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

📘 TOPIC: Loops in JavaScript

(`for`, `while`, `do…while`, `for…of`, `for…in` • Internals & Edge Cases)

🟦 NOTE 1: What loops do internally

- Loops repeat execution of a block
- Each iteration:
  - Evaluates a condition
  - Executes body if condition is truthy
- Loop control relies on:
  - Boolean coercion
  - Increment/update expressions
- Poor loop design = infinite loops / performance issues

🟩 CODE 1: Basic loop idea

```js
let i = 0;

while (i < 3) {
  console.log(i);
  i++;
}
```

🟦 NOTE 2: `for` loop — structure & control

`for` loop has **three parts**:

1. Initialization (runs once)
2. Condition (checked before each iteration)
3. Update (runs after each iteration)

🟩 CODE 2: `for` loop structure

```js
for (let i = 0; i < 3; i++) {
  console.log(i);
}
```

🟦 NOTE 3: Scope behavior in `for` loops

- `let` → block scoped per iteration

- `var` → function scoped (shared)

This affects closures and async code.

🟩 CODE 3: `let` vs `var` in loops (INTERVIEW)

```
js

for (var i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 0);
}
// 3 3 3
```

```
js

for (let i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 0);
}
// 0 1 2
```

🟦 NOTE 4: Infinite loops (danger)

- Occur when condition never becomes falsy
- Can freeze browser or Node process

🟩 CODE 4: Infinite loop example

```
js

// ❌ Dangerous
// for (;;) {
//   console.log("Runs forever");
// }
```

🟦 NOTE 5: `while` loop — condition-first

- Condition checked **before** execution
- Body may run **zero times**

🟩 CODE 5: `while` loop

```
js

let count = 0;

while (count < 3) {
  console.log(count);
```

```
  count++;
}
```

🟦 NOTE 6: `do...while` loop — condition-last

- Body executes **at least once**
- Condition checked **after** execution

🟩 CODE 6: `do...while`

```
js

let num = 5;

do {
  console.log(num);
  num++;
} while (num < 5);
// runs once even though condition is false
```

🟦 NOTE 7: `break` and `continue`

- `break` → exits loop immediately
- `continue` → skips current iteration

🟩 CODE 7: break / continue

```
js

for (let i = 0; i < 5; i++) {
  if (i === 2) continue;
  if (i === 4) break;
  console.log(i);
}
// Output: 0, 1, 3
```

🟦 NOTE 8: `for...of` loop — iterable values

- Iterates over **values**
- Works on:
  - Arrays
  - Strings
  - Maps
  - Sets
- Uses the **iterator protocol**

🟩 CODE 8: `for...of` examples

```js
for (let value of [10, 20, 30]) {
  console.log(value);
}
```

```js
for (let char of "JS") {
  console.log(char);
}
```

🟦 NOTE 9: `for...in` loop — object keys

- Iterates over **enumerable property keys**
- Used for **objects**
- NOT recommended for arrays

🟩 CODE 9: `for...in` examples

```js
let user = { name: "Anoop", age: 22 };

for (let key in user) {
  console.log(key, user[key]);
}
```

🟦 NOTE 10: Why `for...in` is bad for arrays

- Iterates over:
  - Indexes (as strings)
  - Custom properties
- Order is not guaranteed

🟩 CODE 10: `for...in` array problem

```js
let arr = [10, 20, 30];
arr.extra = 99;

for (let i in arr) {
  console.log(i, arr[i]);
```

```
}
// includes "extra"
```

🟦 NOTE 11: Looping arrays — best practices

- Use:
  - `for`
  - `for…of`
  - Array methods (`map`, `forEach`) — later topic
- Avoid `for…in` for arrays

🟦 NOTE 12: Nested loops (performance warning)

- Loop inside loop = multiplicative complexity
- Use carefully for large datasets

🟩 CODE 11: Nested loop example

```js

for (let i = 0; i < 3; i++) {
  for (let j = 0; j < 2; j++) {
    console.log(i, j);
  }
}
```

🟦 NOTE 13: Loop control variables & memory

- Loop variables live in:
  - Block scope (`let`)
  - Function/global scope (`var`)
- Each `let` iteration gets its own binding

🟦 NOTE 14: Common interview traps

- Infinite loops
- `var` in loops with async
- Using `for…in` on arrays
- Forgetting `break` in loops
- Misunderstanding loop condition coercion

🟦 FINAL MENTAL MODEL

- Loops rely on boolean coercion
- `for` → compact, controlled
- `while` → condition-first
- `do…while` → at least once

- `for...of` → values (iterables)
- `for...in` → keys (objects)
- `let` in loops is safer than `var`

🛑 **STOP HERE**

Next **recommended continuation**:

1️⃣ **Functions** (declaration, expression, execution context)
2️⃣ **Scope & Hoisting** (deep, memory-level)
3️⃣ **Array methods** (`map` , `filter` , `reduce`)

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