SELF ASSIGNMENT

1.

JavaScript is a programming language that is commonly used in web development. It allows developers to add dynamic and interactive elements to web pages, such as pop-up alerts, animations, and form validations. JavaScript can also be used to create more complex applications, such as web-based games and online tools.

2.

Object-oriented programming (OOP) in JavaScript is a programming model that allows developers to organize their code into objects, which are collections of related data and methods. OOP is a way of structuring code that emphasizes the organization of data and functionality into reusable components.

3.

DOM stands for Document Object Model. It is a programming interface for HTML and XML documents that represents the structure of the document as a tree-like object. DOM manipulation refers to the process of modifying the content or structure of an HTML or XML document using JavaScript.

4.

Advantages of Using JavaScript:

* JavaScript is supported by all major browsers and is a widely used programming language.
* It is a client-side scripting language, which means that it runs on the user's computer and does not require server-side processing, which can reduce the load on the server.
* It can be used for both front-end and back-end development.
* It is easy to learn and can be integrated with HTML and CSS, making it a popular choice for web development.

Disadvantages of Using JavaScript:

* It can be vulnerable to security issues such as cross-site scripting (XSS) and cross-site request forgery (CSRF).
* Different browsers may interpret JavaScript differently, which can cause compatibility issues.
* It can make websites slower if not optimized properly, as it is a client-side language and requires the user's computer to process the code.
* It is not suitable for heavy computational tasks as it is a single-threaded language and cannot perform multiple tasks simultaneously.

5.

In CSR, the rendering process happens entirely on the client-side, meaning the browser downloads the entire HTML, CSS, and JavaScript files and then renders the page in the browser. This approach provides a faster user experience because the page loads more quickly while in SSR, the rendering process occurs on the server-side. The server processes the request and generates the final HTML code, which is then sent to the client. However, SSR has slower initial load times because the server has to generate the entire HTML code for every request.

6.

* **push()**: This method adds one or more elements to the end of an array and returns the new length of the array.

<script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo1").innerHTML = fruits;

fruits.push("Kiwi");

document.getElementById("demo2").innerHTML = fruits;

</script>

* **pop()**: This method removes the last element from an array and returns that element.

<script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo1").innerHTML = fruits;

fruits.pop();

document.getElementById("demo2").innerHTML = fruits;

</script>

* **delete()**: Array elements can be deleted using this JavaScript operator

<script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo1").innerHTML =

"The first fruit is: " + fruits[0];

delete fruits[0];

</script>

* **concat()**: This method returns a new array that combines the elements of the original array with one or more additional arrays.

<script>

const myGirls = ["Cecilie", "Lone"];

const myBoys = ["Emil", "Tobias", "Linus"];

const myChildren = myGirls.concat(myBoys);

document.getElementById("demo").innerHTML = myChildren;

</script>

* **splice(): is a method that can be used to add new items to an array:** <script>

const fruits = ["Banana", "Orange", "Apple", "Mango"];

document.getElementById("demo1").innerHTML = fruits;

fruits.splice(0, 0, "Lemon", "Kiwi");

document.getElementById("demo2").innerHTML = fruits;

</script>

7.

Object-oriented programming (OOP) is a programming model that is widely used in PHP. It is a way of organizing code into objects that can contain both data and behavior. In PHP, OOP is implemented through classes and objects.

<script>

let Car = {

    name: "My Car",

    brand: "Toyota",

    make: 2023,

    size: "Medium"}

    </script>

8.

* Inheritance in PHP:

Inheritance is an object-oriented programming concept in PHP that allows classes to inherit properties and methods from a parent class. The child class can then use and modify these inherited properties and methods without having to redefine them. Inheritance is used to create a hierarchy of classes that share common attributes and behaviours.

<?php

//Parent class

class Animal {

public $name;

public $age;

function \_\_construct($name, $age) {

$this->name = $name;

$this->age = $age;

}

public function eat() {

echo "The animal is eating.<br>";

}

}

//Child class that inherits from Animal

class Dog extends Animal {

public function bark() {

echo "The dog is barking.<br>";

}

}

//Create an object of the child class

$dog = new Dog("Max", 5);

//Access properties and methods from the parent class

echo $dog->name; //outputs "Max"

$dog->eat(); //outputs "The animal is eating."

//Access method from the child class

$dog->bark(); //outputs "The dog is barking."

?>

* Traits in PHP:

Traits are a way to reuse code in PHP without using inheritance. A trait is a collection of methods that can be used in multiple classes. Traits are used to group functionality that is not related to the main purpose of the class.

<?php

//Define a trait

trait Greeting {

public function sayHello() {

echo "Hello!<br>";

}

}

//Use the trait in a class

class Person {

use Greeting;

}

//Create an object of the class

$person = new Person();

//Access the method from the trait

$person->sayHello(); //outputs "Hello!"

?>

* + Interface in PHP:

An interface is a blueprint of methods that a class must implement. It defines a set of methods that a class must implement without defining how they are implemented. An interface is used to enforce consistency across different classes that perform similar tasks.

<?php

//Define an interface

interface Animal {

public function eat();

public function sleep();

}

//Implement the interface in a class

class Dog implements Animal {

public function eat() {

echo "The dog is eating.<br>";

}

public function sleep() {

echo "The dog is sleeping.<br>";

}

}

//Create an object of the class

$dog = new Dog();

//Access the methods from the interface

$dog->eat(); //outputs "The dog is eating."

$dog->sleep(); //outputs "The dog is sleeping."

?>

* + Polymorphism:

Polymorphism is the ability of an object to take on many forms. In PHP, polymorphism is achieved through method overriding and method overloading. Method overriding is the ability of a subclass to provide a specific implementation of a method that is already defined in its parent class. Method overloading is the ability to define the same method with different parameters in a single class.

class Animal {

public function makeSound() {

echo "The animal makes a sound";

}

}

class Cat extends Animal {

public function makeSound() {

echo "Meow";

}

}

class Dog extends Animal {

public function makeSound() {

echo "Woof";

}

}

$animal = new Animal();

$cat = new Cat();

$dog = new Dog();

$animal->makeSound(); // Output: The animal makes a sound

$cat->makeSound(); // Output: Meow

$dog->makeSound(); // Output: Woof

* + Namespacing:

Namespacing is a way of organizing code into logical groups to prevent naming collisions. Namespaces can contain functions, classes, and constants. In PHP, namespaces are defined using the **namespace** keyword.

namespace MyProject;

<?php  
namespace Html;  
class Table {  
  public $title = "";  
  public $numRows = 0;  
  public function message() {  
    echo "<p>Table '{$this->title}' has {$this->numRows} rows.</p>";  
  }  
}  
$table = new Table();  
$table->title = "My table";  
$table->numRows = 5;  
?>

9.Design patterns

Design patterns in PHP refer to reusable solutions to commonly occurring problems in software design. Design patterns are standard approaches to solving problems that arise during software development. They help to structure code in a way that is easier to read and maintain, making it more efficient and effective.

* Singleton pattern:

This pattern restricts the instantiation of a class to a single object and ensures that only one instance of the class exists throughout the application.

* Factory pattern:

This pattern provides a way to create objects without exposing the creation logic to the client and refers to the use of a special object that creates other objects.

* Strategy pattern:

This pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable, allowing the algorithm to be selected at runtime.

* Observer pattern:

This pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

* MVC (Model-View-Controller) pattern:

This pattern separates an application into three interconnected components: the model (data storage), the view (user interface), and the controller (input handler), making it easier to manage and modify.

* Adapter pattern:

This pattern allows incompatible objects to work together by creating an intermediate adapter that translates the interface of one object into an interface that the client expects.