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:

10,8,8,9,10, undefined

………………………………………………………………………………………………………..

1. Define Global Scope and Local Scope in Javascript.

Answer:

*Variables defined* ***outside*** *a function are […] called* ***global variables****. Variables defined* ***within*** *a function are* ***local variables****.*

*…………………………………………………………………………….*

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?

Answer:

No

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

Answer:

yes

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

Answer:

no

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

Answer:

yes

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

Answer:

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());

answer:

81, 25;

…………………………………………………………………………………………………………………………

1. 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

//////////////////////////////////////////////////////////////////////////////////////

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.

Answer:

var count = (function () {

var counter = 0;

var add=function(){

counter+=1;

}

var reset=function(){

counter=0;

}

Var value=function(){

return counter;

}

Return{

add: add,

reset: reset,

value: value

}

}();

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

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?

Answer:

The free variable is counter.

Free variables are neither locally declared nor passed as 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

answer:

var make\_adder=(function(inc){

var counter=0;

return function() {

counter+=inc;

Return counter;

}

})();

………………………………………………………………………………………..

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?

Answer:

You can just wrapper up them inside a module pattern and the global scope will not be more aware of them.

////////////////////////////////////////////////////////////////////////////////////

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( )

Answer:

Var employee=(function(){

Let name ;

Let age;

Let salary;

Let getName=function(){

Return name;

}

Let getSalary=function(){

Return salary;

}

Let getAge=function(){

Return age;

}

Let setName=function(myName){

This.name=myName;

}

Let setAge=function(myAge){

This.age=myAge;

}

Let setSalary=function(mySalary){

This.salary=mySalary;

}

Let increaseSalary =function(percentage){

Let incSalary= getSalary() +( getSalary()\*percentage);

setSalary(salary +( salary\* percentage));

}

Let incrementAge =function( ){

Var ag=getAge();

Ag++;

setAge(ag);

}

Return {

setAge: getAge,

setSalary: getSalary,

setName : setName ,

increaseSalary : increaseSalary,

incrementAge : incrementAge

}

})();

,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

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

Var employee=(function(){

Let name ;

Let age;

Let salary;

Let getName=function(){

Return name;

}

Let getSalary=function(){

Return salary;

}

Let getAge=function(){

Return age;

}

Return{

Let setName=function(myName){

This.name=myName;

},

Let setAge=function(myAge){

This.age=myAge;

},

Let setSalary=function(mySalary){

This.salary=mySalary;

}

,

Let increaseSalary =function(percentage){

Let incSalary= getSalary() +( getSalary()\*percentage);

setSalary(salary +( salary\* percentage));

},

Let incrementAge =function( ){

Var ag=getAge();

Ag++;

setAge(ag);

}

})();

.///////////////////////////………….////////////////////////////////////

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

Answer:

Var employee=(function(){

Var employee={ };

Let name ;

Let age;

Let salary;

Let getName=function(){

Return name;

}

Let getSalary=function(){

Return salary;

}

Let getAge=function(){

Return age;

}

employee .setName=function(myName){

This.name=myName;

};

employee .setAge=function(myAge){

This.age=myAge;

};

employee .setSalary=function(mySalary){

This.salary=mySalary;

};

employee .increaseSalary =function(percentage){

Let incSalary= getSalary() +( getSalary()\*percentage);

setSalary(salary +( salary\* percentage));

};

employee .incrementAge =function( ){

Var ag=getAge();

Ag++;

setAge(ag);

};

Return employee;

})();

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( ).

Answer:

Let address;

getAddress =function(){

return address;

}

setAddress=function(addr){

this.address=addr;

}

//////////////////////////////////////////////////////////////////////////

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));

answer:

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));

answer:

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';

});

Answer:

Success message then error message

//////////////////////////////////////////// - end -//////////////////////////////////////