7-Day JavaScript Advanced Concepts Learning Plan (1 Hour Daily)

This plan focuses on advanced JavaScript topics while maintaining a practical, hands-on approach. Each day includes theory, practical tasks, and key advanced topics.

Day 1: Advanced Functions

Topics to Cover:

- **Function Closures**:
  - Understand lexical scoping and closures.
- **Higher-Order Functions**:
  - Functions that accept other functions as arguments or return functions.
- **Callback Functions**:
  - Using callbacks in asynchronous operations.
- **IIFE (Immediately Invoked Function Expressions)**.

Practical Task:

1. Write a closure to create a private counter (increment, reset).
2. Implement a higher-order function for filtering numbers from an array.
3. Write an IIFE to log a welcome message.

Resources:

- MDN: Closures

Day 2: Advanced Objects

Topics to Cover:

- Object Methods:
  - `Object.keys()`, `Object.values()`, `Object.entries()`.
- Prototypes and Inheritance:
  - Prototype chaining and `Object.create()`.

- `this` keyword:
  - How `this` works in different contexts (global, object, function, arrow function).
- Object Destructuring and Spread/Rest Operators.

Practical Task:

1. Create an object representing a user and manipulate its properties using `Object.keys()` and `Object.values()`.
2. Create a prototype chain for an object.
3. Use destructuring to extract specific properties from an object.

Resources:

- MDN: Prototypes

Day 3: Asynchronous JavaScript

Topics to Cover:

- The Event Loop and Call Stack.
- Promises:
  - Chaining with `.then()`, `.catch()`.
- Async/Await:
  - Writing asynchronous code in a cleaner way.
- Fetch API:
  - Making HTTP requests and handling responses.

Practical Task:

1. Write a promise that resolves or rejects based on a condition.
2. Use `async/await` to fetch and display data from a public API.
3. Visualize the Event Loop with a simple `setTimeout()` example.

Resources:

- MDN: Promises
- Event Loop Visualization

Day 4: ES6+ and Advanced Syntax

Topics to Cover:

- Modules:
  - `import` and `export` syntax.
- Default and Named Exports.
- Dynamic Imports.
- JavaScript Iterators and Generators.
- Symbol and `for...of`.

Practical Task:

1. Create and use a module system with `import/export`.
2. Write a generator function to yield Fibonacci numbers.
3. Use `for...of` to iterate through an iterable (array, map).

Resources:

- MDN: Modules

Day 5: Error Handling and Debugging

Topics to Cover:

- Try-Catch Blocks:
  - Handling exceptions.
- Custom Errors:
  - Creating custom error classes.
- Debugging Techniques:
  - Using browser DevTools.
- Performance Profiling:
  - Measuring code execution time.

Practical Task:

1. Create a function that throws an error when invalid data is passed.
2. Use `try-catch` to handle the error gracefully.
3. Use browser DevTools to debug a sample script with errors.

Resources:

- MDN: Error Handling

Day 6: Advanced Data Structures

Topics to Cover:

- Sets and Maps:
  - Operations and use cases.
- WeakMap and WeakSet:
  - Memory-sensitive collections.
- Working with Typed Arrays.
- Custom Iterables:
  - Creating custom data structures.

Practical Task:

1. Implement a Set for unique elements and perform operations like union, intersection.
2. Use a Map to store key-value pairs, then iterate through it.
3. Create a custom iterable object.

Resources:

- MDN: Set
- MDN: Map

Day 7: JavaScript Patterns and Performance

Topics to Cover:

- Design Patterns:
  - Singleton, Factory, Module, Observer Patterns.
- Functional Programming:
  - Immutability, Pure Functions, Currying.
- Memory Management:
  - Garbage Collection.
- Performance Optimization:

    - Debouncing and Throttling.
- Web Workers:
    - Running scripts in the background.

Practical Task:

1. Implement a Singleton pattern for an app configuration object.
2. Use debouncing to optimize a search input field.
3. Create a basic Web Worker to perform a time-intensive calculation.

Resources:

- JavaScript Design Patterns
- MDN: Web Workers

Bonus (Optional Topics)

- **JavaScript Frameworks**: Introduction to React, Vue, or Angular.
- **Unit Testing**: Jest or Mocha for testing JavaScript code.
- **Node.js**: Learn backend development with JavaScript.

This advanced plan ensures you learn practical implementations and deep dive into JavaScript concepts. Let me know if you'd like resources or more examples for specific days!