**Assignment 6**

**Q.1 What’s Constructor And Its Purpose?**

In object-oriented programming, a constructor is a special method or function that is used to initialize an object of a class. It is called automatically when an object is created and is responsible for setting initial values to the data members of the object.

The purpose of a constructor is to ensure that an object is properly initialized before it can be used. It is used to allocate memory and initialize the data members of an object. Constructors help in defining the initial state or values of an object, and they ensure that the object is in a valid and usable state.

By defining constructors in a class, you can control how objects are created, initialized, and set up their initial state, ensuring that they are ready for use. Constructors are an essential part of object-oriented programming and play a crucial role in creating and initializing objects in a class hierarchy.

**Q.2 Explain This Keyword and Its Purpose?**

In JavaScript, the "this" keyword is a special identifier that refers to the context in which a function is executed. The specific value of "this" depends on how a function is invoked and can vary dynamically.

The purpose of the "this" keyword in JavaScript is to provide access to the object that is currently being operated upon or accessed within a function or method. It allows functions to access and manipulate the properties and methods of the object it is called upon.

Here are some common use cases of the "this" keyword in JavaScript:

As a reference to the current object: When a function is invoked as a method of an object, the "this" keyword refers to that object. It allows you to access and modify the object's properties and invoke its methods. For example:

| const myObject = {  property: 'value',  method: function() {  console.log(this.property); // Accessing object property using "this"  } };  myObject.method(); // Outputs: "value" |
| --- |

**Q.3 What’s Call Apply Bind Method & Difference Between them ?**

In JavaScript, the call, apply, and bind methods are used to manipulate the value of the this keyword within a function and control its execution context. Although they serve a similar purpose, there are differences in how they are used and how they handle function arguments.

call method: The call method is a built-in function available on every function object in JavaScript. It allows you to invoke a function explicitly, specifying the object to which this should refer. The arguments are passed individually after the context object.

| **function myFunction(arg1, arg2) {  console.log(this); // Refers to the context object  console.log(arg1, arg2); }  const obj = { name: 'John' };  myFunction.call(obj, 'Hello', 'World');** |
| --- |

apply method: Similar to call, the apply method allows you to invoke a function explicitly and specify the object to which this should refer. However, the arguments are passed as an array or an array-like object.

| **function myFunction(arg1, arg2) {  console.log(this); // Refers to the context object  console.log(arg1, arg2); }  const obj = { name: 'John' };  myFunction.apply(obj, ['Hello', 'World']);** |
| --- |

bind method: The bind method returns a new function that has this permanently set to a specific object. It allows you to create a bound function that can be invoked later with a specific context. Unlike call and apply, bind does not immediately invoke the function.

| **function myFunction(arg1, arg2) {  console.log(this); // Refers to the context object  console.log(arg1, arg2); }  const obj = { name: 'John' };  const boundFunction = myFunction.bind(obj);  boundFunction('Hello', 'World');** |
| --- |

**Q.4 Explain OOPS ?**

OOPS stands for Object-Oriented Programming, which is a programming paradigm that focuses on organizing code into objects that represent real-world entities. JavaScript, despite being a prototype-based language, supports object-oriented programming principles. Here's an explanation of OOPS concepts in JavaScript:Objects: Objects are fundamental in JavaScript. They are instances of classes (or prototypes) that encapsulate data and behavior. Objects consist of properties (variables) and methods (functions). You can create objects using object literals or constructor functions.

| **// Object literal const person = {  name: 'John',  age: 25,  sayHello: function() {  console.log(`Hello, my name is ${this.name}.`);  } };  // Constructor function function Person(name, age) {  this.name = name;  this.age = age;  this.sayHello = function() {  console.log(`Hello, my name is ${this.name}.`);  }; }  const john = new Person('John', 25);** |
| --- |

**Q.5 Whats Abstraction and Its Purpose?**

Abstraction is a fundamental concept in computer science and programming that involves simplifying complex systems by focusing on the essential features while hiding the unnecessary details. It allows you to create models or representations of real-world entities in a simplified and manageable way.

The purpose of abstraction is to provide a higher-level view of a system or object, enabling users to interact with it without needing to understand the intricate implementation or inner workings. It offers several benefits:

Simplifying complexity: Abstraction allows you to break down a complex system or object into simpler and more understandable components. It provides a clear and concise representation that helps users comprehend and work with the system or object more easily. By abstracting away the complexities, it simplifies the understanding and usage of the system.

Managing complexity: By hiding unnecessary details, abstraction helps in managing the complexity of a system. It allows you to focus on the important aspects and ignore the irrelevant or low-level implementation specifics. This promotes better organization and structure of code, making it more maintainable and easier to modify or extend.

Encapsulating implementation: Abstraction encapsulates the implementation details, providing a separation between the interface and the implementation. It defines an abstract interface that exposes the essential features and behavior of an object or system, while hiding the internal complexities. This encapsulation improves modularity, code reusability, and helps in managing dependencies between components.

Promoting code reusability: Abstraction enables the creation of reusable components or classes. By defining abstract interfaces, you can create generic and adaptable code that can be used in various contexts. This promotes code reuse, reduces duplication, and improves productivity by leveraging existing solutions and avoiding reinvention.

**Q.6 Whats Polymorphism and Purpose of it?**

Polymorphism is a concept in object-oriented programming that allows objects of different classes to be treated as objects of a common superclass or interface. It refers to the ability of an object to take on different forms or behaviors depending on the context in which it is used.

The purpose of polymorphism is to provide a unified interface or contract that can be shared by multiple objects, even if they have different underlying implementations. It offers several benefits:

Code reusability: Polymorphism enables you to write code that can work with objects of different classes, as long as they adhere to a common interface or inherit from a common superclass. This promotes code reuse, as you can create generic algorithms and functions that operate on a wider range of objects. By designing code to rely on the shared interface, you can avoid writing redundant code for similar functionality.

Flexibility and extensibility: Polymorphism allows you to add new classes or objects that conform to the common interface without modifying existing code. This promotes flexibility and extensibility, as you can introduce new functionality by simply adding new classes that adhere to the existing interface. It enables you to easily incorporate new features or variations into your codebase.

Simplified code structure: Polymorphism helps in simplifying the code structure by promoting a more modular and loosely coupled design. By relying on shared interfaces or superclasses, you can separate the implementation details of individual objects from the code that uses them. This separation of concerns improves code organization, readability, and maintainability.

**Q.7 Whats Inheritance and Purpose of it?**

Inheritance is a fundamental concept in object-oriented programming that allows objects to acquire properties and methods from a parent or base class. It enables the creation of a hierarchy of classes, where child classes inherit and extend the characteristics of a parent class.

In JavaScript, inheritance is implemented using prototype-based inheritance, as JavaScript does not have built-in class-based inheritance like some other programming languages. Instead, objects have a prototype, which serves as a template or blueprint for creating new objects.

The purpose of inheritance is to promote code reuse, enhance modularity, and facilitate the organization and abstraction of code. Here are some key purposes of inheritance in JavaScript:

Code Reusability: Inheritance allows you to define common properties and methods in a parent class and reuse them in multiple child classes. Child classes inherit the behavior and characteristics of the parent class, eliminating the need to duplicate code. This promotes code reuse and reduces redundancy, leading to more efficient and maintainable code.

Extensibility: Inheritance enables you to extend and customize the functionality of existing classes. Child classes can add new properties and methods or override existing ones inherited from the parent class. This allows you to tailor the behavior of classes to specific needs while leveraging the existing functionality provided by the parent class. It simplifies the process of adding new features or variations to your codebase.

**Q.8 Whats Encapsulation and Purpose of it ?**

Encapsulation is a fundamental concept in object-oriented programming that combines data and the methods or functions that operate on that data into a single unit called an object. It involves bundling data and related functionality together, while hiding the internal implementation details from the outside world.

The purpose of encapsulation is to provide data protection, improve code maintainability, and promote modular and reusable code. Here are the key purposes of encapsulation:

Data Protection: Encapsulation ensures that the internal state or data of an object is protected from direct external access. By making the data private or providing controlled access through methods (getters and setters), encapsulation prevents unauthorized modification or manipulation of the object's data. It establishes data integrity and reduces the risk of unintended side effects or inconsistent states.

Information Hiding: Encapsulation hides the internal implementation details of an object. It allows you to expose only the necessary information or functionality through a well-defined interface, while keeping the inner workings hidden. This abstraction protects the complexity and intricacies of the implementation, making it easier to understand and use the object.

Code Maintainability: Encapsulation improves code maintainability by providing a clear separation between the interface (public methods) and implementation (private methods and data) of an object. Modifying the internal implementation does not affect the code that uses the object, as long as the interface remains consistent. It allows for easier updates, bug fixes, and enhancements, as changes can be localized to the encapsulated object.

**Q.9 Explain Class in JavaScript?**

**In JavaScript, a class is a blueprint or a template for creating objects with shared properties and methods. Introduced in ECMAScript 2015 (ES6), the class syntax in JavaScript provides a more structured and familiar way to define and create objects in an object-oriented programming (OOP) style.**

**The class syntax in JavaScript is primarily syntactic sugar over the existing prototype-based inheritance model. It provides a cleaner and more intuitive syntax for creating objects and defining their behavior. Here's an example of how to define a class in JavaScript:**

| **class MyClass {  constructor(param1, param2) {  this.property1 = param1;  this.property2 = param2;  }   method1() {  // Method code  }   method2() {  // Method code  } }  const myObject = new MyClass('value1', 'value2');** |
| --- |

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

The super keyword in JavaScript is used to call and access the parent class's properties and methods from within a subclass. It is primarily used in the context of inheritance to invoke the superclass's constructor, methods, and access its properties. The super keyword can be used in two main ways:

Calling the Superclass Constructor: The super() function is used within the constructor of a subclass to invoke the constructor of its superclass. It is essential when the subclass needs to initialize the inherited properties from the superclass.

| **class Parent {  constructor(name) {  this.name = name;  } }  class Child extends Parent {  constructor(name, age) {  super(name); // Calling the superclass constructor  this.age = age;  } }  const child = new Child('John', 25); console.log(child.name); // Outputs: "John" console.log(child.age); // Outputs: 25** |
| --- |

The super keyword ensures that the context is correctly set to the superclass, allowing you to access its properties and methods. It is used to establish and maintain the inheritance chain between classes, providing a way to access and extend the behavior of the superclass within the subclass.