JavaScript

Exception Handling in JavaScript

In JavaScript, **exception handling** is a mechanism to handle **runtime errors** (unexpected situations) so that the program does not crash suddenly.

An **exception** is an error that occurs during execution and disrupts the normal flow of the program.

Example:

- Accessing an undefined variable
- Dividing by zero (logic error)
- JSON parsing errors
- Network request failures
- Wrong input types

Why Exception Handling is Needed?

Without exception handling, a program may stop immediately when an error occurs.

Benefits:

- Prevents program crash
- Improves user experience
- Helps in debugging
- Allows graceful recovery
- Useful in real-world apps (forms, API calls, file reading, etc.)

Types of Errors in JavaScript

(A) Syntax Error

Occurs when code structure is incorrect.

```
console.log("Hello"
```

- *Missing closing bracket → syntax error.*

(B) Runtime Error

Occurs during execution.

```
console.log(x);  // x is not defined
```

(C) Logical Error

Program runs but gives wrong output.

```
let sum = 10 - 5;  // should be +
but written -
```

4. The `try...catch` Block

*JavaScript provides **try-catch** for handling exceptions.*

Syntax:

```
try {
    // risky code
} catch (error) {
    // error handling code
}
```

Example:

```
try {
  let result = 10 / 0;
  console.log(result);
} catch (err) {
  console.log("Error occurred:", err);
}
```

- Here no exception happens, because division by 0 in JS returns `Infinity`.

Example with real exception:

```
try {
  console.log(a); // a is not defined
} catch (err) {
  console.log("Caught Error:",
err.message);
}
```

Output:

```
Caught Error: a is not defined
```

5. *Understanding the `error` Object*

Inside `catch(error)` JavaScript provides an ***Error object****.*

Common properties:

- *`error.name` → type of error*
- *`error.message` → message of error*
- *`error.stack` → stack trace (debugging)*

Example:

```
try {
  let x = y + 10;
} catch (e) {
  console.log("Name:", e.name);
  console.log("Message:", e.message);
}
```

6. The `finally` Block

`finally` always executes whether error occurs or not.

Syntax:

```
try {
  // code
} catch (err) {
  // handle error
} finally {
  // always runs
}
```

Example:

```
try {
  console.log("Try block executed");
  throw new Error("Custom Error!");
} catch (e) {
  console.log("Catch block:", e.message);
} finally {
  console.log("Finally block executed");
}
```

Output:

```
Try block executed
Catch block: Custom Error!
Finally block executed
```

7. Throwing Custom Exceptions (`throw`)

We can create our own errors using `throw`.

Example:

```
let age = -5;

try {
  if (age < 0) {
    throw "Age cannot be negative!";
  }
  console.log("Age is valid");
} catch (err) {
  console.log("Error:", err);
}
```

Throwing Error Objects (Recommended)

```
let marks = 120;

try {
  if (marks > 100) {
    throw new Error("Marks cannot be more
than 100!");
  }
} catch (e) {
  console.log(e.message);
}
```

8. Real-Life Example: Form Validation

```
function validateEmail(email) {
  try {
    if (!email.includes("@")) {
      throw new Error("Invalid Email Format");
    }
    console.log("Valid Email");
  } catch (e) {
    console.log("Validation Error:", e.message);
  }
}

validateEmail("abcgmail.com");
```

9. Nested try-catch

You can use try-catch inside another try-catch.

```
try {
  try {
    JSON.parse("{name: 'Ashwini'}"); // invalid JSON
  } catch (innerErr) {
    console.log("Inner Error:", innerErr.message);
  }
} catch (outerErr) {
  console.log("Outer Error:", outerErr.message);
}
```

10. Handling JSON Parsing Errors

```
let data = '{"name":"Ashwini"}';

try {
  let obj = JSON.parse(data);
  console.log(obj.name);
} catch (e) {
  console.log("JSON Error:", e.message);
}
```

11. Exception Handling in Async JavaScript

(A) *Using `try-catch` with `async/await`*

```
async function fetchData() {
  try {
    let res = await fetch("https://wrong-url.com");
    let data = await res.json();
    console.log(data);
  } catch (e) {
    console.log("Fetch Error:", e.message);
  }
}

fetchData();
```

✔ *`try-catch` works properly with `await`.*

(B) *Using `.catch()` with Promises*

```
fetch("https://wrong-url.com")
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.log("Promise
Error:", err.message));
```

Can `finally` execute without `catch`?

Yes.

```
try {
  console.log("Try");
} finally {
  console.log("Finally");
}
```

Summary Table

Keyword	Purpose
`try`	risky code
`catch`	handles exception
`finally`	always executes
`throw`	manually create exception

```
function divide(a, b) {
  try {
    if (b === 0) {
      throw new Error("Division by zero is not allowed!");
    }
    console.log("Result:", a / b);
  } catch (e) {
    console.log("Error:", e.message);
  } finally {
    console.log("Operation Completed");
  }
}

divide(10, 2);
divide(10, 0);
```

Summary Table

Error Type	Meaning
`Error`	General error
`SyntaxError`	Wrong syntax in code
`ReferenceError`	Variable not defined
`TypeError`	Wrong data type usage
`RangeError`	Value out of range
`URIError`	Bad URI encoding/decoding
`EvalError`	Related to eval (rare)
`AggregateError`	Multiple errors together
`DOMException`	Browser API errors