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);**//undefined,** x local var x;

document.write(a);**//8** a: argument c(**8**,9,10)

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

b = a;

document.write(b);**//8 b = a = 8**

b = c;

var x = 5;

}

f(a,b,c);

document.write(b);**//9** b: argument c(8,**9**,10)

var x = 10;

}

c(8,9,10);

document.write(b);**//10** b in var b = 10;

document.write(x);**//1** x in x = 1;

}

|  |
| --- |
| **Result:**  undefined  8  8  9  10  1 |

2. **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.

**3. 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**

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

var x = 9;

function myFunction() {

return x \* x; }

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

x = 5;

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

|  |
| --- |
| **Answer**  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’.)?

|  |
| --- |
| **Answer**  alert(10); |

//undefined, empty, null, 0, false, empty array is false

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 count = (function () {  var counter = 0;  return {  add: function () {  return counter += 1;  },  reset: function () {  counter = 0;  return counter;  }  }  })();  count.add();  count.add();  count.add();  count.reset();  count.add();  count.add(); |
| **Or**  var count = (function () {  var counter = 0;  var countObj = {};  var add = function () {  return counter += 1;  };  var reset = function () {  counter = 0;  return counter;  };  countObj.add = add;  countObj.reset = reset;  return countObj;  })();  count.add();  count.add();  count.add();  count.reset();  count.add();  count.add(); |

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?**

|  |
| --- |
| * In the definition of add( ) shown in question 6, identify the "free" variable   var counter = 0;  counter is the free variable   * In the context of a function closure, what is a "free" variable?   A free variable in this context is a **private** variable of the add function |

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

|  |
| --- |
| var make\_adder = (function (inc) {  var counter = 0;  return function () {  return counter += inc;  } }); |
| add5 = make\_adder(5);  add5( ); add5( ); add5( ); // final counter value is 15  add7 = make\_adder(7);  add7( ); add7( ); add7( ); // final counter value is 21 |

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?**

|  |
| --- |
| (function(){  //all code move to here  }  )(); |
| var Module = (function () {  // code  })() |

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)

Public Method: incrementAge( ) // uses private getAge( )

|  |
| --- |
| **var *Employee*** = (**function**(){   **let** name;  **let** age;  **let** salary;   **let** *setAge* = **function**(newAge){  age=newAge;  };  **let** *setSalary* = **function**(newSalary){  salary = newSalary;  };  **let** *setName* = **function**(newName){  name = newName;  };  **let** *getAge* = **function**(){  **return** age;  };  **let** *getSalary* = **function**(){  **return** salary;  };  **let** *getName* = **function**(){  **return** name;  };  **let** *increaseSalary* = **function**(percentage){  salary+=salary\* percentage;  };  **let** *incrementAge* = **function**(newAge){  *// uses private getAge( )  setAge*(*getAge*()+1);  };   **return**{  setAge: *setAge*,  setSalary:*setSalary*,  setName: *setName*,  increaseSalary: *increaseSalary*,  incrementAge: *incrementAge* }; })(); |
| var emp1 = Employee; |

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

|  |
| --- |
| *//Create Employee Object using the Anonymous Object Literal Return Pattern* **var *Employee*** = (**function** () {   **let** name;  **let** age;  **let** salary;  **let** *getAge* = **function** () {  **return** age;  };  **let** *getSalary* = **function** () {  **return** salary;  };  **let** *getName* = **function** () {  **return** name;  };  *//return object with public methods* **return** {  setAge: **function** (newAge) {  age = newAge;  }  , setSalary: **function** (newSalary) {  salary = newSalary;  }  , setName: **function** (newName) {  name = newName;  }  , increaseSalary: **function** (percentage) {  salary += salary \* percentage;  }  , incrementAge: **function** (newAge) {  *// uses private getAge( )* setAge(*getAge*() + 1);  }  } })(); |
| **var *emp1*** = ***Employee***; ***console***.log(***emp1***); |

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

|  |
| --- |
| *//Create Employee Object using the Locally Scoped Object Literal Pattern* **var *Employee*** = (**function** () {   **let** name;  **let** age;  **let** salary;  **let** myObject={};  **let** *getAge* = **function** () {  **return** age;  };  **let** *getSalary* = **function** () {  **return** salary;  };  **let** *getName* = **function** () {  **return** name;  };  myObject.setAge= **function** (newAge) {  age = newAge;  };  myObject.setSalary= **function** (newSalary) {  salary = newSalary;  };  myObject.setName= **function** (newName) {  name = newName;  };  myObject.increaseSalary= **function** (percentage) {  salary += salary \* percentage;  };  myObject.incrementAge= **function** (newAge) {  *// uses private getAge( )* setAge(*getAge*() + 1);  };  **return** myObject;  })(); |
| **var *emp1*** = ***Employee***; ***console***.log(***emp1***); |

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( ).**

|  |
| --- |
| **var *Employee*** = (**function** () {   **let** name;  **let** age;  **let** salary;   **let** *setAge* = **function** (newAge) {  age = newAge;  };  **let** *setSalary* = **function** (newSalary) {  salary = newSalary;  };  **let** *setName* = **function** (newName) {  name = newName;  };  **let** *getAge* = **function** () {  **return** age;  };  **let** *getSalary* = **function** () {  **return** salary;  };  **let** *getName* = **function** () {  **return** name;  };  **let** *increaseSalary* = **function** (percentage) {  salary += salary \* percentage;  };  **let** *incrementAge* = **function** (newAge) {  *// uses private getAge( )  setAge*(*getAge*() + 1);  };   **return** {  setAge: *setAge*,  setSalary: *setSalary*,  setName: *setName*,  increaseSalary: *increaseSalary*,  incrementAge: *incrementAge* }; })(); |
| *//Add extension public field* ***Employee***.**address** = **null**; *//Add extension public method* ***Employee***.setAddress = **function** (newAddress) {  **this**.**address** = newAddress; }; *//Add another extension public method* ***Employee***.getAddress = **function** () {  **return this**.**address**; }; |
| *//Testing only* **var *emp1*** = ***Employee***; ***emp1***.setAddress(**"1000 N 4th St"**); ***console***.log(***emp1***.**address**); ***console***.log(***emp1***.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 output shows an alert message “Error: Hattori” |

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 output shows an alert message “Success: Hattori” |
| A promise is an object which can be returned synchronously from an asynchronous function. It will be in one of 3 possible states:   * **Fulfilled:** onFulfilled() will be called (e.g., resolve() was called) * **Rejected:** onRejected() will be called (e.g., reject() was called) * **Pending:** not yet fulfilled or rejected   A promise is **settled** if it’s not pending (it has been resolved or rejected). Sometimes people use resolved and settled to mean the same thing: not pending.  Once settled, a promise cannot be resettled. Calling resolve() or reject()again will have no effect. The immutability of a settled promise is an important feature.  Reference: <https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-promise-27fc71e77261> |

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'**;  
  
 });

|  |
| --- |
| **'success'** |
| **'error'** |
| **let *promise*** = *job*(**true**);  ***promise***.then(**function**(data) {   ***console***.log(data);//write data (**'success')** to console window  **return** *job*(**false**);//return another promise with state = false  //so the .cacth will be invoked  .catch(**function**(error) {  ***console***.log(error);//write error (**'error')** to console window   **return 'Error caught'**;   }); |
|  |