Great. We continue **step-by-step**, **NotebookLM-ready**, with **NOTES and CODE strictly separated**.

📘 TOPIC: Comparison Operators (Deep Rules + Edge Cases)

🟦 NOTE 1: What comparison operators do

- Comparison operators compare **two values**
- Result is **always a boolean**
- Some comparisons trigger **type coercion**
- Others are **strict and safe**

🟩 CODE 1: Basic comparison operators

```
js

10 > 5;     // true
10 < 5;     // false
10 >= 10;   // true
10 <= 9;    // false
```

🟦 NOTE 2: Equality operators (two categories)

There are **two equality systems** in JavaScript:

1. **Loose equality** → `==`
2. **Strict equality** → `===`

They behave very differently internally.

🟩 CODE 2: Loose vs strict equality

```
js

5 == "5";    // true  (type coercion)
5 === "5";   // false (no coercion)
```

🟦 NOTE 3: Loose equality (`==`) internal rules

- Converts operands to a **common type**
- Uses coercion rules
- Special-cases `null` and `undefined`
- Source of many bugs

🟩 CODE 3: `==` coercion examples (IMPORTANT)

```
js

0 == false;        // true
"" == false;       // true
```

```
"0" == false;       // true
null == undefined;  // true
[] == false;        // true
[] == "";           // true
```

🟦 NOTE 4: `null` and `undefined` special rule

- `null` and `undefined` are **loosely equal only to each other**
- They are NOT equal to any other value

🟩 CODE 4: null / undefined cases

```
js

null == undefined;  // true
null == 0;          // false
undefined == 0;     // false
```

🟦 NOTE 5: Strict equality (`===`) rules

- No type conversion
- Compares **type + value**
- Predictable and safe
- Always preferred in production

🟩 CODE 5: Strict equality clarity

```
js

0 === false;        // false
"" === false;       // false
null === undefined; // false
```

🟦 NOTE 6: Object comparison behavior

- Objects are compared by **reference**
- Even identical objects are NOT equal

🟩 CODE 6: Object comparison

```
js

let a = {};
let b = {};

a == b;   // false
a === b;  // false
```

🟦 NOTE 7: Same reference comparison

- If two variables point to the same object
- Comparison returns true

🟩 CODE 7: Same reference

```
js

let obj = {};
let ref = obj;

obj === ref;  // true
```

🟦 NOTE 8: Relational operators (`<` , `>`) rules

- Operands are converted to **primitives**
- Then compared as:
  - Numbers OR
  - Strings (lexicographically)

🟩 CODE 8: Numeric comparison

```
js

"10" > 2;    // true
"5" < 10;    // true
```

🟦 NOTE 9: String comparison (lexicographical)

- Compared **character by character**
- Based on Unicode values
- NOT numeric comparison

🟩 CODE 9: String comparison traps

```
js

"2" > "10";   // true
"apple" > "app"; // true
```

🟦 NOTE 10: Mixed comparisons (string + number)

- Strings converted to numbers
- Then numeric comparison occurs

🟩 CODE 10: Mixed cases

```
js
```

```
"10" > 9;    // true
"abc" > 1;   // false (NaN comparison)
```

🟦 NOTE 11: Comparisons with NaN

- `NaN` is **never equal to anything**
- Even itself

🟩 CODE 11: NaN behavior

```
js

NaN == NaN;     // false
NaN === NaN;    // false
NaN > 0;        // false
NaN < 0;        // false
```

🟦 NOTE 12: How to correctly check NaN

🟩 CODE 12: NaN checks

```
js

isNaN("abc");       // true (coercion!)
Number.isNaN("abc"); // false
Number.isNaN(NaN);   // true
```

🟦 NOTE 13: Comparison chaining DOES NOT work like math

JavaScript does NOT support mathematical chaining.

🟩 CODE 13: Chaining trap (INTERVIEW)

```
js

1 < 2 < 3;   // true
3 < 2 < 1;   // true (WHY?)
```

Explanation:

- `1 < 2` → true
- `true < 3` → `1 < 3` → true

🟦 NOTE 14: `Object.is()` (advanced equality)

- Similar to `===`
- Handles edge cases differently

🟩 CODE 14: Object.is differences

```js
Object.is(NaN, NaN);    // true
Object.is(0, -0);       // false
0 === -0;               // true
```

🟦 NOTE 15: Best practices (professional)

- Always use `===`
- Never chain comparisons
- Be careful with string comparisons
- Avoid comparing objects directly
- Use `Object.is` for edge cases

🟦 FINAL MENTAL MODEL

- `==` → coercion (dangerous)
- `===` → strict (safe)
- Objects compare by reference
- Strings compare lexicographically
- `NaN` is never equal
- `Object.is` handles edge cases

🛑 STOP HERE

Next **recommended continuation**:

1️⃣ Logical Operators (`&&` , `||` , `!`) + Short-Circuiting
2️⃣ Decision Making (`if / else / switch`)
3️⃣ Nullish Coalescing (`??`) vs `||`

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