "required",

                    username: {

                        required: true,

                        minlength: 2 //for length of lastname

                    },

                    password: {

                        required: true,

                        minlength: 5

                    },

},

 messages: {

                    firstname: " Please enter your firstname",

                    lastname: " Please enter your lastname",

                    username: {

                        required: " Please enter a username",

                        minlength: " Your username must consist of at least 2 characters"

                    },

                    password: {

                        required: " Please enter a password",

                        minlength: " Your password must be consist of at least 5 characters"

                    },

}

 });

        });

    </script>

</head>

<body>

    <form class="cmxform" id="signupForm" method="get" action="form-handler.html" autocomplete="off">

        <fieldset>

            <legend>GFG sign-up Form</legend>

<p>

                <label for="firstname">Firstname</label>

                <input id="firstname" name="firstname" type="text"></input>

            </p>

<p>

                <label for="lastname">Lastname</label>

                <input id="lastname" name="lastname" type="text"></input>

            </p>

<p>

                <label for="username">Username</label>

                <input id="username" name="username" type="text"></input>

            </p>

<p>

                <label for="password">Password</label>

                <input id="password" name="password" type="password"></input>

            </p>

 </fieldset>

    </form>

</body>

</html>

**Get state data by country selection (Ajax).**

<script>

function getState(country\_id)

{

$.ajax

({

type: "POST",

url: "statedata",

data:"btn=" + country\_id,

success: function(data)

{

$("#state\_id").html(data);

}

});

}

</script>