/\*

USN: 1MS22CS120

Name: Sakshi R K.

Section: 'C'.

\*/

/\*

About Closures:

\* 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.

\* JavaScript functions are lexically scoped, meaning that they will have access to the local variables of their environment.

\* This happens by it maintaining a reference to its lexical environment.

\* 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,.

\*/

/\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

/\*

Q1 : Creating a function makeCounter that returns a counter function.

Each time the counter function is called, it increments the count and returns it.

The count is private and not is accessible outside the counter function.

\*/

let makeCounter = () => {

let count = 0;

let incrementCount = () => {

count++;

return count;

}

return incrementCount; // inner function is returned.

}

let myCnter = makeCounter();

console.log(myCnter()); // OP : 1

console.log(myCnter()); // OP : 2

console.log(myCnter()); // OP : 3

/\*

Explaination :

'myCnter' stores the function which 'makeCounter()' returns.

=> the inner 'incrementCount' function has closure access to 'count'.

\*/

/\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

/\*

Q2 : 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.

\*/

let createGreeter = (name) => {

// Anonymous Inner Arrow-Function => has closure access to 'name' of outer function.

return () => {

return `Hai ${name} .. how are you?!`;

}

}

let greetABC = createGreeter("ABC");

console.log(greetABC()); // OP : Hai ABC .. how are you?!

/\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

/\*

Q3 : Create a function multiplierFactory that takes a factor parameter and returns a function that multiplies any given number by that factor.

Demonstrate the use of closures by creating multiple multiplier functions with different factors.

\*/

let multiplierFactory = (factor) => {

return (number) => {

return factor \* number;

}

}

let scale10x = multiplierFactory(10);

let scaleHalf = multiplierFactory(0.5);

let value = 10;

console.log(scale10x(value)); // OP : 100

console.log(scaleHalf(value)); // OP : 5

/\*

Explaination :

'scale10x' => stores a closure function that multiplies any num by 10.

'scaleHalf' => stores a closure function that multiplies any num by 0.5.

\*/

/\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

/\*

Q4 : 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.

Fibonacci sequence => F(n) = F(n-1) + F(n-2),

with F(0) = 0 and F(1) = 1.

\*/

let makeFibonacci = () => {

// Private object for caching only.

const cache = {};

// Inner function which leverages the 'cache' by virtue of closure.

let fibo = (n) => {

if(n in cache) return cache[n];

if(n == 0) return 0;

if(n == 1) return 1;

const res = fibo(n-1) + fibo(n-2);

cache[n] = res;

return res;

}

// Inner function's reference is returned.

return fibo;

}

let generateFibo = makeFibonacci();

console.log(generateFibo(10)); // OP : 55

console.log(generateFibo(15)); // OP : 610

/\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

/\*

Q5 : 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.

\*/

let createTodoModule = () => {

// todoTasks => private and not externally exposed or returned.

let todoTasks = [];

// Returns an anonymous object with the 4 requried functions.

// These functions are publically accessible by the returned object.

return {

addTodo(task)

{

if(typeof task === 'string' && task.length > 0) {

todoTasks.push(task.trim());

}

else {

console.error(`Invalid Task Input.`);

}

},

removeTodo(taskIdx)

{

taskIdx -= 1

if(typeof taskIdx !== 'number' || taskIdx >= todoTasks.length || taskIdx < 0)

{

console.error(`Invalid Task Index.`);

}

else {

const taskRemoved = todoTasks.splice(taskIdx, 1);

console.log(`Removed Task : ${taskRemoved[0]}`);

}

},

listTodos()

{

let taskString = ``;

if(todoTasks.length === 0) {

taskString += `[] \nNo Tasks To-Do!\n`;

}

else {

for(var idx in [...todoTasks]) {

taskString += `${Number(idx)+1} : ${todoTasks[idx]}\n`;

}

}

taskString += "\*\*\*\*\* \n";

return taskString;

},

clearTodos()

{

todoTasks = [];

console.log(`All todo-tasks are cleared!`);

}

};

}

// Creating reference to the inner function, via the outer function return.

const myTodoList = createTodoModule();

// Demonstrating the required functions.

// 1. addTodo()

myTodoList.addTodo(`Learn React.`);

myTodoList.addTodo(`Finish Sliding Window.`);

myTodoList.addTodo(`See about Wireshark.`);

// 2. listTodos()

console.log(`\nCurrent Todo-List : \n${myTodoList.listTodos()}`);

// 3. removeTodo()

// Number is passed as per how the list is visible.

myTodoList.removeTodo(2);

// 4. clearTodos()

myTodoList.clearTodos();

console.log(`\nFinal Todo-List : \n${myTodoList.listTodos()}`);

/\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

END OF FILE!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/