**Javascript Assignment 9**

**1a) Is the InnerFunction() a closure?**

function OuterFunction()

{

var outerVariable = 100;

function InnerFunction()

{

alert(outerVariable);

}

return InnerFunction;

}

var innerFunc = OuterFunction();

innerFunc();

In the above example, InnerFunction is a closure, outervariable belongs to the outerfunction is captured or used in InnerFunction.

**1 b) output is 100**

**2. What is the difference between a closure and a scope ?**

In the above example, outer variable is scope of OuterFunction.

In the inner function, outer variable is used or accessed in innerFunction, so

function InnerFunction()

{ alert(outer Variable) } // is Closure

**3. What is a lexical scope and how is it related to closure?**

Lexical scope describes how innerfunction (child) functions have access to “outerVariable” variables defined in outerfunction(parent) scopes.

i.e inner function can access outer function variables/data is defined as lexical scope.

function OuterFunction()

{ let outerVariable = 100;

  console.log(`the previous value of outerVariable is ${outerVariable}`);

function InnerFunction()

{

outervariablenew=outerVariable+2;

console.log("lexical scopes defines how the outervariable value is accessed in innerfunction(child)from parent OuterFunction")

console.log(`now the value of outer variable is changed to ${outervariablenew} from ${outerVariable}`)

}

return InnerFunction();

}

OuterFunction();

**In case of closure,**

A closure is a function having access to the parent scope, even after the parent function has closed.

const OuterFunction=()=>

{

     let outerVariable = 100;

     console.log(`Printing the value of outerVariable from OuterFunction is ${outerVariable}`)

     console.log("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

const InnerFunction=()=>

   {

    outerVariable=outerVariable+2;

    console.log(`Printing the value of outerVariable from InnerFunction is ${outerVariable}`)

}

return InnerFunction;

}

const result = OuterFunction();

// console.log("Describing the properties of closure:To access the value of variable outside both the function")

// console.log(`the value of outervariablenew is ${outerVariable}`)

result();

result();

result();

InnerFunction as a child of OuterFunction, this function has access to the “outerVariable” variable which is inside OuterFunction even after it has closed!

The closure we created now allows us to continue to increase the value of the myValue variable every time we call result().

**Another example of closure:**

const myFunction = () => {

    let myname = "Rukshana";

   console.log(`First name From ParentFunction myFunction:${myname}`);

    const childFunction = () => {

         console.log(`name From childFunction:${myname=myname+" "+"Begam"}`);

    }

    return childFunction;

}

const result = myFunction();

result();

 //when called first time, myvalue used is 2 which is from parent function myFunction() 2+1=3

result();

 //calling again: previous value "myvalue" from child function used here is 3......3+1=4

 result();

//calling again: previous value "myvalue" from child function used here is 4 ......4+1=5

**output:**

First name From ParentFunction myFunction: Rukshana

name From childFunction: Rukshana Begam

name From childFunction: Rukshana Begam Begam

name From childFunction: Rukshana Begam Begam Begam

As a child of myFunction, this anonymous function has access to the myname variable inside myFunction even after it has closed.

The closure we created now allows us to continue to update the value of the myname variable every time we call result().

// Another example of closure:

In the below example, num1 value defined inside the outer function is lexical scope. In case of closure, the variable defined in lexical scope of outer function can be accessed inside the inner function.

function outer()

{

    let num1=20

    console.log("num1 from outer func");

    console.log(num1)

    return function()

    {

        let num2=10;

        console.log("num2 from inner function inner",num2)

        console.log("here we can access the num1 value defined in outer:",num1)

    }

}

let inner=outer();

inner();

output:

num1 from outer func

20

num2 from inner function inner 10

here we can access the num1 value defined in outer: 20

**4. Output of following closure ?**

**for (var i = 0; i < 3; i++) {**

**setTimeout(function log() {**

**console.log(i); // What is logged?**

**}, 1000);**

**}**

**Answer:**

In the above example, SetTimeout function is used…here setTimeout executes after the loop comes out..when i value becomes 3. So the output is

**3**

**3**

**3**

When i=0 it enters, SetTimeOut function is encountered so it comes out without execution(console.log) again i gets incremented to 1,2 then same repeats when i becomes 3, then it comes out of loop and executes console lines and print the current i value as 3. For loop executes 3 times so 3 times i value(i.e 3 )is printed.

A function within another function, second function i.e setTimeout use value from first function i.e for loop…so this is considered as closure.