# 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);

a) 1 8 8 9 10 1

1. Define Global Scope and Local Scope in Javascript.

a) Global scope is when the variables are declared outside any function and accessible from any part of the Javascript code. Local scope is when a variable is declared inside a function and cannot be accessed from outside of it.

1. Consider the following structure of Javascript code:

// Scope A

function XFunc () {

// Scope B

function YFunc () {

// Scope C

};

};

1. Do statements in Scope A have access to variables defined in Scope B and C?

No.

1. Do statements in Scope B have access to variables defined in Scope A?

Yes.

1. Do statements in Scope B have access to variables defined in Scope C?

No.

1. Do statements in Scope C have access to variables defined in Scope A?

Yes.

1. 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());

81 25

1. What will the alert print out? (Answer without running the code. Remember ‘hoisting’.)?

var foo = 1;

function bar() {

if (!foo) {

var foo = 10;

}

alert(foo);

}

bar();

1

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 counter = 0;

var add = function () {

return counter += 1;

}

Var reset = function () {

counter = 0;

}

var count = (function () {

return {

counter: counter

add: add,

reset: 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?

The free variables are counter, add and reset. It’s the variable that’s neither declared inside the count scope, nor passed as a parameter.

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 counter = 0;

var make\_adder = (function (inc) {

return function(){

counter += inc;

}

})();

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?

Write the code inside a self-invoking function.

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( )

let employee = (function(){

let name;

let age;

let salary;

let getName = function () {

return name;

}

let getAge = function () {

return age;

}

let getSalary = function(){

return salary;

}

let setName = function(newName){

name = newname;

}

let setAge = function(newAge){

age = newAge;

}

let setSalary = function(newSalary){

salary = salary;

}

let increaseSalary = function(percentage){

setSalary(getSalary() \* percentage);

}

let incrementAge = function(){

setAge(getAge() + 1);

}

return {

setName: setName,

setAge: setAge,

setSalary: setSalary,

increaseSalary: increaseSalary,

incrementAge: incrementAge

}

})();

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

let employee = (function(){

let name;

let age;

let salary;

let getName = function () {

return name;

}

let getAge = function () {

return age;

}

let getSalary = function(){

return salary;

}

return {

setName: function(newName){

name = newname;

},

setAge: function(newAge){

age = newAge;

},

setSalary: function(newSalary){

salary = salary;

},

increaseSalary: function(percentage){

setSalary(getSalary() \* percentage);

},

incrementAge: function(){

setAge(getAge() + 1);

}

}

})();

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

let employee = (function(){

let emp = {}

let name;

let age;

let salary;

let getName = function () {

return name;

}

let getAge = function () {

return age;

}

let getSalary = function(){

return salary;

}

emp.setName: function(newName){

name = newname;

}

emp.setAge: function(newAge){

age = newAge;

}

emp.setSalary: function(newSalary){

salary = salary;

}

emp.increaseSalary: function(percentage){

setSalary(getSalary() \* percentage);

}

emp.incrementAge: function(){

setAge(getAge() + 1);

}

return emp;

})();

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( ).

employee.address = “”;

employee.getAddress = function(){

return this.address;

}

employee.setAddress = function(newAddress){

this.address = newAddress;

};

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));

Popping out this message: “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));

Popping out: “Success: Hattori” then popping out: “Error: Yoshi”

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';

});