**Question 1: What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.**

**Error Handling in JavaScript**

Error handling in JavaScript is a mechanism to catch and handle runtime errors to prevent the script from crashing unexpectedly. JavaScript provides the try...catch...finally statement to handle errors gracefully.

**Try, Catch, and Finally Blocks**

1. **try block**: The code that might throw an error is written inside the try block.
2. **catch block**: If an error occurs in the try block, it is caught in the catch block, where we can handle the error.
3. **finally block**: The finally block always executes after try and catch, regardless of whether an error occurred or not. It is usually used for cleanup operations.

**Example: Using try, catch, and finally**

function divideNumbers(a, b) {

try {

if (b === 0) {

throw new Error("Cannot divide by zero!");

}

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

} catch (error) {

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

} finally {

console.log("Execution completed.");

}

}

// Test cases

divideNumbers(10, 2);

divideNumbers(5, 0);

**Explanation**

* If b is 0, an error is thrown and caught in the catch block.
* Whether an error occurs or not, the finally block always executes.

**Question 2: Why is error handling important in JavaScript applications?**

**Importance of Error Handling in JavaScript Applications**

Error handling is crucial in JavaScript applications to ensure smooth execution and a better user experience. Here are some key reasons why it is important:

1. **Prevents Application Crashes**
   * Errors can cause an application to stop unexpectedly. Proper error handling prevents this by catching errors and allowing the program to continue running.
2. **Enhances User Experience**
   * Instead of showing technical error messages, user-friendly error messages can be displayed, improving the overall user experience.
3. **Debugging and Troubleshooting**
   * Helps developers identify and fix issues efficiently by providing meaningful error messages and stack traces.
4. **Ensures Code Reliability**
   * Proper handling of exceptions ensures that critical parts of an application, such as authentication, payments, or form submissions, function correctly even if an error occurs elsewhere.
5. **Security Benefits**
   * Prevents exposure of sensitive system information to users, reducing potential security vulnerabilities.
6. **Improves Maintainability**
   * Well-structured error handling makes it easier to maintain and update the code, especially in large-scale applications.
7. **Asynchronous Operations Handling**
   * JavaScript applications often rely on asynchronous operations (e.g., API calls). Handling errors in promises and async/await prevents failures from propagating and breaking the application.

Example: Handling API Errors Gracefully

async function fetchData() {

try {

let response = await fetch("https://api.example.com/data");

if (!response.ok) {

throw new Error("Failed to fetch data");

}

let data = await response.json();

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

} catch (error) {

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

}

}

fetchData();

* If the API call fails, the catch block prevents the application from crashing and logs an appropriate error message.