**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);**

**}**

**Output:**

undefined

8

8

9

10

1

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

* Global scope: Before you write a line of JavaScript, you are in the Global Scope. If we declare a variable, it is defined globally:

// global scope

var name = 'Todd';

* Local scope:
  + There is one global scope, and each function defined has its own (nested) local scope.
  + If we define a function and create variables inside it, those variables are locally scoped:

// Scope A: Global scope out here var

myFunction = function () {

// Scope B: Local scope in here

};

* + Any locally scoped items are not visible in the global scope.

1. **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

1. **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());

**Output:**

81

25



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’.)?**

Alert print: 10

1. **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.**

var count = (function () {  
 var counter = 0;  
  
 let add = function () {  
 counter++;  
 };  
  
 let reset = function () {  
 counter = 0;  
 };  
  
 return {  
 add: add,  
 reset: reset  
 };  
})();

// count.add();

// count.reset();

1. **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?**

counter is a free variable.

1. **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**

var make\_adder = (function (inc) {  
 var counter = 0;  
 return function() {  
 return counter += inc;  
 }  
});  
  
var add5 = make\_adder(5);  
console.log(add5()); //5  
console.log(add5()); //10  
console.log(add5()); //15

1. **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?**

Make it as a Module:

var Module = (function () {

// code: function and variable declarations.

})();

1. **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( )**

var Employee = (function() {  
 let name, age, salary;  
 let that = this;  
  
 let setAge = function(newAge) {  
 that.age = newAge;  
 }  
 let setName = function(newName) {  
 that.name = newName;  
 }  
 let setSalary = function(newSalary) {  
 that.salary = newSalary;  
 }  
  
 let getAge = function() {  
 return that.age;  
 }  
  
 let getName = function() {  
 return that.name;  
 }  
  
 let getSalary = function() {  
 return that.salary;  
 }  
  
 let increaseSalary = function(percentage) {  
 that.salary = getSalary() + (getSalary()\*percentage)/100;  
 }  
  
 let incrementAge = function() {  
 that.age = getAge() + 1;  
 }  
  
 return {  
 setAge: setAge,  
 setSalary: setSalary,  
 setName: setName,  
 increaseSalary: increaseSalary,  
 incrementAge: incrementAge,  
 // for test only  
 info: function () {  
 return "---" + that.name + "---" + that.age + '---' + that.salary;  
 }  
 }  
  
})();  
  
  
Employee.setAge(10);  
Employee.setName('Wick');  
Employee.setSalary(1000);  
Employee.increaseSalary(50);  
Employee.incrementAge();  
console.log(Employee.info()); // ---Wick---11---1500

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

var Employee = (function() {  
 let name, age, salary;  
 let that = this;  
  
 let getAge = function() {  
 return that.age;  
 }  
  
 let getName = function() {  
 return that.name;  
 }  
  
 let getSalary = function() {  
 return that.salary;  
 }  
  
 return {  
 setAge: function (newAge) {  
 that.age = newAge;  
 },  
 setSalary: function(newSalary) {  
 that.salary = newSalary;  
 },  
 setName: function(newName) {  
 that.name = newName;  
 },  
 increaseSalary: function(percentage) {  
 that.salary = getSalary() + (getSalary()\*percentage)/100;  
 },  
 incrementAge: function() {  
 that.age = getAge() + 1;  
 },  
 // for test only  
 info: function () {  
 return "---" + that.name + "---" + that.age + '---' + that.salary;  
 }  
 }  
  
})();  
  
Employee.setAge(10);  
Employee.setName('Wick');  
Employee.setSalary(1000);  
Employee.increaseSalary(50);  
Employee.incrementAge();  
console.log(Employee.info()); // ---Wick---11---1500

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

var Employee = (function() {  
 let name, age, salary;  
 let that = this;  
   
 let empl = {};  
  
 empl.setAge = function(newAge) {  
 that.age = newAge;  
 }  
 empl.setName = function(newName) {  
 that.name = newName;  
 }  
 empl.setSalary = function(newSalary) {  
 that.salary = newSalary;  
 }  
  
 let getAge = function() {  
 return that.age;  
 }  
  
 let getName = function() {  
 return that.name;  
 }  
  
 let getSalary = function() {  
 return that.salary;  
 }  
  
 empl.increaseSalary = function(percentage) {  
 that.salary = getSalary() + (getSalary()\*percentage)/100;  
 }  
  
 empl.incrementAge = function() {  
 that.age = getAge() + 1;  
 }  
   
 empl.info = function () {  
 return "---" + that.name + "---" + that.age + '---' + that.salary;  
 }  
  
 return empl;  
  
})();  
  
  
Employee.setAge(10);  
Employee.setName('Wick');  
Employee.setSalary(1000);  
Employee.increaseSalary(50);  
Employee.incrementAge();  
console.log(Employee.info()); // ---Wick---11---1500

1. **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( ).**

The code below extend Employee Module

var Employee = (function() {  
 let name, age, salary;  
 // […]  
 return empl;  
  
})();

// Extend Employee Module  
Employee.address  
Employee.setAddress = function(newAddress) {  
 Employee.address = newAddress;  
}  
Employee.getAddress = function() {  
 return Employee.address;  
}  
  
// Test  
Employee.setAddress("Test");  
console.log(Employee.getAddress()); // Test

1. **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));

**Alert display:** Error: Hattori

1. **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));

**Alert display:** Success: Hattori

1. **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';

});

**First, it will log “success” and then log “error”:**

success

error