Javascript Module Exercises

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

   undefined, 8, 8, 9, 10, 1

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

   Global scope is a scope that accessible from each part of the code in js file. Local scope is a scope that accessible only in the block for example in function block.

3. **Consider the following structure of Javascript code:**

   (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)?**

   81, 25

5. **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: 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 add = (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?**

Counter. Free variable is variable that exist outside the block of the function bit used in this block.

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:**

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

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?**

If the task is to remove all functions from global scope the decision is to use revealing module pattern. Put all methods in the function scope as local and return as result all functions from this 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:**

```
  var employee = (
   var name = name;
  var age = age;
  var salary = salary;
 var getAge  = function (){
    return age;
 var getSalary =  function (){
return salary;
   }
 var getName =  function (){
      return name;
   }
 var setAge: function(newAge){
      age = newAge;
   }
```

```
       var setSalary: function(newSalary){
           salary = newSalary;
        }
        var setName: function(newName){
           name = newName;
        }
       var increaseSalary: function(percentage){
           let sal = getSalary();
           setSalary(sal + ((sal * percentage)/100));
        }
        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 = (

       var name = name;

        var age = age;

        var salary = salary;

       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){

           let sal = getSalary();

           setSalary(sal + ((sal * percentage)/100));
        }

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

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( ).

```
var employee = (function(){
var name;
var age;
var salary;
var address;
var getAge  = function (){return age;}
var getSalary =  function (){return salary;}
 var getName =  function (){return name;}
 var emp = {
      setAge: function(newAge){age = newAge;}
      setSalary: function(newSalary){salary = newSalary;}
      setName: function(newName){name = newName;}
      increaseSalary: function(percentage){
```

```
                let sal = getSalary();
                setSalary(sal + ((sal * percentage)/100));
            }
            incrementAge: function(){setAge(getAge()+1);}
            setAddress: function(addr){
             address = addr;
          }
    }
      return emp;})();
```

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));**

    Error: Hattory

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));**

    Error: Hattory