JS - Object-Oriented Programming (OOP)

1. What is OOP?

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data (properties) and code (methods). In JavaScript, we use objects and classes to implement OOP principles like encapsulation, inheritance, and polymorphism.

2. Creating an Object

In JavaScript, an object can be created using the object literal syntax or the `new Object()` syntax:

```
// Creating an object using object literal
let car = {
    make: "Toyota",
    model: "Camry",
    year: 2020,
    drive: function() {
        console.log("The car is driving.");
    }
};

// Accessing object properties
console.log(car.make); // Output: Toyota
car.drive();           // Output: The car is driving.
```

3. Constructor Functions

Constructor functions are used to create multiple instances of similar objects. They are regular functions but written with a capital letter to indicate that they are used for creating objects.

```
// Constructor function
function Car(make, model, year) {
    this.make = make;
    this.model = model;
    this.year = year;
    this.drive = function() {
        console.log("The car is driving.");
    };
}

// Creating an instance of the Car object
let myCar = new Car("Honda", "Civic", 2022);
console.log(myCar.make); // Output: Honda
myCar.drive();           // Output: The car is driving.
```

4. Classes (ES6)

ES6 introduced the `class` syntax, which provides a more convenient and clear way to define constructor functions and methods.

```
// Defining a class
class Car {
    constructor(make, model, year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }

    drive() {
        console.log("The car is driving.");
    }
}

// Creating an instance of the Car class
let myCar = new Car("Ford", "Mustang", 2023);
console.log(myCar.make); // Output: Ford
```

```
myCar.drive();           // Output: The car is driving.
```

5. Inheritance

Inheritance allows one class to inherit properties and methods from another class. In JavaScript, we use the `extends` keyword to create a subclass that inherits from a superclass:

```
// Base class
class Vehicle {
    constructor(make, model) {
        this.make = make;
        this.model = model;
    }

    start() {
        console.log("The vehicle is starting.");
    }
}

// Subclass inherits from Vehicle
class Car extends Vehicle {
    constructor(make, model, year) {
        super(make, model); // Call the parent class constructor
        this.year = year;
    }

    drive() {
        console.log("The car is driving.");
    }
}

// Creating an instance of the Car class
let myCar = new Car("Chevrolet", "Impala", 2021);
console.log(myCar.make);  // Output: Chevrolet
myCar.start();            // Output: The vehicle is starting.
myCar.drive();            // Output: The car is driving.
```

6. Polymorphism

Polymorphism allows objects of different classes to be treated as objects of a common superclass. The specific method to be invoked is determined at runtime.

```
// Base class
class Animal {
    speak() {
        console.log("The animal makes a sound.");
    }
}

// Subclass
class Dog extends Animal {
    speak() {
        console.log("The dog barks.");
    }
}

// Another subclass
class Cat extends Animal {
    speak() {
        console.log("The cat meows.");
    }
}

// Creating instances
let dog = new Dog();
let cat = new Cat();

dog.speak(); // Output: The dog barks.
cat.speak(); // Output: The cat meows.
```

7. Encapsulation

Encapsulation is the bundling of data and methods that operate on that data into a single unit, often restricting direct access to some of the object's components. In JavaScript, we can use getter and setter methods to control access to properties:

```
// Class with encapsulation
class Person {
    constructor(name, age) {
        this._name = name;
        this._age = age;
    }

    // Getter for name
    get name() {
        return this._name;
    }

    // Setter for name
    set name(newName) {
        this._name = newName;
    }

    // Getter for age
    get age() {
        return this._age;
    }

    // Setter for age
    set age(newAge) {
        if (newAge < 0) {
            console.log("Age cannot be negative.");
        } else {
            this._age = newAge;
        }
    }
}

// Creating an instance
let person = new Person("Alice", 30);
console.log(person.name); // Output: Alice
person.age = -5;          // Output: Age cannot be negative.
person.age = 31;          // Sets age to 31
console.log(person.age);  // Output: 31
```

8. Summary of OOP Concepts in JavaScript

Concept	Description
Object	A collection of properties and methods.
Constructor	A function used to create instances of objects.
Class	A blueprint for creating objects, introduced in ES6.
Inheritance	Mechanism to allow one class to inherit properties/methods from another.
Polymorphism	The ability for different objects to be treated as instances of the same class.
Encapsulation	The concept of bundling data and methods, with restricted access to certain components.