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);
    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);
```

⇨ 1 8 8 9 10 1

2. Define Global Scope and Local Scope in JavaScript.

There is one global scope that is environment for functions, vars, etc.
And each function defined has its own (nested) local scope (function 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? 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());
```

⇨ 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'?)

It will alert value 1.

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.

```
let counter = (() => {
    let counter = 0;
    const add = () => counter++;
    const reset = () => counter = 0;
    return { add, 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?

The "free" variable is `counter`. The "free" variable is variable that is used locally but defined in an enclosing scope.

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
```

```
const make_adder = number => {
    let count = 0;
    return () => count += number;
}
```

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?

We can put all those functions and variables into a module pattern with IIFE method.

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)
Public Method: incrementAge( ) // uses private getAge( )

```
let Employee = (() => {
    let _name;
    let _age;
    let _salary;
    const _getName = () => _name;
    const _getAge = () => _age;
    const _getSalary = () => _salary;
    const setName = newName => _name = newName;
    const setAge = newAge => _age = newAge;
    const setSalary = newSalary => _salary = newSalary;
    const increaseSalary = percentage => {
        let salary = _getSalary() * (1 + percentage / 100);
        setSalary(salary);
        return salary;
    }
    const increaseAge = () => {
        let age = _getAge();
        age++;
        setAge(age);
        return age;
    }
    return { setAge, setName, setSalary, increaseSalary, increaseAge };
})();
```

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

```
let Employee = (() => {
    let _name;
    let _age;
    let _salary;
    const _getName = () => _name;
    const _getAge = () => _age;
    const _getSalary = () => _salary;
    return {
        setName: newName => _name = newName,
        setAge: newAge => _age = newAge,
        setSalary: newSalary => _salary = newSalary,
        increaseSalary: percentage => {
            let salary = _getSalary() * (1 + percentage / 100);
            setSalary(salary);
            return salary;
        },
        increaseAge: () => {
            let age = _getAge();
            age++;
            setAge(age);
            return age;
        }
    };
})();
```

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

```
let Employee = (() => {
    let _name;
    let _age;
    let _salary;
    let employee = {};
    const _getName = () => _name;
    const _getAge = () => _age;
    const _getSalary = () => _salary;
    employee.setName = newName => _name = newName;
    employee.setAge = newAge => _age = newAge;
    employee.setSalary = newSalary => _salary = newSalary;
    employee.increaseSalary = percentage => {
        let salary = _getSalary() * (1 + percentage / 100);
        setSalary(salary);
        return salary;
    }
    employee.increaseAge = () => {
        let age = _getAge();
        age++;
        setAge(age);
        return age;
    }
    return employee;
})();
```

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 Employee1 = ((Employee) => {
    Employee.address = "";
    Employee.setAddress = newAddress => address = newAddress;
    Employee.getAddress = () => 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))
```

⇨ Open alert box with text: “Error: Hattori”

15. What is the output of the following code?

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

⇨ Open alert box with text: “Success: Hattor”

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';
    });
```

- success
- error
- *Promise {<resolved>: "Error caught"}*
    - __proto__: Promise
    - *[[PromiseStatus]]*: "resolved"
    - *[[PromiseValue]]*: "Error caught"