# CSCI 3403 01 Assignment 08

# Total : 100 Points

# Advanced JavaScript Concepts Assignment

This assignment focuses on understanding and implementing advanced JavaScript concepts like functions, asynchronous programming, and object-oriented design. By completing this assignment, you will gain hands-on experience with JavaScript techniques such as arrow functions, closures, callbacks, promises, async/await, and methods for manipulating and working with data.

## Part 1: Theory - Short Answer Questions

Write short answers for the following questions:

1. What are arrow functions, and how are they different from traditional functions?

Arrow functions are functions defined by the => arrow operator. Arrow functions are different because they will return without the return keyword.

2. Explain the concept of closures with a practical example.

A closure is a function that retains access to its outer function's variables even after the outer function has executed.

3. What are callbacks, and how are they used in asynchronous programming?

A callback is a function passed to another function that gets executed later, usually after an asynchronous task like loading data or waiting for a timer finishes.

4. Define promises. How do they improve handling asynchronous code compared to callbacks?

A promise is an object that represents the result of an asynchronous operation, which will eventually be either resolved or rejected. It improves handling asynchronous code by allowing easier chaining of tasks and more structured error handling compared to traditional callbacks.

5. Explain async/await with an example.

Async and await are used to write asynchronous code in a more synchronous, readable way. The async keyword is added to a function to indicate it will return a promise and await is used inside an async function to pause execution until the promise is resolved or rejected. The most common use for async/await is fetch requests.

6. What is the prototype chain in JavaScript, and how does inheritance work?

The prototype chain in JavaScript is a mechanism by which objects inherit properties and methods from other objects, with each object having a prototype that points to another object, allowing inheritance and shared behavior across instances.

7. Describe how the `map()` function works. How is it different from `forEach()`?

The map() function creates a new array by applying a given function to each element of the original array, while forEach() simply executes a function for each element without returning a new array.

8. What is the purpose of the `filter()` function in JavaScript?

The filter() function in JavaScript is used to create a new array with all elements that pass a specified test (i.e., meet a condition), without modifying the original array.

9. How does the `reduce()` function work? Provide an example to illustrate its use.

The reduce() function in JavaScript applies a callback to each element of an array, accumulating a single result (such as a sum, product, or concatenation), with an optional initial value for the accumulator.

10. What are JavaScript modules, and why are they important in modern development?

JavaScript modules are files that encapsulate code into reusable pieces, allowing for better organization, easier maintenance, and the ability to import/export functionality between different parts of an application, which is crucial for scaling modern web development.

## Part 2: Practical Implementation

### Task 1: Arrow Functions and Closures

1. Create a function using an arrow function to calculate the square of a number.

2. Implement a closure that keeps track of how many times a button is clicked on a webpage.  
Example:

const clickCounter = () => {  
 let count = 0;  
 return () => {  
 count++;  
 console.log(`Button clicked ${count} times`);  
 };  
};  
const buttonClick = clickCounter();

### Task 2: Callbacks, Promises, and Async/Await

1. Write a function that takes a callback and uses it to log the current time every second for 5 seconds.  
2. Create a promise-based function that resolves after 2 seconds with a greeting message ("Hello, World!") and demonstrate its use.  
3. Rewrite the above promise-based function using async/await.

### Task 3: Prototypes and Inheritance

1. Create a `Person` constructor with properties `name` and `age`. Add a method `introduce` to the prototype that logs a greeting.  
2. Create a `Student` constructor that inherits from `Person` and adds a property `grade`. Add a method `study` to `Student`'s prototype.  
Example:

const alice = new Student("Alice", 20, "A");  
alice.introduce(); // Logs: "Hi, I'm Alice and I'm 20 years old."  
alice.study(); // Logs: "Alice is studying."

### Task 4: Map, Filter, and Reduce

1. Use the `map()` method to double the values in an array `[1, 2, 3, 4, 5]`.  
2. Use the `filter()` method to extract even numbers from the array `[1, 2, 3, 4, 5]`.  
3. Use the `reduce()` method to calculate the sum of all numbers in the array `[1, 2, 3, 4, 5]`.

### Task 5: Modules

1. Create two JavaScript files: `math.js` and `main.js`.  
 - math.js: Export two functions: `add(a, b)` and `subtract(a, b)`.  
 - main.js: Import the functions from math.js and use them to perform addition and subtraction on two numbers.  
Example:

// math.js  
export const add = (a, b) => a + b;  
export const subtract = (a, b) => a - b;  
  
// main.js  
import { add, subtract } from './math.js';  
console.log(add(10, 5)); // Output: 15  
console.log(subtract(10, 5)); // Output: 5

## Part 3: Problem Solving

### Task 1: Task Scheduler

Write a function `taskScheduler(tasks, delay)` that takes an array of tasks (functions) and a delay (in milliseconds). The function should execute the tasks sequentially with the given delay between each.  
Example:

const tasks = [  
 () => console.log("Task 1"),  
 () => console.log("Task 2"),  
 () => console.log("Task 3"),  
];  
taskScheduler(tasks, 1000);

### Task 2: Data Processing

Given an array of objects:  
const users = [  
 { id: 1, name: "Alice", age: 25 },  
 { id: 2, name: "Bob", age: 30 },  
 { id: 3, name: "Charlie", age: 35 },  
];  
  
Perform the following:  
- Use `map()` to create an array of user names.  
- Use `filter()` to create an array of users older than 28.  
- Use `reduce()` to calculate the total age of all users.

## Part 4: Deliverables

1. Source Code: Submit all JavaScript files containing your solutions.  
2. Screenshots: Provide screenshots of the console output for each task.  
3. Documentation: Write a brief explanation (1-2 sentences) for each solution in a separate text or word document.

## Grading Criteria

| Task | Points |  
|------------------------------|--------|  
| Theory Questions (Part 1) | 20 |  
| Practical Tasks (Part 2) | 50 |  
| Problem-Solving (Part 3) | 30 |  
| \*\*Total\*\* | 100 |