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.

ANSWER:

Variables declared outside of a function, which isn't limited to any code block, it is said to have a global scope.

Variables declared within a code block are said to be in a Local 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?

A: No. Scope A cannot access to local scope.

(b) Do statements in Scope B have access to variables defined in Scope A?

A: Yes. Local Scope can access Global scope.

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

A: No. Scope B cannot access to child Scope C

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

A: Yes. Local Scope can access Global Scope

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

A: Yes. Scope C can access outer Scope B.

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. 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); // alert shows 10
}
bar();
```

6. Consider the following definition of an *add*( ) function to increment a *counter* variable:

```
var count = (function () {
        var counter = 0;
        const add = function() {
                return counter += 1
        }
        const reset = function() {
                return counter = 0;
        }
        return {
                add: add,
                reset: reset
        };
})();
console.log(count.add());
console.log(count.add());
console.log(count.add());
console.log(count.reset());
console.log(count.add());
console.log(count.add());
```

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.

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 variable is counter. Free variables are the variables that are neither locally declared nor passed as parameter.

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) {
  var counter = 0;
  return function () {
    counter += inc;
    console.log(counter); // print the values
  };
};

const add5 = make_adder(5);
add5( ); add5( ); add5( ); // final counter value is 15
const 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?

ANSWER: Just use Module Pattern in order to remove names from the Global namespace.

10. Using the *Revealing Module Pattern*, write a Javascript definition of a Module that creates an *Employee* Object with the following fields and methods:

```
const EmployeeModule = (function functionName(){
        let age = 0;
        let name = "";
        let salary = 0;
        const setAge = function(newAge) {
                age = newAge;
        }
        const setSalary = function(newSalary) {
                salary = newSalary;
        }
```

```
        const setName = function(newName) {
                name = newName;
        }
        const getAge = function() {
                return age;
        }
        const getSalary = function() {
                return salary;
        }
        const getName = function() {
                return name;
        }
        const increaseSalary = function (percentage) {
                setSalary(getSalary() + (percentage * getSalary())/ 100);
        }
        const incrementAge = function () {
                setAge(getAge() + 1);
        }
        return {
                setAge,
                setSalary,
                setName,
                increaseSalary,
                incrementAge
        }
})();
```

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

```
const EmployeeModule = (function functionName(){
        let age = 0;
        let name = "";
        let salary = 0;
        const getAge = function() {
                return age;
        }
        const getSalary = function() {
                return salary;
        }
        const getName = function() {
                return name;
        }
        return {
                setAge : function(newAge) {
                        age = newAge;
                },
                setSalary: function(newSalary) {
```

```
                salary = newSalary;
        },
        setName: function(newName) {
                name = newName;
        },
        increaseSalary: function (percentage) {
                setSalary(getSalary() + (percentage * getSalary())/ 100);
        },
        incrementAge: function () {
                setAge(getAge() + 1);
        }
    }
})();
```

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

```
const EmployeeModule = (function functionName(){
    let age = 0;
    let name = "";
    let salary = 0;
    const employeeObject = {}
    const getAge = function() {
        return age;
    }
    const getSalary = function() {
        return salary;
    }
    const getName = function() {
        return name;
    }
    employeeObject.setAge = function(newAge) {
        age = newAge;
    };
    employeeObject.setSalary = function(newSalary) {
        salary = newSalary;
    };
    employeeObject.setName = function(newName) {
        name = newName;
    };
    employeeObject.increaseSalary = function (percentage) {
        this.setSalary(getSalary() + (percentage * getSalary())/ 100);
    };
    employeeObject.incrementAge = function () {
        this.setAge(getAge() + 1);
    }
    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*( ).

```
EmployeeModule.address = "";

EmployeeModule.getAddress = function (){

        return this.address;

}

EmployeeModule.setAddress = function (newAddress){

        this.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));
```

OUTPUT: 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));
```

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

OUTPUT:

success

error