Q.1 What's Constructor And Its Purpose?

In JavaScript, a constructor is a special function that is used to create and initialize objects created from a class (or constructor function). It is typically defined within a class and is responsible for setting the initial state and behavior of the objects.
The purpose of a constructor is to provide a blueprint or template for creating multiple objects of the same type with consistent initial values and behavior. It is called automatically when an object is instantiated using the new keyword.

```
class Person {
    constructor(name, age) {
      this.name = name;
      this.age = age;
    }

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

  const person1 = new Person('John Doe', 25);
  person1.greet(); // Output: Hello, my name is John Doe and I'm 25 years old.
```

In the example above, Person is a class with a constructor function defined within it.
The constructor takes the name and age parameters and assigns them to the corresponding properties (this.name and this.age) of the object being created. The greet method is also defined within the class to demonstrate additional behavior.

When the new keyword is used to instantiate a new object from the Person class (const person1 = new Person('John Doe', 25)),
the constructor function is automatically invoked, creating an object with the specified name and age values.

The purpose of a constructor can include:

1. **Initializing Object State**: The constructor allows you to initialize the properties and state of an object to specific values when it is created. This ensures that newly created objects start with the desired initial state.

2. **Setting Up Object Behavior:** Along with initializing properties, a constructor can also define methods and behaviors that are associated with the objects created from the class. These methods can be called on the objects to perform certain actions or computations.

3. **Encapsulation:** Constructors can encapsulate the logic for creating objects of a specific type. They provide a consistent way to create instances of a class, ensuring that the objects have the necessary properties and behaviors.

4. **Inheritance:** Constructors play a crucial role in inheritance hierarchies. Subclasses can extend the behavior of the base class by invoking the constructor of the superclass and adding their specific initialization logic.

Constructors are essential in object-oriented programming to ensure that objects are created with the desired initial state and behavior. They help establish consistency and provide a con

Q.2 Explain This Keyword and Its Purpose?

In JavaScript, the this keyword is a special keyword that refers to the current execution context, typically the object that the function is invoked upon. The value of this is dynamically determined based on how a function is called or how an object method is accessed.

The purpose of the this keyword is to provide a reference to the context in which a function is executed, allowing access to the object's properties and methods within the function. It allows functions to be reusable and flexible, as they can operate on different objects dynamically based on the execution context.

```
const person = {
    name: 'Rajan Prajapati',
    greet: function() {
      console.log(`Hello, my name is ${this.name}.`);
    }
  };

  person.greet(); //

  function sayHello() {
    console.log(`Hello, ${this.name}.`);
  }

  const name = 'Rajan Prajapati';
  sayHello(); // Output: Hello, undefined.
```

Q.3 What's Call Apply Bind Method & Difference Between them

In JavaScript, call, apply, and bind are methods that allow you to manipulate the this value of a function and control how the function is invoked. They provide a way to explicitly set the this value and pass arguments to the function.

1. call Method:

The call method is used to invoke a function with a specified this value and individual arguments passed in order.

```
functionName.call(thisValue, arg1, arg2, ...);
function sayHello() {
  console.log(`Hello, ${this.name}.`);
}

const person = { name: 'John Doe' };
sayHello.call(person); // Output: Hello, John Doe.
```

2. apply Method:

The apply method is similar to call, but it takes an array-like object or an actual array of arguments instead of individual arguments.

```
functionName.apply(thisValue, [arg1, arg2, ...]);
function sayHello() {
  console.log(`Hello, ${this.name}.`);
}

const person = { name: 'John Doe' };
sayHello.apply(person); // Output: Hello, John Doe.
```

3. bind Method:

The bind method creates a new function with a specified this value and, optionally, pre-filled arguments. It doesn't immediately invoke the function but returns a new function that can be invoked later.

```
const newFunction = functionName.bind(thisValue, arg1, arg2, ...);
function sayHello() {
  console.log(`Hello, ${this.name}.`);
}

const person = { name: 'John Doe' };
const greetPerson = sayHello.bind(person);
greetPerson(); // Output: Hello, John Doe.
```

Q.4 Explain OOPS ?

OOPS (Object-Oriented Programming) is a programming paradigm that organizes code around objects, which are instances of classes or blueprints for creating objects. It aims to provide a structured approach to design and build software systems by emphasizing the concepts of encapsulation, inheritance, polymorphism, and abstraction.

Q.5 Whats Abstraction and Its Purpose?

Abstraction focuses on representing the essential features of an object, hiding unnecessary implementation details. It allows for the creation of abstract classes and interfaces that define the structure and contract for implementing classes. Abstraction provides a high-level view, enabling code modularization and separation of concerns.

Q.6 Whats Polymorphism and Purpose of it?

Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables objects to be used interchangeably by sharing a common interface or base class. Polymorphism allows methods to be overridden in subclasses, providing different implementations while maintaining a consistent interface.

Q.7 Whats Inheritance and Purpose of it?

Inheritance enables the creation of new classes (derived classes or subclasses) by inheriting properties and behaviors from existing classes (base classes or superclasses). It promotes code reuse and allows for the creation of hierarchical relationships between classes. Subclasses can inherit and extend the functionality of their parent classes, adding or modifying behavior as needed.

Q.8 Whats Encapsulation and Purpose of it ?

Encapsulation refers to the bundling of data and related methods within a class. It provides data hiding and access control by defining access levels (public, private, protected) to the class members. Encapsulation ensures that data is accessed and modified through controlled interfaces, improving code maintainability and reusability.

Q.9 Explain Class in JavaScript?

Classes are templates or blueprints that define the structure and behavior of objects. They encapsulate data (attributes/properties) and methods (functions) that operate on that data.

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

In JavaScript, the super keyword is used to call functions or access properties on an object's parent class.
It is primarily used within subclasses (derived classes) to invoke the constructor or methods of the parent class

1. Accessing the Parent Class's Constructor:
When defining a constructor in a subclass, the super keyword is used to call the constructor of the parent class, allowing the subclass to inherit and initialize the properties defined in the parent class.
Example:

```
class Parent {
    constructor(name) {
      this.name = name;
    }
  }

  class Child extends Parent {
    constructor(name, age) {
      super(name); // Call the constructor of the parent class
      this.age = age;
    }
  }
```

```
const child1 = new Child('Rajan', 25);
console.log(child1.name); // Output: Rajan
console.log(child1.age); // Output: 25
```

2. Invoking Methods from the Parent Class:
The super keyword can also be used to call methods defined in the parent class from within the subclass.
This allows the subclass to inherit and extend the behavior of the parent class's methods.
Example:

```
class Parent {
    greet() {
      console.log('Hello from the parent class!');
    }
  }

  class Child extends Parent {
    greet() {
      super.greet(); // Call the greet() method of the parent class
      console.log('Hello from the child class!');
    }
  }

  const child1 = new Child();
  child1.greet();
```