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); // output undefined document.write(a); // output 8

var f = function(a, b, c) {

b = a;

document.write(b); // output 8

b = c; var x = 5;

}

f(a,b,c);

document.write(b); // output 9

var x = 10;

}

c(8,9,10);

document.write(b);//output 10 document.write(x);//output undefined

}

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

Ans

Global scope is Any **variable** declared outside of a **function** , and is therefore accessible from anywhere in your code.

*Local Scope* **function** has its own **scope**, and any **variable** declared within that **function** is only accessible from that **function** and any nested functions

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

True

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

False

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

True

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

True

4. What will be printed by the following (answer without running it)?

var x = 9;

function myFunction() { return x \* x;

}

document.write(myFunction());// output 81

x = 5;

document.write(myFunction());// output 25

5.

var foo = 1;

function bar() {

if (!foo) { var foo = 10;

}

alert(foo); //outout alert 10

} bar();

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

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.

Ans

var count = (function() {

var counter = 0;

return {add:function() {

counter +=1;

return counter;

},reset:function() {

counter=0;

return counter;

}}

})();

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?

Ans

The free variable is a variable decleared not parameter nor the local variable of the function

Which declared out side of the existingfunction

At the above add() the free variable is counter;

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

Ans

var count = (function() {

var counter = 0;

return {add:function() {

counter +=1;

return counter;

},reset:function() {

counter=0;

return counter;

},

make\_adder: function(inc) {

return function(){return 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?

Ans

Use either of Module Patterns or Object Literals.

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( )

Ans

var employee =

(function(){

//fields

let name;

let age;

let salary;

//getter & setter methods

let setAge = function(newAge){this.age = newAge};

let setSalary = function(newSalary){this.salary = newSalary};

let setName = function(newName){this.name = newName};

let getAge = function(){return this.age;};

let getSalary = function(){return this.salary;};

let getName = function(){return this.name;};

//extra methods

let increaseSalary = function(percentage){

setSalary(getSalary()+(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*.

Ans

var employee =

(function(){

//fields

let name;

let age;

let salary;

//getter & setter methods

let getAge = function(){return age;};

let getSalary = function(){return salary;};

let getName = function(){return name;};

return {

setName : function(newName){name = newName},

setAge : function(newAge){age = newAge},

setSalary: function(newSalary){salary = newSalary},

increaseSalary : function(percentage){salary = getSalary() + (getSalary()\*percentage/100);},

incrementAge: function(){age =getAge()+1;}

};

})();

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

Ans

var employee =

(function(){

//fields

let name;

let age;

let salary;

//getter & setter methods

let getAge = function(){return age;};

let getSalary = function(){return salary;};

let getName = function(){return name;};

let empO = {};

empO.setName = function(newName){name = newName};

empO.setAge = function(newAge){age = newAge};

empO.setSalary = function(newSalary){salary = newSalary};

empO.increaseSalary = function(percentage){salary = getSalary() + (getSalary()\*percentage/100);};

empO.incrementAge = function(){age =getAge()+1;};

return empO;

})();

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*( ).

Ans

employee.address = "";

employee.setAddress = function(newAddress){this.address = newAddress;};

employee.getAddress = function(){return this.address;};

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));

Ans

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));

Ans

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';

});

Ans

success error