Day 6 → JavaScript Error Handling and Debugging

I. Introduction to JavaScript Error Handling

A. What are JavaScript errors?

- Explanation: JavaScript errors occur when the interpreter encounters code that it cannot execute properly.

B. Common types of JavaScript errors

- Explanation: Common types include SyntaxError, ReferenceError, TypeError, and RangeError, among others.

C. Importance of error handling in JavaScript

- Explanation: Proper error handling ensures that errors are caught, logged, and appropriately dealt with, improving code reliability and user experience.

II. Debugging Techniques

A. Console logging

1. Using console.log()

- Explanation: console.log() is used to print values and debug information to the browser console.

- Example:

```javascript
let x = 5;
console.log(x); // Output: 5
```

2. Printing variables and values

- Explanation: Printing variables helps verify their values during code execution.

- Example:

```javascript
```

```
let x = 5;
console.log("The value of x is:", x); // Output: The value of x is: 5
```

3. Debugging with console.assert()
   - Explanation: console.assert() checks if a condition is true and logs an error if it's false.
   - Example:
   ```javascript
   let x = 5;
   console.assert(x > 10, "x should be greater than 10"); // Assertion failed, error message logged
   ```

B. Breakpoints and Step-through Debugging
1. Setting breakpoints
   - Explanation: Breakpoints pause code execution at specific points, allowing inspection of variables and code flow.
2. Executing code step-by-step
   - Explanation: Step-by-step execution helps identify the cause of errors and inspect variable values at each step.
3. Inspecting variables and their values
   - Explanation: Inspecting variables helps understand their current state and detect issues.

C. Error Tracking and Reporting
1. Handling uncaught errors
   - Explanation: Uncaught errors stop code execution unless they are handled.

2. Try-Catch statements
   - Explanation: Try-Catch blocks allow catching and handling errors gracefully.
   - Example:
   ```javascript
   try {
     let x = 10;
     let y = x.toUpperCase(); // Error: toUpperCase is not a function
   } catch (error) {
     console.log("An error occurred:", error.message);
   }
   ```
3. Error objects and properties
   - Explanation: Error objects contain information about the error, such as name, message, and stack trace.
   - Example:
   ```javascript
   try {
     // Some code that may throw an error
   } catch (error) {
     console.log("Error name:", error.name);
     console.log("Error message:", error.message);
     console.log("Stack trace:", error.stack);
   }
   ```

III. Common JavaScript Errors and their Solutions
A. Syntax errors

- Explanation: Syntax errors occur when code violates the rules of the JavaScript language.
- Example:
  ```javascript
  let x = 5;
  console.log(x // Missing closing parenthesis
  ```

B. Reference errors
- Explanation: Reference errors happen when code tries to access a variable or function that doesn't exist.
- Example:
  ```javascript
  let x = 5;
  console.log(y); // Error: y is not defined
  ```

C. Type errors
- Explanation: Type errors occur when a value is of the wrong type for the operation being performed.
- Example:
  ```javascript
  let x = 5;
  console.log(x.toUpperCase()); // Error: toUpperCase is not a function
  ```

D. Handling asynchronous errors (promises, async/await)
- Explanation: Asynchronous errors require specific error-handling techniques when using promises or async/await.
- Example with promises:
  ```javascript
  ```

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Request failed');
    }
    return response.json();
  })
  .catch(error => {
    console.log('Error:', error.message);
  });
```

- Example with async/await:
```javascript
async function fetchData() {
  try {
    const response = await
fetch('https://api.example.com/data');
    if (!response.ok) {
      throw new Error('Request failed');
    }
    const data = await response.json();
    console.log('Data:', data);
  } catch (error) {
    console.log('Error:', error.message);
  }
}
```

IV. Project: Debugging and Fixing Errors in JavaScript Code

- Explanation: The project involves identifying and fixing intentional errors in a given JavaScript code snippet.
- Example Project Code:
  ```javascript
  let x = 10; // Error: Missing semicolon at the end
  console.log(x);

  let y = 20;
  console.log(y);

  let result = x + y;
  console.log(result);

  console.log(z); // Error: z is not defined
  ```