JavaScript (ES6) Advanced Concepts Assignment 4

Topics Covered

- call, apply, bind
- Closures
- Object-Oriented Programming (OOP) in JavaScript
- Asynchronous JavaScript

Assignment Overview

This assignment focuses on **how JavaScript actually works at runtime**, especially around function context, memory, objects, and asynchronous behavior.

You are expected to **think like the JavaScript engine**: write code, run it multiple times, change values, observe execution order, and learn from unexpected results.

The tasks are intentionally designed so that **AI-generated answers or copied solutions will fail** unless the code is genuinely executed, modified, and understood.

Learning Objectives

By completing this assignment, students will be able to:

- Control function context using `call`, `apply`, and `bind`
- Understand and prove how closures work in memory
- Design object-oriented systems using JavaScript
- Reason about synchronous vs asynchronous execution
- Predict execution order using the event loop

Rules & Academic Integrity (Strict)

- Use **vanilla JavaScript (ES6+) only**
- No frameworks, libraries, or async helpers
- Code must be written, executed, and modified by **you**
- Each task must produce **unique runtime output**
- Identical logic or predictable outputs across submissions will be rejected

AI Usage Policy: This assignment requires real execution evidence. Submissions that appear auto-generated without runtime experimentation will be marked invalid.

Section 1: call, apply & bind (Context Control)

Task 1.1 – Losing and Fixing `this`

Create an object with a method that uses `this`.

Steps:

- Call the method normally
- Assign the method to a variable and call it
- Fix the broken context using `call`

Log outputs at each step and observe differences.

Task 1.2 – call vs apply (Argument Handling)

Create a function that accepts multiple parameters.

Invoke it using:

- `call`
- `apply`

Use different argument values and log results.

Mandatory:

- Change argument count at least once

Task 1.3 – bind and Delayed Execution (Hard)

Create a function that:

- Uses `this`
- Is executed later using a timer

Use `bind` to permanently attach context and compare behavior.

Section 2: Closures (Memory & Scope)

Task 2.1 – Closure Proof

Write a function that:

- Declares a private variable
- Returns an inner function that uses it

Call the inner function multiple times and observe memory behavior.

Task 2.2 – Independent Closures

Create a function factory that:

- Produces multiple closure instances
- Each instance maintains independent state

Log outputs to prove isolation.

Section 3: OOP with JavaScript

Task 3.1 – Constructor Functions

Build an object system using:

- Constructor functions
- Shared methods via prototype

Create multiple instances and show shared vs unique properties.

Task 3.2 – ES6 Classes (Comparison)

Rewrite Task 3.1 using ES6 `class` syntax.

Log similarities and differences through output behavior.

Task 3.3 – Inheritance & Method Overriding (Hard)

Create a base class and at least one child class.

Override a method and use `super` correctly.

Demonstrate polymorphism using runtime logs.

Section 4: Asynchronous JavaScript

Task 4.1 – Sync vs Async Execution Order

Write code that mixes:

- Normal functions
- Timers

Log output order and explain it using execution timing (via logs).

Task 4.2 – Callback Hell Simulation

Create nested callbacks that:

- Depend on previous results
- Clearly show execution depth

Log timestamps and nesting levels.

Task 4.3 – Promises vs Callbacks (Hard)

Rewrite Task 4.2 using Promises.

Compare readability and execution flow using logs.

Final Project (Medium → Hard)

Asynchronous Task Manager App

Build a small **JavaScript task manager application** that:

- Uses OOP principles (classes or constructors)
- Stores tasks with private state (closures)
- Executes tasks asynchronously
- Controls context using `bind`

Proof of Work Requirements:

- Add a section called `Runtime Observations`

- Log at least **3 behaviors that surprised you**
- Observations must be based on actual execution

Submission Requirements

1. GitHub Repository (Mandatory)

Repository structure:

```
/assignment
  ├── index.js
  ├── README.md
  └── observations.md
```

README.md must include:

- How to run the project
- What concepts are demonstrated
- Challenges faced
- Link to demo video

---

2. Demo Video (Mandatory for Final Project)

- 3–5 minutes maximum
- Show code execution
- Explain logic briefly
- Show output changing in real time

Evaluation Criteria

- Depth of understanding via runtime behavior
- Correct context handling and async control
- Clean and readable code structure
- Evidence of experimentation
- Original problem-solving

Final Note: This assignment is designed to reward thinking, experimentation, and execution.

Shortcuts will fail.

Write it. Run it. Observe it. Understand it.