JS Error Handling: Throw, Try, and Catch

Managing Errors in JavaScript

What is Error Handling?

Definition: Error handling is the process of anticipating, detecting, and resolving programming errors or exceptions.

Importance: Ensures that your code runs smoothly and can handle unexpected situations gracefully.

Common JavaScript Errors

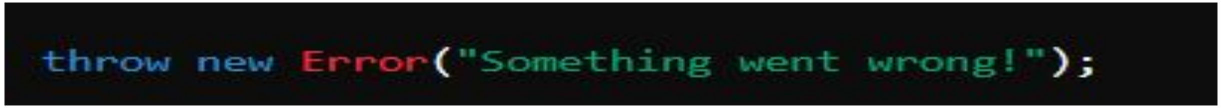
- **Types of Errors**:
    - **Syntax Errors**: Mistakes in the code syntax.
    - **Reference Errors**: Trying to access variables that are not declared.
    - **Type Errors**: Performing operations on incompatible types.
    - **Range Errors**: Using numbers outside of allowed ranges.
    - **URI Errors**: Errors in encoding/decoding URIs.

Throwing Errors

The `throw` Statement

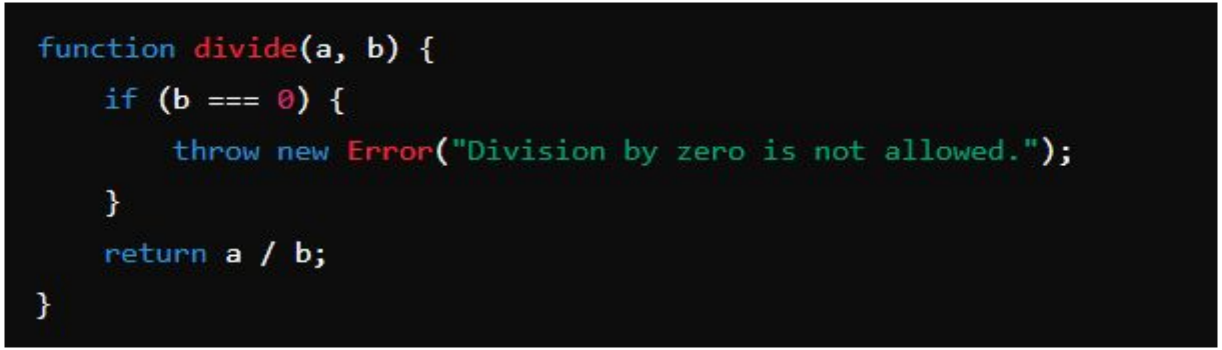
- **Purpose**: Manually trigger an error.

Syntax :



```
throw new Error("Something went wrong!");
```

Example :



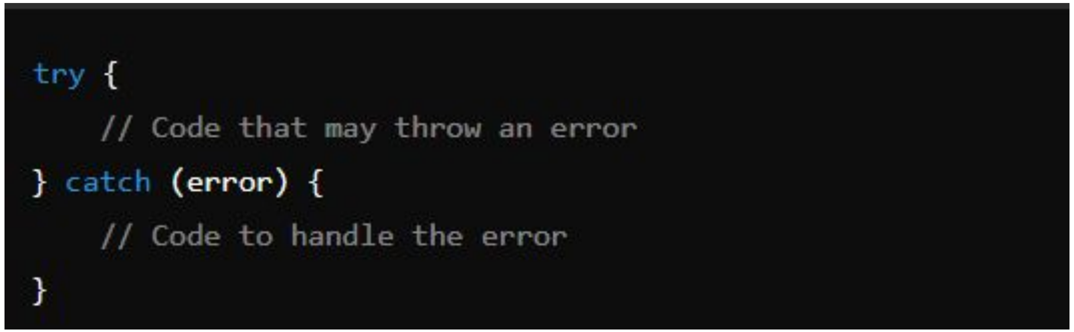
```
function divide(a, b) {
    if (b === 0) {
        throw new Error("Division by zero is not allowed.");
    }
    return a / b;
}
```

Try-Catch Error Handling

The `try-catch` Block

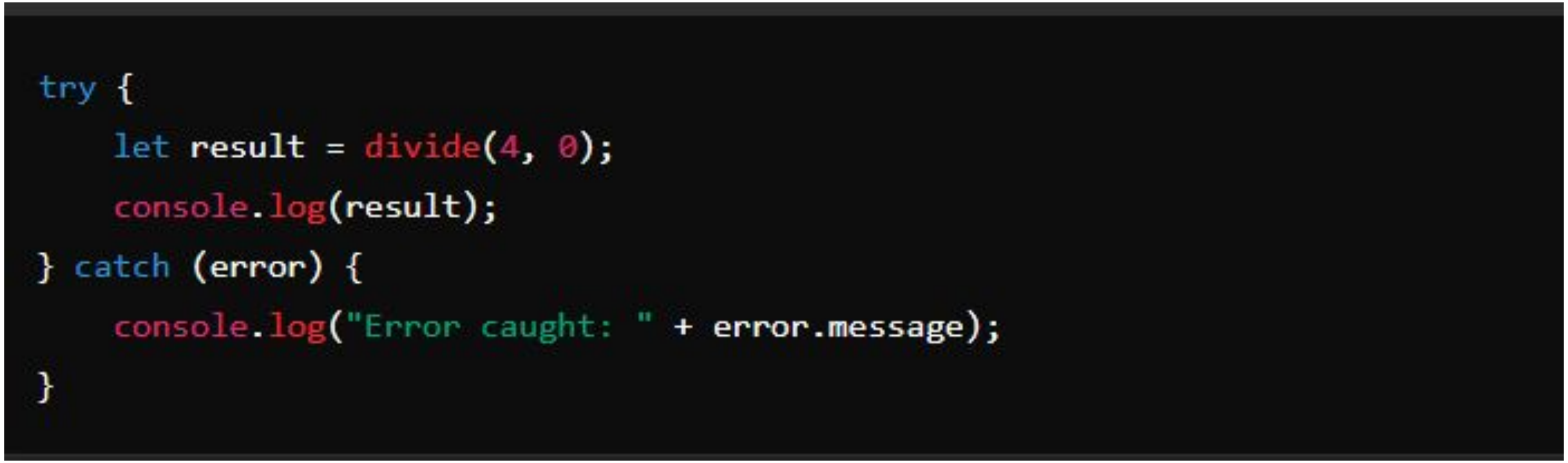
- **Purpose**: To handle errors gracefully and prevent them from stopping the program.

  Syntax :



```
try {
    // Code that may throw an error
} catch (error) {
    // Code to handle the error
}
```

Example: Basic Try-Catch



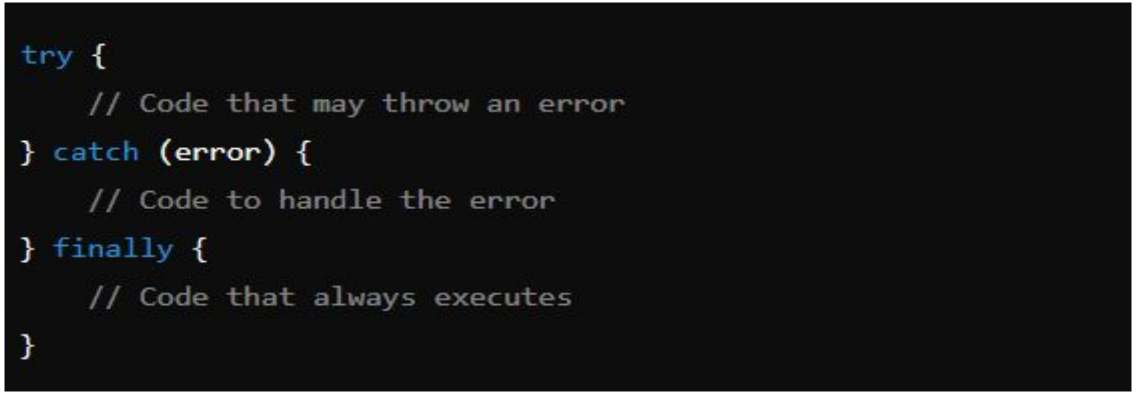
```
try {
    let result = divide(4, 0);
    console.log(result);
} catch (error) {
    console.log("Error caught: " + error.message);
}
```

Advanced Error Handling

Using `finally`

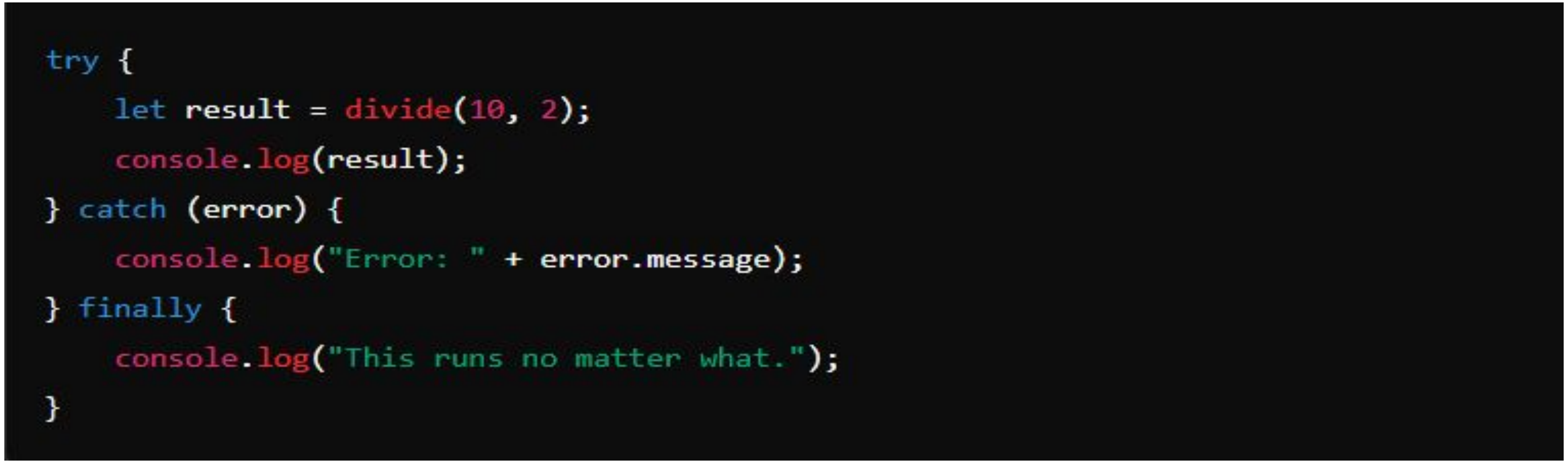
- **Purpose**: The `finally` block executes code after `try` and `catch`, regardless of whether an error occurred.

Syntax ::



```
try {
    // Code that may throw an error
} catch (error) {
    // Code to handle the error
} finally {
    // Code that always executes
}
```

Example : Advanced Error Handling



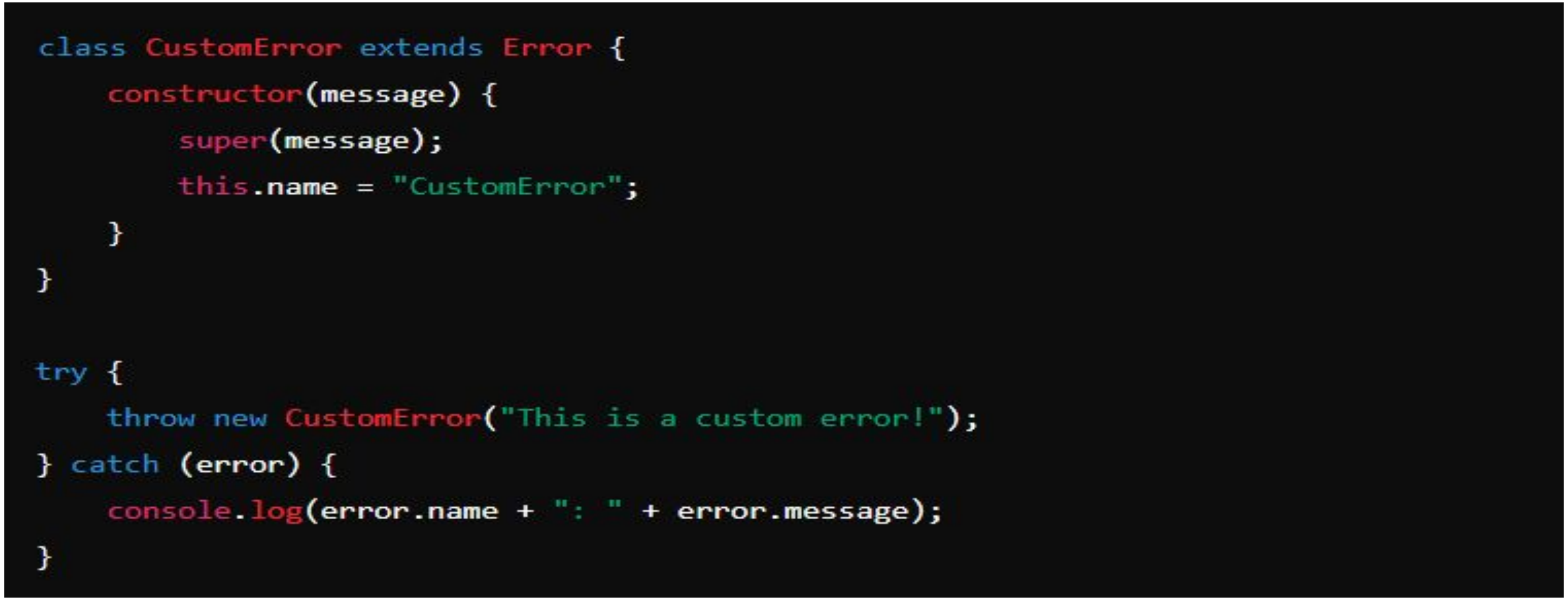
```javascript
try {
    let result = divide(10, 2);
    console.log(result);
} catch (error) {
    console.log("Error: " + error.message);
} finally {
    console.log("This runs no matter what.");
}
```

Custom Error Types

Creating Custom Errors

- **Why Create Custom Errors?**: To provide more meaningful error messages specific to your application.

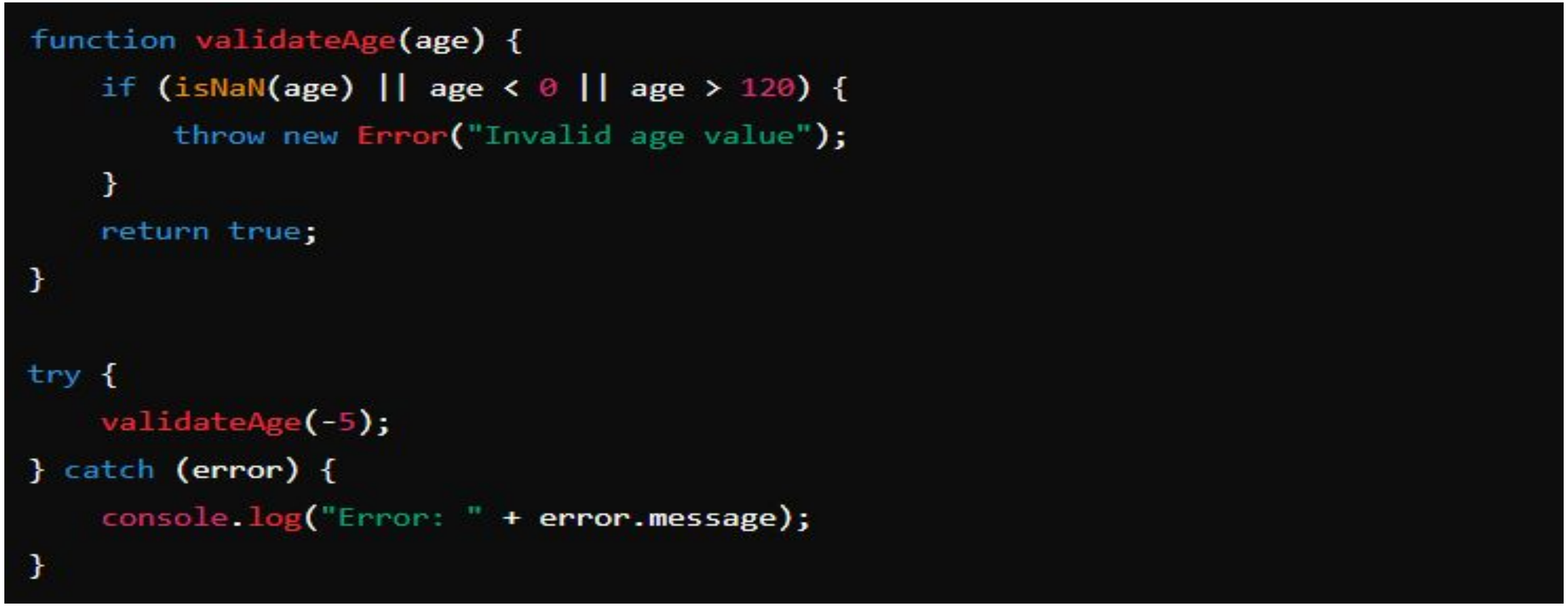
Example : Custom Errors



```
class CustomError extends Error {
    constructor(message) {
        super(message);
        this.name = "CustomError";
    }
}

try {
    throw new CustomError("This is a custom error!");
} catch (error) {
    console.log(error.name + ": " + error.message);
}
```

Example : Validating User Input



```
function validateAge(age) {
    if (isNaN(age) || age < 0 || age > 120) {
        throw new Error("Invalid age value");
    }
    return true;
}

try {
    validateAge(-5);
} catch (error) {
    console.log("Error: " + error.message);
}
```

Summary

Key Takeaways:

- `throw` is used to manually trigger errors.
- `try-catch` blocks are essential for managing errors without crashing the program.
- `finally` can be used for cleanup tasks.
- Custom errors can make debugging easier.