# Constructor

**In JavaScript, a constructor is a special method that is automatically called when a new instance of a class is created using the new keyword. The constructor method is used to initialize the newly created object's properties or perform any setup that is necessary before the object is ready for use.**

**Here's a basic example of a constructor in JavaScript:**

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

const john = new Person('John', 30);
console.log(john.name); // Output: John
console.log(john.age); // Output: 30
```

**In this example:**

- **We define a class Person with a constructor method.**
- **The constructor method takes two parameters (name and age) and assigns them to the object's properties (this.name and this.age).**
- **We create a new instance of the Person class using the new keyword and pass 'John' and 30 as arguments to the constructor.**

- **The john object is created with the specified name and age, and we can access these properties using dot notation (john.name and john.age).**

**Constructors can perform various tasks such as:**

- **Initializing instance variables.**
- **Setting up default property values.**
- **Initializing object state.**
- **Executing any setup logic required before the object can be used.**

**Constructors are not required in JavaScript classes, but they are commonly used to set initial state or perform initialization tasks when creating new objects. If a class does not have a constructor, JavaScript will provide a default constructor that initializes the object with no properties or behavior.**

**It's important to note that a class can have only one constructor method. If multiple constructors are defined in the same class, it will result in a syntax error.**

## Basic Constructor Example:

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

const john = new Person('John', 30);
console.log(john.name); // Output: John
console.log(john.age); // Output: 30
```

## *Explanation:*

***In this example, we define a Person class with a constructor method that takes name and age parameters. When a new Person object is created with new Person('John', 30), the constructor sets the name and age properties of the object to 'John' and 30, respectively.***

## Default Constructor:

```
class Car {
    constructor() {
        this.brand = 'Toyota';
        this.model = 'Camry';
        this.year = 2020;
    }
}

const myCar = new Car();
console.log(myCar.brand); // Output: Toyota
console.log(myCar.model); // Output: Camry
console.log(myCar.year); // Output: 2020
```

***Explanation:***

***In this example, we define a Car class with a constructor method that initializes default values for brand, model, and year. When a new Car object is created without passing any arguments, the constructor sets default values for these properties.***

## Complex Constructor with Initialization Logic:

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

const rectangle1 = new Rectangle(5, 10);
console.log(rectangle1.area); // Output: 50
```

```
console.log(rectangle1.isSquare); // Output: false

const rectangle2 = new Rectangle(6, 6);
console.log(rectangle2.area); // Output: 36
console.log(rectangle2.isSquare); // Output: true
```

***Explanation:***

***In this example, we define a Rectangle class with a constructor method that takes width and height parameters. The constructor calculates the area of the rectangle (width * height) and sets the isSquare property based on whether the rectangle is a square (i.e., width === height).***

***Constructors in JavaScript classes provide a way to initialize object properties and perform setup logic when creating new instances of a class. They are essential for setting up the initial state of objects and ensuring that they are properly initialized before use.***

## Constructor with Initialization Logic and Error Handling:

```
class Circle {
    constructor(radius) {
        if (typeof radius !== 'number' || radius <=
0) {
            throw new Error('Radius must be a
positive number');
        }
        this.radius = radius;
        this.area = Math.PI * radius ** 2;
    }
}

try {
    const invalidCircle = new Circle(-5); // This
will throw an error
} catch (error) {
    console.error(error.message); // Output: Radius
must be a positive number
}

const validCircle = new Circle(7);
console.log(validCircle.area.toFixed(2)); // Output:
153.94
```

***Explanation:***

***In this example, we define a Circle class with a constructor method that checks if the radius parameter is a positive number. If the radius is invalid (negative or not a number), it throws an error. Otherwise, it calculates the area of the circle (Math.PI * radius ** 2).***

## Constructor with Object Initialization:

```
class Product {
    constructor({ name, price }) {
        this.name = name;
        this.price = price;
    }
}

const laptop = new Product({ name: 'Laptop', price:
999 });
console.log(laptop.name); // Output: Laptop
console.log(laptop.price); // Output: 999
```

***Explanation:***

***In this example, we define a Product class with a constructor method that accepts an object containing name and price properties. This allows for object initialization using an object literal { name, price } during instantiation, providing a convenient way to initialize object properties.***

***Constructors in JavaScript provide flexibility in initializing objects, allowing for default values, initialization logic, error handling, and object initialization using different approaches. They are essential for creating instances of classes with properly initialized properties and behaviors.***