

Session 7

Javascript Error Handling

During script execution, errors are inevitable. They can stem from syntax mistakes, invalid user input, network failures, or unforeseen runtime conditions. Without proper error handling, a single failure can crash an entire application or leave users with broken interfaces. This session introduces JavaScript's structured error-handling mechanisms, enabling developers to gracefully catch, diagnose, and recover from runtime exceptions while keeping applications stable and user-friendly.

Objectives

✗ Do not copy. Read for understanding and the viva

- Understand the `try...catch...finally` execution flow
- Implement custom `error` throwing using the `throw` statement
- Utilize JavaScript's built-in `Error` object to diagnose failures
- Apply nested `try...catch` blocks for complex validation scenarios
- Practice robust error handling through hands-on lab exercises

Error Handling Syntax

JavaScript uses a structured block system to manage exceptions:

```
1 try {  
2   // Code that might throw an error  
3 } catch(error) {  
4   // Executes ONLY if an error occurs in try  
5   // `error` is an instance of the built-in Error object  
6 } finally {  
7   // ALWAYS executes, regardless of success or failure  
8   // Ideal for cleanup, closing connections, resetting UI  
9 }
```

JS

Execution Flow

1. `try` block runs.
2. If **no error** → skips `catch` → runs `finally`.
3. If **error occurs** → jumps to `catch` → runs `catch` → runs `finally`.

The `throw` Statement

Allows you to create custom errors or re-throw caught errors:

```
1 throw "Email cannot be empty";      // Throws a string
2 throw 404;                          // Throws a number
3 throw new Error("Custom validation failed"); // Throws an Error object (reco
```

JS

Built-in `Error` Object

When an error is caught, JavaScript provides an error object with two key properties:

Property	Description
<code>error.name</code>	Type of error (e.g., "TypeError", "ReferenceError")
<code>error.message</code>	Human-readable description of what went wrong

Error Types

Error Name	Trigger Condition
SyntaxError	Invalid JS syntax (e.g., missing brackets, 2 = x)
ReferenceError	Accessing an undeclared variable
TypeError	Invalid operation on a value type (e.g., "abc".toUpperCase on undefined)
RangeError	Number outside valid range (e.g., invalid array length)
EvalError	Error inside eval() function
URIError	Malformed URI in encoding/decoding functions

Nested `try...catch` Blocks

For complex validation, you can nest blocks to handle errors at different levels:

```
1 try {
2   // Outer validation
3   try {
4     // Inner risky operation
5     JSON.parse(invalidJSON);
6   } catch(innerErr) {
7     console.warn("Inner failed:", innerErr.message);
8     throw new Error("Data parsing failed"); // Re-throw to outer catch
9   }
10 } catch(outerErr) {
11   console.error("Outer caught:", outerErr.message);
12 } finally {
13   console.log("Cleanup complete");
14 }
```

JS

Best Practices & Common Pitfalls

✗ Do not copy. Read for understanding and the viva

Pitfall	Best Practice
Empty catch <code>{}</code> blocks	Always log or handle the error; never silently swallow failures
Throwing non-Error values	Use <code>throw new Error("msg")</code> for consistent stack traces
Overusing try...catch for control flow	Use conditionals (if/else) for expected logic; reserve try...catch for true exceptions
Ignoring finally for cleanup	Use finally to reset UI states, close streams, or hide loading spinners
Assuming all errors are recoverable	Some errors (e.g., <code>SyntaxError</code>) should be fixed at development time, not caught at runtime

Quick Guide

✖ Do not copy. Read for understanding and the viva

```
1 // 🔍 Basic Error Handling
2 try { riskyOperation(); }
3 catch(err) { console.error(err.name, err.message); }
4 finally { console.log("Runs no matter what"); }
5
6 // 🚀 Custom Error
7 if (!userInput) throw new TypeError("Input is missing");
8
9 // 🔄 Retry Logic (Lab Exercise 4 pattern)
10 function safeMultiply(a, b) {
11   while (true) {
12     try { return primitiveMultiply(a, b); }
13     catch (e) { if (e instanceof MultiplierUnitFailure) continue; else th
14   }
15 }
```

JS

Question 1

Problem Statement

■ Write in lab record

Write an HTML code using JavaScript to add two numbers taken as input from user and display the result of division operation on them i.e. a/b . If an error occurs, show the error. Use error handling concepts in JavaScript.

Preview

✕ Do not copy. Read for understanding and the viva

Divide a / b

Calculate

Waiting for input...

Code

■ Write in lab record

add-division.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Ex1: Division Error Handling</title>
6     <style>
7       body {
8         font-family: sans-serif;
9         padding: 20px;
10        max-width: 400px;
11      }
12    </style>
13  </head>
14  <body>
15    <h2>Divide a / b</h2>
16    <input
17      id="numA"
18      type="number"
19      placeholder="Number a"
20      style="width: 100%; margin-bottom: 8px; padding: 5px"
21    />
22    <input
23      id="numB"
24      type="number"
25      placeholder="Number b"
26      style="width: 100%; margin-bottom: 8px; padding: 5px"
27    />
28    <button id="calcBtn" style="padding: 8px 16px">Calculate</button>
29    <p id="output" style="margin-top: 10px; font-weight: bold"></p>
30
31    <script>
32      document.getElementById("calcBtn").addEventListener("click", ()
33        try {
34          const a = parseFloat(document.getElementById("numA").val
35          const b = parseFloat(document.getElementById("numB").val
36
37          if (isNaN(a) || isNaN(b))
38            throw new TypeError(
39              "Both fields must be valid numbers.",
40            );
41          if (b === 0)
42            throw new RangeError("Division by zero is undefined.
43
```

```
44         document.getElementById("output").textContent =
45             `Result: ${a / b}`;
46     } catch (err) {
47         document.getElementById("output").textContent =
48             `❌ Error (${err.name}): ${err.message}`;
49     }
50 });
51 </script>
52 </body>
53 </html>
```

HTML View

Question 2

Problem Statement

■ Write in lab record

A teacher has created a `gradeLabs` function that verifies if student programming labs work. This function loops over an array of JavaScript objects that should contain a `student` property and `runLab` property. The `runLab` property is expected to be a function containing the student's code. The `runLab` function is called and the result is compared to the expected result. If the result and expected result don't match, then the lab is considered a failure.

Demo

✕ Do not copy. Read for understanding and the viva

Auto-Grader

Run gradeLabs()

Code

■ Write in lab record

grade-lab.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Ex2: Grade Labs</title>
6     <style>
7       body {
8         font-family: sans-serif;
9         padding: 20px;
10      }
11      pre {
12        background: #f4f4f4;
13        padding: 10px;
14        border-radius: 5px;
15      }
16    </style>
17  </head>
18  <body>
19    <h2>Auto-Grader</h2>
20    <button id="gradeBtn">Run gradeLabs()</button>
21    <pre id="results"></pre>
22
23    <script>
24      const students = [
25        { name: "Alice", runLab: () => 2 + 2, expected: 4 },
26        {
27          name: "Bob",
28          runLab: () => {
29            throw new ReferenceError("x is not defined");
30          },
31          expected: 5,
32        },
33        { name: "Charlie", runLab: () => 10 / 2, expected: 6 }, // W
34        { name: "Diana", runLab: () => "5" * 2, expected: 10 }, // T
35      ];
36
37      function gradeLabs(students) {
38        let logs = [];
39        for (const s of students) {
40          try {
41            const actual = s.runLab();
42            if (actual === s.expected)
43              logs.push(`✅ ${s.name}: PASS`);
```

```
44         else
45             logs.push(
46                 `❌ ${s.name}: FAIL (Got ${actual}, Expected
47             );
48         } catch (err) {
49             logs.push(
50                 `⚠️ ${s.name}: EXCEPTION - ${err.name}: ${err.me
51             );
52         }
53     }
54     return logs.join("\n");
55 }
56
57 document
58     .getElementById("gradeBtn")
59     .addEventListener("click", () => {
60         document.getElementById("results").textContent =
61             gradeLabs(students);
62     });
63 </script>
64 </body>
65 </html>
```

HTML View

Question 3

Problem Statement

■ Write in lab record

There is an array of animals. The user is asked to enter the index for the animal they want to see.

1. If the user enters an index that does NOT contain an animal, the code will throw a `TypeError` when `name` is referenced on an undefined value.
2. Update the above program to print out the index the user entered. We want this message to be printed EVERY time the code runs

Demo

✕ Do not copy. Read for understanding and the viva

Animal Lookup

Code

■ Write in lab record

animals.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Ex3: Animal Lookup</title>
6     <style>
7       body {
8         font-family: sans-serif;
9         padding: 20px;
10      }
11      input,
12      button {
13        padding: 8px;
14        margin-right: 5px;
15      }
16    </style>
17  </head>
18  <body>
19    <h2>Animal Lookup</h2>
20    <input id="idxInput" type="number" placeholder="Enter index (0-2)" />
21    <button id="lookupBtn">Find Animal</button>
22    <p id="animalRes" style="margin-top: 10px; font-weight: bold"></p>
23    <p id="finallyMsg" style="color: #555"></p>
24
25    <script>
26      const animals = [
27        { id: 0, name: "Lion" },
28        { id: 1, name: "Elephant" },
29        { id: 2, name: "Tiger" },
30      ];
31
32      document
33        .getElementById("lookupBtn")
34        .addEventListener("click", () => {
35          const idx = parseInt(
36            document.getElementById("idxInput").value,
37          );
38          let animalName = "";
39
40          try {
41            const animal = animals[idx];
42            if (!animal)
43              throw new TypeError(
```

```
44         `Index ${idx} does not contain an animal.` ,
45     );
46     animalName = animal.name;
47     document.getElementById("animalRes").textContent =
48         `🐾 Animal: ${animalName}`;
49     } catch (err) {
50         document.getElementById("animalRes").textContent =
51             `❌ ${err.message}`;
52     } finally {
53         document.getElementById("finallyMsg").textContent =
54             `📝 Checked index: ${idx} (Executed via finally)
55     }
56     });
57     </script>
58 </body>
59 </html>
```

HTML View

Question 4

Problem Statement

■ Write in lab record

Assume you have a function `primitiveMultiply` that in 20 percent of cases multiplies two numbers and in the other 80 percent of cases raises an exception of type `MultiplicatorUnitFailure`. Write a function that wraps this clunky function and just keeps trying until a call succeeds, after which it returns the result. Make sure you handle only the exceptions you are trying to handle.

Demo

✗ Do not copy. Read for understanding and the viva

Reliable Multiply

Input: **6 × 7** (20% success rate per attempt)

Run Wrapper

Code

■ Write in lab record

primitive-multipln.html

```
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <title>Ex4: Retry Multiplication</title>
6     <style>
7       body {
8         font-family: sans-serif;
9         padding: 20px;
10      }
11      button {
12        padding: 10px 20px;
13      }
14    </style>
15  </head>
16  <body>
17    <h2>Reliable Multiply</h2>
18    <p>Input: <strong>6 × 7</strong></p>
19    <button id="multiplyBtn">Run Wrapper</button>
20    <p id="res" style="margin-top: 10px; font-weight: bold"></p>
21
22    <script>
23      class MultiplierUnitFailure extends Error {
24        constructor(msg) {
25          super(msg);
26          this.name = "MultiplierUnitFailure";
27        }
28      }
29
30      function primitiveMultiply(a, b) {
31        if (Math.random() < 0.2) return a * b;
32        else
33          throw new MultiplierUnitFailure(
34            "Random failure (80% chance)",
35          );
36      }
37
38      function reliableMultiply(a, b) {
39        let attempts = 0;
40        while (true) {
41          attempts++;
42          try {
43            return { result: primitiveMultiply(a, b), attempts }
```

```
44         } catch (e) {
45             if (!(e instanceof MultiplierUnitFailure)) throw
46         }
47     }
48 }
49
50 document
51     .getElementById("multiplyBtn")
52     .addEventListener("click", () => {
53         const { result, attempts } = reliableMultiply(6, 7);
54         document.getElementById("res").textContent =
55             `✅ Result: ${result} (Took ${attempts} attempt${att
56             });
57     </script>
58 </body>
59 </html>
```

HTML View

[✎ Edit this page](#)

[< Previous](#)
Session 6

Session 8 [Next >](#)