Great. We continue **cleanly**, **one topic**, **NotebookLM-ready**, with **NOTES and CODE strictly separated**.

📘 TOPIC: JavaScript Data Types (Internal + Memory Model)

🟦 NOTE 1: JavaScript is dynamically typed

- Variables do **not** have fixed types
- **Values** have types
- Type is decided **at runtime**, not compile time

🟩 CODE 1: Dynamic typing

```js
let x = 10;     // number
x = "hello";   // string
x = true;      // boolean
```

🟦 NOTE 2: JavaScript type categories (core model)

JavaScript types are divided into **two internal categories**:

1. **Primitive types** → stored by value
2. **Non-primitive types (Objects)** → stored by reference

This division is **fundamental to memory behavior**.

🟦 NOTE 3: Primitive data types (7 total)

Primitive types are:

1. Number
2. String
3. Boolean
4. Undefined
5. Null
6. BigInt
7. Symbol

Properties of primitives:

- Immutable
- Stored directly in stack
- Compared by value

🟩 CODE 2: Primitive examples

```js
```

```
let n = 10;          // Number
let s = "JS";        // String
let b = true;        // Boolean
let u;               // Undefined
let z = null;        // Null
let big = 123n;      // BigInt
let sym = Symbol();  // Symbol
```

🟦 NOTE 4: Number (internal details)

- JavaScript has **only one numeric type**
- No int / float separation
- Uses **IEEE-754 floating-point**

This explains precision issues.

🟩 CODE 3: Number quirks

```js
0.1 + 0.2;      // 0.30000000000000004
typeof NaN;     // "number"
```

🟦 NOTE 5: String (immutability)

- Strings are immutable
- Any “change” creates a **new string in memory**

🟩 CODE 4: String immutability

```js
let s = "Hi";
s[0] = "h";      // no effect
s = s + " JS";   // new string created
```

🟦 NOTE 6: Boolean

- Only two values: `true`, `false`
- Extensively used in control flow
- Result of comparisons and logical operations

🟩 CODE 5: Boolean usage

```js
let isLoggedIn = true;
let result = 10 > 5; // true
```

🟦 NOTE 7: Undefined

- Means: declared but **not initialized**
- Automatically assigned by JS engine

🟩 CODE 6: Undefined example

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

🟦 NOTE 8: Null (intentional absence)

- `null` means “no value intentionally”
- Used by developers, not engine

⚠️ Historical bug:

`typeof null === "object"`

🟩 CODE 7: Null quirks

```js
let x = null;
typeof x;          // "object" (bug)
```

🟦 NOTE 9: BigInt

- Used for very large integers
- Ends with `n`
- Cannot mix with Number directly

🟩 CODE 8: BigInt rules

```js
let big = 100000000000000000000n;
// big + 10;        // TypeError
big + 10n;          // OK
```

🟦 NOTE 10: Symbol

- Always unique
- Used as object keys
- Prevents naming collisions

🟩 CODE 9: Symbol uniqueness

```js
let a = Symbol("id");
let b = Symbol("id");

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

🟦 NOTE 11: Non-primitive type (Object)

- Objects include:
    - Object
    - Array
    - Function
    - Date, Map, Set, etc.
- Stored in **heap**
- Variables store **reference (address)**

🟩 CODE 10: Object reference behavior

```js
let obj1 = { x: 10 };
let obj2 = obj1;

obj2.x = 20;
console.log(obj1.x); // 20
```

🟦 NOTE 12: Primitive vs Object (core difference)

Aspect	Primitive	Object
Storage	Stack	Heap
Copy	Value	Reference
Mutability	Immutable	Mutable
Comparison	By value	By reference

🟩 CODE 11: Comparison difference

```js
let a = 10;
let b = 10;
```

```
a === b; // true

let o1 = {};
let o2 = {};
o1 === o2; // false
```

🟦 NOTE 13: `typeof` operator (important)

- Returns type as string
- Has **known edge cases**

🟩 CODE 12: typeof results

```
js

typeof 10;          // "number"
typeof "hi";        // "string"
typeof true;        // "boolean"
typeof undefined;   // "undefined"
typeof null;        // "object" (bug)
typeof {};          // "object"
typeof [];          // "object"
typeof function(){}; // "function"
```

🟦 NOTE 14: Why arrays are objects

- Arrays are specialized objects
- Indexed keys + length property
- Stored in heap

🟩 CODE 13: Array proof

```
js

Array.isArray([]);  // true
typeof [];          // "object"
```

🟦 NOTE 15: Memory mental model (important)

- Primitive → value copied
- Object → reference copied
- Explains side effects and bugs

🟦 FINAL MENTAL MODEL

- JS is dynamically typed
- Primitives = value semantics
- Objects = reference semantics

- `typeof null` is a bug
- Arrays & functions are objects

🛑 **STOP HERE**

Next **logical continuation** (same format, recommended order):

1️⃣ **Type Conversion & Coercion** (VERY important)

2️⃣ Truthy / Falsy values

3️⃣ Comparison operators (`==` vs `===`)

4️⃣ Control flow (`if`, `switch`)

Reply with the **number** to continue.