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

2. Define Global Scope and Local Scope in Javascript.
   - Global scope is global environment for functions, vars, etc.
   - Local Scope is scope inside the function including nested variables, objects, methods, etc.

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()); // => 81
x = 5;
document.write(myFunction()); // => 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'.)?
=>The alert print out 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;
        function add() {
                return counter += 1;
        }
        function reset(){
                return counter = 0;
        }
        return{
                add: add,
                reset: reset
        }
})();
```

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?
=> A variable referred to by a function that is not one parameters of its or local variables. (counter is free variable here).

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
```

=> Solution:

```
var counter = 0;
function make_adder(inc){
        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?

=> Using Closeure-Module pattern

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( )

Solution:

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

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

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

```
 var Module = (function(){
        let name; let age; let salary;
        let getAge = function( ){ return age;}
        let getSalary = function ( ){ return salary;}
        let getName = function ( ){ return name;}
        //locally scope object
        let Employee = {};
                Employee .setAge = function (newAge){ age = newAge;},
```

```
        Employee .setAge = function (newSalary){salary = newSalary;},
        Employee .setAge = function (newName){name = newName;},
        Employee .setAge = function (percentage){
            let newSalary = getSalary() + (getSalary()*percentage);
            setSalary(newSalary);
        },
        Employee .setAge = function ( ){
            let newAge = getAge()+1;
            setAge(newAge);
        }
    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( ).

```
Module.address = null;
Module.getAddress = function(){
    return Module.address;
}

Module.setAddress = function(newAddress){
    Module.address = newAddress;
}
```

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));
```

=>Alert message: '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));
```

=>Alert message: 'Success: Hattori'

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);     // => success
		return job(false);})
	.catch(function(error) {
		console.log(error);  // => error
		return 'Error caught';
});

=> success
=> error
```