**Closures Assisnment -1MS22CS114**

**Part 1: Theoretical Understanding (20 points)**

1. What is a closure in JavaScript?

In JavaScript, a **closure** is formed when a function is defined within another function and retains access to the outer function’s variables, even after the outer function has completed execution. This is possible because of **lexical scoping**, where functions are executed using the scope in which they were defined, not where they are called.

2. How does a closure relate to lexical scope?

Closures are tightly connected to the idea of **lexical scope**. They allow inner functions to “remember” variables from their parent scopes. This enables developers to write more modular, encapsulated, and secure code by keeping certain data private.

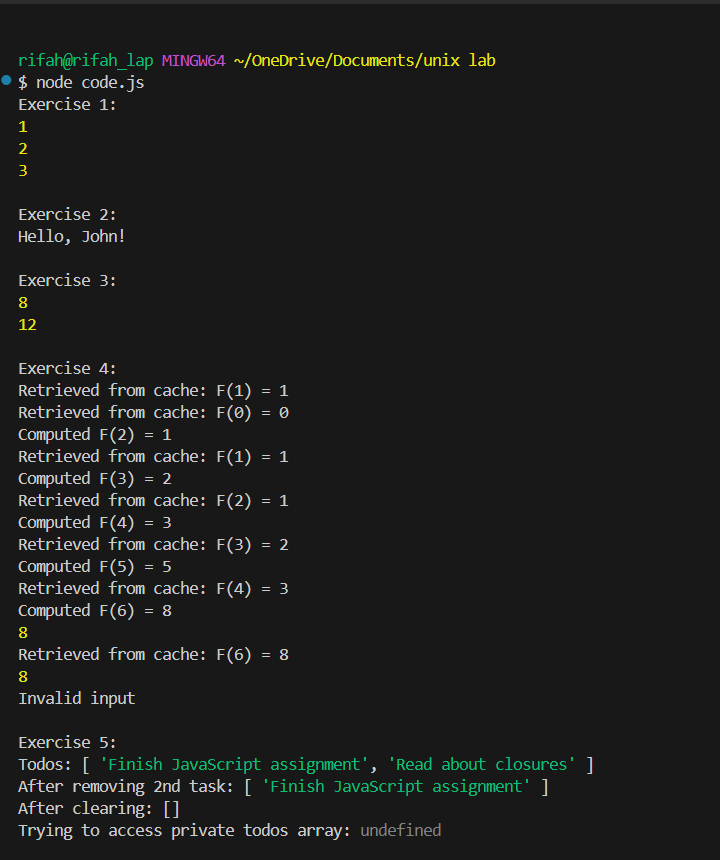
3.Provide one real-world example where closures are useful (e.g., data privacy, event handlers).

One common and powerful use of closures is in **data privacy**. For instance, a counter function can use a closure to maintain a private count variable, preventing unauthorized access or modifications. This is essential in avoiding global state pollution.

**Real-world examples**:

* At **Google**, closures are extensively used in complex **event handling** and **asynchronous operations** in apps like Gmail and Google Docs. Closures help ensure that each user interaction has access to the correct context, even when executed later via callbacks or promises.
* **Netflix** uses closures in their frontend JavaScript for dynamic UI rendering and to handle user-specific configurations and preferences. For example, playback settings or viewing history logic is often managed using closures to encapsulate user state.
* In **React** (used by Facebook and many others), hooks like useState and useEffect internally rely on closures to maintain state across renders.
* **Chrome DevTools**, maintained by Google, uses closures to manage debugging states across sessions without exposing sensitive debugging internals to the global environment.

**Part-2 output snippet**



**Part 3: Questions**

**Question 1: Closures vs Global/Local Variables**

Closures differ from global or local variables in how they preserve access to variables. Global variables are accessible from anywhere, which risks accidental overwriting. Local variables are temporary and disappear after function execution. Closures preserve local variables in memory by wrapping them inside another function. In Exercise 1 (makeCounter), using a closure keeps the count private and prevents other functions from modifying it. If count were global, any part of the code could change it, potentially causing bugs. Using closures promotes encapsulation and safer code design.

**Question 2: Closure and Factor Retention in Multiplier**

In Exercise 3, closures ensure that each multiplier function retains its own factor because the returned function closes over the factor variable. Even though multiplierFactory finishes executing, the inner function still has access to its factor. If factor were a global variable, all multiplier functions would share the same value, breaking their individual behavior. Closures allow each function to "remember" the correct factor even after the outer function is gone, enabling the creation of multiple, independently configured multiplier functions.