**try...catch in JavaScript**

The try...catch statement in JavaScript is used to **handle errors** gracefully. It allows the program to continue executing instead of crashing when an error occurs.

**Syntax:**

try {

// Code that may cause an error

} catch (error) {

// Code to handle the error

}

**Example 1: Handling an Error**

try {

let x = y + 10; // 'y' is not defined (ReferenceError)

console.log(x);

} catch (error) {

console.log("An error occurred:", error.message);

}

**Explanation:**

* The try block contains code that may cause an error (y is not defined).
* When an error occurs, the control is transferred to the catch block.
* The error message is displayed instead of stopping the entire script.

**Example 2: Using finally (Runs Regardless of Error)**

try {

let num = 10 / 0; // This does not cause an error (Infinity)

console.log("Result:", num);

} catch (error) {

console.log("An error occurred:", error.message);

} finally {

console.log("This runs whether an error occurs or not.");

}

**Explanation:**

* The finally block always executes, regardless of whether an error occurs or not.
* It is useful for clean-up operations like closing files or resetting variables.

**Example 3: Custom Error Handling**

function divide(a, b) {

try {

if (b === 0) {

throw new Error("Division by zero is not allowed.");

}

console.log("Result:", a / b);

} catch (error) {

console.log("Error:", error.message);

}

}

divide(10, 0); // Error: Division by zero is not allowed.

divide(10, 2); // Result: 5

**Explanation:**

* The throw statement generates a custom error if b is 0.
* The catch block catches and displays the custom error message.

**Key Points:**

1. **try Block**: Contains the code that might throw an error.
2. **catch Block**: Handles the error and prevents program termination.
3. **finally Block (Optional)**: Runs always, regardless of an error.
4. **throw Statement**: Used to create custom errors.

**Handling Errors in Asynchronous Code**

When working with asynchronous operations (like API calls, database queries, or file operations), errors can occur. try...catch ensures that your program handles these errors gracefully.

**Example 1: Handling Errors in an Async Function**

async function fetchData() {

try {

let response = await fetch("https://jsonplaceholder.typicode.com/invalid-url"); // Invalid URL

if (!response.ok) {

throw new Error(`HTTP error! Status: ${response.status}`);

}

let data = await response.json();

console.log(data);

} catch (error) {

console.log("Error fetching data:", error.message);

}

}

fetchData();

**Explanation:**

1. fetch() is an asynchronous function that retrieves data from a URL.
2. If the response status is not OK (e.g., 404 Not Found), we **throw** an error manually.
3. The catch block catches any errors (like network issues or invalid URLs) and logs them.

**Example 2: Using try...catch Inside async/await with Promises**

function getUserData() {

return new Promise((resolve, reject) => {

setTimeout(() => {

let success = false; // Simulating failure

if (success) {

resolve({ name: "Alice", age: 25 });

} else {

reject(new Error("Failed to fetch user data"));

}

}, 2000);

});

}

async function displayUser() {

try {

let user = await getUserData();

console.log("User Data:", user);

} catch (error) {

console.log("Error:", error.message);

}

}

displayUser();

**Explanation:**

1. getUserData() is a function that returns a **Promise**.
2. If success is false, the Promise is **rejected**, triggering an error.
3. The async function displayUser() uses await to wait for the Promise.
4. If an error occurs, the catch block handles it gracefully.

**Example 3: Using finally with Async/Await**

async function fetchDataWithFinally() {

try {

let response = await fetch("https://jsonplaceholder.typicode.com/posts/1");

let data = await response.json();

console.log("Fetched Data:", data);

} catch (error) {

console.log("Error:", error.message);

} finally {

console.log("Fetch operation complete.");

}

}

fetchDataWithFinally();

**Explanation:**

* The finally block **always runs**, whether the fetch is successful or not.
* It's useful for **clean-up actions**, like closing connections or showing UI messages.

**Key Takeaways**

✅ try...catch works with **async/await** to handle errors effectively.  
✅ Always use catch to prevent unhandled promise rejections.  
✅ Use finally for **cleanup** operations that should run no matter what.