**1 ans:** A constructor is a special method or function within a class that initializes an object when it is created. It sets the initial state of the object, performs any necessary setup tasks, and ensures that the object is ready for use. Constructors can have parameters, allowing you to pass initial values, and they can contain initialization logic for member variables.

**class Person {**

**constructor(name, age) {**

**this.name = name;**

**this.age = age;**

**}**

**}**

**const person1 = new Person("John Doe", 25);**

**console.log(person1.name);**

**console.log(person1.age);**

**2 ans:** The this keyword is a special keyword in JavaScript that refers to the current object or context in which a function or method is being executed. It allows you to access and manipulate the properties and methods of the object within its scope.

**const person = {**

**name: "John",**

**age: 25,**

**sayHello() {**

**console.log(`Hello, my name is ${this.name} and I'm ${this.age} years old.`);**

**}**

**};**

**person.sayHello(); // Output: Hello, my name is John and I'm 25 years old.**

**3 ans:** The `call`, `apply`, and `bind` are methods in JavaScript that allow you to control the value of `this` when invoking functions. They differ in how they are used and how they handle arguments.

Here's a brief explanation of each method and their differences:

1. `call`: The `call` method is used to invoke a function and explicitly specify the value of `this`. It accepts the object to be used as `this` as the first argument, followed by the function arguments passed individually.

Example:

**```javascript**

**const person = {**

**name: "John",**

**sayHello: function() {**

**console.log(`Hello, my name is ${this.name}.`);**

**}**

**};**

**const anotherPerson = {**

**name: "Jane"**

**};**

**person.sayHello.call(anotherPerson); // Output: Hello, my name is Jane.**

**```**

2. `apply`: The `apply` method is similar to `call`, but it accepts the object to be used as `this` as the first argument, followed by an array-like or iterable object that contains the function arguments.

Example:

**```javascript**

**const person = {**

**name: "John",**

**sayHello: function() {**

**console.log(`Hello, my name is ${this.name}.`);**

**}**

**};**

**const anotherPerson = {**

**name: "Jane"**

**};**

**person.sayHello.apply(anotherPerson); // Output: Hello, my name is Jane.**

**```**

3. `bind`: The `bind` method creates a new function that is bound to a specific `this` value. It returns a new function with the specified `this` value, allowing you to call it later. Unlike `call` and `apply`, `bind` does not immediately invoke the function.

Example:

**```javascript**

**const person = {**

**name: "John",**

**sayHello: function() {**

**console.log(`Hello, my name is ${this.name}.`);**

**}**

**};**

**const anotherPerson = {**

**name: "Jane"**

**};**

**const boundFunction = person.sayHello.bind(anotherPerson);**

**boundFunction(); // Output: Hello, my name is Jane.**

**```**

The main difference between `call` and `apply` lies in how function arguments are passed: `call` accepts arguments individually, while `apply` takes them in an array-like object. On the other hand, `bind` returns a new function with the bound `this` value, without immediately invoking it.

In summary, `call`, `apply`, and `bind` provide different ways to control the `this` value when invoking functions, allowing you to specify the context in which the function is executed.

**4 ans:** OOPS (Object-Oriented Programming) is a programming paradigm that focuses on organizing code into objects, which are instances of classes. It emphasizes the concepts of encapsulation, inheritance, and polymorphism to structure and model code.

In short, OOPS can be summarized as follows:

1. Objects: OOPS revolves around objects, which are instances of classes. Objects encapsulate data (properties) and behavior (methods) into a single unit.

2. Classes: Classes define the blueprint or template for creating objects. They specify the properties and methods that objects of that class will have.

3. Encapsulation: Encapsulation is the concept of bundling data and methods together within an object, hiding the internal implementation details. It promotes data integrity and provides abstraction.

4. Inheritance: Inheritance allows objects or classes to inherit properties and methods from other objects or classes. It enables code reuse and supports the creation of hierarchical relationships.

5. Polymorphism: Polymorphism allows objects or classes to exhibit different behaviors while sharing a common interface. It allows for flexible and extensible code by providing multiple implementations for the same method.

6. Abstraction: Abstraction focuses on defining the essential features and behavior of an object, hiding unnecessary details. It simplifies complex systems by breaking them into manageable and understandable components.

Overall, OOPS provides a structured approach to software development by modeling real-world entities as objects and defining their relationships and behaviors. It promotes code reusability, maintainability, and flexibility.

**5 ans:** Abstraction is a fundamental concept in object-oriented programming that involves simplifying complex systems by focusing on the essential features while hiding unnecessary details. Its purpose is to provide a high-level view or interface for interacting with objects or classes, allowing users to work with the system without being concerned about the internal implementation.

FEATURES:-

- Simplifies complex systems

- Focuses on essential features

- Hides unnecessary details

- Provides a clear and simplified interface

- Encapsulates complexity

- Enhances code reusability

- Promotes flexibility and extensibility

- Improves code organization and maintainability

**6 ans:** Polymorphism allows objects or classes to exhibit different behaviors while sharing a common interface. Its purpose is to provide flexibility, extensibility, and simplified usage of code. It promotes modular design and enables dynamic behavior determination at runtime.

FEATURES:-

- Objects or classes can have multiple forms or behaviors.

- Objects or classes can be used interchangeably.

- Different objects or classes can share a common interface.

- Method calls can be dynamically resolved at runtime.

- Enables code reuse and modularity.

- Supports extensibility without modifying existing code.

- Simplifies code usage and promotes abstraction.

**7 ans:** Inheritance is a fundamental concept in object-oriented programming where one class inherits properties and methods from another class. The purpose of inheritance is to facilitate code reuse, promote hierarchical relationships between classes, and allow for the creation of specialized classes based on existing ones.

In brief, the purpose of inheritance can be summarized as follows:

- Code Reuse: Inheritance enables the reuse of existing code by allowing a new class (derived or child class) to inherit properties and methods from an existing class (base or parent class). This promotes code efficiency and reduces redundancy.

- Hierarchical Relationships: Inheritance supports the creation of hierarchical relationships between classes, where a derived class inherits characteristics from its parent class. This allows for the modeling of real-world relationships and the organization of code into logical and manageable structures.

- Specialization: Inheritance allows for the creation of specialized classes that inherit the common features of a base class but also have their unique attributes and behaviors. This facilitates the customization and extension of functionality while maintaining a consistent interface.

- Method Overriding: Derived classes have the ability to override methods inherited from their parent class, allowing them to provide their own implementation of the method. This enables customization and flexibility in behavior based on the specific needs of the derived class.

- Polymorphism: Inheritance is closely related to polymorphism, where objects of different derived classes can be treated as objects of the base class. This promotes flexibility and allows for the creation of code that can work with multiple related objects interchangeably.

Overall, inheritance provides a mechanism for code reuse, hierarchical organization, specialization, and flexibility in object-oriented programming. It enhances modularity, promotes efficient use of resources, and simplifies code maintenance and extension.

**8 ans:** Encapsulation is a fundamental concept in object-oriented programming that involves bundling data and methods together within an object and hiding the internal implementation details. Its purpose is to promote data integrity, code organization, and provide abstraction and access control.

In brief, the purpose of encapsulation can be summarized as follows:

- Data Protection: Encapsulation ensures that the internal state or data of an object is protected and cannot be directly accessed or modified from outside the object.

- Code Organization: Encapsulation helps organize code by grouping related data and methods together within an object. It promotes modular design and improves code readability and maintainability.

- Abstraction: Encapsulation provides abstraction by exposing a public interface or API to interact with an object, while hiding the internal implementation details.

- Access Control: Encapsulation allows for controlling the visibility and access of object members (properties and methods) through access modifiers like public, private, and protected. It enforces encapsulation boundaries and enables proper information hiding.

- Code Flexibility: Encapsulation facilitates changes and modifications within an object without affecting other parts of the code. By encapsulating implementation details, it reduces dependencies and promotes code flexibility and modifiability.

Overall, encapsulation promotes data protection, code organization, abstraction, and access control in object-oriented programming. It enhances code maintainability, flexibility, and security by hiding implementation details and providing a well-defined public interface for interacting with objects.

**9 ans:** In JavaScript, a class is a template or blueprint for creating objects. It defines the structure, behavior, and initial state of objects of that class. It serves as a blueprint from which multiple objects can be created, sharing the same properties and methods.

In short, a class in JavaScript:

- Defines the blueprint for creating objects.

- Specifies the properties and methods that objects of that class will have.

- Acts as a template for creating multiple objects with similar characteristics.

- Allows for code reuse and promotes object-oriented programming principles.

**10 ans:** The `super` keyword in JavaScript is used to call functions or access properties on an object's parent or superclass. It allows a subclass or derived class to access and invoke methods or constructors from its parent class.

Here's a summary of what the `super` keyword does:

1. Accessing Parent's Properties: With `super`, you can access properties or methods defined in the parent class. It is commonly used in the constructor of a subclass to access the parent's properties before initializing its own.

2. Invoking Parent's Methods: By using `super`, you can invoke methods defined in the parent class from within the subclass. This allows the subclass to extend or override the behavior of the parent class's methods while still utilizing their functionality.

3. Calling the Parent's Constructor: When creating an instance of a subclass, `super` is used to call the constructor of the parent class. This ensures that the parent class's initialization logic is executed before initializing the subclass.

4. Passing Arguments: `super` can pass arguments to the parent class's constructor or methods. This enables you to provide specific values or modify behavior based on the subclass's needs.

In summary, the `super` keyword in JavaScript is used to access and call functions or properties in the parent class. It allows for code reuse, extension, and customization in class hierarchies by facilitating communication between parent and subclass.