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);
}
```

ANSWER :10 8 8 9 10 1

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

Global Scope : If a variable is defined outside of a function, then the variable can be called as global variable, that means this variable has global scope through out the program..

Local Scope: If a variable is defined inside of a function, then the variable can be called as local variable, that means this variable has local scope. It can not be used outside of the function.

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) 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());
```

Answer : 81 and 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'.)?

Ans: 1

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.

Answer :

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

   Answer : **Free variables** are simply the **variables** that are neither locally declared nor passed as parameter.

   - var counter is the 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
```

   ANSWER :

```
add() = make_adder(inc) {
        Return add(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?

   Answer: Module Pattern will help us to remove all the names from the global namespace by wrapping up set of variables and functions together in a single scope.

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( )

ANSWER:

```
var Employee = (function () {
      let Private Field:  name
      let Private Field:  age
      let Private Field:  salary
     var setAge = function(newAge) {
          age = newAge;              //public
     };
     var setSalary = function(newsalary) {
          salary = newSalary;          //public
     };
    var setName= function(newName) {
          name = newName;            //public
     };
    var getAge = function() {
          return newAge;             //private
    };
   var getSalary = function() {
          return newSalary;          //private
   };
```

```
var getName = function() {
        return newName;        //private
 };
var increasingSalary = function() {
        return getSalary();          //public
 };
var incrementAge = function() {
        return getAge();             //public
 };

return {
   newAge : setAge,
   newSalary: setSalary,
   newName : setName,
    newSalary : increasingSalary,
    newAge : incrementAge,
   };
} ();
```

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

Answer:

```
var Employee = function() {
  let getAge = getAge( )  {};
  let getSalary = getSalary( )  {};
  let getName = getName( )  {};
    return {
          Public MethodOne() : setAge() {
           // I can call privateMethod()
            },
          Public MethodOne() : setSalary() {  },
          Public MethodOne() : setName()   { }
   }
}) (
```

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

Answer

```
var Employee = (function () {
      let Employee = { } ;

      getAge = getAge () { };

     Employee.setAge = function () {
            setAge(newAge)
    };
    Employee.setSalary = function () {
            setAge(newSalary)
  };
  Employee.setName = function () {
            setAge(newName)
  };
   return Employee;
```

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*( ).

Answer :
```
Employee.extension = function () {
      setAddress(newAddress);
   }
: Employee.extension = function () {
      getAddress();
   }
```

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));
```

ANSWER: Error: Hattori

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));
```

Answer: success : Hattori

The executor should call only one `resolve` or one `reject`. Any state change is final.

All further calls of `resolve` and `reject` are ignored:

16.What is the output of the following code?

```
function job(state) {
   return new Promise(function(resolve, reject) {
if (state) {
    resolve('success');
 } else {
     reject('error');
       }
   });
}
let promise = job(true);
 promise.then(function(data) {
console.log(data);
 return job(false);})

  .catch(function(error) {
console.log(error);
 return 'Error caught';
});
```

Answer : error