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); //i
    document.write(a); //ii
    var f = function(a, b, c) {
        b = a;
        document.write(b); //iii
        b = c;
        var x = 5;
    }
    f(a,b,c);
    document.write(b); //iv
    var x = 10;
}
c(8,9,10);
document.write(b);
document.write(x);
```

Answers:

c(8,9,10) ->

undefined //i x

8 //ii a

8 //iii b

9 //iv b

document.write(b) -> 10

document.write(x) -> undefined

2.Define Global Scope and Local Scope in Javascript.

Global Scope: In Javascript, there is one global scope. Variable defined in that scope can be accesed anywhere in file.

Local Scope: Function define its own scope called local scope. The variable defined in that scope are only accessible to that 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?

(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:

(a). false

(b). true

(c). false

(d). true

(e). true

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());
```

Answers:

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:

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.

Answer:

```
        var count = {
          counter: 0,
          add: function (input) {
                if (input) {
                   this.counter += input;
                }
                else {
                   this.counter++;
                }
                return this.counter;
               },
          reset: function () {
             this.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?

Answer:

In the cntext of javascript closure, free variables are those variables that a child scope can use because those variable were declared in the enclosing or parent scope.

var counter = 0; counter is a free variable in above add().

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:

Answer:

```
var add = (function(){
        let counter = 0;
        return function(input){
            if(input){
                counter = counter+input;
             }
             else{
             counter = counter+1;
             }
             return counter;
           }
    })();
  var make_adder = function (inc) {
            return function () {
               return add(inc);
```

```
    };
};
```

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:

Using Module pattern we can remove all the names from 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:

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, age, salary;

    var getAge = function () {
            return age;
```

```
        };
var getSalary = function () {
            return salary;
        };
var getAgePubFunc = function () {
            return getAge();
        };
var getSalaryPubFunc = function(){
            return getSalary();
        };
var setAge = function (newAge) {
        age = newAge;
      };
var setSalary = function(newSalary){
            salary = newSalary;
        };
var setName = function(newName){
          name = newName;
      };
var increaseSalary = function(percentage){
            var sal = getSalary();
           setSalary(sal + (sal*percentage)*0.01);
        };
var incrementAge = function(){
            var newAge = getAge() + 1;
            setAge(newAge);
        };
return {
   setAge: setAge,
```

```
        setName:setName,
        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, age, salary;

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

```
    increaseSalary: function(percentage){
                var sal = getSalary();
                setSalary(sal + (sal*percentage)*0.01);
            },
    incrementAge: function(){
                var newAge = getAge() + 1;
                setAge(newAge);
            }
    };
})();
```

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

Answer:

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

    let getAge = function () {
            return age;
        };
    let getSalary = function () {
            return salary;
        };
    let employeeObj = {};
    employeeObj.setAge = function(newAge) {
            age = newAge;
        };
    employeeObj.setName = function(newName) {
            name = newName;
```

```
            };
        employeeObj.setSalary = function(newSalary) {
                salary = newSalary;
             };
        employeeObj.increaseSalary= function(percentage){
                  var sal = getSalary();
                  setSalary(sal + (sal*percentage)*0.01);
                };
        employeeObj.incrementAge = function(){
                  var newAge = getAge() + 1;
                  setAge(newAge);
                };
    };
    return employeeObj;
})();
```

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:

```
Employee.address = "";
Employee.setAddress = function(newAddress) {
            this.setAddress = newAddress;
};
Employee.getAddress = function() {
            return this.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));
```

Answers:

```
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));
```

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:

```
sucess
error
Error caught
```