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:

undefined 8 8 9 10 1

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

Answer:

Global Scope: Global variables are variables that are defined outside of functions.

Local/Function Scope: local variables are variables that are defined within functions.

3. Consider the following structure of Javascript code:

```
// Scope A
function XFunc () {
      // Scope B
      function YFunc () {
            // Scope C
      };
};
```

Answer:

(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());
x = 5;
document.write(myFunction());
```

Answer:

81 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'.)?

Answer:

10 - because there is hoisting, and the value of foo is undefined, then foo will be assigned a value 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;

  var add = function(){
      counter += 1;
  };

  var reset = function(){
      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?

Answer: **'counter' is free variable.**

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:

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

```
}
```

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: **We can use functions that can be invoked Immediately**

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;
            var age;
            var salary;
            var getName = () => {return name};
            var getAge = () => {return age};
            var getSalary = () => {return salary};

            var setName = (newName) => {name = newName;},
            var setAge = (newAge) => {age = newAge;},
            var setSalary = (newSalary) => {salary =
newSalary;},
            var increaseSalary = (percentage) => {salary =
salary * (1 + percentage / 100);},
            var incrementAge = () => {age ++ ;},

            return {
                'setName' : setName,
                'setAge' : setAge,
                '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;
            var age;
            var salary;
            var getName = () => {return name};
            var getAge = () => {return age};
            var getSalary = () => {return salary};

            return {
                'setName' : (newName) => {name = newName;},
                'setAge' : (newAge) => {age = newAge;},
                'setSalary' : (newSalary) => {salary =
newSalary;},
                'increaseSalary' : (percentage) => {salary =
salary * (1 + percentage / 100);},
                'incrementAge' : () => {age ++ ;},
            }
        })();
```

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

Answer:

```
var Employee = (function () {
            var Contructor = {};

            var name;
            var age;
            var salary;
            var getName = () => {return name};
            var getAge = () => {return age};
            var getSalary = () => {return salary};

            Contructor.setName = (newName) => {name = newName;},
            Contructor.setAge = (newAge) => {age = newAge;},
            Contructor.setSalary = (newSalary) => {salary =
newSalary;},
            Contructor.increaseSalary = (percentage) => {salary
= salary * (1 + percentage / 100);},
            Contructor.incrementAge = () => {age ++ ;},

            return Contructor;
        })();
```

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:

```
var Employee = (function () {
            var name;
            var age;
            var salary;

            var getName = () => {return name};
            var getAge = () => {return age};
            var getSalary = () => {return salary};

            var setName = (newName) => {name = newName;},
            var setAge = (newAge) => {age = newAge;},
            var setSalary = (newSalary) => {salary =
newSalary;},
            var increaseSalary = (percentage) => {salary =
salary * (1 + percentage / 100);},
            var incrementAge = () => {age ++ ;},

            return {
                'address':null,
                'getAddress': () => {return address;},
                'setAddress': (newAddress) => {address =
newAddress;},

                'setName' : setName,
                'setAge' : setAge,
                'setSalary' : setSalary,
                'increaseSalary' : increaseSalary,
                'incrementAge' : incrementAge,
            }
        })();
```

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:

```
It is going to alert => "Error: Hattori"
        then it is going to change the state of promise as a `fullfilled`
```

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:

```
            It is going to alert => "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';
});
```

Answer:

```
            In console:
                    success
                    error

                    promise result will be `Error caught`
```