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.

Global Scope : a scope which defines the global environment for function and variables
Local Scope: the inner scope defined by a function to determine the visibility and accessibility of variables

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?

Answer: B,D,E are Yes…while A and C are No

4. What will be printed by the following (answer without running it)?

```
var x = 9;
function myFunction() {
```

```
return x * x;
} D
ocument.write(myFunction());
x = 5;
document.write(myFunction());
```

Answer: 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);
}
bar();
```

Answer: 10

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

```
var count = (function() {
                var counter = 2;
                var add = function() {
                          return counter += 1;
                       }
                var reset = function(){
                             return counter;
                          }
                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 the free variable

Free variable are those variables which are not locally defined inside the closure function or not passed as a parameter to the closure but can be used by the closure.

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 (){
    var counter = 0;
        return function(inc){
            counter+=inc;
            document.write(counter);

        }
    })();
    make_adder(5);
    make_adder(5);
```

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:

The simplest modification to do would be to use module pattern, wrapping the the entire code construction inside an anonymous function.

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

```
var emp = function(){

    var employee ={
       'nam': 'John',
       age:40,
       salary:6000,

       getAge:function(){return this.age},
       getSalary:function(){return this.salary},
       getName:function(){return this.nam},
       setAge:function(newAge){age = newAge;},
       setSalary: function(newSalary){salary = newSalary},
       setName: function(newName){nam = newName},
       increaseSalary:function(percentage){return this.getSalary() + this.getSala
ry() * percentage},
       incrementAge: function() {return  this.getAge() + 10}
    }
  function setName(name){
  return  employee.setName(name);
```

```
  }

    function setSalary(){
        return  employee.setSalary();

        }
    function setAge(age){
            return  employee.setAge(age);

            }

    function increaseSalary(p){
        return  employee.increaseSalary(p);

    }
    function incrementAge(){
        return  employee.incrementAge();

    }
       return {
                setName: setName,
                setSalary: setSalary,
                setAge: setAge,
                increaseSalary:increaseSalary,
                incrementAge:incrementAge
       }

}();

console.log(emp.increaseSalary(2));
console.log(emp.incrementAge());
```

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

```
var emp = function(){

    var employee ={
       'nam': 'John',
       age:40,
       salary:6000,
```

```
      getAge:function(){return this.age},
      getSalary:function(){return this.salary},
      getName:function(){return this.nam},
      setAge:function(newAge){age = newAge;},
      setSalary: function(newSalary){salary = newSalary},
      setName: function(newName){nam = newName},
      increaseSalary:function(percentage){return this.getSalary() + this.getSala
ry() * percentage},
      incrementAge: function() {return  this.getAge() + 10}
    }
  function setName(name){
  return  employee.setName(name);

  }

    function setSalary(){
        return  employee.setSalary();

        }
    function setAge(age){
            return  employee.setAge(age);

            }

    function increaseSalary(p){
        return  employee.increaseSalary(p);

    }
    function incrementAge(){
        return  employee.incrementAge();

    }
       return {
                setName:  function(n){return setName(n)},
                setSalary: function(s){return setSalary(s)},
                setAge: function(a){return setAge(a)},
                increaseSalary:function(n){return increaseSalary(n)},
                incrementAge:function(){return incrementAge()},
       }

}();

console.log(emp.increaseSalary(2));
console.log(emp.incrementAge());
```

12.Rewrite your answer to Question 10 using the Locally scoped object literal Object Literal Pattern

```
var emp = function(){

    var employee ={
       'nam': 'John',
       age:40,
       salary:6000,

       getAge:function(){return this.age},
       getSalary:function(){return this.salary},
       getName:function(){return this.nam},
       setAge:function(newAge){age = newAge;},
       setSalary: function(newSalary){salary = newSalary},
       setName: function(newName){nam = newName},

    }


    function getSalary(){
        return  employee.getSalary();

        }
    function getAge(){
            return  employee.getAge();

            }

    employee.increaseSalary = function(percentage){return getSalary() +getSalary(
) * percentage}
    employee.incrementAge = function() {return  getAge() + 10} ;

       return employee

}();

console.log(emp.increaseSalary(2));
console.log(emp.incrementAge());
```

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.extension= function(){
    varaddress;
    functionsetAddress= function(newAddress){
    address= newAddress;
     }
    functiongetAddress=function(){
        returnaddress;
    }
return{
     address:address,
     setAddress:setAddress,
     getAddress:getAddress
    }

}
```

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 : Error: Hattori

15 . What is the output of the following code?

```
const promise = new Promise((resolve, reject) => {
reject(“Hattori”);
setTimeout(()=> reject(“Yoshi”), 500);
});
promise.then(val => alert(“Success: “ + val))
.catch(e => alert(“Error: “ + e));
```

Answer : 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: success

error