

JavaScript (ES6) Core Concepts Assignment

Topics Covered

- var vs let vs const
 - Template Literals
 - Arrow Functions
 - Iterators & for..of
-

Assignment Overview

This assignment is designed to help students **deeply understand modern JavaScript fundamentals** through practical coding tasks rather than theory or memorization.

The tasks are intentionally written in a way that requires **logical thinking, experimentation, and hands-on coding**. Students are expected to write original code and explain their reasoning through implementation, not copied patterns.

This assignment focuses on *how JavaScript behaves*, not just *how it looks*.

Learning Objectives

By completing this assignment, students will be able to:

- Clearly distinguish between `var`, `let`, and `const` through real behavior
 - Use template literals for dynamic and readable string construction
 - Understand how arrow functions differ from regular functions
 - Work with iterables using `for..of` and iterator concepts
 - Write cleaner, modern, and predictable JavaScript code
-

Rules & Instructions

- Use **plain JavaScript (ES6+) only**
- Do **not** use frameworks or libraries
- Code must be written and tested by **you**
- Avoid copying examples from the internet
- Write small experiments and observe results
- Console output must clearly show behavior

Important (Academic Integrity):

- Each task must include **runtime output that depends on your own variable values** (names, numbers, sequences chosen by you).
 - Generic answers or identical outputs across submissions will be considered invalid.
 - You must be able to **explain your code line-by-line if asked**.
 - Code that looks auto-generated without experimentation evidence may be rejected.
-

Section 1: var vs let vs const (Deep Understanding)

Instruction: You must change variable names, values, and block structures in every task. Do not reuse examples shown in class or online.

Task 1.1 – Scope Exploration

Create a program where:

- A variable is declared using `var` inside a block
- The same variable name is declared using `let` inside another block
- Access both variables outside the blocks

Requirement:

- Use **non-obvious variable names** of your own choosing
- Print the value *before*, *inside*, and *after* the block
- Change the values at least once and re-run

Do not explain in words. The **console output sequence** must make the behavior obvious.

Task 1.2 – Re-declaration & Re-assignment

Write code that:

- Re-declares a variable using `var`
- Attempts the same with `let`
- Attempts to modify a `const` value indirectly (object or array)

Constraint:

- The program must **continue running** after errors using controlled execution (e.g., try/catch)
 - Log **exactly what succeeds and what fails**, in your own wording via output labels.
-

Task 1.3 – Loop Behavior Test

Create three loops:

- One using `var`
- One using `let`
- One using `const`

Inside each loop, log values using a delayed operation.

Mandatory:

- Use different delay timings
 - Add a final log after all delays complete
 - Your output order must clearly show *why* the behavior differs
-

Section 2: Template Literals (Practical Usage)

Task 2.1 – Dynamic Message Generator

Create a function that accepts:

- User name
- Course name
- Completion percentage

Return a message using **template literals only** that dynamically formats the message across multiple lines.

Task 2.2 – Conditional Templates

Generate different template literal outputs based on:

- Pass / Fail status
- Grade boundaries

Avoid string concatenation completely.

Section 3: Arrow Functions (Behavior Matters)

Task 3.1 – Arrow vs Regular Function

Create an object with:

- One regular function
- One arrow function

Both should attempt to access `this`.

Log results and compare behavior.

Task 3.2 – Implicit vs Explicit Return

Write arrow functions that:

- Use implicit return
- Use explicit return

Apply them to an array transformation.

Task 3.3 – Arrow Functions in Callbacks

Rewrite a callback-based operation twice:

- Once using a normal function
- Once using an arrow function

Observe readability and behavior differences.

Section 4: Iterators & for..of

Task 4.1 – Manual Iteration

Create a custom iterable object and manually iterate over it using `for..of`.

The iterable should return a controlled sequence of values.

Task 4.2 – Comparing Loops

Use the same data set and iterate using:

- `for`
- `for..in`
- `for..of`

Log what each loop outputs and compare results.

Task 4.3 – Real-World Iterator Usage

Simulate a real-world scenario (e.g., queue, playlist, task list) and iterate over it using `for..of`.

Focus on clarity and structure.

Final Challenge (Medium → Hard)

Build a **Mini JavaScript Analyzer** that:

- Accepts a list of mixed variable declarations **you design yourself**
- Uses iterators (not array helpers only)
- Formats output using multi-line template literals
- Uses arrow functions **only where behavior is correct**

Proof of Work Requirement:

- Add a final section called `Reflection Logs`
 - Log at least **3 unexpected behaviors** you observed while building this
 - These logs must be based on *your own runtime results*, not definitions
-

Submission Guidelines

- Submit a single JavaScript file in a public repository
 - Code must be runnable without modification
 - Output must clearly demonstrate experimentation
 - Include a short `RUN NOTES` section explaining how many times you ran and changed the code
 - No screenshots, PDFs, or copied explanations
-

Evaluation Criteria

- Concept clarity through code behavior
- Correct usage of ES6 features
- Clean and readable structure
- Logical reasoning in implementation
- Depth of practice

Note: This assignment is designed to test *understanding through execution*. Shortcuts will not help. Practice and experimentation are the only way forward.

Best of luck — code, break it, and learn from it.