Javascript Module Exercises

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

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

undefined 8 8 9 10 1

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

   Global Scope: the scope where all variables and functions defined in are accessible to the rest of the program. It is outside of any function definition.

   Local Scope: also known as function scope, is the scope associated with functions, where variables and functions inside the local scope are only accessible within the enclosing function. It is inside any function definition.

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? **No**
(b) Do statements in Scope B have access to variables defined in Scope A? **Yes**
(c) (c) Do statements in Scope B have access to variables defined in Scope C? **No**
(d) Do statements in Scope C have access to variables defined in Scope A? **Yes**
(e) Do statements in Scope C have access to variables defined in Scope B? **Yes**

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

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

81 25

5.

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

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

10

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.

```
var count = (function() {
        var counter = 0;
        return {
                add: function() {
                        return counter += 1;
                },
                reset: function() {
                        counter = 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?

   The free variable in question 6 is counter. A free variable is a variable accessible to a function, but not locally defined within the function.

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
```

```
make_adder = function(inc) {
        var counter = 0;
        return function() {
                return counter += inc;
        };
};
```

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?

```
(function() {
        // the original code
}) ();
```

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( )

```
var employee = (function() {
        var name = "";
        var age = 0;
        var salary = 0;

        var setAge = function(newAge) { age = newAge; };
        var setSalary = function(newSalary) { salary = newSalary; };
        var setName = function(newName) { name = newName; };
        var getAge = function() { return age; };
        var getSalary = function() { return salary; };
        var getName = function() { return name; };
        var increaseSalary = function(percentage) { setSalary( (1+percentage/100) * getSalary() ); };
        var incrementAge = function() { setAge( getAge() + 1); };

        return { setAge: setAge, setSalary: setSalary, setName: setName, increaseSalary: increaseSalary,
incrementAge: incrementAge };

}) ();
```

11. Rewrite your answer to Question 10 using the *Anonymous Object Literal Return Pattern*.

```
var employee = (function() {
        var name = "";    var age = 0;        var salary = 0;

        var getAge = function() { return age; };
        var getSalary = function() { return salary; };
        var getName = function() { return name; };
        return {
            setAge: function(newAge) { age = newAge; },
            setSalary: function(newSalary) { salary = newSalary; },
            setName: function(newName) { name = newName; },
            increaseSalary: function(percentage) { setSalary( (1+percentage/100) * getSalary() ); },
            incrementAge: function() { setAge( getAge() + 1); }        };
    }) ();
```

12. Rewrite your answer to Question 10 using the *Locally Scoped Object Literal Pattern*.

```
var employee = (function() {
    var name = "";    var age = 0;        var salary = 0;

    var getAge = function() { return age; };
    var getSalary = function() { return salary; };
    var getName = function() { return name; };

    var myObject = {};
    myObject. setAge = function(newAge) { age = newAge; };
    myObject.setSalary = function(newSalary) { salary = newSalary; };
    myObject.setName = function(newName) { name = newName; };
    myObject.increaseSalary: function(percentage) { setSalary( (1+percentage/100) * getSalary() ); };
    myObject.incrementAge: function() { setAge( getAge() + 1); };

    return myObject;
}) ();
```

13. 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*( ).

```
employee.address = "";
employee.setAddress = function(newAddress) { address = newAddress; };
employee.getAdress = function() { return this.address; };
```

14. What is the output of the following code?

```
const promise = new Promise((resolve, reject) => {
      reject("Hattori");
});
promise.then(val => alert("Success: " + val))
       .catch(e => alert("Error: " + e));
```

The promise is explicity rejected, so the error handler in invoked.
We get an error message.

15. What is the output of the following code?

```
const promise = new Promise((resolve, reject) => {
        resolve("Hattori");
        setTimeout(()=> reject("Yoshi"), 500);
});
promise.then(val => alert("Success: " + val))
         .catch(e => alert("Error: " + e));
```

The success handler will be invoked. The error handler will not be invoked because one a promise has settled, it can't be changed. Rejecting a promise after 500ms has no effect.