🌟 What is ES6?

> ES6 (also called ECMAScript 2015) is the sixth edition of JavaScript's standard. It introduced **new features** to make coding faster, cleaner, and more fun! Think of it as a "major upgrade" to JavaScript.

Why Learn ES6?

- Most modern websites and apps use ES6+ features.
- It simplifies complex tasks (like loops and object handling).
- It's widely supported in browsers today.

💡 *Example:* Before ES6, you'd write `var myArray = new Array();`. Now, you'd just do `let myArray = [1, 2, 3];` with `let` (a newer variable type).

🧠 Key Features of ES6

Let's break down the big ones!

1️⃣ Variables: `let` vs `const` vs `var`

Before ES6, JavaScript used `var` to declare variables. But it had flaws (like hoisting and scope issues). ES6 introduced `let` and `const` to fix these!

- `var`: Function-scoped (can be modified/overwritten).
- `let`: Block-scoped (works with `{}` blocks). Can change value.
- `const`: Also block-scoped. **Cannot change value** once set.

Keywords	var	let	const
Eample	var a = 10	let b = 10	const c = 10
Initialization	Can be declared without an initial value	Can be declared without an initial value	Must be assigned an initial value when declared
Re-declaration	Can be redeclared within the same scope	Cannot be redeclared within the same block scope	Cannot be redeclared within the same block scope
Re-initialization	Can be reassigned	Can be reassigned	Cannot be reassigned
Scope	Function-scoped	Block-scoped	Block-scoped

Example:

```
let name = "Alice"; // OK to change later
const age = 25;    // Error if you try to change this!
```

Self-check:

- What happens if you try to reassign a `const` variable?
- When should you use `let` vs `const`?

2️⃣ Arrow Functions

Arrow functions (`⇒`) offer a **shorter syntax** for writing functions. They're great for small tasks!

Old Way (ES5):

```
function add(a, b) { return a + b; }
```

New Way (ES6 Arrow Function):

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

Self-check:

- What's the benefit of using an arrow function instead of a regular function?
- How does `this` work differently in arrow functions?

3️⃣ Template Literals

This is a fun way to create strings! Use **backticks (`)** instead of quotes (`"` or `'`). You can even **embed variables** inside strings using `${}`.

Example:

```
let greeting = `Hello, ${name}!`; // Instead of "Hello, " + name + "!"
```

Self-check:

- How would you add a variable inside a string using template literals?
- What's the advantage of template literals over regular strings?

4️⃣ Destructuring Assignment

This feature lets you **unwrap values from arrays or objects** into separate variables. No more messy `array[0]`, `obj.property`!

Example with Arrays:

```
let [first, second] = [10, 20]; // first = 10, second = 20
```

Example with Objects:

```
let person = { name: "Bob", age: 30 };
let { name, age } = person; // name = "Bob", age = 30
```

Self-check:

- How would you get the first and last name from an object like `{ first: "John", last: "Doe" }`?
- What's the benefit of destructuring?

5️⃣ Classes

ES6 introduced a simpler way to create **objects** using classes. It's like a blueprint for objects!

```
class Car {
  constructor(model) {
    this.model = model;
  }
  start() {
    console.log(`${this.model} is starting!`);
  }
}
```

You can now create car objects like:

```
let myCar = new Car("Tesla");
```

Self-check:

- What does the `constructor` do in a class?
- How is a class different from an object in JavaScript?

🔁 Quick Recap 🌟

- **ES6** is a modern version of JavaScript with cool new features.
- Use `let` (changeable) and `const` (fixed) for variables.
- Arrow functions (`⇒`) are shorter and handle `this` differently.
- Template literals (`` ` ``) make strings easier with embedded variables.
- Destructuring simplifies extracting values from arrays/objects.
- Classes help organize code like blueprints.