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

Ans:

1. A constructor is a special method used to create and initialize objects in JavaScript.

2. It is invoked using the `new` keyword followed by the constructor function name.

3. Constructors are functions that define the blueprint for creating objects with specific properties and behaviors.

4. They allocate memory for the object and set its initial state.

5. They allow for the creation of multiple instances of objects with similar characteristics.

6. Constructors can be defined within a class (ES6 syntax) or as standalone functions (pre-ES6 syntax).

The purpose of a constructor in JavaScript are:-

1. Object Creation: Constructors allow you to create new instances of objects based on a predefined blueprint or class. They provide a convenient way to create multiple objects with similar characteristics and behaviors.

2. Property Initialization: Constructors enable you to set initial values for the properties of the newly created object. You can pass arguments to the constructor to initialize object properties dynamically.

3. Encapsulation: Constructors encapsulate the logic for object creation and initialization within a single function. This helps organize and structure your code by keeping related object creation tasks in one place.

## Q.2 Explain This Keyword and Its Purpose.

Ans:

1. Dynamic Reference: The `this` keyword in JavaScript is a special reference that represents the current execution context or the object on which a function is being invoked.

2. Object Context: When a function is called as a method of an object, `this` refers to the object itself. It allows the function to access and modify the object's properties and methods.

3. Global Context: Outside of any function or object, `this` refers to the global object. In a browser environment, it is typically `window`, and in Node.js, it is `global`.

4. Explicit Binding: JavaScript provides methods like `call()`, `apply()`, and `bind()` that allow you to explicitly bind `this` to a specific object. They provide control over the context in which a function is executed.

5. Constructor Context: When a function is used as a constructor with the `new` keyword, `this` refers to the newly created object. It allows the constructor to set properties and behaviors specific to that instance.

The purpose of the `this` keyword in JavaScript are:

1. Accessing Object Properties: The main purpose of the `this` keyword is to provide a convenient way to access and modify properties and methods within an object. It allows functions to refer to the object they belong to, enabling interaction with its specific data.

2. Context Flexibility: The `this` keyword allows functions to be reused across different objects, providing flexibility in how they operate. By dynamically referencing the current object, functions can adapt their behavior based on the specific context in which they are invoked.

3. Method Invocation: When functions are used as methods of an object, `this` is used to refer to the object itself. This allows methods to access and manipulate the object's properties, providing a way to encapsulate related functionality within the object.

4. Constructor Functions: When used in a constructor function with the `new` keyword, `this` refers to the newly created object. Constructors use `this` to set initial property values specific to each instance of an object, enabling the creation of multiple objects with shared behavior.

5. Dynamic Context Binding: JavaScript provides methods like `call()`, `apply()`, and `bind()` that allow explicit binding of `this` to a specific object. This purpose enables fine-grained control over the context in which a function is executed, facilitating code reusability and flexibility.

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

Ans:

The `call`, `apply`, and `bind` methods in JavaScript are used to explicitly set the value of `this` in a function and invoke it in a specific context.

1. `call` method:

   - The `call` method is a function method that allows you to invoke a function with a specified `this` value and pass arguments individually.

   - Syntax: `functionName.call(thisValue, arg1, arg2, ...)`

   - The `thisValue` parameter represents the object that will be bound to `this` inside the function.
   - Arguments are passed individually after the `thisValue` parameter.

2. `apply` method:

   - The `apply` method is similar to `call` but allows you to invoke a function with a specified `this` value and pass arguments as an array or an array-like object.

   - Syntax: `functionName.apply(thisValue, [arg1, arg2, ...])`

   - The `thisValue` parameter represents the object that will be bound to `this` inside the function.
   - Arguments are passed as an array or an array-like object after the `thisValue` parameter.

3. `bind` method:

   - The `bind` method returns a new function that has its `this` value explicitly set to a specified object and allows pre-filling of arguments.

   - Syntax: `functionName.bind(thisValue, arg1, arg2, ...)`

- The `thisValue` parameter represents the object that will be bound to `this` inside the returned function.
- Additional arguments can be provided after the `thisValue` parameter to pre-fill them in the returned function.
- The `bind` method does not invoke the function immediately; it returns a new function that can be called later.

Differences:

- The main difference between `call` and `apply` lies in how arguments are passed to the function. `call` requires arguments to be passed individually, while `apply` takes arguments as an array or an array-like object.

- Both `call` and `apply` invoke the function immediately.

- The `bind` method, on the other hand, returns a new function with the specified `this` value and pre-filled arguments, but it doesn't invoke the function right away.

- `bind` is commonly used when you want to create a new function with a specific `this` value and reuse it later.

## Q.4 Explain OOPS.

Ans:

OOPS stands for Object-Oriented Programming Paradigm.
It is a programming paradigm that organizes code into objects, which are instances of classes or prototypes. OOPS focuses on creating reusable, modular, and maintainable code by emphasizing concepts such as encapsulation, inheritance, polymorphism, and abstraction.

1. Classes and Objects:

   - Classes are blueprints or templates that define the structure and behavior of objects.

   - Objects are instances of classes that contain data (properties) and behavior (methods).

2. Encapsulation:

   - Encapsulation refers to the bundling of data and methods within an object, hiding internal details and providing an interface to interact with the object.

   - It helps in achieving data abstraction and protection, preventing direct access to internal implementation.

3. Inheritance:

- Inheritance allows classes to inherit properties and methods from other classes, creating a hierarchical relationship.

- It promotes code reuse, as common attributes and behaviors can be defined in a base (parent) class and inherited by derived (child) classes.

4. Polymorphism:

- Polymorphism allows objects of different classes to be treated as objects of a common superclass.

- It enables methods to be overridden in derived classes, providing different implementations while maintaining a consistent interface.

5. Abstraction:

- Abstraction focuses on providing simplified and essential representations of complex systems.

- It involves hiding unnecessary details and exposing only relevant information and functionalities to the users.

## Q.5 Whats Abstraction and Its Purpose?

Ans/;

Abstraction, in the context of object-oriented programming (OOP), refers to the concept of simplifying complex systems by focusing on the essential aspects and hiding unnecessary details.

It provides a high-level view or representation of an object or system, emphasizing what it does rather than how it does it.

The purpose of abstraction is to:

1. Simplify Complexity: Abstraction helps manage the complexity of software systems by breaking them down into manageable and understandable components. It allows developers to focus on the essential features and behaviors of an object or system without getting bogged down by intricate implementation details.

2. Hide Implementation Details: By hiding internal implementation details, abstraction provides a clear separation between the interface of an object and its internal workings. This helps in reducing dependencies and making the code more modular and maintainable.

3. Provide a Consistent Interface: Abstraction defines a clear and consistent interface for interacting with an object. It presents a set of methods or functions that can be used to perform specific actions on the object, while the underlying implementation remains hidden.

4. Encapsulate Complexity: Abstraction allows complex operations to be encapsulated within objects, providing a higher level of abstraction. It allows developers to interact with objects using simple and intuitive methods without needing to understand the intricate inner workings.

5. Enhance Code Reusability: By abstracting away specific implementation details, reusable components can be created. These components can be easily integrated into different parts of an application or reused in future projects, reducing redundant code and promoting code reusability.

6. Facilitate System Design and Maintenance: Abstraction assists in designing and maintaining software systems by providing a clear and concise representation of objects and their interactions. It helps in managing complexity, identifying relationships between objects, and making the system more modular, scalable, and adaptable.

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

Ans:
Polymorphism, in object-oriented programming (OOP), is the ability of an object to take on different forms or behaviors depending on the context in which it is used.

It allows objects of different classes to be treated as objects of a common superclass or interface.

The purpose of polymorphism is to:

1. Code Flexibility: Polymorphism enables the writing of code that can work with objects of different types without the need for explicit type checking and casting. It provides flexibility in handling different object types interchangeably.

2. Code Reusability: Polymorphism promotes code reuse by allowing shared behavior to be defined in a common superclass or interface. Objects of various subclasses can then inherit or implement that behavior, reducing redundancy and improving code organization.

3. Method Overriding: Polymorphism allows derived classes to override methods of the base class. This means that objects of different classes can have different implementations of the same method. It enables specialized behavior in each subclass while maintaining a consistent interface.

4. Simplified Interface: Polymorphism provides a simplified and unified interface to interact with objects. It allows code to call methods on objects without needing to know the specific class of the object. This abstraction enhances code readability and maintainability.

5. Run-Time Binding: Polymorphism allows dynamic binding of methods at runtime. The appropriate method implementation is determined based on the actual type of the object, allowing for late binding and flexible behavior based on the context of object usage.

6. Method Overloading: Polymorphism also encompasses method overloading, where multiple methods with the same name but different parameter lists can exist within a class. It allows for different variations of the same method to handle different input types or numbers of parameters.

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

Ans:
Inheritance, in object-oriented programming (OOP), is a mechanism that allows one class (called the child or derived class) to inherit properties and methods from another class (called the parent or base class).

The purpose of inheritance is to:

1. Code Reusability: Inheritance promotes code reuse by allowing derived classes to inherit and reuse the properties and methods of the base class. Instead of writing the same code multiple times, inheritance enables the sharing of common functionality among related classes.

2. Hierarchical Organization: Inheritance provides a hierarchical organization of classes. It allows for the creation of a class hierarchy, where classes at higher levels represent more general concepts, while classes at lower levels represent more specific or specialized concepts.

3. Overriding and Extending Functionality: Derived classes have the ability to override the inherited methods of the base class. This allows them to provide their own implementation of those methods to suit their specific requirements. Inheritance also allows derived classes to add new methods or properties, thus extending the functionality inherited from the base class.

4. Polymorphism: Inheritance plays a vital role in achieving polymorphism, where objects of different classes can be treated as objects of a common superclass. Polymorphism allows for a unified interface, enabling objects of different derived classes to be used interchangeably through a shared base class reference.

5. Code Organization and Maintainability: Inheritance helps in organizing code by structuring related classes into a logical hierarchy. It improves code maintainability as changes made in the base class automatically propagate to the derived classes. This reduces the need for modifying code in multiple places, enhancing code consistency and reducing potential errors.

6. Specialization and Generalization: Inheritance allows for the specialization of classes, where derived classes can add additional features or modify existing ones to represent more specific types. It also enables generalization, where common attributes and behaviors are defined in the base class, and specific characteristics are introduced in derived classes.

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

Ans:

Encapsulation, in object-oriented programming (OOP), is a concept that combines data and the methods that operate on that data within a single unit called an object.

It involves bundling data (attributes or properties) and methods (functions or behaviors) together and controlling access to them.

The purpose of encapsulation is to:

1. Data Protection and Security: Encapsulation ensures that the internal state of an object is hidden from the outside world. By encapsulating data within an object, access to that data is controlled through methods, allowing for data protection and preventing unauthorized modification.

2. Modularity and Code Organization: Encapsulation helps in creating modular and organized code. By encapsulating related data and methods within an object, code becomes more manageable, allowing for better readability, reusability, and maintainability. It also reduces dependencies between different parts of the codebase.

3. Abstraction and Information Hiding: Encapsulation supports abstraction by exposing only essential information to the external world while hiding unnecessary implementation details. It allows objects to provide a clear and simplified interface, abstracting away complex internal workings and focusing on what the object does rather than how it does it.

4. Code Flexibility and Versioning: Encapsulation facilitates code flexibility and versioning. By encapsulating data and methods within an object, changes can be made to the internal implementation without affecting the external interface. This allows for easier updates, bug fixes, and enhancements without impacting other parts of the codebase.

5. Encourages Consistent State: Encapsulation promotes the maintenance of a consistent state within an object. By encapsulating data and providing controlled access through methods, the object can enforce business rules and validations to ensure data integrity. This helps in preventing invalid or inconsistent states of the object.

6. Encourages Collaboration and Teamwork: Encapsulation promotes collaboration among developers in a team. By defining clear interfaces for objects, encapsulation allows different team members to work on different components independently, as long as the interfaces are respected. This enables concurrent development and reduces the chances of code conflicts.

## Q.9 Explain Class in JavaScript?

Ans:

- A class in JavaScript is a blueprint or template for creating objects.

- It is declared using the `class` keyword followed by the class name.

- Example:

```
  class Person {
```

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

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

- The `constructor` method is a special method that is called when an object is instantiated from the class. It is used to initialize object properties.

- Instance methods are defined within the class and are accessible on objects created from the class. They define the behavior of individual objects.

- Static methods are defined on the class itself and are not accessible on individual objects. They are called directly on the class and are useful for utility functions or operations that don't depend on object-specific data.

- Classes in JavaScript support inheritance using the `extends` keyword. It allows one class to inherit properties and methods from another class.

- Objects are created from classes using the `new` keyword followed by the class name and any required arguments.

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

Ans:

In JavaScript, the `super` keyword is used to call and access the parent class's constructor and methods from within a subclass.

It allows for proper inheritance and enables the subclass to extend or override behavior inherited from the parent class.

The `super` keyword does:

1. Accessing the Parent Class's Constructor:

   - In a subclass constructor, the `super` keyword is used to call the constructor of the parent class. It initializes the inherited properties and sets up the object based on the parent class's initialization logic.

2. Accessing the Parent Class's Methods:

   - The `super` keyword can also be used to access and invoke methods from the parent class within the subclass. It allows the subclass to extend or override the inherited behavior while still utilizing the parent class's functionality.

3. Passing Arguments to the Parent Class's Constructor:

   - When calling the parent class's constructor using `super`, arguments can be passed to it if the parent class expects them. This allows the subclass to provide additional initialization values or override parent class properties.