



Live Cohort

Day 92



1. OOP (Object-Oriented Programming)

◆ Definition

OOP is a programming paradigm based on the concept of "objects", which contain data (properties) and methods (functions) to operate on that data.

◆ Code (Example)

```
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    greet() {  
        console.log(`Hello, I'm ${this.name}`);  
    }  
}  
  
const p1 = new Person("Rahul", 25);  
p1.greet(); // Hello, I'm Rahul
```

◆ Use Case

OOP is used to build modular, scalable, and reusable code for real-world applications like web apps, games, and enterprise software.

◆ Interview Q&A

Q: What are the main concepts of OOP?

A: Encapsulation, Abstraction, Inheritance, Polymorphism.

2. Why Do We Need OOP?

◆ Definition

OOP helps in organizing code better, making it reusable, easy to maintain, and scalable for complex applications.

Code (Not applicable directly)

◆ Use Case

- Helps manage large-scale projects
- Encourages DRY (Don't Repeat Yourself) coding
- Simplifies debugging and testing

◆ Interview Q&A

Q: Why is OOP preferred over procedural programming?

A: OOP promotes modularity and reusability, while procedural programming is more linear and less flexible for large projects.

3. Class and Object

◆ Definition

- Class: Blueprint for creating objects.
- Object: An instance of a class.

◆ Code (Example)

```
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    greet() {  
        console.log(`Hello, I'm ${this.name}`);  
    }  
}  
  
const p1 = new Person("Rahul", 25);  
p1.greet();
```

◆ Use Case

- Use classes to define templates (like User, Product) and instantiate multiple consistent objects.

◆ Interview Q&A

Q: What is the difference between class and object?

A: A class is a template, while an object is a specific instance created from that template.

4. Literal Object

◆ Definition

An object created directly using curly braces {}, without a class or constructor function.

◆ Code (Example)

```
const car = {  
    brand: "Toyota",  
    start() {  
        console.log("Car started");  
    },  
};  
  
car.start(); // Car started
```

◆ Use Case

Useful for creating simple objects without needing reusable templates.

◆ Interview Q&A

Q: How is a literal object different from a class-based object?

A: Literal objects are quick and simple, but lack structure and reusability provided by classes.

5. Constructor & Constructor Function

◆ Definition

- **Constructor Function:** A function used to create and initialize objects (Pre-ES6).
- **ES6 Constructor:** Defined inside a class using the `constructor()` method.

◆ Code (Example)

```
function Student(name, roll) {  
    this.name = name;  
    this.roll = roll;  
    this.display = function () {  
        console.log(`Name: ${this.name}, Roll: ${this.roll}`);  
    };  
}  
  
const s1 = new Student("Ravi", 101);  
s1.display();
```

ES6 Class Construction

```
class Student {  
    constructor(name, roll) {  
        this.name = name;  
        this.roll = roll;  
    }  
  
    display() {  
        console.log(`Name: ${this.name}, Roll: ${this.roll}`);  
    }  
}
```

◆ Use Case

Constructors are used to initialize object properties when an instance is created.

◆ Interview Q&A

Q: What is the difference between a constructor function and a class constructor?

A: Literal objects are quick and simple, but lack structure and reusability provided by classes.

6. **this** Keyword

◆ Definition

Refers to the current object from which the method is being called.

◆ Code (Example)

```
const user = {  
    name: "Ankit",  
    showName() {  
        console.log(this.name); // 'Ankit'  
    },  
};  
  
user.showName();
```

⚠ In arrow functions, this is lexically scoped:

```
const user = {  
    name: "Ravi",  
    showName: () => {  
        console.log(this.name); // undefined  
    },  
};  
  
user.showName();
```

◆ Use Case

Access properties and methods of the current object inside methods.

◆ Interview Q&A

Q: What is the difference in this behavior between regular functions and arrow functions?

A: Arrow functions don't have their own this; they inherit it from the parent scope.

7. Prototype Object

◆ Definition

Each constructor function has a prototype object. All instances created from it share the prototype's properties and methods.

```
function Animal(name) {  
    this.name = name;  
}  
  
Animal.prototype.speak = function () {  
    console.log(`#${this.name} makes a noise`);  
};  
  
const dog = new Animal("Dog");  
dog.speak(); // Dog makes a noise
```

◆ Use Case

Use prototype to add shared methods to all instances, improving memory efficiency.

◆ Interview Q&A

Q: Why is the prototype used in JavaScript?

A: To share methods among all instances, conserving memory and enabling inheritance.

8. Create Custom Prototype Object

◆ Definition

You can manually assign a prototype to an object using `__proto__` or `Object.setPrototypeOf()`.

◆ Code (Example)

```
const parent = {
  greet() {
    console.log("Hello from parent");
  },
};

const child = {};
child.__proto__ = parent;
child.greet(); // Hello from parent
```

Using Object.create:

```
const customProto = {  
  sayHi() {  
    console.log("Hi from prototype!");  
  },  
};  
  
const obj = Object.create(customProto);  
obj.sayHi(); // Hi from prototype!
```

◆ Use Case

Allows custom inheritance and shared method setup between objects.

◆ Interview Q&A

Q: How can you manually assign a prototype to an object?

A: Using __proto__, Object.setPrototypeOf(), or Object.create().