***Javascript Assignment 9***

***Carefully observe this example.***

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

***b) What is output of this program***

*function OuterFunction() {*

*var outerVariable = 100;*

*function InnerFunction() {*

*alert(outerVariable);*

*}*

*return InnerFunction;*

*}*

*var innerFunc = OuterFunction();*

*innerFunc();*

***ANSWER:***

*a) Yes, InnerFunction() is a closure as it accesses a variable (outerVariable) declared in the parent function (OuterFunction()).*

*b) The output of this program will be an alert dialog box displaying the value 100, as the InnerFunction() accesses the outerVariable which was set to 100 in the parent function. When the variable innerFunc is called, it executes the InnerFunction which alerts the value of outerVariable. So, the output will be an alert message showing the number 100.*

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

*A scope is a region of code where a variable, function, or object is defined and can be accessed. Scopes can be nested, meaning that a scope can contain other scopes. The scope chain is the hierarchy of scopes that exist in a program, and it determines where a variable or function can be accessed from.*

*A closure, on the other hand, is a function that has access to variables from its outer (enclosing) scope, even after that scope has returned. A closure is created when a function is defined inside another function and accesses one or more variables declared in the outer function's scope. The inner function can continue to access the outer function's variables even after the outer function has returned.*

*In summary, scope refers to the region of code where a variable, function, or object can be accessed, while a closure is a function that has access to variables from its outer scope, even after that scope has returned.*

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

*Lexical scope refers to the way variables are resolved in a program based on the physical structure of the code. In a lexically scoped language, the scope of a variable is determined by its location in the source code. In other words, variables are resolved based on their position in the program's source code.*

*In JavaScript, which is a lexically scoped language, closures are created by functions that are defined inside other functions. A closure is formed when an inner function accesses variables from the outer function's lexical scope, even after the outer function has returned. The inner function has access to the variables of the outer function, and these variables are "captured" by the closure. The closure allows the inner function to continue to access the variables of the outer function, even after the outer function has completed execution.*

*So, in JavaScript, closures are closely related to lexical scope because they are formed by the combination of functions and their lexical environments. Closures allow functions to access variables from their lexical environments even when those variables are no longer in scope. This is a powerful feature of JavaScript that enables developers to create functions that have persistent state and can retain information between calls.*

***4. Output of following closure ?***

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

*setTimeout(function log() {*

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

*}, 1000);*

*}*

***ANSWER:***

*The output of this closure will be three times the number 3, each on a separate line.*

*This happens because the setTimeout() function is executed asynchronously, meaning that it is executed after the rest of the program has finished executing. By the time the setTimeout() function is executed, the loop has already finished executing, and the variable i has been incremented to 3. So, when the console.log(i) statement is executed inside the setTimeout function, it always prints the value of i, which is 3.*

*Therefore, after a delay of one second (due to the 1000ms delay in the setTimeout() function), the console.log(i) statement is executed three times, each time printing the value of the variable i, which is 3 in all cases, resulting in the output of "3" being logged to the console three times.*