**Assignment 6 – PPT Web Dev**

Q.1 What’s Constructor And Its Purpose?

A constructor is a special method or function within a class that is responsible for initializing objects of that class. Its purpose is to set the initial state of an object when it is created. Constructors plays a vital role in object creation and initialization, allowing for the proper setup of objects and ensuring their integrity and usability throughout the program's execution.

Q.2 Explain This Keyword and Its Purpose?

The "this" keyword is a special reference variable that is used within a class or object to refer to itself. It holds a reference to the current instance of the class, allowing us to access its members (variables and methods) and use them within the class's scope.

The purpose of the "this" keyword is to disambiguate between the instance variables of the class and the local variables or parameters that have the same names. When a method or constructor is called, it may receive arguments with the same names as the class's instance variables. In such cases, using "this" helps the program distinguish between the local variables and instance variables.

Q.2 What’s Call Apply Bind Method & Difference Between them?

In JavaScript, the `call`, `apply`, and `bind` methods are used to control the execution context (the value of the `this` keyword) of a function. They allow you to explicitly specify the object on which a function should be invoked.

a. `call` method:

The `call` method allows you to call a function with a specified `this` value and optional arguments passed individually. The syntax is as follows:

functionName.call(thisValue, arg1, arg2, ...);

b. `apply` method:

The `apply` method is similar to `call`, but it takes an array-like object as the second argument, which represents the arguments to be passed to the function. The syntax is as follows:

functionName.apply(thisValue, [arg1, arg2, ...]);

c. `bind` method:

The `bind` method is used to create a new function with a specified `this` value and, if provided, pre-filled with some initial arguments. The original function is not called immediately but returned as a new function that can be invoked later. The syntax is as follows:

const newFunction = functionName.bind(thisValue, arg1, arg2, ...);

Q.4 Explain OOPS ?

OOPS stands for Object-Oriented Programming (OOP). It is a programming paradigm based on the concept of "objects," which are data structures that contain both data (attributes or properties) and behavior (methods or functions). OOP focuses on organizing code into reusable and self-contained objects, making it easier to model and solve complex problems.

Key principles of Object-Oriented Programming:

* **Encapsulation**
* **Abstraction**
* **Inheritance**
* **Polymorphism**

Q.5 Whats Abstraction and Its Purpose?

Abstraction is a fundamental concept in object-oriented programming (OOP) that focuses on representing the essential features of an object while hiding unnecessary details. Abstraction provides a clear and well-defined interface for interacting with objects, shielding the underlying implementation complexity.

The purpose of abstraction is to manage complexity and increase code maintainability by emphasizing what an object does rather than how it does it. By abstracting away the implementation details, developers can work with higher-level concepts and focus on the functionality of the objects, making the code more readable and less error-prone.

Let's consider an example of a car. From a user's perspective, a car is a simple entity with functionalities like starting the engine, accelerating, braking, and turning. As a user, we don't need to know the intricate details of how these actions are performed internally by the car's engine, transmission, and other components. This is abstraction in action, where the internal complexities of the car's mechanism are hidden, and we interact with it through a simplified interface (e.g., a steering wheel, pedals)

Q.6 Whats Polymorphism and Purpose of it?

Polymorphism is a key concept in object-oriented programming (OOP) that allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to represent various types of objects, providing flexibility and extensibility in the code.

The purpose of polymorphism is to create a unified and consistent way of interacting with objects, regardless of their specific types. It allows code flexibility, reuse, and adaptability by treating objects of different classes as instances of their common superclass or interface, enabling a single interface to represent various types of objects

E.g-

class Shape:

def draw(self):

pass

class Circle(Shape):

def draw(self):

print("Drawing a circle")

class Rectangle(Shape):

def draw(self):

print("Drawing a rectangle")

# Polymorphism in action

shapes = [Circle(), Rectangle()]

for shape in shapes:

shape.draw()

Q.7 Whats Inheritance and Purpose of it?

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a new class (the derived or subclass) to inherit properties and behaviors from an existing class (the base or superclass). The derived class can extend and specialize the functionalities of the base class while inheriting its common characteristics.

The purpose of inheritance is to promote the following key benefits:

* Code Reusability: Inheritance allows us to reuse the code and functionalities defined in the base class without duplicating them in the derived class.
* Hierarchy and Organization: Inheritance supports the concept of a class hierarchy, where classes are organized in a parent-child relationship.
* Specialization and Extension: Derived classes can specialize and extend the functionalities of the base class. This means that we can create more specific subclasses that inherit the common properties of the base class while adding or overriding methods to cater to specific requirements.
* Polymorphism: Inheritance plays a significant role in achieving polymorphism, as objects of derived classes can be treated as objects of their superclass. This allows code to work with different objects through a common interface, promoting flexibility and adaptability.

Q.8 Whats Encapsulation and Purpose of it ?

Encapsulation is one of the four fundamental principles of object-oriented programming (OOP). It refers to the bundling of data (attributes or properties) and methods (functions) that operate on that data within a single unit called an "object." Encapsulation ensures that the internal state of an object is hidden from the outside world and can only be accessed through well-defined public interfaces.

The purpose of encapsulation is to achieve the following key objectives:

* Data Protection: By hiding the internal data and implementation details, encapsulation prevents direct access to an object's state from outside the object. Instead, access to the data is controlled through public methods, allowing for better control over how the data is modified and ensuring its integrity.
* Modularity and Reusability: Encapsulation allows objects to be designed as modular and self-contained units. Each object encapsulates its data and behavior, making it easier to reuse in different parts of the code or in other projects without affecting other parts of the codebase.
* Code Maintenance: Encapsulation makes code maintenance easier. Since the internal implementation is hidden from the external code, modifications to the internal implementation do not affect other parts of the code that use the object. This reduces the risk of introducing bugs and improves the overall maintainability of the codebase.
* Information Hiding: Encapsulation promotes the concept of "information hiding," where the details of how an object is implemented are concealed, and only the necessary information is exposed through the public interface. This reduces complexity and increases code readability.

Q.9 Explain Class in JavaScript?

In JavaScript, a class is a blueprint or a template that defines the structure and behavior of objects of that class. It provides a way to create objects with similar properties and methods in a consistent and organized manner.

Syntax:

class ClassName {

constructor(parameter1, parameter2, ...) {

// Constructor: initializes the object's properties

this.property1 = parameter1;

this.property2 = parameter2;

// ... additional property assignments

}

method1() {

// Method definition

// You can access properties using 'this'

// ... method logic

}

method2() {

// Another method definition

// ... method logic

}

// ... additional methods

}

Q.10 What’s Super Keyword & What it does?

The super keyword is used to call functions or access properties of a parent class (or superclass) from within a subclass. It is commonly used in classes that inherit from another class to refer to the parent class's constructor, methods, or properties. The super keyword allows for efficient code reuse and ensures that the parent class's behavior is maintained in the subclass.

e.g-

class ChildClass extends ParentClass {

constructor(...args) {

super(...args); // Calls the constructor of the parent class with arguments

// Additional subclass-specific initialization

}

}