# 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);
}
```

Output: undefined 8 8 9 10 1

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

Global scope: A variable declared outside a function, becomes a global. It has global scope: all scripts and functions on a web page can access it. --outside any function

Local scope: Variables declared within a JavaScript function, become LOCAL to the function. These variables have local scope: they can only be accessed within the function. -- inside any 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?*
No.

*(e) Do statements in Scope C have access to variables defined in Scope B?*
No

**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());
```

Output: 81 25

**5.**

```
var foo = 1;
function bar() {
        if (!foo) { // foo is not boolean so false
                var foo = 10;
        }
        alert(foo);
}
bar();
```

**What will the alert print out? (Answer without running the code. Remember 'hoisting'.)?**
output : 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() {
                        return counter  = 0;
                }
        }
 })();
count.add(); count.add(); count.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?**

counter is a “free variable”. Free variable is a variable which is used by a function which is neither parameter nor local variables of 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:**

```
var make_adder = function(inc) {
        counter = 0;
        return function() {
                return counter += inc;
        };
};
add5 = make_adder(5);
add5( );        add5( );        add5( );        // final counter value is 15

add7 = make_adder(7);
add7( );        add7( );        add7( );        // final counter value is 21
```

**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?**

Module pattern can be used to remove all the names from the global namespace. For example

```
(function() {
        //declaration and initializations of functions and variables;
})();
```

**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){
                salary= getSalary()+ (percentage * getSalary()) / 100;
        }
        var incrementAge= function(){
                age = getAge()+1;
        }
        return {
                setAge: setAge, // fieldname : functionName
                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){
                        salary= getSalary()+ (percentage * getSalary()) / 100;
                        },
                incrementAge: function(){age = getAge()+1;}
                };
})();
```

**12. Rewrite your answer to Question 10 using the Stacked 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 = {
                setAge:  function(newAge) {age = newAge;},
                setSalary: function(newSalary) {salary = newSalary;},
                setName: function(newName) {name = newName;},
                increaseSalary: function(percentage){
                        salary= getSalary()+ (percentage * getSalary()) / 100;
                        },
                incrementAge: function(){age = 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; // or Employee.address = "";
Employee.setAddress = function(newAddress) {
                        this.address = newAddress;
                };
Employee.getAddress = function() {
                        return this.address;
                };
```