Programming Paradigm:

1. Imperative Paradigm
    - Procedural Programming
    - Structured Programming
2. Declarative Paradigm
    - Functional Programming
    - Reactive Programming (e.g., RxJS)
3. Object-Oriented Programming
    - Class-based
    - Prototype-based
4. Event-Driven Programming
    - Based on event listeners, callbacks, DOM events, etc.
5. Asynchronous Programming (cross-paradigm)
    - Callback-based
    - Promise-based
    - async/await (syntactic sugar over Promises)

A programming paradigm is a style or way of programming.

Common paradigms include:

1. Imperative programming (writing step-by-step instructions)

2. Declarative Programming (What it is: Focuses on what to do, not how to do it.)

3. Event-driven programming (responding to events like clicks or messages)

4. Functional programming (using functions as the core building blocks)

5. Object-oriented programming (organizing code around objects)

Most Used Programming Paradigm:

Paradigm	Common in JS?	Description
1. Imperative Programming	✅ Very common	Writing step-by-step logic like loops, conditionals, and variable updates.
2. Declarative Programming	✅ Very common	What it is: Focuses on what to do, not how to do it. Seen in Array.map, JSX in React, etc.
3. Event-Driven Programming	✅ Core in JS	Central to DOM events, and browser interactivity.
4. Functional Programming	✅ Increasingly popular	Emphasized in modern JS with arrow functions, map, filter, reduce, immutability, etc and avoid shared state & side effects etc.
5. Object-Oriented Programming	✅ Still widely used	JavaScript supports both prototype-based and class-based OOP. Used in frameworks and app architecture.

Imperative Programming v/s Declarative Programming:

Imperative Programming

Concept: You tell the computer how to do something step by step.

Think of it like: Giving someone exact instructions to make a sandwich — one step at a time.

```
let numbers = [1, 2, 3];
let doubled = [];

for (let i = 0; i < numbers.length; i++) {
  doubled.push(numbers[i] * 2);
}

console.log(doubled); // [2, 4, 6]
```

Declarative Programming:

Concept: You describe what you want, not how to do it.

Think of it like: Ordering a coffee — you just say "I want a cappuccino", you don't explain how to make it.

```
let numbers = [1, 2, 3];
let doubled = numbers.map(num => num * 2);

console.log(doubled); // [2, 4, 6]
```

Event-Driven Programming

Concept: Code responds to events like button clicks, form submissions, or messages.

Think of it like: A doorbell — your action (pressing the bell) triggers a response (someone answers).

```
<button onclick="greet()">Click Me</button>

<script>
    function greet() {
        alert("Hello, School4U!");
    }
</script>
```

Functional Programming

Concept: Uses pure functions, avoids changing variables, and focuses on data transformation.

Think of it like: A machine that always gives the same output for the same input.

```
const add = (a, b) => a + b;

const result = add(2, 3); // 5
console.log(result);
```

```
const add = (a, b) => a + b;

const result = add(2, 3); // 5
console.log(result);
```

Object-Oriented Programming (OOP)

Concept: Organize code into objects with properties (data) and methods (behavior).

Think of it like: A car object — it has properties (color, brand) and methods (drive()).

```
class Car {
    constructor(brand) {
        this.brand = brand;
    }

    start() {
        console.log(`${this.brand} is starting...`);
    }
}

const myCar = new Car("Buggati");
myCar.start(); // Buggati is driving
```