**Java script Assignment on Closures**

Part 1: Theoretical Understanding (20 points)

**Deliverable**: Write a 200-300 word summary answering the following:

* What is a closure in JavaScript?
* How does a closure relate to lexical scope?
* Provide one real-world example where closures are useful (e.g., data privacy, event handlers).
* Submit your summary as a text file or PDF.

**Grading Criteria**:

* Clarity and correctness of the definition (10 points).
* Explanation of lexical scope and example (10 points).

### Part 2: Coding Exercises (60 points)

**Task**: Implement the following JavaScript programs to demonstrate your understanding of closures. Each exercise should be submitted as a separate .js file with comments explaining the code.

1. Create a function makeCounter that returns a counter function. Each time the counter function is called, it should increment and return a count. The count should be private and not accessible outside the counter function.

**Grading Criteria**:

* Correct implementation (10 points).
* Privacy of the count variable (5 points).
* Clear comments explaining the closure (5 points).

1. Write a function createGreeter that takes a name parameter and returns a greeter function. The greeter function should return a personalized greeting message using the name provided when createGreeter was called.

**Grading Criteria**:

* Correct implementation (10 points).
* Proper use of closure to retain name (5 points).
* Clear comments (5 points).

1. Create a function multiplierFactory that takes a factor parameter and returns afunction that multiplies any given number by that factor. Demonstrate the use of closures by creating multiple multiplier functions with different factors.

**Grading Criteria**:

* Correct implementation (10 points).
* Demonstration of multiple closures (5 points).
* Clear comments (5 points).

**Task**: Answer the following questions in a separate text file or PDF. Each answer should be 100-150 words.

1. **Question 1**: Explain how closures are different from global or local variables. Why is it beneficial to use closures instead of global variables in the counter example from Exercise 1?
2. **Question 2**: In Exercise 3, how does the closure ensure that each multiplier function retains its own factor value? What would happen if the factor was stored in a global variable instead?

**Grading Criteria**:

* Depth and clarity of explanation for Question 1 (10 points).
* Depth and clarity of explanation for Question 2 (10 points).

1. Create a function makeFibonacci that returns a memoized function to calculate Fibonacci numbers. The returned function should use a closure to cache previously computed Fibonacci values to improve performance for repeated calls. The Fibonacci sequence is defined as F(n) = F(n-1) + F(n-2), with F(0) = 0 and F(1) = 1.

**Requirements**:

* Use a closure to maintain a cache (e.g., an object or array) of computed Fibonacci numbers.
* The returned function should accept a single parameter n (a non-negative integer) and return F(n).
* Optimize performance by checking the cache before computing new values.
* Handle edge cases: n = 0, n = 1, and invalid inputs (e.g., negative numbers or non-integers).
* Add comments explaining how the closure enables memoization.
* Include a console.log statement in the function to indicate when a value is retrieved from the cache vs. computed.

**Grading Criteria**:

* Correct Fibonacci calculation (8 points).
* Proper memoization using a closure (8 points).
* Handling of edge cases (4 points).
* Clear comments explaining the closure and cache (5 points).

1. Create a function createTodoModule that returns an object representing a private todo list module. The module should use closures to maintain a private array of todos and expose public methods to interact with it (add, remove, list, and clear todos). Ensure the internal array is not directly accessible from outside the module.

**Requirements**:

* Use a closure to maintain a private todos array.
* Return an object with at least four methods:
  + addTodo(task): Adds a task (string) to the array.
  + removeTodo(index): Removes a task at the given index.
  + listTodos(): Returns a copy of the todos array.
  + clearTodos(): Empties the todos array.
* Prevent direct access to the todos array (demonstrate this in your code).
* Handle edge cases (e.g., invalid index for removeTodo, empty task strings).
* Add comments explaining how the closure ensures data privacy.
* Include a test script that demonstrates all methods and attempts to access the private array.

**Grading Criteria**:

* Correct implementation of all methods (8 points).
* Data privacy via closure (8 points).
* Handling of edge cases (4 points).
* Clear comments and test script (5 points).

**Submission Format:**

* A single .js file or .html file with embedded JS.
* Include a comment block at the top with:
  + Name, USN,Section
  + A short paragraph explaining what you learned about Closure

<https://forms.gle/i6Y17PkKj91Cvkvh6>