JavaScript OOPs (Object-Oriented Programming)

Understanding Object-Oriented Concepts in JavaScript

Introduction to OOPs

What is Object-Oriented Programming?

- **Definition**: Object-Oriented Programming (OOP) is a programming paradigm that uses objects to represent data and methods to operate on that data.
- **Key Concepts**: Encapsulation, Abstraction, Inheritance, Polymorphism.

Why Use OOP in JavaScript?

Benefits:

- **Code Reusability**: Reuse code through inheritance and classes.
- **Modularity**: Break down complex problems into smaller, manageable objects.
- **Ease of Maintenance**: Simplifies debugging and updates.
- **Real-World Modeling**: Objects in code can represent real-world entities.

Introduction to Constructors

What is a Constructor?

- **Definition**: A constructor is a special function in JavaScript used to create and initialize objects.
- **Purpose**: Constructors allow you to create multiple instances of an object with the same properties and methods.

Defining a Constructor Function

```
function ConstructorName(parameters) {
   this.property1 = value1;
   this.property2 = value2;
}
```

Example :

```
function Person(firstName, lastName, age) {
   this.firstName = firstName;
   this.lastName = lastName;
   this.age = age;
}
```

Creating Object Instances

```
var person1 = new Person("Pawan", "Maurya", 30);

var person2 = new Person("Manish", "Mishra", 25);
```

Objects and Classes

Definition: Objects are collections of properties, where each property has a key and a value.

Example : Objects

```
let person = {

    name: "Rahul",

    age: 30,

    greet: function() {

        console.log("Hello, my name is " + this.name);

    }

};

person.greet(); // Output: Hello, my name is Rahul
```

Introduction to Classes

Definition: Classes are templates for creating objects.

Example : Classes

```
class Person {    constructor(name, age) {

  this.name = name;   this.age = age;

  }

 greet() {    console.log("Hello, my name is " + this.name);    }

}

let rahul = new Person("Rahul", 30);

rahul.greet(); // Output: Hello, my name is Rahul
```

Key OOP Concepts in JavaScript

Encapsulation

- **Definition**: Encapsulation is the bundling of data and methods that operate on that data within one unit, typically a class.

Example : Encapsulation

```
Class BankAccount {    constructor(balance) {        this._balance = balance;    }
deposit(amount) {   this._balance += amount;    }
getBalance() {      return this._balance;     }
}
let account = new BankAccount(1000);
account.deposit(500);
console.log(account.getBalance()); // Output: 1500
```

Inheritance

Definition: Inheritance allows a class to inherit properties and methods from another class.

Example :

```
class Animal {    constructor(name) {        this.name = name;    }
    speak() {        console.log(this.name + " makes a sound.");     }
}
class Dog extends Animal {    speak() {        console.log(this.name + " barks.");    }
}
let dog = new Dog("Tommy");
dog.speak(); // Output: Tommy barks.
```

Polymorphism

Definition: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It also allows methods to be overridden.

Example : Polymorphism

```
class Shape {    area() {       console.log("Calculating area...");    } }

class Circle extends Shape {    area() { console.log("Area of Circle: " + Math.PI * r * r);    } }

class Rectangle extends Shape {  area() {  console.log("Area of Rectangle: " + l * w);   } }

let shape1 = new Circle(5);

let shape2 = new Rectangle(4, 6);


shape1.area(); // Output: Area of Circle: ...

shape2.area(); // Output: Area of Rectangle: ...
```

Abstraction

Definition: Abstraction means hiding complex implementation details and showing only the essential features of an object.

Example : Abstraction

```
class Car {
    start() {    console.log("Car started");    }
    stop() {    console.log("Car stopped");    }
}
let myCar = new Car(); myCar.start();  myCar.stop();
```