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.
   Anything declared in global scope can be accessed anywhere in the script whereas local scoped variables/functions can only be accessed within that local function/object.

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());
```

   Answer : 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'.)?

   Answer : 10

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

```
var counterModule = (function() {
  var counter = 0;

  return{
       add : function(){
           counter+=1;
      document.write(counter);
    },
    reset: function(){
       counter = 0;
      document.write(counter);
    }
  }
})();
```

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?
   The free variable in given code snippet is “counter” because as the definition says it’s a variable that is not declared within function nor passed inside via 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:

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

   By wrapping up all functions and variables in a module.

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 () {
  //private fields
  var name, age, salary;
  let getName = function getName() {
  return this.name;
  }
  let setName = function (name) {
  this.name = name;
  }
  let getAge = function () {
  return this.age;
  }
  let setAge = function (age) {
  this.age = age;
  }
  let getSalary = function () {
  return this.salary;
  }
  let setSalary = function (salary) {
  this.salary = salary;
  }
  let increaseSalary = function (percentage) {
  setSalary(parseFloat(getSalary()) + parseFloat(getSalary() * percentage / 100));
  }
  let incrementAge = function () {
  setAge(parseInt(getAge()) + 1);
  }
```

```
  return {
  getName: function () { return getName(); },
  setName: function (name) { setName(name); },
  getAge: function () { return getAge(); },
  setAge: function (age) { setAge(age); },
  getSalary: function () { return getSalary(); },
  setSalary: function (salary) { return setSalary(salary); },
  increaseSalary: function (percentage) { increaseSalary(percentage); },
  incrementAge: function () { incrementAge(); }
  };
 };


 var emp = Object.create(Employee);
 emp.setAge(100);
 emp.setName('Shree');
 emp.setSalary(12000);
 emp.increaseSalary(2);

 console.log(emp);
```

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

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

  let getAge = function() {
    return age;
  }

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

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

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

  }
};


var emp = employee();
emp.setAge(100);
emp.setName('Shree');
emp.setSalary(12000);
console.log(emp);

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

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

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

  let getAge = function() {
    return age;
  }

  let getSalary = function() {
    return salary;
  }

  let getName = function() {
    return name;
  }

  let employeePublicFunctions = {};

  employeePublicFunctions.setAge = function(newAge) {
    age = parseFloat(newAge);
  };
  employeePublicFunctions.setSalary = function(newSalary) {
    salary = parseFloat(newSalary);
  };

  employeePublicFunctions.setName = function(newName) {
    name = newName;
  };
  employeePublicFunctions.increaseSalary = function(percentage) {
    this.setSalary(getSalary() + getSalary() * (percentage / 100));
    return getSalary();
  };

  employeePublicFunctions.incrementAge = function() {
    this.setAge(getAge() + 1);
    return getAge();
```

```
  };

  return employeePublicFunctions;


};


var emp = employee();
emp.setAge(100);
emp.setName('Shree');
emp.setSalary(12000);
console.log(emp);

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( ).

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

  let getAge = function() {
    return age;
  }

  let getSalary = function() {
    return salary;
  }

  let getName = function() {
    return name;
  }

  let employeePublicFunctions = {};
```

```
  employeePublicFunctions.setAge = function(newAge) {
    age = parseFloat(newAge);
  };
  employeePublicFunctions.setSalary = function(newSalary) {
    salary = parseFloat(newSalary);
  };

  employeePublicFunctions.setName = function(newName) {
    name = newName;
  };
  employeePublicFunctions.increaseSalary = function(percentage) {
    this.setSalary(getSalary() + getSalary() * (percentage / 100));
    return getSalary();
  };

  employeePublicFunctions.incrementAge = function() {
    this.setAge(getAge() + 1);
    return getAge();
  };

  return employeePublicFunctions;


})();

(function() {
  employee.address = '';
  employee.setAddress = function(adr) {
    this.address = adr;
  };
  employee.getAddress = function() {
    return this.address;
  };

})();
```

```
var emp = Object.create(employee);
emp.setAge(100);
emp.setName('Shree');
emp.setSalary(12000);
emp.setAddress("Iowa");
console.log(emp);

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

It shows alert with message -> 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));
```

Shows : 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';
});
```

It logs success & error in the console.