Javascript Module Exercises

1. Determine what this Javascript code will print out (without running it):

The result:

undefined

8

8

9

10

1

2. Define Global Scope and Local Scope in Javascript.

a = 1;

var b = 2;

var c = function(){

var d = 3;

//…

};

a and b are global variables, c is global function

d is local variable

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());

The result is:

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’.)?

The result is:

10

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.

var add = (function () {

var counter = 0;

return {

add:function (){

counter++;

},

reset:function(){

counter = 0;

}

}

})();

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?

Answer: The variable ‘counter’ is free variable

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

Answer:

var add = (function (inc) {

var counter = 0;

return {

add:function (){

counter++;

},

make\_adder:function (inc){

counter += inc;

},

reset:function(){

counter = 0;

}

}

})();

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?

Answer: The simple way is creating a module pattern and adding all to that 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( )

The answer:

var Employee = (function(){

let name;

let age;

let salary;

let getAge = function(){

return age;

};

let getSalary = function(){

return salary;

};

let getName = function(){

return name;

};

let setAge=function(newAge){

age = newAge;

};

let setSalary=function(newSalary){

salary = newSalary;

};

let setName=function(newName){

name = newName;

};

let increaseSalary=function(percentage){

salary = getSalary() + getSalary()\*percentage;

return salary;

};

let increamentAge=function(){

age = getAge() + 1;

return age;

};

return {

setAge: setAge,

setSalary: setSalary,

setName:setName,

increaseSalary: increaseSalary,

increamentAge: increamentAge

};

})();

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

The answer:

var myModule = (function(){

let name;

let age;

let salary;

let getAge = function(){

return age;

};

let getSalary = function(){

return salary;

};

let getName = function(){

return name;

};

return {

setAge:function(newAge){

age = newAge;

},

setSalary:function(newSalary){

salary = newSalary;

},

setName:function(newName){

name = newName;

},

increaseSalary:function(percentage){

salary = getSalary() + getSalary()\*percentage;

return salary;

},

increamentAge:function(){

age = getAge() + 1;

return age;

}

};

})();

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

The answer:

var myModule = (function(){

let Employee = {};

let name;

let age;

let salary;

let getAge = function(){

return age;

};

let getSalary = function(){

return salary;

};

let getName = function(){

return name;

};

Employee.setAge = function(newAge){

age = newAge;

};

Employee.setSalary = function(newSalary){

salary = newSalary;

};

Employee.setName = function(newName){

name = newName;

};

Employee.increaseSalary = function(percentage){

salary = getSalary() + getSalary()\*percentage;

return salary;

};

Employee.increamentAge = function(){

age = getAge() + 1;

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( ).

Employee.extension = function(){

let address;

let setAddress = function(newAddress){

address = newAddress;

};

let getAddress = function(){

return address;

}

return{

setAddress: setAddress;

getAddress: 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));

The result is: alert ‘Error: Hettori’

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));

The result is: alert ‘Success: Hattori’ and each 500ms it will alert ‘Error: Yoshi’

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'; });

The result is:

It shows in the console:

success

error