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

📘 TOPIC: Error Handling in JavaScript

(Sync vs Async • `try/catch` • Error Objects • Pitfalls • Interview Traps)

🟦 NOTE 1: What an “error” really is

- An error is a **runtime exception**
- It immediately **stops normal execution**
- JS creates an **Error object**
- Unhandled errors can:
  - Crash Node apps
  - Break browser scripts
  - Cause silent failures (worst case)

🟦 NOTE 2: Error vs Bug vs Exception

Term	Meaning
Bug	Logical mistake
Error	Runtime failure
Exception	Thrown error object

👉 We **handle errors**, not bugs.

🟦 NOTE 3: Built-in Error types (IMPORTANT)

JavaScript provides standard error objects:

- `Error`
- `ReferenceError`
- `TypeError`
- `SyntaxError`
- `RangeError`
- `URIError`

🟩 CODE 1: Common error examples

```js
x;              // ReferenceError
null.foo();     // TypeError
JSON.parse("{"); // SyntaxError
```

🟦 NOTE 4: How JS executes errors internally

1. Error occurs
2. Stack unwinds
3. If `try/catch` exists → control jumps to `catch`
4. Otherwise → runtime terminates execution

🟦 NOTE 5: `try...catch` (sync only!)

- Catches **runtime errors**
- Works **only for synchronous code**
- Does NOT catch async errors automatically

🟩 CODE 2: Basic try/catch

```js
try {
  let x = y + 1;
} catch (err) {
  console.log(err.message);
}
```

🟦 NOTE 6: What `catch(err)` contains

The error object includes:

- `name`
- `message`
- `stack`

🟩 CODE 3: Error object inspection

```js
try {
  foo();
} catch (err) {
  console.log(err.name);
  console.log(err.message);
  console.log(err.stack);
}
```

🟦 NOTE 7: `finally` block

- Always runs
- Executes **after try/catch**
- Used for cleanup

🟩 CODE 4: finally example

```js
try {
  console.log("Try");
} catch {
  console.log("Catch");
} finally {
  console.log("Cleanup");
}
```

🟦 NOTE 8: Throwing custom errors

- Use `throw`
- Can throw:
  - Error objects (recommended)
  - Strings (bad practice)

🟩 CODE 5: Custom error

```js
function withdraw(amount) {
  if (amount <= 0) {
    throw new Error("Invalid amount");
  }
}
```

🟦 NOTE 9: Why always throw Error objects

Bad:

```js
throw "Error occurred";
```

Good:

```js
throw new Error("Error occurred");
```

👉 Error objects preserve stack trace.

🟦 NOTE 10: `try/catch` does NOT catch async errors (TRAP)

🟩 CODE 6: Async error NOT caught

```js
try {
 setTimeout(() => {
   throw new Error("Boom");
 }, 1000);
} catch (e) {
 console.log("Caught");
}
// ❌ NOT caught
```

🟦 NOTE 11: Correct async error handling (Promises)

- Use `.catch()`
- Or `try/catch` with `await`

🟩 CODE 7: Promise error handling

```js
fetch(url)
 .then(res => res.json())
 .catch(err => console.log(err));
```

🟩 CODE 8: Async/await error handling (BEST)

```js
async function load() {
 try {
   const res = await fetch(url);
   if (!res.ok) throw new Error("HTTP error");
   const data = await res.json();
 } catch (err) {
   console.log(err.message);
 }
}
```

🟦 NOTE 12: Error propagation

- Errors bubble up
- First matching `catch` handles it
- Unhandled promise rejections are dangerous

🟩 CODE 9: Error bubbling

```js
async function a() {
  throw new Error("Fail");
}

async function b() {
  await a();
}

b().catch(err => console.log(err.message));
```

🟦 NOTE 13: Global error handling (Browser)

Browser provides:

- `window.onerror`
- `unhandledrejection`

🟩 CODE 10: Browser global handlers

```js
window.onerror = function (msg) {
  console.log("Global error:", msg);
};

window.addEventListener("unhandledrejection", e => {
  console.log("Unhandled promise:", e.reason);
});
```

🟦 NOTE 14: Global error handling (Node.js)

Node provides:

- `process.on("uncaughtException")`
- `process.on("unhandledRejection")`

🟩 CODE 11: Node global handlers

```js
process.on("uncaughtException", err => {
  console.error(err);
});

process.on("unhandledRejection", reason => {
  console.error(reason);
});
```

🟦 NOTE 15: When NOT to use try/catch

Avoid:

- Wrapping entire functions blindly
- Swallowing errors silently
- Using try/catch for normal control flow

🟦 NOTE 16: Interview traps

- `try/catch` doesn't catch async errors
- Fetch 404 ≠ rejected promise
- Throwing strings is bad practice
- `finally` always executes
- Unhandled promise rejections crash Node apps

🟦 FINAL MENTAL MODEL

- Errors stop execution
- `try/catch` is sync only
- Async errors need `.catch` or `await`
- Always throw Error objects
- Let errors bubble when appropriate
- Handle globally as safety net

🛑 STOP HERE

Next **recommended continuation**:

1️⃣ **Memory Management & Garbage Collection (deep internals)**

2️⃣ **Browser Storage APIs (localStorage, sessionStorage)**

3️⃣ **Modules & Import/Export (real-world structure)**

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