JavaScript
Error Handling

In Detail

JS

Error Handling

Error handling in JS involves dealing with **runtime errors and exceptions** that may occur **during the execution** of your code.

There are **several techniques** and mechanisms available to handle errors effectively.

- **try-catch** Statement
- **throw** Statement:
- **finally** Block
- **Error** Objects

try-catch | Statement

The **try** block contains the code that may **throw an exception**, and the **catch** block is where you can **handle the exception.**

```
try {
  // Code that may throw an exception
} catch (error) {
  // Handling the exception
}
```

throw | Statement

The **throw** statement is used to **manually throw a custom error**.

It allows you to create and throw your **own errors,** which can be caught and handled using **try-catch**

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

try {
  let result = divide(10, 0);
  console.log(result);
} catch (error) {
  console.log("An error occurred:", error.message);
}

// An error occurred: Division by zero is not allowed.
```

finally | Statement

The **finally** block is an **optional block** that follows the **try-catch** block. The code within the finally block **executes regardless** of whether an exception occurred or not.

It is commonly used to perform cleanup tasks or release resources.

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

Error Objects

JavaScript provides **built-in error** objects that represent different types of errors.

These objects, such as Error, TypeError, ReferenceError, etc., have properties like **message** and **name**, which can provide valuable information about the error.

You can also create **custom error** objects by extending the Error object.

Error Objects

- **Error**: The base error object from which other error objects inherit. It provides a name and message property that contain information about the error.

- **TypeError**: Represents an error when a value is not of the expected type.

- **SyntaxError**: Occurs when there is a syntax error in the code.

- **ReferenceError**: Indicates an invalid reference, typically when trying to access an undeclared variable or non-existing object.

Error Objects

- **RangeError**: Occurs when a numeric value is outside the range of acceptable values.
- **EvalError**: Represents an error that occurs during the eval() function's execution.
- **URIError**: Indicates an error when working with malformed URIs.

Error Objects | Example

```
try {
  throw new TypeError("Invalid argument type");
} catch (error) {
  console.log(error.name);
  // Output: TypeError

  console.log(error.message);
  // Output: Invalid argument type
}
```