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);
}
```

Soln: undefined 8 8 9 10 1

2. Define Global Scope and Local Scope in Javascript.

Soln: Global scope is a scope where anyone anywhere can access our code (variables or functions). Local scope is limited to the scope of our javascript code either inside modules or functions.

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?
    (b) Do statements in Scope B have access to variables
defined in Scope A?
    (c) Do statements in Scope B have access to variables
defined in Scope C?
    (d) Do statements in Scope C have access to variables
defined in Scope A?
    (e) Do statements in Scope C have access to variables
defined in Scope B?
```

Soln: a). No b). Yes c). No d).Yes e). 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());
```

soln: 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’.)?

Soln: 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.

Soln:

```
var count=(function () {
    var counter=0;
    return {
        add:function () {
            return counter+=1;
        },
        reset:function () {
            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?

Free variables are simply the variables that are neither locally declared nor passed as parameter. In this case counter is a 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
```

soln:

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

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?

Soln: Modularize it to remove it from global namespace, we use notation like this: (the Javascript code here)();

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()
```

Soln:

```
var Employee = (function () {
    let name;
    let age;
    let salary;
    let setAge=function (newAge) {
        age=newAge;
    };
    let setSalary=function (newSalary) {
        salary=newSalary;
    };
    let setName=function (newName) {
        name=newName;
    };
    let getAge=function () {
        return age;
    };
    let getSalary=function () {
       return salary;
    };
    let getName=function () {
        return name;
    };
    let increaseSalary=function (percentage) {
        let s=getSalary();
       return s+=((s*percentage)/100);
    };
    let incrementAge=function () {
        let ag=getAge();
        return ag++;
    };
    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.

Soln:

```
var Employee = (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 s=getSalary();
            return s+=((s*percentage)/100);},
        incrementAge: function () {
            let ag=getAge();
            return ag++;}
    };
})();
```

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

Soln:

```
var Employee = (function () {
    let name;
    let age;
    let salary;
    let myObject= {};
    let getAge=function () {
```

```
        return age;
    };
    let getSalary=function () {
       return salary;
    };
    let getName=function () {
        return name;
    };
    myObject.setAge = function (newAge) {
        age=newAge;};
    myObject.setSalary = function (newSalary) {
        salary=newSalary;};
    myObject.setName = function (newName) {
        name=newName;};
    myObject.increaseSalary = function (percentage) {
        let s=getSalary();
        return s+=((s*percentage)/100);};
    myObject.incrementAge = function () {
        let ag=getAge();
        return ag++;};
    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().

Soln:

```
var Employee = (function () {
    let name;
    let age;
    let salary;
    let setAge=function (newAge) {
        age=newAge;
    };
    let setSalary=function (newSalary) {
        salary=newSalary;
    };
    let setName=function (newName) {
        name=newName;
    };
    let getAge=function () {
        return age;
    };
    let getSalary=function () {
```

```
        return salary;
    };
    let getName=function () {
        return name;
    };
    let increaseSalary=function (percentage) {
        let s=getSalary();
        return s+=((s*percentage)/100);
    };
    let incrementAge=function () {
        let ag=getAge();
        return ag++;
    };
    return {
        setAge:setAge,
        setSalary:setSalary,
        setName:setName,
        increaseSalary:increaseSalary,
        incrementAge:incrementAge
    };
})();
var address;
Employee.setAddress = function (newAddress) {
    address=newAddress;
};
Employee.gerAddress = function () {
    return 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));
```

Soln: 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));
```

Soln: 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';
});
```

Soln:

```
    success
    error
```