

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);
}
```

Ans Undefined 8 8 9 10 1

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

Ans

Global Scope : A variable defined outside a function has a global scope

```
let name = "sujan";  
// code here can use name  
  
function myFunction() {  
  // code here can also use variable name  
}
```

Local Scope : Variables declared within a JavaScript function, become local to the function.

```
// code here can NOT use carName  
  
function myFunction() {  
  let carName = "Volvo";  
  // code here CAN use carName  
}  
  
// code here can NOT use carName
```

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

ANS 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’.)?

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 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?

ANS

variable counter is a free variables

a closure instance is created when the inner function is returned or assigned to a variable and it attaches itself to the free variables from the surrounding environment to "close" up those stray references

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 make_adder = (function(inc)) {  
    var counter = 0;  
    return function () {  
        return counter += inc;  
    }  
});  
  
add5 = make_adder(5); add
```

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?

→ by using module pattern

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()

```
let Employee = (function (name,age,salary) {  
    let setName = function (n) {  
        name=n;  
    };  
    let setAge = function (a) {  
        age=a;  
    };  
    let setSalary = function (s) {  
        salary=s;  
    };  
    let getAge = function () {  
        return age;  
    };  
    let getName = function () {  
        return name;  
    };  
    let getSalary = function () {  
        return salary;  
    };  
});
```

```
};  
let increaseSalary = function (percentage) {  
    salary = (percentage/100)*(this.getSalary()+this.getSalary());  
};  
let increaseAge = function () {  
    age++;  
    return getAge();  
}  
return {  
    setName: setName,  
    setAge: setAge,  
    setSalary:setSalary,  
    increaseSalary:increaseSalary,  
    increaseAge:increaseAge  
};  
});
```

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

```
let Employee = (function (name,age,salary) {  
    let getAge = function () {  
        return age;  
    };  
    let getName = function () {  
        return name;  
    };  
    let getSalary = function () {  
        return salary;  
    };  
  
    return {  
        setName: function (n) {  
            name=n;  
        },  
        setAge: function (a) {  
            age=a;  
        },  
        setSalary:function (s) {  
            salary=s;  
        },  
        increaseSalary:function (percentage) {  
            salary = (percentage/100)*(this.getSalary()+this.getSalary());  
        },  
        increaseAge:function () {  
            age++;  
            return getAge();  
        }  
    };  
});
```


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

```
let Employee = (function (name,age,salary) {
    let getAge = function () {
        return age;
    };
    let getName = function () {
        return name;
    };
    let getSalary = function () {
        return salary;
    };
    let employeeObject = {};

    employeeObject.setName = function (n) {
        name=n;
    };

    employeeObject.setAge= function (a) {
        age=a;
    };

    employeeObject.setSalary=function (s) {
        salary=s;
    };

    employeeObject.increaseSalary=function (percentage) {
        salary = (percentage/100)*(getSalary()+getSalary());
    };

    employeeObject.increaseAge=function () {
        age++;
        return getAge();
    };
    return employeeObject;
});
```

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()*.

```
let extendedEmployee = Object.create(employee);
extendedEmployee.address = "Fairfield";
extendedEmployee.setAddress = function( newAddress){
    address=newAddress;
}
extendedEmployee.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));
```

Ans 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));
```

Ans 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';  
    });
```