Series 1: JavaScript Foundations.

---

1. `var`, `let`, `const`

👨‍💻 Interview Q&A:

Q: What's the difference between `var`, `let`, and `const`?

> `var` is function-scoped and hoisted; `let` and `const` are block-scoped.
> `let` allows reassignment, `const` does not.
> `const` ensures the variable name can't be reassigned, but the contents (like in objects/arrays) can still be mutated.

Example:

```
function testScope() {
  if (true) {
    var a = 1;
    let b = 2;
    const c = 3;
  }
  console.log(a); // 1
  console.log(b); // ReferenceError
  console.log(c); // ReferenceError
}
testScope();
```

🎯 Analogy:

Imagine `var` as an old-school locker that anyone in the class can access all day (function-wide). `let` and `const` are like modern smart lockers: access limited to a specific hallway (block) and keycard.

---

2. Data Types

Interview Q&A:

Q: What are JavaScript's primitive data types?

> JS has 7 primitives: `string`, `number`, `boolean`, `null`, `undefined`, `symbol`, and `bigint`.
> These are immutable and stored by value. Non-primitive types like objects and arrays are stored by reference.

Example:

```
let x = 10;
let y = x;
y++;
console.log(x); // 10
console.log(y); // 11
```

🎯 Analogy:

Think of primitives as a photocopy — changing the copy doesn't affect the original.
Objects are like links to a Google Doc — changes affect all who share the link.

---

3. Hoisting

👨‍💻 Interview Q&A:

Q: What is hoisting in JavaScript?

> In JavaScript, variable and function declarations are moved to the top of their scope at runtime.
> `var` is hoisted but initialized with `undefined`.
> `let` and `const` are hoisted but not initialized — they exist in the *Temporal Dead Zone* until the line where they are declared.

Example:

```
console.log(a); // undefined
```

```
var a = 5;

console.log(b); // ReferenceError
let b = 10;
```

🎯 Analogy:

Think of hoisting like setting up chairs at the front of the room before class.
`var` gets a chair but it's empty (`undefined`).
`let/const` also have chairs but they're roped off — you can't use them yet.

---

4. Scope

👨‍💻 Interview Q&A:

Q: What are the types of scope in JavaScript?

JavaScript has 3 scopes:

- Global scope (outside all functions)
- Function scope (`var` inside functions)
- Block scope (`let` and `const` inside `{}`)

Example:

```
let globalVar = 'I am global';

function greet() {
  let localVar = 'I am local';
  console.log(globalVar); // accessible
  console.log(localVar);  // accessible
}

console.log(globalVar); // accessible
console.log(localVar);  // ReferenceError
```

🎯 Analogy:

Global scope is like a city — everyone has access.
Function scope is like your house — only family inside knows what's there.
Block scope is like your bedroom — even more private.

---

5. Closures

👨‍💻 Interview Q&A:

Q: What is a closure in JavaScript?

> A closure is when a function "remembers" the variables from its outer lexical scope even after that outer function has finished executing.

Example:

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    console.log(count);
  };
}

const counter = outer();
counter(); // 1
counter(); // 2
```

🎯 Analogy:

A closure is like a child remembering their parent's advice even after they've moved out.
The child (inner function) carries that memory (outer variable `count`) with them.