What is Dynamic Polymorphism?

✅ Definition:

Dynamic Polymorphism (also called **runtime polymorphism**) occurs when a method call is resolved **at runtime**, not at compile-time.It usually happens through **method overriding**, where a child class provides its **own version** of a method defined in the parent class.

✅ Key Concepts:

- Achieved via **inheritance** and **method overriding**.
- Method call is decided at **runtime based on object type**.
- Supports **flexibility and scalability** in OOP design.

✅ JavaScript Example:

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

// Child class overrides the speak method
class Dog extends Animal {
  speak() {
    console.log("The dog barks.");
  }
}

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

// Function that uses polymorphism
function makeAnimalSpeak(animal) {
  animal.speak(); // The correct version is chosen at runtime
}

// Runtime behavior
const a1 = new Animal();
const d1 = new Dog();
const c1 = new Cat();

makeAnimalSpeak(a1); // Output: The animal makes a sound.
makeAnimalSpeak(d1); // Output: The dog barks.
makeAnimalSpeak(c1); // Output: The cat meows.
```

✅ Explanation (Point-by-Point):

1. The speak() method is **overridden** in Dog and Cat.
2. The function makeAnimalSpeak() takes an object of any subclass of Animal.
3. At **runtime**, the JavaScript engine determines **which version** of speak() to call based on the actual object.
4. This is **dynamic polymorphism** — method resolution happens **while the code is running**, not while it's written.

✅ Real-Life Analogy:

You call a general method makeSound() on different objects:

- If it’s a **Dog**, it barks.
- If it’s a **Cat**, it meows.Same call, **different behavior** – decided at **runtime**.