Defensive Programming

HyperionDev

Muhammad Zahir
Junejo

Lecture – Housekeeping

- ❑ The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
    - ❑ Please review Code of Conduct (in Student Undertaking Agreement) if unsure
- ❑ No question is daft or silly - **ask them!**
- ❑ Q&A session at the end of the lesson, should you wish to ask any follow-up questions.
- ❑ Should you have any questions after the lecture, please schedule a mentor session.
- ❑ For all non-academic questions, please submit a query: www.hyperiondev.com/support

Lecture Objectives

1. Understand Defensive Programming basics and its significance.
2. Learn the key principles of Defensive Programming.
3. Explore techniques for input validation in JavaScript.
4. Grasp error handling strategies using try-catch blocks.

Introduction to Defensive Programming

- ❑ Definition: Defensive programming is a coding style focused on anticipating and guarding against potential errors and issues.
- ❑ Importance: Protects your code from unexpected inputs and chaotic environments.
- ❑ Analogy: Think of it as adding seat belts and airbags to your code.
- ❑ JavaScript runs in unpredictable environments: browsers, servers, IoT devices, etc.
- ❑ User input, network delays, and hardware failures can disrupt your code.
- ❑ Examples: Slow network, unexpected user interactions, and server outages.

Defensive Programming Principles

- Principle 1: Assume Nothing
  - Never assume inputs will be correct or functions will succeed.
  - Always validate, sanitize, and handle errors.

```
function divide(a, b) {
  if (typeof a !== 'number' || typeof b !== 'number') {
    throw new Error('Both arguments must be numbers');
  }
  if (b === 0) {
    throw new Error('Division by zero is not allowed');
  }
  return a / b;
}
```

Defensive Programming Principles

- Principle 2: Fail Fast
  - Identify issues as soon as they occur, don't let them propagate.
  - Use early checks and validations to catch errors.

```
function findUser(userId) {
  if (!userId) {
    throw new Error('Invalid user ID');
  }
  // ... rest of the function ...
}
```

Defensive Programming Principles

- ❑ Principle 3: Use Proper Validation
    - ❑ Validate user inputs, API responses, and data from external sources.
    - ❑ Sanitize and validate data before processing.
- ❑ Principle 4: Handle Errors Gracefully
    - ❑ Use try-catch blocks to catch and handle exceptions.
    - ❑ Provide meaningful error messages for debugging.

```
try {
  // Risky code
} catch (error) {
  console.error('An error occurred:', error.message);
}
```

Defensive Programming Principles

- Principle 5: Fail-Safe Defaults
  - Provide default values or fallback mechanisms when possible.
  - Prevent code from breaking if expected data is missing.

```
    function getUserProfile(userId) {
  if (!userId) {
    userId = 'defaultUserId';
  }
  // ... rest of the function ...
}
```

Hyperiondev

Input Validation in JavaScript

- Numeric Input:

```
function isNumeric(value) {
  return !isNaN(value);
}
const numericInput = '42';
console.log(isNumeric(numericInput) ? 'Valid input: ' + numericInput : 'Invalid input');
```

- Non-Empty Strings:

```
function isNonEmptyString(input) {
  return typeof input === 'string' && input.trim() !== '';
}
const userInput = '  Hello, World!  ';
console.log(isNonEmptyString(userInput) ? 'Valid input: ' + userInput.trim() : 'Invalid
input');
```

Types of Errors

- ❑ Syntax Errors: Occur during code parsing due to incorrect syntax.
- ❑ Reference Errors: Happen when trying to access undeclared variables or properties.
- ❑ Type Errors: Occur when incompatible operations are performed, like calling a non-function or accessing properties on undefined values.
- ❑ Examples:

```
function add(a, b { // Syntax Error: Missing closing parenthesis
  return a + b;
}
console.log(x); // Reference Error: Variable 'x' is not defined

const arr = undefined; // Type Error: Cannot read property 'length' of undefined
console.log(arr.length);
```

The try...catch Block

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

try {
  const result = 10 / 0; // Division by zero
  console.log(result);
} catch (error) {
  console.error('An error occurred:', error.message);
}
```

Questions and Answers

Thank You!