Lab 2

- Due Feb 13 by 11:59pm
- Points 100
- Submitting a file upload
- File Types zip
- Available until Feb 20 at 11:59pm

This assignment was locked Feb 20 at 11:59pm.

CS-546 Lab 2

The purpose of this lab is to familiarize yourself with Node.js modules and further your understanding of JavaScript syntax.

In addition, you must have error checking for the arguments of all your functions. If an argument fails error checking, you should throw a string describing which argument was wrong, and what went wrong. You can read more about error handling on the **MDN (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/throw)** .

You can download the starter template here: **lab2_stub.zip (https://sit.instructure.com/courses/71954/files/13006061?wrap=1)** ↓ **(https://sit.instructure.com/courses/71954/files/13006061/download?download_frd=1)** **PLEASE NOTE: THE STUB DOES NOT INCLUDE THE PACKAGE.JSON FILE. YOU WILL NEED TO CREATE IT! DO NOT FORGET TO ADD THE START COMMAND AND DO NOT FORGET TO ADD THE "type": "module" PROPERTY TO THE PACKAGE.JSON. THERE IS A HELPERS.JS FILE IN THE STUB, YOU CAN USE THIS IF YOU CREATE ANY HELPER FUNCTIONS THAT YOU CAN CALL FROM YOUR OTHER MODULES. YOU DO NOT HAVE TO USE THIS FILE FOR THE ASSIGNMENT IF YOU DO NOT WANT/NEED TO. DO NOT ADD ANY ADDITIONAL FILES OR FOLDERS TO THE ZIP FILE OTHER THAN WHAT'S INCLUDED IN THE STUB EXCEPT FOR THE PACKAGE.JSON!**

Initializing a Node.js Package

For all of the labs going forward, you will be creating Node.js packages, which have a `package.json`. To create a package, simply create a new folder and within that folder, run the command `npm init`. When it asks for a package name, name it **cs-546-lab-2**. You may leave the version as default and add a description if you wish. The entry file will be `app.js`.

All of the remaining fields are optional **except** author. For the author field, you **must** specify your first and last name, along with your CWID. **In addition,** You must also have a start script for your package, which will be invoked with `npm start`. You can set a start script within the `scripts` field of your `package.json`. Also add the `"type": "module"` property to the `package.json`

Here's an example of a valid package.json:

```
{
   "name": "cs-546-lab-2",
   "version": "1.0.0",
```

```
   "description": "My lab 2 module",
   "main": "app.js",
   "type": "module",
   "scripts": {
      "start": "node app.js"
   },
   "author": "John Smith 12345678",
   "license": "ISC"
}
```

General requirement for ALL functions that take in a string/strings as input: You need to trim input strings using the trim function! A string with just spaces is not usually valid input! You will be expected to trim all string inputs from lab 2 forward unless the specs for that function say spaces or untrimmed strings are ok.

arrayUtils.js

This file will export 3 functions, each of which will pertain to arrays.

Remember: You must check if all input parameters are supplied, that the input parameters are the correct data type and that the input parameters are the correct range(if there is a range constraint). If they are not, then you throw an error.

arrayPartition(arrayToPartition, partitionFunc)

For this function You will take in an array (arrayToPartition) and a partition function (partitionFunc) as input. The goal of this function is to partition the elements of the array into two subarrays based on the result of applying the partition function to each element. The partition function should return true for elements that belong to the first subarray and false for elements that belong to the second subarray. Ensure the input array is valid, meaning it is an actual array, and the partition function is a valid function. The function should then return an array containing two subarrays: one with elements that satisfy the partition function, and another with elements that do not.

```
const arrayToPartition1 = [1, 2, 3, 4, 5];
const partitionFunc1 = (num) => num % 2 === 0;
const partitionedArrays1 = arrayPartition(arrayToPartition1, partitionFunc1); // Expected Result: [[2,
4], [1, 3, 5]]

const arrayToPartition2 = [10, 15, 20, 25, 30];
const partitionFunc2 = (num) => num > 18;
const partitionedArrays2 = arrayPartition(arrayToPartition2, partitionFunc2); // Expected Result: [[20, 2
5, 30], [10, 15]]

const arrayToPartition3 = ["apple", "banana", "cherry", "date"];
const partitionFunc3 = (fruit) => fruit.length >= 6;
const partitionedArrays3 = arrayPartition(arrayToPartition3, partitionFunc3); // Expected Result: [['bana
na', 'cherry'], ['apple', 'date']]

const arrayToPartition4 = [0, -5, 10, -3, 7];
const partitionFunc4 = (num) => num >= 0;
```

```
const partitionedArrays4 = arrayPartition(arrayToPartition4, partitionFunc4); // Expected Result: [[0, 1
0, 7], [-5, -3]]
```

You must check:

- That the `arrayToPartition` parameter exists and is of proper type (an array).
- That the `arrayToPartition` parameter is not an empty array
- That the `arrayToPartition` parameter has at LEAST 2 elements
- That the `partitionFunc` parameter exists and is of proper type (a function).

If any of those conditions fail, you will throw an error.

arrayShift(arr, n)

For this function, you will take in two arguments, an array `arr`, and an integer `n`. You will rotate the elements in the array `n` positions. If n is a positive integer, you will rotate the elements to the right. If `n` is a negative integer, you will rotate the elements to the left. If `n` is 0, you will not rotate the array. Your function will return the newly shifted array.

You must check:

- That the `arr` parameter exists and is proper type (an array),
- That the `arr` parameter has at least two elements
- That the `n` parameter exists and is proper type (number)
- That the `n` is a whole number and not a decimal (positive, negative and 0 are all valid)

If any of those conditions fail, you will throw an error.

```
arrayShift([3,4,5,6,7], 3)   // returns [5,6,7,3,4]
arrayShift(["Hello",true, 5,"Patrick","Goodbye"], 4)   // returns [true, 5, "Patrick", "Goodbye", "Hell
o"]
arrayShift([1,2,3,4], -2)   // returns [3,4,1,2]
arrayShift([7,8,9,10], 0) // returns [7,8,9,10]
arrayShift([7,11,15], 3.5) // throws error
```

matrixOne(matrix)

This function takes one m x n integer matrix, as input and set its entire row and column to 1's if an element is 0. The resulting matrix is then returned.

You must check:

- That the `matrix` argument exists and is in proper type (array)
- That each element in the `matrix` argument is an array
- That each element in the `matrix` array is not an empty array
- That the `matrix` itself is not an empty array
- That each sub-array element inside the `matrix` array is a number
- That the sub-arrays in the matrix array has no missing elements (Eg: Ensure that all rows in the matrix have the same number of elements.)

If any of those conditions fail, you will throw an error.

```
matrixOne([[2,2,2],[2,0,2],[2,2,2]]) //returns [[2,1,2],[1,1,1],[2,1,2]]
matrixOne([[0,1,2,0],[3,5,4,2],[1,7,3,5]]) //returns [[1,1,1,1],[1,5,4,1],[1,7,3,1]]
matrixOne([[0,1,2,0],[3,5,4]])// throws error
matrixOne([])// throws error
```

stringUtils.js

This file will export 3 functions, each are useful functions when dealing with strings in JavaScript.

swapChars(string1, string2)

Given `string1` and `string2` return the concatenation of the two strings, separated by a space and swapping the first 4 characters of each.

You must check:

- That both the `string1` and `string2` input parameters exist
- That both the `string1` and `string2` input parameters are of the proper type(strings)
- The length of both the `string1` and `string2` input parameters are at least 4 characters each. (a string with just spaces is not valid)

If any of those conditions fail, the function will throw.

```
swapChars("Patrick", "Hill"); //Returns "Hillick Patr"
swapChars("hello", "world"); //Returns "worlo helld"
swapChars("Patrick", ""); //Throws error
swapChars(); // Throws Error
swapChars("John") // Throws error
swapChars ("h", "Hello") // Throws Error
swapChars ("h","e") // Throws Error
```

NOTE: In the above example of the output, you should NOT return it with quotes. The quotes are there to denote that you are returning a string. In your function, you just return the string. If you add quotes to your output, points will be deducted.

```
let returnValue = "myFunctionRocks"
return returnValue //This is the correct way to return it
return `"${returnValue}"` //This is NOT correct
return '"' + returnValue + '"' //This is NOT correct
```

longestCommonSubstring(str1, str2)

For this function, you take in two valid strings, str1 and str2, as input and returns a string with the longest common substring present in both strings.

You must check:

- That both `str1` and `str2` input parameters exist
- That both `str1` and `str2` and are of valid type (string)
- That both `str1` and `str2` each have at least 5 characters each.
- That both `str1` and `str2` are not just empty strings or strings with just spaces.

If any of those conditions fail, you will throw an error.

```
const str1 = "abcdxyz";
const str2 = "xyzabcd";
const commonSubstring = longestCommonSubstring(str1, str2); // Expected Result: "abcd"

const str1 = "programming";
const str2 = "programmer";
const commonSubstring = longestCommonSubstring(str1, str2); // Expected Result: "programm"

const str1 = "abcdef";
const str2 = "123456";
const commonSubstring = longestCommonSubstring(str1, str2); // Expected Result: "" // No common substring
in this case

const str1 = "abcdef";
const str2 = "acdfgh";
const commonSubstring = longestCommonSubstring(str1, str2); // Expected Result: "cd"
```

palindromeOrIsogram(arrStrings)

For this function, you will check if each string in the array is a **palindrome, isogram, both or neither.** You will return an object; The keys of the object should correspond to the elements in the input array, and the values should indicate whether the respective element is a Palindrome, Isogram, Both or Neither. The values for the keys MUST match 100% for you to get credit: "Palindrome", "Isogram", "Both", "Neither". You MUST match the case of the values.

A palindrome is a phrase that is spelled the same way, backwards and forwards (ignoring spacing and punctuation; only alphanumeric characters matter). For example, the following phrases are palindromes:

- Madam
- Was it a cat I saw?
- He did, eh?
- Go hang a salami, I'm a lasagna hog.
- Poor Dan is in a droop
- Taco cat? Taco cat.

An isogram is a word or phrase in which no letter occurs more than once. (ignoring spacing and punctuation; only alphanumeric characters matter). In other words, all the letters in an isogram are unique. Here are a few examples:

- "Lumberjack" - This is an isogram because each letter appears only once.
- "Honey" - This is an isogram as well, as there are no repeating letters.
- "Background" - This word is an isogram as there are no repeating letters.
- "Alphabet" - This is not an isogram due to the repeated letter 'a'.

You must check:

- That the `arrStrings` argument exists and is of proper type (an array)
- The array contains ONLY elements that are strings.
- That there are at least two string elements in the array.

- Each string element cannot be empty strings with just empty spaces.

If any of those conditions fail, the function will throw an error.

```
const checkStrings = (["Madam", "Lumberjack", "He did, eh?", "Background", "Taco cat? Taco cat.", "Invali
d String"]);
const results = palindromeOrIsogram(checkStrings);
console.log(results);
//returns and then logs:
{ "Madam": "Palindrome", "Lumberjack": "Isogram", "He did, eh?": "Palindrome", "Background": "Isogram",
"Taco cat? Taco cat.": "Palindrome", "Invalid String": "Neither" }

const strings1 = ["level", "Racecar", "Palindrome", "Isogram"];
const results1 = palindromeOrIsogram(strings1);
console.log(results1);
//returns and then outputs:
{ "level": "Palindrome", "Racecar": "Palindrome", "Palindrome": "Isogram", "Isogram": "Isogram" }

const strings2 = ["hello", "world", "Java", "Python"]; const
results2 = palindromeOrIsogram(strings2);
console.log(results2);
//returns and then outputs
{ "hello": "Neither", "world": "Isogram", "Java": "Neither", "Python": "Isogram" }

const strings3 = ["abba", "abcd", "Is it OK?", "No lemon, no melon", "a"];
const results3 = palindromeOrIsogram(strings3);
console.log(results3);
//returns and then outputs
{ "abba": "Palindrome", "abcd": "Isogram", "Is it OK?": "Neither", "No lemon, no melon": "Palindrome",
"a": "Both" }
```

objectUtils.js

This file will export 3 functions that are useful when dealing with objects in JavaScript.

objectStats(arrObjects)

This function will take in an array of objects will return an object that has statistical measures of the values in the objects within the array. The statistical measures include the mean, median, mode, range, minimum, maximum, count, and sum. Before performing calculations, the function first extracts the numerical values from all objects, sorts them from lowest to highest, and then computes the required statistics.

You will first extract the values of the object and sort them from lowest to highest numbers before performing your calculations.

Note: If there is no mode, you will return 0 for that key. If there is more than one mode, you will return an array for the mode that has all the modes as elements (sorted by lowest to highest number).

Reminder: The order of the keys of an object does not matter. For example: {a: 1, b: 2, c: 3} is the same as/equal to {c: 3, a: 1, b: 2}

You must check:

• That the input parameter `arrObjects` exists and is of proper type (an array)

• That each element in the array is an `object`

• That each object in the array is not empty and has at least 1 key/value pair

• Each object value for each key is a number (can be positive, negative, decimal, zero) and Decimal numbers should be rounded to a maximum of three decimal places.

If any of those conditions fail, you will throw an error.

Examples:

```
const arrayOfObjects1 = [ { a: 12, b: 8, c: 15, d: 12, e: 10, f: 15 }, { x: 5, y: 10, z: 15 }, { p: -2,
q: 0, r: 5, s: 3.5 }, ];
const statsResult1 = objectStats(arrayOfObjects1);
// Expected Result:{ mean: 8.346, median: 10, mode: 15, range: 17, minimum: -2, maximum: 15, count: 13, s
um: 108.5 }
const arrayOfObjects2 = [ { p: 10, q: 15, r: 20 }, { x: -5, y: 8, z: 10 }, { a: 5, b: 5, c: 5 }, ];
const statsResult2 = objectStats(arrayOfObjects2);
// Expected Result:{ mean: 8.111, median: 8, mode: 5, range: 25, minimum: -5, maximum: 20, count: 9, sum:
73 }
const arrayOfObjects3 = [ { alpha: 3.5, beta: 7.2, gamma: 4.8 }, { x: 0, y: 0, z: 0 }, { p: -2, q: -8, r:
-5 }, ];
const statsResult3 = objectStats(arrayOfObjects3);
// Expected Result: { mean: 0.056, median: 0, mode: 0, range: 15.2, minimum: -8, maximum: 7.2, count: 9,
sum: 0.5 }
```

nestedObjectsDiff(obj1, obj2)

This function compares two nested objects, obj1 and obj2, and returns an object representing the differences between them. It considers differences at every level, including nested objects and arrays.

You must check:

• That both input parameters `obj1` and `obj2` exists and is of proper type (both objects)

• That each object input parameter is not empty and has at least 1 key/value pair

If any of those conditions fail, you will throw an error.

```
const obj1 = { key1: "value1", key2: { nestedKey: "nestedValue", arrayKey: [1, 2, 3], }, };
const obj2 = { key1: "value1", key2: { nestedKey: "differentValue", arrayKey: [1, 2, 4], }, key3: "newKe
y", };
const differences = nestedObjectsDiff(obj1, obj2);
// Example Output:   { key2: { nestedKey: "differentValue", arrayKey: [1, 2, 4], }, key3: "newKey" }

const obj1 = { a: 1, b: { c: 2, d: [3, 4] }, e: "hello" };
const obj2 = { a: 1, b: { c: 2, d: [3, 5] }, f: "world" };
const differences1 = nestedObjectsDiff(obj1, obj2); // Expected Result: { b: { d: [3, 5] }, e: undefined,
f: "world" }

const obj3 = { x: { y: { z: 1 } } };
const obj4 = { x: { y: { z: 1 } } };
const differences2 = nestedObjectsDiff(obj3, obj4); // Expected Result: {} // Both objects are identical,
so no differences are found.
```

mergeAndSumValues(...args)

This function takes a **variable number of objects** (that's what ...args signifies: You can see **this** (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments) and **this**

(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters) for more information) as input and merges them into a new object. The values corresponding to the same key are summed. The result is a new object with keys from all input objects and values representing the sum. The key values can be either numbers OR strings, i.e. 42 or "42". If the key value is a string with a valid number, you will need to convert it to a regular number. If it is a string that is not a valid number i.e. "hello", you would throw an error.

You must check:

- That each argument passed into the function is the correct type: object
- That each object has **at least 1 key/value**.
- That each object's values for each key are either numbers are strings that represent numbers i.e. either 12 or "12" (if the value is a string number, you will need to convert it into a number first!)

If these conditions are not met, you will throw an error.

Examples:

```
const object1 = { a: 3, b: 7, c: "5" };
const object2 = { b: 2, c: "8", d: "4" };
const object3 = { a: 5, c: 3, e: 6 };
const resultMergedAndSummed = mergeAndSumValues(object1, object2, object3);
console.log(resultMergedAndSummed);
// Expected Result: { a: 8, b: 9, c: 16, d: 4, e: 6 }

const obj1 = { a: 1, b: 2, c: 3 };
const obj2 = { b: 3, c: 4, d: 5 };
const obj3 = { a: 2, c: 1, e: 6 };
const result1 = mergeAndSumValues(obj1, obj2, obj3); // Expected Result: { a: 3, b: 5, c: 8, d: 5, e: 6 }

const obj4 = { x: 10, y: 5, z: 3 };
const obj5 = { x: 5, y: 2, z: 7 };
const obj6 = { x: 3, y: 8, z: 1 }; const
result2 = mergeAndSumValues(obj4, obj5, obj6); // Expected Result: { x: 18, y: 15, z: 11 }

const obj7 = { one: 15, two: 20 };
const obj8 = { one: 5, two: 10 };
const obj9 = { two: 5, three: 8 };
const result3 = mergeAndSumValues(obj7, obj8, obj9); // Expected Result: { one: 20, two: 35, three: 8 }

const obj10 = { a: 1, b: "2", c: 3 };
const obj11 = { b: 3, c: 4, d: 5 };
const obj12 = { a: 2, c: "hello", e: 6 };
const result4 = mergeAndSumValues(obj10, obj11, obj12); // Throws an error
```

Testing

In your `app.js` file, you must import all the functions the modules you created above export and create one passing and one failing test case for each function in each module. So you will have a total of 18 function calls (there are 9 total functions)

For example: (these are just generic function call examples, you would use the functions that you created, specified above)

```
try {
   // Should Pass
   const meanOne = mean([2, 3, 4]);
```

```
   console.log('mean passed successfully');
} catch (e) {
   console.error('mean failed test case');
}
try {
   // Should Fail
   const meanTwo = mean(1234);
   console.error('mean did not error');
} catch (e) {
   console.log('mean failed successfully');
}
```

Requirements

Write each function in the specified file and export the function so that it may be used in other files.

Ensure to properly error check for different cases such as arguments existing and of the proper type as well as throw if anything is out of bounds such as invalid array index.

Import ALL exported module functions and write 2 test cases for each in `app.js`.

Submit all files (including `package.json`) in a zip with your name in the following format: `LastName_FirstName.zip`.

do NOT have the files in any folders, they should be in the root of the zip file

You are not allowed to use any npm dependencies for this lab.

> **Lab 2 Rubric Spring 2024**

Criteria	Ratings		Pts
arrayUtils.js - arrayPartition(arrayToPartition, partitionFunc) Test cases used for grading will be different from assignment examples.	**13.5 to >0.0 pts** **1.5 Points/Error Handling Test Case & 3.75 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	13.5 pts
arrayUtils.js -matrixOne(matrix) Test cases used for grading will be different from assignment examples.	**13.5 to >0.0 pts** **1.5 Points/Error Handling Test Case & 3.75 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	13.5 pts
arrayUtils.js - arrayShift(arr, n) Test cases used for grading will be different from assignment examples.	**6 to >0.0 pts** **1 Points/Error Handling Test Case & 1 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	6 pts
stringUtils.js - swapChars(string1, string2) Test cases used for grading will be different from assignment examples.	**13.5 to >0.0 pts** **1.5 Points/Error Handling Test Case & 3.75 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	13.5 pts
stringUtils.js - longestCommonSubstring(str1, str2) Test cases used for grading will be different from assignment examples.	**6 to >0.0 pts** **1 Points/Error Handling Test Case & 1 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	6 pts
stringUtils.js - palindromeOrIsogram(arrStrings)	**13.5 to >0.0 pts**	**0 pts**	13.5 pts

Criteria	Ratings		Pts
Test cases used for grading will be different from assignment examples.	**1.5 Points/Error Handling Test Case & 3.75 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**All Test Cases Failed** Incorrect implementation or none of the test cases pass.	
objUtils.js - objectStats(arrObjects) Test cases used for grading will be different from assignment examples.	**13.5 to >0.0 pts** **1.5 Points/Error Handling Test Case & 3.75 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	13.5 pts
objUtils.js - nestedObjectsDiff(obj1, obj2) Test cases used for grading will be different from assignment examples.	**6 to >0.0 pts** **1 Points/Error Handling Test Case & 1 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	6 pts
objUtils.js - mergeAndSumValues(...args) Test cases used for grading will be different from assignment examples.	**13.5 to >0.0 pts** **1.5 Points/Error Handling Test Case & 3.75 Points/Valid Input Test Case** 4 Error Handling Test Cases & 2 Valid Input Test Cases. The function is implemented correctly, and few or all test cases pass.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	13.5 pts
Assignment Submitted	**1 pts** **Assignment Submitted** At least one of the test cases passes in the assignment.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass in the assignment.	1 pts
			Total Points: 100