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:
→ 1889101

2. Define Global Scope and Local Scope in Javascript.
Answer:
→ Global Scope: is global environment for functions, vars, etc. Before you write a line of JavaScript, you are in the Global Scope. If we declare a variable, it is defined globally. Global scope is needed to access functions defined in other files.
→ Local Scope: Every function gets its own inner scope. If we define a function and create variables inside it, those variables are locally scoped. Any locally scoped items are not visible in the global scope.

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?
Answer: → No.
(b) Do statements in Scope B have access to variables defined in Scope A?
Answer: → Yes.
(c) Do statements in Scope B have access to variables defined in Scope C?
Answer: → No.

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

Answer: → Yes.

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

Answer: → 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:

→ 8125

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

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

Answer:

→ 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.

Answer: →

```
var count = (
              var counter = 0;
              return {
                       add : function (){
                             return counter += 1;
                          },
                       reset : function(){
                             return 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:
   → The “free” is *counter*.
   Free variable is a variable referred to by a function that is not one of its parameters or local variables.

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: →

```
var make_adder = function(inc){
                        let 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?
   Answer: → Use module pattern, put all the code inside this function:

```
(function(){
        …….
 })();
```

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(){
		var name, age, 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){
							var sal = getSalary();
							setSalary(sal + (sal * percentage)*0.01);
				};
		var incrementAge = function(){
						var newAge = getAge() + 1;
						setAge(newAge);
				};
		return {
			setAge: setAge,
			setName: setName,
			setSalary: setSalary,
			increaseSalary: increaseSalary,
			incrementAge: incrementAge
		};
})();
```

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

Answer: →

```
var Employee = (function(){
		var name, age, salary;
		var getAge = function(){ return age; };
		var getSalary = function(){ return salary; };
		var getName = function() { return name; };
		return {
			var setAge = function(newAge) { age = newAge; },
			var setName = function(newName) { name = newName},
			var setSalary = function(newSalary) { salary = newSalary },
			var increaseSalary = function(percentage){
							var sal = getSalary();
							setSalary(sal + (sal * percentage)*0.01);
					},
			var incrementAge = function(){
						var newAge = getAge() + 1;
						setAge(newAge);
					}
		};
})();
```

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

Answer: →

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

            var employeeObject = {};

            employeeObject.setAge = function(newAge) { age = newAge; };
            employeeObject.setSalary = function(newSalary) { salary = newSalary };
            employeeObject.setName = function(newName) { name = newName};
            employeeObject.increaseSalary = function(percentage){
                                    var sal = getSalary();
                                    setSalary(sal + (sal * percentage)*0.01);
                             };
            employeeObject.incrementAge = function(){
                                var newAge = getAge() + 1;
                                setAge(newAge);
                            };
            return employeeObject;
})();
```

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.address = "";
Employee.setAddress = function(newAddress) {
                        this. address = newAddress;
};
Employee.getAddress = 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));
```

Answer: → Output is : 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: → Output is : Success Hattori