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

📘 TOPIC: Nullish Coalescing Operator (`??`)

(vs Logical OR `||` • Edge Cases • Real-World Patterns)

🟦 NOTE 1: Why `??` was introduced

- `||` treats **all falsy values** as "missing"
- This causes bugs when **valid falsy values** exist (`0`, `""`, `false`)
- `??` fixes this by checking **only**:
  - `null`
  - `undefined`

🟩 CODE 1: The core problem with `||`

```js
let count = 0;
let result = count || 10;

result; // 10 ❌ (0 is valid but ignored)
```

🟦 NOTE 2: Definition of `??`

- `a ?? b` returns:
  - `b` **only if** `a` is `null` or `undefined`
  - otherwise returns `a`
- No other falsy values are considered "missing"

🟩 CODE 2: Basic `??` usage

```js
let count = 0;
let result = count ?? 10;

result; // 0 ✅
```

🟦 NOTE 3: Exact comparison logic (internal rule)

Internally, this is how JS treats `??`:

```js
a !== null && a !== undefined ? a : b
```

No boolean coercion involved.

🟩 CODE 3: Internal equivalence

```
js

let value = undefined;
let output = value ?? "default";

// Equivalent to:
let output2 =
  value !== null && value !== undefined
    ? value
    : "default";
```

🟦 NOTE 4: `??` vs `||` — side-by-side behavior

Value	`value || "X"`	`value ?? "X"`
`0`	`"X"` ❌	`0` ✅
`""`	`"X"` ❌	`""` ✅
`false`	`"X"` ❌	`false` ✅
`null`	`"X"` ✅	`"X"` ✅
`undefined`	`"X"` ✅	`"X"` ✅

🟩 CODE 4: Side-by-side examples

```
js

0 || 100;       // 100
0 ?? 100;       // 0

"" || "text";   // "text"
"" ?? "text";   // ""

false || true;  // true
false ?? true;  // false
```

🟦 NOTE 5: Common real-world use cases

- Default values from APIs
- Configuration values
- User input handling
- Optional fields

🟩 CODE 5: API data example

```js
let user = {
  name: "Anoop",
  age: 0
};

let age = user.age ?? 18;
age; // 0 (correct)
```

🟦 NOTE 6: `??` with function return values

- Safe when a function may return `null` or `undefined`
- Avoids overwriting valid falsy returns

🟩 CODE 6: Function defaulting

```js
function getScore() {
  return 0;
}

let score = getScore() ?? 50;
score; // 0
```

🟦 NOTE 7: Short-circuit behavior

- `??` is **short-circuiting**
- Right side executes **only if needed**

🟩 CODE 7: Short-circuit example

```js
let value = null;

value ?? console.log("Runs"); // console.log runs
```

```js
let value2 = 10;

value2 ?? console.log("Does not run");
```

🟦 NOTE 8: Operator precedence (IMPORTANT)

- `??` has **lower precedence** than `||` and `&&`
- Mixing without parentheses is **not allowed**

🟩 CODE 8: Syntax error example

```js
// ❌ SyntaxError
// let x = null || 10 ?? 20;
```

🟦 NOTE 9: Correct way to mix operators

- Always use parentheses
- Be explicit

🟩 CODE 9: Correct mixing

```js
let x = (null || 10) ?? 20;
x; // 10

let y = null ?? (false || 50);
y; // 50
```

🟦 NOTE 10: `??` vs ternary (`? :`)

- `??` is **more readable** for null checks
- Ternary is more flexible but verbose

🟩 CODE 10: Comparison

```js
let value = undefined;

let a = value ?? "default";

let b =
  value === null || value === undefined
    ? "default"
    : value;
```

🟦 NOTE 11: When NOT to use `??`

- When **any falsy value** should trigger fallback
- Example: authentication, boolean flags

🟩 CODE 11: Wrong usage example

```js
let isLoggedIn = false;

// ❌ wrong
let status = isLoggedIn ?? true;

// ✅ correct
let status2 = isLoggedIn || true;
```

🟦 NOTE 12: Interview traps

- `??` checks only `null` and `undefined`
- `||` checks **all falsy values**
- Cannot mix `??` with `&&` / `||` without parentheses
- `??` is short-circuiting

🟦 FINAL MENTAL MODEL

- `||` → fallback on **falsy**
- `??` → fallback on **nullish**
- `??` = safer defaults
- Always use parentheses when mixing
- Prefer `??` for data, `||` for logic

🛑 STOP HERE

Next **recommended continuation**:

1️⃣ **Loops** (`for`, `while`, `do…while`, `for…of`, `for…in`)
2️⃣ **Functions** (declaration, expression, execution context)
3️⃣ **Scope & Hoisting** (deep, memory-level)

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