Su Lai Myint - 613735

Week 2 Day 3 Lab – Closure and Module Pattern

1. Determine what this Javascript code will print out (without running it):

```
 x = 1;
var a = 5;
var b = 10;
                   8   9   10
var c = function(a, b, c) { f = undefined; x = undefinde
                  document.write(x); //undefined
                  document.write(a); //8
                                          8   9   10
                  var f = function(a, b, c) { x = undefined
                                    b = a; 8
                                    document.write(b); // 8
                                    b = c; 10
                                    var x = 5;
                               }
                    8 9 10
                  f(a,b,c);
                  document.write(b); // 9
                  var x = 10;
             }
c(8,9,10);
document.write(b); //10
document.write(x); //1
}
```

This doesn't effect cos its within Function.

← not current scope as this.(scope chain)

1) Answer:
undefined889101

Explanation:
Pls refer the stack trace notes

2. Define *Global Scope* and *Local Scope* in Javascript.

2) Answer:
When the JavaScript engine executes a script, it creates a global execution context. Also, it also assigns variables that declare outside of functions to the global execution context. These variables are in the global scope. They are also known as global variables.

The variables that declare inside a function are local to the function. They are called local variables.

The way that JavaScript resolves a variable is by looking at it in its current scope, if it cannot find the variable, it goes up to the outer scope, which is called the scope chain.

3. Consider the following structure of Javascript code:

```
// Scope A
function XFunc () {
      // Scope B
      function YFunc () {
            // Scope C
      };
};
```

(a) Do statements in Scope A have access to variables defined in Scope B and C?

✓ (b) Do statements in Scope B have access to variables defined in Scope A?

(c) Do statements in Scope B have access to variables defined in Scope C?

✓ (d) Do statements in Scope C have access to variables defined in Scope A?

✓ (e) Do statements in Scope C have access to variables defined in Scope B?

3) Explanation:
Scope chain works from current scope to outer scope, not from outer to inner.

4. What will be printed by the following (answer without running it)?

```
var x = 9;
function myFunction() {
     return x * x;
}
document.write(myFunction());// 81
x = 5;
document.write(myFunction()); //25
```

4) Answer:
8125

Explanation:
x is global scope variable, when it assigns new values, it effects globally and that product 25 for second doc. write.

5. foo=undefined;

```
var foo = 1;
function bar() {   Foo= undefined;
          if (!foo) {
                    var foo = 10;
          }
          alert(foo);// 10
}
bar();
```

5) Answer:
Yes, alert will pop with value 10.

Explanation:
When reaching to function bar, JS hoist all variables declaration, that makes foo=undefined at first. When it come to 'if', foo is falsy and !foo is true, then it meets the condition and reaches to value assignment to foo.
As 'if' is not the block scope, it become foo = 10;

What will the *alert* print out? (Answer without running the code. Remember 'hoisting'.)?

6. Consider the following definition of an *add*( ) function to increment a *counter* variable:

```
var add = (function () {
     var counter = 0;
     return function () {
             return counter += 1;
             }
     })();
```

Modify the above module to define a *count* object with two methods: *add*( ) and *reset*( ). The *count.add*( ) method adds one to the *counter* (as above). The *count.reset*( ) method sets the *counter* to 0.

7. In the definition of *add*( ) shown in question 6, identify the "free" variable. In the context of a function closure, what is a "free" variable?

8. The *add*( ) function defined in question 6 always adds 1 to the *counter* each time it is called. Write a definition of a function *make_adder*(*inc*), whose return value is an *add* function with increment value *inc* (instead of 1). Here is an example of using this function:

```
add5 = make_adder(5);
add5( );    add5( );    add5( );   // final counter value is 15

add7 = make_adder(7);
add7( );    add7( );    add7( );   // final counter value is 21
```

9. Suppose you are given a file of Javascript code containing a list of many function and variable declarations. All of these function and variable names will be added to the Global Javascript namespace. What simple modification to the Javascript file can remove all the names from the Global namespace?

10. Using the *Revealing Module Pattern*, write a Javascript definition of a Module that creates an *Employee* Object with the following fields and methods:

Private Field: name
Private Field: age
Private Field: salary

Public Method: setAge(newAge)
Public Method: setSalary(newSalary)
Public Method: setName(newName)
Private Method: getAge( )
Private Method: getSalary( )
Private Method: getName( )
Public Method: increaseSalary(percentage) // uses private getSalary( )
Public Method: incrementAge( ) // uses private getAge( )

11. Write a few Javascript instructions to extend the Module of Question 10 to have a public *address* field and public methods *setAddress*(*newAddress*) and *getAddress*( ).