**JavaScript Theory Assignment on Closures**

USN : 1MS22CS150

Name : Suvan.B.U

Section : C

**Q1 :** 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).

Ans : Closures are a mechanism in Javascript by which an inner function in a nested environment is able to access the variables of its outer function, even after the outer function has completed its execution. The inner function maintains a set of references and it remembers the variables and parameters of the environment where the function itself was defined. Thus, the local variables of the outer function get preserved in the function where they are defined. This concept is what is termed as Lexical Scoping.

JavaScript functions are lexically scoped, meaning that they will have access to the local variables of their environment. So, depending on where they a function is defined its lexical scope is determined and not by where it is called. It is the concept of lexical scoping that enables for the implementation of closures. When a function is defined, it maintains a reference to its lexical environment, containing the variables and other parameters where it was defined. This enables access to it outer scope even after the outer scope has executed.

Closures are employed for a number of applications such as data encapsulation, information hiding, memoization, event handlers etc,.

A real world application is the use of closures for data encapsulation. Here, the internal variables are prevented from direct access from the outside code. They can be accessed only through an already defined set of functions. This is analogous to the way how an ATM works. The money which is stored in the ATM, cannot be directly accessed by the customer. It can be done so only through the buttons / the interface provided. Thus, closures help in achieving controlled access to variables, similar to private properties of OOP.

**Q2 :** Each answer should be 100-150 words.

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?

**Ans :** Global or local variables are defined in the entire program or local within a block or function. They are accessible only within the scope in which they are defined => either the global or the local scope.

But in the case of closures, they are defined locally inside a function, but it is preserved through the inner function, which maintains a reference to its lexical scope. So, in a closure, even after the execution of the outer function, that is, even outside the scope of its definition, the variables are accessible, but only through the inner function.

In case of the counter example, if a global variable was used .. it would be as :

let count = 0;

function increment() {

count++;

return count;

}

Here, the count can be modified easily by directly referring the count variable. It is fully exposed.

But in the case of closures, no such direct access is provided and it is through inner functions only.

Also, we look at another inconsistency :

let event\_one\_cnter = increment();

let event\_two\_cnter = increment();

Here ... both counters will have inconsistent values .. as the ‘count’ is shared.

This can be avoided, and each of them can have independent & private ‘count’ by the use of closures.

Further, Closures help in avoiding naming conflicts and to handle the state of multiple instances of the function reference in an efficient and modular manner.

1. 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?

**Ans :** In the Exercise 3, the multiplier function, multiplierFactory() takes an argument factor. And it returns an anonymous function reference, which is the inner function. It is this inner function which is responsible for the unique factor value of each of the multiplierFactory() instance.

At the time of creating the function reference, an argument for the ‘factor’ is passed into the multiplierFactory(), which will accordingly return the inner function reference. This reference will store the value of ‘factor’ in its lexical scope. As a result the function literal storing this reference will have indirect access to the factor value through closure scope.

If the factor was stored in a global variable, we would have to re-assign the required value of factor, everytime we wish to call it. This would make debugging of the code very difficult. The ‘factor’ would be a shared global variable, rather than a private variable for each instance as in the case of closures.

Hence by using closure, we ensure, modularity, code reusability and private-assignment of ‘factor’ for each instance of the use of the function.