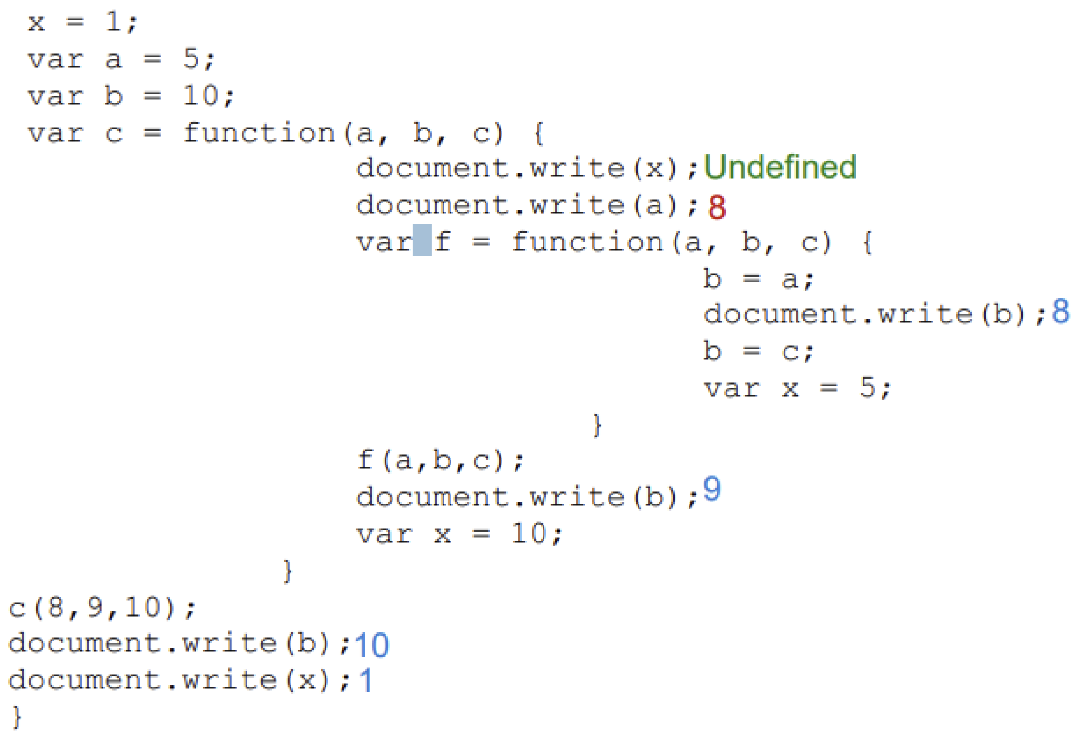
Javascript Module Exercises

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);Undefined
                document.write(a);8
                var f = function(a, b, c) {
                                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
}
```

Answer : Undefined , 8 , 8 , 9, 10, 1

2. Define Global Scope and Local Scope in Javascript

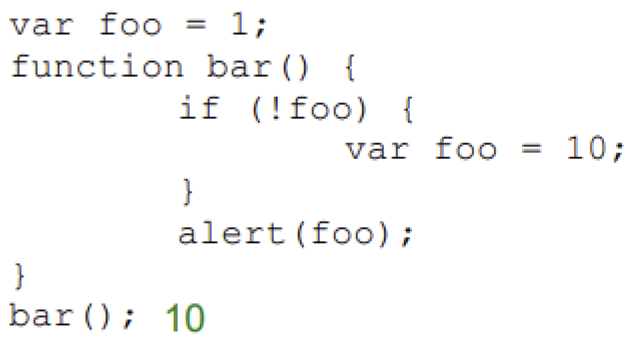
- **Local scope** is a scope within a function. If we have a function and declare values inside , those are locally scoped. They can only be accessed from within the function
- **Global scope -** any variable declared in windows level . Global variables are variables that are defined outside of functions. These variables have global scope, so they can be used by any function without passing them to the function as parameters.

3. Consider the following structure of Javascript code:

(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)?
   Answer : **81, 25,**

5.
Answer : 10



```
var foo = 1;
function bar() {
        if (!foo) {
                var foo = 10;
        }
        alert(foo);
}
bar();  10
```

Answer : **<u>10</u>**

6. **<u>Solution</u>**

```
var count=( function(){

      var counter=0; //private

      var add=function() {
          return ++counter;
          }

      var reset=function() {
                counter=0
            return counter;
          }

     return {
      add:add,
     reset:reset
     }

}
)();
```

7. **<u>Solution</u>**
   counter
   variables that are neither arguments nor local variables

8. **solution**

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

console.log(make_adder);
add5=make_adder(5);

add5();
add5();
console.log(add5());
```

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?

<u>Solution</u> : wrapping the code in module pattern design specifically revealing

10.

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

  var increaseSalary=function(percentage) // uses private getSalary( )
  {
  salary+=parseFloat(getSalary()*percentage/100);
  }
  var incrementAge=function() // uses private getAge( )
  {
  age=getAge()+1;
  }
return {
    setAge:setAge,
    setSalary:setSalary,
    setName:setName,
    increaseSalary:increaseSalary,
    incrementAge:incrementAge
}

}

)();
```

11.

```
var employee = (function () {

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

        return {
            setAge: function (newAge) {
                age = newAge;
            },
            setSalary: function (newSalary) {
                salary = newSalary;
            },
            setName: function (newName) {
                name = newName;
            },
            increaseSalary: function (percentage) // uses private getSalary( )
            {
                salary += parseFloat(getSalary() * percentage / 100);
            },
            incrementAge: function () // uses private getAge( )
            {
                age = getAge() + 1;
            }
        }

    }

)();
```

12.

```
var employee = (function () {

        // locally scoped Object
        var myObject = {};

        var name;
        var age;
        var salary;
        var getAge = function () {
            return age;
        }
        var getSalary = function () {
            return salary;
        }
        var 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) // uses private getSalary( )
        {
            salary += parseFloat(getSalary() * percentage / 100);
        };
        myObject.incrementAge = function () // uses private getAge( )
        {
            age = getAge() + 1;
        };

        return myObject;

    }

)();
```

13.

```
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 setAge = function (newAge) {
            age = newAge;
        }
        var setSalary = function (newSalary) {
            salary = newSalary;
        }
        var setName = function (newName) {
            name = newName;
        }

        var increaseSalary = function (percentage) // uses private getSalary( )
        {
            salary += parseFloat(getSalary() * percentage / 100);
        }
        var incrementAge = function () // uses private getAge( )
        {
            age = getAge() + 1;
        }
        return {
            setAge: setAge,
            setSalary: setSalary,
            setName: setName,
            increaseSalary: increaseSalary,
            incrementAge: incrementAge
        }

    }
```

```
var employee2 = (function (employee) {

    var address;
    employee.setAddress = function (newAddress) {
        address = newAddress;
    };
    employee.getAddress = function () {
        return address;
    };

    return employee;

})(employee || {});
```

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) => {
      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: Success
Error