Perfect. Continuing **systematically**, **NotebookLM-ready**, with **NOTES and CODE strictly separated**.

📘 TOPIC: Logical Operators (`&&`, `||`, `!`)

(Short-Circuiting • Return Values • Edge Cases)

🟦 NOTE 1: What logical operators do

- Logical operators combine or evaluate conditions
- They **do NOT always return booleans**
- They use **short-circuit evaluation**
- Very important for real-world JS patterns

🟩 CODE 1: Basic logical operators

```
js

true && true;   // true
true || false;  // true
!true;        // false
```

🟦 NOTE 2: Logical AND (`&&`) — core rule

- Evaluates operands **left to right**
- Returns the **first falsy value**
- If no falsy value → returns **last value**

🟩 CODE 2: `&&` behavior examples

```
js

true && true;      // true
true && false;     // false
false && true;     // false

1 && 2;          // 2
0 && 100;        // 0
"JS" && "Code";    // "Code"
"" && "Hello";     // ""
```

🟦 NOTE 3: Why `&&` returns non-boolean values

- JS does **not** force boolean conversion
- It returns the **actual operand**
- Enables guard patterns and chaining

🟩 CODE 3: Guard pattern with `&&`

```
js
```

```
let user = { name: "Anoop" };

user && user.name && console.log(user.name);
// Prevents runtime errors
```

🟦 NOTE 4: Logical OR (`||`) — core rule

- Evaluates operands **left to right**
- Returns the **first truthy value**
- If no truthy value → returns **last value**

🟩 CODE 4: `||` behavior examples

```
js

true || false;      // true
false || true;      // true

0 || 100;          // 100
"" || "default";    // "default"
"JS" || "Code";     // "JS"
null || "fallback"; // "fallback"
```

🟦 NOTE 5: Default value pattern using `||`

- Commonly used to set defaults
- Can cause bugs if valid falsy values exist

🟩 CODE 5: Default value trap

```
js

let count = 0;
let value = count || 10;

value; // 10 ❌ (unexpected if 0 is valid)
```

🟦 NOTE 6: NOT operator (`!`)

- Forces boolean conversion
- Then negates the result
- Often used to normalize values

🟩 CODE 6: `!` and `!!` usage

```
js
```

```
!true;      // false
!0;         // true
!"";        // true

!!"JS";     // true
!!0;        // false
```

🟦 NOTE 7: Short-circuit evaluation (IMPORTANT)

- Evaluation stops as soon as result is known
- Later expressions may **never execute**
- Used heavily in performance-sensitive code

🟩 CODE 7: Short-circuit examples

```
js

false && console.log("will not run");
true || console.log("will not run");

true && console.log("runs");
false || console.log("runs");
```

🟦 NOTE 8: Logical operators return VALUES, not booleans

- This is a key JS concept
- Enables functional patterns
- Confuses beginners and interviewees

🟩 CODE 8: Value-return behavior

```
js

let result1 = "JS" && 100;   // 100
let result2 = null || 50;   // 50
```

🟦 NOTE 9: Mixing logical operators

- `&&` has higher precedence than `||`
- Parentheses improve clarity and safety

🟩 CODE 9: Precedence examples

```
js

true || false && false;   // true
(true || false) && false; // false
```

🟦 NOTE 10: Logical operators with functions

- Functions execute only if needed
- Useful for conditional execution

🟩 CODE 10: Conditional execution

```
js

let isLoggedIn = true;

isLoggedIn && showDashboard();

function showDashboard() {
  console.log("Dashboard shown");
}
```

🟦 NOTE 11: Common interview traps

- `&&` returns first falsy, not true/false
- `||` returns first truthy
- `!` forces boolean
- Short-circuit skips execution

🟦 NOTE 12: Best practices (professional)

- Use `&&` for guards
- Use `||` carefully for defaults
- Prefer `??` when falsy values are valid
- Avoid over-complex logical chains

🟦 FINAL MENTAL MODEL

- `&&` → first falsy OR last value
- `||` → first truthy OR last value
- `!` → boolean coercion + negation
- Short-circuiting controls execution
- Logical operators return values

🛑 STOP HERE

Next **recommended continuation**:

1️⃣ **Decision Making (`if / else / switch`)**
2️⃣ **Nullish Coalescing (`??`) vs `||` (very important)**
3️⃣ **Loops (`for`, `while`, `for…of`, `for…in`)**

Reply with **next** or the **number** you want to continue.