Lab 1

- Due Feb 6 by 11:59pm
- Points 100
- Submitting a file upload
- File Types zip
- Available until Feb 13 at 11:59pm

This assignment was locked Feb 13 at 11:59pm.

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 and it should only contain these two files.

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

You can download the lab code starting template here that has the correct file and function names: **Lab1_stub.zip (https://sit.instructure.com/courses/71954/files/12974138?wrap=1)** ↓ **(https://sit.instructure.com/courses/71954/files/12974138/download?download_frd=1)**

Note: For this lab, you do not have to do any error handling or input validation. For lab 1, you can assume only valid inputs will be passed into your functions. From lab 2 forward, you will have to do error handling and input validation, but for this first lab, you can just assume that valid inputs will be passed to your functions.

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 = (index) => {
    // Implement question 1 here
   return //return result
}


export const questionTwo = (arr) => {
    // Implement question 2 here


    return //return result
}


export const questionThree = (str) => {
    // Implement question 3 here
    return //return result
}
```

```
export const questionFour = (arr) => {
    // 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 passing in different inputs
console.log(lab1.questionOne(7)); // calls the function and then outputs the return value: 13

// make 5 calls to questionTwo passing in different inputs
console.log(lab1.questionTwo([5, 3, 10]));
// calls the function and then outputs the return value: {5:true, 3: true, 10: false}


// make 5 calls to questionThree
console.log(lab1.questionThree("The quick brown fox jumps over the lazy dog.")); // returns and then outp
uts: {consonants: 24, vowels: 11, numbers: 0, spaces: 8, punctuation: 1, specialCharacters: 0}

// make 5 calls to questionFour
console.log(lab1.questionFour([1, 1, 1, 1, 1, 1]));
//returns and then outputs: [1]
```

Functions to implement

Important note: Your functions should have a return value using the return keyword. Do NOT just console.log the result of the function. If you do, you will receive full points off for that function. A function that does not return a value using the return keyword, actually returns undefined if there is no return statement, so if you console.log instead of using the return statement, your function is not returning the correct result, it's returning undefined, If you want to log the result, you log the call to the function in the test file as shown above.

questionOne(index);

This function should calculate the **Fibonacci (https://en.wikipedia.org/wiki/Fibonacci_number)** that corresponds to the `index` given.

The Fibonacci value of a number is the sum of the previous two Fibonacci values; the Fibonacci of any number less than 1 is 0; the Fibonacci Value of 1 is 1; the Fibonacci value of all other numbers is

the sum of the previous two Fibonacci numbers.

Index	Value	Description
0	0	Fibonacci of anything less than 1 is 0
1	1	Fibonacci of 1 is 1
2	1	Fibonacci of 2 is Fibonacci(1) + Fibonacci(0)
3	2	Fibonacci of 3 is Fibonacci(2) + Fibonacci(1)
4	3	Fibonacci of 4 is Fibonacci(3) + Fibonacci(2)
5	5	Fibonacci of 5 is Fibonacci(4) + Fibonacci(3)
6	8	Fibonacci of 6 is Fibonacci(5) + Fibonacci(4)
7	13	Fibonacci of 7 is Fibonacci(6) + Fibonacci(5)
8	21	Fibonacci of 8 is Fibonacci(7) + Fibonacci(6)
9	34	Fibonacci of 9 is Fibonacci(8) + Fibonacci(7)
10	55	Fibonacci of 10 is Fibonacci(9) + Fibonacci(8)
11	89	Fibonacci of 11 is Fibonacci(10) + Fibonacci(9)

And so on.

That means that in `lab1.test.js`, running `lab1.questionOne(3)` would return `2`.

To test this function, you will log the result of 5 calls to `lab1.questionOne(index)` in your `lab1.test.js` file with different inputs, like so:

```
console.log(lab1.questionOne(7)); // returns and then outputs: 13

console.log(lab1.questionOne(10)); // returns and then outputs: 55

console.log(lab1.questionOne(4)); // returns and then outputs: 3
```

questionTwo(arr);

For your second function, you will calculate if all numbers in the array are prime numbers or not. You will return an `object` with the number as the key and true/false as the value That means that in `lab1.test.js`, running `lab1.questionTwo([5, 3, 10])` would return `{5: true, 3: true, 10: false}`. If an empty array is passed in or if the function is called without any input parameters, just return an empty object. You do not have to worry about dealing with different data types passed in. You can assume only arrays and numbers as elements will be passed in to your function (we get to type checking and error handling in lecture 2)

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

```
console.log(lab1.questionTwo([5, 3, 10]));
// returns and then outputs: {5:true, 3: true, 10: false}

console.log(lab1.questionTwo([2]));
// returns and then outputs: {2: true}
```

```
console.log(lab1.questionTwo([5, 10, 9]));
// returns and then outputs: {5: true, 10: false, 9: false}

console.log(lab1.questionTwo([2, 7, 9, 1013]));
// returns and then outputs: {2: true, 7: true, 9: false, 1013: true}

console.log(lab1.questionTwo([]));
// returns and then outputs: {}

console.log(lab1.questionTwo());
// returns and then outputs: {}
```

questionThree(str)

For your third function, you will return an `object` that contains the number of **consonants, vowels, numbers, spaces, punctuation, and any special characters** in the value `str`. For the purposes of this exercise, we are not counting `y` as a vowel, it would count as a consonant. your output would look like this: `{consonants: 20,  vowels: 10,  numbers: 7, spaces: 3,  punctuation: 5, specialCharacters: 2}`

If an empty string is passed in, just return `0` for each key. You do not have to worry about dealing with different data types passed in. You can assume only strings will be passed in to your function (we get to type checking and error handling in lecture 2)

punctuation characters example : **. , ? ! ' " : ;** etc...

If you are in doubt if it's considered punctuation, you can always Google the character and punctuation or grammar. for example: " punctuation or grammar

Special Characters example: **# $ % & ^** etc..

To test this function, you will log the result of 5 calls to `lab1.questionThree(str)` in your `lab1.test.js` file with different inputs, like so:

```
console.log(lab1.questionThree("The quick brown fox jumps over the lazy dog."));
// returns and then outputs: {consonants: 24, vowels: 11, numbers: 0, spaces: 8, punctuation: 1, specialC
haracters: 0}

console.log(lab1.questionThree("How now brown cow!!!"));
// returns and then outputs: {consonants: 10, vowels: 4, numbers: 0, spaces: 3, punctuation: 3, specialCh
aracters: 0}

console.log(lab1.questionThree("One day, the kids from the neighborhood carried my mother's groceries all
the way home. You know why? It was out of respect."));
// returns and then outputs: {consonants: 61, vowels: 36, numbers: 0, spaces: 22, punctuation: 5, special
Characters: 0}

console.log(lab1.questionThree("CS 546 is going to be fun & I'm looking forward to working with you all t
his semester!!" ));
// returns and then outputs: {consonants: 40, vowels: 23, numbers: 3, spaces: 17, punctuation: 3, special
Characters: 1}

console.log(lab1.questionThree(""));
// returns and then outputs: {consonants: 0, vowels: 0, numbers:0, spaces: 0, punctuation: 0, specialChar
acters: 0}
```

questionFour(arr)

This function will return a new array that contains no duplicated values.

Note: '1' and 1 are not considered the same value.

If an empty array is passed in, just return an empty array `[]`. You do not have to worry about dealing with different data types passed in. You can assume only arrays are passed in. In the arrays, numbers and strings are the only element types that are contained (we get to type checking and error handling in lecture 2).

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

```
console.log(lab1.questionFour([1, 1, 1, 1, 1, 1]));
//returns and then outputs: [1]

console.log(lab1.questionFour([1, '1', 1, '1', 2]));
// returns and then outputs: [1, '1', 2]

console.log(lab1.questionFour([3, 'a', 'b', 3, '1']));
// returns and then outputs: [3, 'a', 'b', '1']

console.log(lab1.questionFour([]));
//returns and then outputs: []
```

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 Spring 2024

Criteria	Ratings		Pts
questionOne(index); 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(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
questionThree(str); 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(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
Student Info	**6 to >0.0 pts** **2 points per field in student info**	**0 pts** **No Marks**	6 pts
		Total Points: 100	