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) {  //8,9,10
         document.write(x);  //1
        document.write(a);    //8
        var f = function(a, b, c) {    //8,9,10
                          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
  }
```

Ans: undefined 8 8 9 10 1

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

Ans: In Global Scope, variables are declared outside of the function and are accessible from anywhere.

In Local scope, variables are declared inside of the function and have its own scope. It is accessible only inside the function and its nested 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?
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())
;// output 81
 x = 5;
document.write(myFunction());// output 25
```

Ans: 81 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); //outout alert 10
}
bar();
```

Ans: 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.

Ans

```
var countVal = (function(){

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



}();
```

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?**

Ans: In the definition of *add*( ) shown in question 6:

Free variable= counter

In the context of a function closure, free variable is a variable referred to function which is not its local variable or 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:**

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


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

Ans:

```
var countVal = (function () {
    var counter = 0;
    var counterObj = {
        add: function () {
            counter += 1;
            return counter;
        },
        reset: function () {
            counter = 0;
            return counter;
        },
        make_adder: function (inc) {

            return function (inc) {
                counter += inc;
                return counter;

            }
        }
    }
    return counterObj;
}();
```

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?
   Ans: To remove all names from Global namespace, **Use Module Patter or Object Literals.**

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( )

Ans:

```
var Employee=(function (){
   var name;
   var age;
   var salary;
   var getAge = function(){
      return age;
   }
   var getSalary = function(){
      return salary;
   }
   var getName = function(){
      return name;
   }
   var setName=function(newName){
      this.name=newName;
   }
   var setAge=function(newAge){
      this.age=newAge;
   }
   var setSalary=function(newSalary){
      this.salary=newSalary;
   }
   var increaseSalary=function(percentage){
      setSalary(getSalary()+(percentage*getSalary())/100);
   }
   var incrementAge=function(){
      setAge(getAge()+1);
   }

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

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

```
    var employee =
        (function(){
          //fields
          let name;
          let age;
          let salary;
    //getter & setter methods

    let getAge = function(){return age;};

    let getSalary = function(){return salary;};

    let getName = function(){return name;};

    return {

        setName : function(newName){name = newName},

        setAge : function(newAge){age = newAge},

        setSalary: function(newSalary){salary = newSalary},

        increaseSalary : function(percentage){salary  = getSalary() + (getSalary()*percentage/100);},

        incrementAge: function(){age =getAge()+1;}

    };

})();
```

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

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

    var localObj = {};

    localObj.setName = function (newName) {
        this.name = newName;
    };
    localObj.setAge = function (newAge) {
        this.age = newAge;
    };
    localObj.setSalary = function (newSalary) {
        this.salary = newSalary;
    };
    localObj.increaseSalary = function (percentage) {
        setSalary(getSalary() + (percentage * getSalary()) / 100);
    };
    localObj.incrementAge = function () {
        setAge(getAge() + 1);
    };

    return localObj;
})();
```

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( ).**

Ans

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

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