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

📘 TOPIC: Decision Making in JavaScript

(`if`, `else`, `else if`, `switch`, ternary)

🟦 NOTE 1: What "decision making" means internally

- Decision making controls **execution flow**
- JavaScript evaluates a condition
- Condition is **coerced to Boolean**
- Based on `true / false`, a code path is chosen

Internally:

- Expression → `Boolean(expression)`
- No comparison with `true` happens explicitly

🟩 CODE 1: Basic `if` statement

```js
let age = 20;

if (age >= 18) {
  console.log("Adult");
}
```

🟦 NOTE 2: How `if` evaluates conditions

- Condition is **not required to be boolean**
- JavaScript applies **truthy / falsy rules**

🟩 CODE 2: Truthy / falsy in `if`

```js
if (1) {
  console.log("runs");
}

if (0) {
  console.log("does not run");
}

if ("") {
  console.log("does not run");
}
```

```
if ("hello") {
  console.log("runs");
}
```

🟦 NOTE 3: `if...else` flow

- If condition is truthy → `if` block runs
- Otherwise → `else` block runs
- Exactly **one block executes**

🟩 CODE 3: `if...else`

```
js

let isLoggedIn = false;

if (isLoggedIn) {
  console.log("Welcome");
} else {
  console.log("Please log in");
}
```

🟦 NOTE 4: `else if` ladder

- Conditions evaluated **top to bottom**
- First matching condition executes
- Remaining conditions are skipped

🟩 CODE 4: `else if` chain

```
js

let score = 85;

if (score >= 90) {
  console.log("Grade A");
} else if (score >= 75) {
  console.log("Grade B");
} else if (score >= 50) {
  console.log("Grade C");
} else {
  console.log("Fail");
}
```

🟦 NOTE 5: Importance of order in `else if`

- Conditions must be ordered **carefully**
- First match wins, even if later conditions are “better”

🟩 CODE 5: Order bug example

```js
let marks = 95;

if (marks >= 50) {
  console.log("Pass");
} else if (marks >= 90) {
  console.log("Excellent");
}
// "Pass" (logical bug)
```

🟦 NOTE 6: Nested `if` statements

- `if` inside another `if`
- Harder to read if overused
- Often refactored using logical operators or early returns

🟩 CODE 6: Nested `if`

```js
let user = { loggedIn: true, isAdmin: false };

if (user.loggedIn) {
  if (user.isAdmin) {
    console.log("Admin panel");
  } else {
    console.log("User dashboard");
  }
}
```

🟦 NOTE 7: `switch` statement (decision by value)

- Used when comparing **same variable** against many values
- Uses **strict equality (`===`) internally**
- Better readability than long `else if` chains

🟩 CODE 7: Basic `switch`

```js
```

```
let day = 3;

switch (day) {
  case 1:
    console.log("Monday");
    break;
  case 2:
    console.log("Tuesday");
    break;
  case 3:
    console.log("Wednesday");
    break;
  default:
    console.log("Invalid day");
}
```

🟦 NOTE 8: Why `break` is required

- Without `break`, execution **falls through**
- This is by design (C-style behavior)

🟩 CODE 8: Fall-through behavior

```
js

let x = 1;

switch (x) {
  case 1:
    console.log("One");
  case 2:
    console.log("Two");
  default:
    console.log("Done");
}

// Output:
// One
// Two
// Done
```

🟦 NOTE 9: Intentional fall-through (advanced use)

- Sometimes used deliberately

- Must be documented clearly

🟩 CODE 9: Controlled fall-through

```js
let role = "admin";

switch (role) {
  case "admin":
  case "manager":
    console.log("Access granted");
    break;
  default:
    console.log("Access denied");
}
```

🟦 NOTE 10: `switch` vs `if...else`

Aspect	if...else	switch
Condition	Any expression	Single value
Comparison	Boolean	`===`
Readability	Medium	High for enums
Flexibility	High	Limited

🟦 NOTE 11: Ternary operator (`?:`)

- Short form of `if...else`
- Expression-based (returns a value)
- Should be used for **simple decisions only**

🟩 CODE 10: Basic ternary

```js
let age = 20;

let result = age >= 18 ? "Adult" : "Minor";
console.log(result);
```

🟦 NOTE 12: Nested ternary (readability warning)

- Technically valid

- Hard to read and debug
- Avoid in production code

🟩 CODE 11: Nested ternary

```
js

let marks = 80;

let grade =
  marks >= 90 ? "A" :
  marks >= 75 ? "B" :
  marks >= 50 ? "C" :
  "Fail";
```

🟦 NOTE 13: Decision making + logical operators

- Logical operators often replace nested `if`
- Improves readability and performance

🟩 CODE 12: Logical decision pattern

```
js

let user = { loggedIn: true, name: "Anoop" };

user.loggedIn && console.log("Welcome", user.name);
```

🟦 NOTE 14: Common interview traps

- `if (value)` uses coercion
- `switch` uses strict equality
- Missing `break` causes fall-through
- Ternary returns a value, `if` does not

🟦 FINAL MENTAL MODEL

- `if` → boolean coercion
- `else if` → first match wins
- `switch` → strict equality checks
- `break` controls flow
- Ternary → expression, not statement

🛑 STOP HERE

Next **recommended continuation**:

1️⃣ **Nullish Coalescing (`??`) vs `||`**

2️⃣ **Loops (`for`, `while`, `for…of`, `for…in`)**

3️⃣ **Functions (declaration, expression, execution context)**

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