Lab 1

Due Feb 6, 2023 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** zip
Available after Jan 30, 2023 at 12am

CS-546 Lab 1

An Intro to Node

For this lab, you will be creating and running several functions to practice JavaScript syntax.

For this lab, you will make two files: `lab1.mjs` and `lab1.test.mjs` and submit them in a zip file that's named `LastName_FirstName.zip`. For example: Hill_Patrick.zip

You **should not** have any folders inside the zip file.

You **must** submit your files with the format specified, named as specified.

You do not have to take input validation and error handling into account for lab 1. You can assume that we will be testing with only valid data. We will begin input validation and error handling in lecture/lab 2.

You can download the lab code starting template here: **Lab1_stub.zip (https://sit.instructure.com/courses/64637/files/11082296?wrap=1)** **(https://sit.instructure.com/courses/64637/files/11082296/download?download_frd=1)**

lab1.mjs

In this file, you will update the content of the functions and update the `firstName`, `lastName`, and `studentId` with the appropriate information. The function specifications are listed in the section below.

```
export const questionOne = (arr) => {
    // Implement question 1 here
   return //return result
}

export const questionTwo = (numArray) => {
    // Implement question 2 here
    return //return result
}

export const questionThree = (obj1, obj2) => {
    // Implement question 3 here
    return //return result
}

export const questionFour = (string) => {
    // Implement question 4 here
    return //return result
```

```
}

export const studentInfo= {
    firstName: "YOUR FIRST NAME",
    lastName: "YOUR LAST NAME",
    studentId: "YOUR STUDENT ID",

};
```

lab1.test.mjs

In this file, you will import your functions from lab1.mjs and call EACH function 5 times, passing in different input each time to ensure your function is working properly.

```
import * as lab1 from "./lab1.mjs";

// make 5 calls to questionOne
console.log(lab1.questionOne([5, 3, 10])); // should return and then output {'1152': false}

// make 5 calls to questionTwo
console.log(lab1.questionTwo([1, 2, 4, 3]); //Returns and then outputs [false, 2, 3]

// make 5 calls to questionThree
console.log(lab1.questionThree({a:1,b:2,c:3}, {c:10, a:20, b:30}); //Returns and then outputs {a: true, b:true, c:tru

// make 5 calls to questionFour
console.log(lab1.questionFour(`Patrick,Hill,cs546
Jared,Bass,cs115
Shudong,Hao,cs570`)); //Returns and then outputs [["Patrick", "Hill", "cs546"],["Jared", "Bass", "cs115"], ["Shudong'
```

Functions to implement

questionOne(arr);

For your first function, you will first calculate the cube for each element in the supplied array and then sum the cubes. Your function will return an `object` that has the value of the sum of cubes for all elements as the key/property name and a boolean stating if that value is prime or not as the value. That means that in `lab1.test.mjs`, running `lab1.questionOne([5, 3, 10])` would return `{'1152': false}`.

To test this function, you will log the result of 5 calls in your lab1.test.mjs to `lab1.questionOne([x, y, z])` with different inputs, like so:

```
console.log(lab1.questionOne([5, 3, 10]));  // Returns and then outputs {'1152': false}
console.log(lab1.questionOne([2, 1, 2])); // Returns and then outputs {'17': true}
console.log(lab1.questionOne([512, 1007, 17389])); //Returns and then outputs {'5259194599940': false}
console.log(lab1.questionOne([0, 14159, 785])); //Returns and then outputs {'2839041558304', false}
console.log(lab1.questionOne([11, 4])); //Returns and then outputs {'1395': false}
```

questionTwo(numArray);

This function will take in an array of numbers and will 1.determine if the array is sorted in ascending order and 2. Report on the indexes that are out of order. You will return an array that has a boolean to note if the given numArray is sorted as one element. if the array is not sorted, you will also have one element that gives that starting index and one element that gives the ending index where they are not in order (You only have to return the indexes of the first instance where the values are not in order)

For example in `lab1.test.mjs`, running `lab1.questionTwo([2,10,4])` would return `[false, 1, 2]`.

To test this function, you will log the result of 5 calls to `lab1.questionTwo([x, y, z])` with different inputs, like so:

```
console.log(lab1.questionTwo([1, 2, 3, 4, 5, 6, 7]));  // Returns and then outputs [true]
console.log(lab1.questionTwo([1, 2, 4, 3])); // Returns and then outputs [false, 2, 3]
console.log(lab1.questionTwo([10, 7, 6, 11])); //Returns and then outputs [false, 0, 1]
console.log(lab1.questionTwo([28,45,1002, 10000])); //Returns and then outputs [true]
```

questionThree(obj1, obj2)

This function will return an `object`. You will return an `object` with each key from both objects as the key and for the value, a boolean stating if that key appears in both obj1 and obj2. You do not have to consider the values for the keys. Also remember that the order of the keys of an object do not matter.

```
console.log(lab1.questionThree({a:1,b:2,c:3}, {c:10, a:20, b:30}));  // Returns and then outputs {a:true, b:true, c:true}

console.log(lab1.questionThree({a:1,b:2,c:3, d:4}, {f:10, b:20, e:30, d: 40, c:50, a:60})); // Returns and then outputs {a:true, b:true, c:true, d:true, e:false, f:false}

console.log(lab1.questionThree({foo:"I'm foo", bar: "I'm bar", fizz: "I'm fizz" , buzz: "I'm buzz" }, {fizz: "I'm not buzz", foo:"I'm not bar", buzz: "I'm not fizz", bar: "I'm not foo", c:50, a:60})); // Returns and then outputs {foo:true, bar: true, fizz: true, buzz: true, c:false, a:false}

console.log(lab1.questionThree({a:10, b:20, c:30, d: 40, e:50, f:60}, {a:1,b:2,c:3} )); //Returns and then outputs {a: true, b: true, c:true, d: false, e: false, f: false}
```

questionFour(string)

For the fourth function, you will convert a comma-separated value (CSV) string to an array of arrays where a new line in the string indicates a new row in the array.

For example, calling (notice I am using a template literal string for the new lines):

```
console.log(lab1.questionFour(`Patrick,Hill,cs546
Jared,Bass,cs115
Shudong,Hao,cs570`));

//should return and then log [["Patrick", "Hill", "cs546"],["Jared", "Bass", "cs115"], ["Shudong", "Hao", "cs570"]]
```

Requirements

1. You will have to write each function
2. You must submit all files, zipped up, not contained in any folders
3. You must not use any npm dependenices in this lab

Lab 1 Rubric

Criteria	Ratings		Pts
questionOne(arr); Test cases used for grading will be different from assignment examples.	**23.5 to >0.0 pts** **5.875 Points Per Test Case** The function is implemented correctly, and test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	23.5 pts
questionTwo(numArray); Test cases used for grading will be different from assignment examples.	**23.5 to >0.0 pts** **5.875 Points Per Test Case** The function is implemented correctly, and test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	23.5 pts
questionThree(obj1,obj2); Test cases used for grading will be different from assignment examples.	**23.5 to >0.0 pts** **5.875 Points Per Test Case** The function is implemented correctly, and test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	23.5 pts
questionFour(string); Test cases used for grading will be different from assignment examples.	**23.5 to >0.0 pts** **5.875 Points Per Test Case** The function is implemented correctly, and test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	23.5 pts
Student Info	**6 to >0.0 pts** **2 points per field in student info**	**0 pts** **No Marks**	6 pts
			Total Points: 100