JavaScript classes

JavaScript classes are a syntax feature introduced in ECMAScript 2015 (ES6) that provide a more convenient and familiar way to define object-oriented programming (OOP) constructs in JavaScript. Classes allow you to create blueprints for objects with properties and methods, facilitating code organization, inheritance, and code reuse.

Here's a basic example of how to define and use classes in JavaScript:

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

    speak() {
        console.log(`${this.name} makes a sound.`);
    }
}

const dog = new Animal('Dog');
dog.speak(); // Output: Dog makes a sound.
```

In this example:

- We define a class named Animal using the class keyword.
- Inside the class, we define a constructor method using the constructor keyword. This method is automatically called when a new instance of the class is created. In this case, it initializes the name property of the instance.
- We define a speak method within the class, which logs a message to the console.
- We create an instance of the Animal class using the new keyword and pass 'Dog' as an argument to the constructor.
- We call the speak method on the dog object, resulting in the output "Dog makes a sound.".

JavaScript classes also support inheritance, allowing you to create subclasses that inherit properties and methods from parent classes:

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

    speak() {
        console.log(`${this.name} makes a sound.`);
    }
}

const dog = new Animal('Dog');
dog.speak(); // Output: Dog makes a sound.

class Dog extends Animal {
    constructor(name, breed) {
        super(name);
        this.breed = breed;
    }

    speak() {
        console.log(`${this.name} barks.`);
    }
}

const myDog = new Dog('Buddy', 'Labrador');
myDog.speak(); // Output: Buddy barks.
```

In this example, we define a subclass Dog that extends the Animal class. We use the super keyword in the constructor of the subclass to call the constructor of the superclass (Animal). We also override the speak method of the superclass in the subclass to provide a specialized implementation for dogs.

JavaScript classes provide a more intuitive and structured way to work with objects and inheritance in JavaScript, making code more organized, readable, and maintainable.

Basic class with methods:

```
class Rectangle {
    constructor(width, height) {
        this.width = width;
        this.height = height;
    }

    calculateArea() {
        return this.width * this.height;
    }
}

const rectangle = new Rectangle(5, 10);
console.log(rectangle.calculateArea()); // Output: 50
```

Class with static method:

```
class MathUtils {
    static add(a, b) {
        return a + b;
    }

    static subtract(a, b) {
        return a - b;
    }
}

console.log(MathUtils.add(5, 3)); // Output: 8
console.log(MathUtils.subtract(10, 7)); // Output: 3
```

Class with getter and setter methods:

```
class Circle {
    constructor(radius) {
        this.radius = radius;
    }

    get diameter() {
        return this.radius * 2;
    }

    set diameter(diameter) {
        this.radius = diameter / 2;
    }

    get area() {
        return Math.PI * this.radius ** 2;
    }
}

const circle = new Circle(5);
console.log(circle.diameter); // Output: 10
console.log(circle.area); // Output: 78.53981633974483

circle.diameter = 12;
console.log(circle.radius); // Output: 6
console.log(circle.area); // Output: 113.09733552923255
```

Inheritance:

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

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

class Student extends Person {
    constructor(name, grade) {
        super(name);
        this.grade = grade;
    }

    study() {
        console.log(`${this.name} is studying.`);
    }
}

const student = new Student('Alice', 10);
student.introduce(); // Output: Hello, my name is
Alice.
student.study(); // Output: Alice is studying.
```