Lab 3

Start Assignment

- Due Tuesday by 11:59pm
- Points 100
- Submitting a file upload
- Available Feb 13 at 12am - Feb 27 at 11:59pm

CS-546 Lab 3

The purpose of this lab is to familiarize yourself with asynchronous programming in JavaScript, as well as using modules from the Node.js Package Manager (**npm (https://www.npmjs.com/)**).

For this lab, you **must** use the `async/await` keywords (not Promise .then syntax or callbacks). You will also be using **axios (https://github.com/axios/axios)** , which is a HTTP client for Node.js; you can install it with `npm i axios`. For this lab and all labs going forward, you can use any package you like from npm!

Hint: Some of these functions can be done very easily using built-in array functions, find, filter, map, forEach etc..!

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 will be creating three `.js` files: `authors.js`, `books.js` and `app.js`.

Note: Remember that the order of the keys in the objects does not matter so `{firstName: "Patrick", lastName: "Hill"}` is the same as: `{lastName: "Hill", firstName: "Patrick"}`

You can download the starter template here: **lab3_stub.zip (https://sit.instructure.com/courses/71954/files/13049641?wrap=1)** (https://sit.instructure.com/courses/71954/files/13049641/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 "type": "module". DO NOT ADD ANY OTHER FILE OR FOLDER APART FROM PACKAGE.JSON FILE.**

Network JSON Data

You will be downloading JSON files from the following GitHub Gists:

- **authors.json (https://gist.githubusercontent.com/graffixnyc/a086a55e04f25e538b5d52a095fe4467/raw/e9f835e9a5439a**
- **books.json (https://gist.githubusercontent.com/graffixnyc/3381b3ba73c249bfcab1e44d836acb48/raw/e14678cd750a4**

For every function you write, you will download the necessary JSONs with `axios`. DO NOT just save the data into a local file, you MUST use Axios to get the data. Here is an example of how to do so:

```
async function getAuthors(){
  const { data } = await axios.get('https://..gist_url../authors.json')
  return data // this will be the array of author objects
}
```

Instead of making the request in every single function, remember that code reuse is key. So if you see that you are making the same axios request in all of your functions, it's best to put it in a function like noted above and then call that function in all of the functions that need to get the data from whichever json file you're working with. Always do this when you see you are doing the same thing over and over again in multiple different places. It's much easier to maintain. Say if the URL of the file ever changes, then you only need to change it in one place, not 10 different places.

General note: For all functions that receive strings as input, you must trim the input! Also, for this lab, the input should be case in-sensitive(with exception to strings that are IDs as ids are usually always case sensitive.). So for example, calling: `getBooksByState("ny")` should return the same results as if you ran `getBooksByState("NY")` which would also return the same results as `getBooksByState("   NY   ")`. Empty strings or strings with just spaces are not valid input for any of the functions!

authors.js

This file will export the following five functions:

getAuthorById(id)

This will return the author for the specified id within the `authors.json` array. **Note: The `id` is case sensitive**. Notice you are returning one single object, not an array with an object as an element! Full points will be deducted if you return this as an array!

You must check:

- That the `id` parameter exists and is of proper type (string). If not, throw an error.
- If the id exists and is in the proper type but the `id` is not found in the array of authors, throw an 'author not found' error.
- if the `id` parameter is just empty spaces, throw an error.

```
await getAuthorById("1871e6d7-551f-41cb-9a07-08240b86c95c");
// Returns:
{
id: '1871e6d7-551f-41cb-9a07-08240b86c95c',
first_name: 'Derward',
last_name: 'Ticic',
date_of_birth: '6/3/1932',
HometownCity: 'Garden Grove',
HometownState: 'CA',
books: ['4efdb199-5a0f-4410-bded-ce07990c6aa4']
}

await getAuthorById(-1); // Throws Error
await getAuthorById(1001); // Throws Error
await getAuthorById(); // Throws Error
await getAuthorById('7989fa5e-5617-43f7-a931-46036f9dbcff');// Throws Author not found Error
```

searchAuthorsByAge(age)

For this function, you will return an array of author names whose age is equal to or older than the age argument passed into the function. So for example, if the age is 35, you will return all of the authors who are 35 years old or older. You should concatenate the authors first and last name when adding it to the array of names.

You must check:

- That the `age` parameter exists
- That the `age` parameter is the correct type (a number)
- the `age` parameter is a positive whole number between 1-100

If any of these checks fail, you will throw an error.

```
await searchAuthorsByAge(40);
// Returns:
["Mayer Staddart", "Madelaine Armatage", "Adorne Byrant"...] //Only the first three are shown

await searchAuthorsByAge(5000); // Throws Error since there are no results
await searchAuthorsByAge(" "); // Throws Error
await searchAuthorsByAge("abc"); // Throws Error
await searchAuthorsByAge(); // Throws Error
```

getBooksByState(state)

For this function, you will return an array of book names from authors whose hometown state is the same as the state passed into the function (i.e. all of the book names from authors from NJ). You must look up the book names in books.js!

You must check:

- That the `state` parameter exists
- That the `state` parameter is a string
- That the `state` parameter is only two characters
- That the `state` parameter is a valid state abbreviation

If any of these checks fail, you will throw an error.

```
await getBooksByState("NJ");
// Returns
["Summertime","Crime and Punishment"] // there are others, but this an example of just a few NJ book
s

await getBooksByState(123); // Throws Error
await getBooksByState(" "); // Throws Error
await getBooksByState("Patrick"); // Throws Error because there is no state Patrick in authors.json
await getBooksByState(); // Throws Error
```

searchAuthorsByHometown(town, state)

For this function, you will return an array of author names whose hometown matches the town and state passed into the function (i..e all of the authors from jacksonville, MI). You should concatenate the authors first and last name together when adding it to the array of names. If no authors are found, you will return an empty array. You should sort the authors names in AZ order by last name.

You must check:

- That the `town` and `state` parameters exist
- That both the `town` and `state` input parameters are strings
- That the `state` parameter is only two characters
- That the `state` parameter is a valid state abbreviation

If any of these checks fail, you will throw an error.

```
await searchAuthorsByHometown("New York City", "NY");
// Returns
["Maurice McKinless","Mayer Stadart"] // there are others, but this an example of just a few authors

await searchAuthorsByHometown(123, 456); // Throws Error
await searchAuthorsByHometown("", ""); // Throws Error
await searchAuthorsByHometown("Patrick", "Hill"); // Throws Error because there is no state hill or
town patrick in authors.json
await searchAuthorsByHometown(); // Throws Error
```

getAuthorBooks(authorid)

For this function, you will return all of the books for the author with the given id. But there is a catch! Instead of returning the book ids, you will return the names of the books. You will have to take each bookId and look up in books.js what the actual name is. If the author has no books, return an empty array.

You must check:

- That the `authorid` parameter exists
- That the `authorid` parameter is a string
- That the `authorid` parameter is a valid author id (meaning if can be found in the data)

If any of these checks fail, you will throw an error.

```
await getAuthorBooks("69b3f32f-5690-49d1-b9a6-9d2dd7d6e6cd"); // Returns: ["Jason X", "Nanny McPhe
e"]
await getAuthorBooks("2155574a-80b0-4389-8bb3-3240da52b770"); // Returns: []
await getAuthorBooks(""); // Throws Error
await getAuthorBooks(230); // Throws Error
await getAuthorBooks(); // Throws Error:
```

books.js

This file will export five functions:

getBookById(id)

This will return the Book object for the specified id within the `books.json` array. **Note: The `id` is case sensitive.** Notice you are returning one single object, not an array with an object as an element! Full points will be deducted if you return this as an array! **You MUST trim your inputs for this function!**

You must check:

- That the `id` parameter exists and is of proper type (string). If not, throw an error.
- If the id exists and is in the proper type but the `id` is not found in the array of books, throw a 'book not found' error.
- if the `id` parameter is just empty spaces, throw an error.

```
await getBookById("99875ad8-a1d3-42ea-8d7b-5ac4cd4edb9e");
// Returns:
{
  id: '99875ad8-a1d3-42ea-8d7b-5ac4cd4edb9e',
  title: 'No habrá paz para los malvados',
  genres: ['Art', 'Travel'],
  publicationDate: '5/7/2018',
  publisher: 'Avamm',
  summary:   'Maecenas ut massa quis augue luctus tincidunt. Nulla mollis molestie lorem. Quisque ut
erat.\n\nCurabitur gravida nisi at nibh. In hac habitasse platea dictumst. Aliquam augue quam, solli
citudin vitae, consectetuer eget, rutrum at, lorem.\n\nInteger tincidunt ante vel ipsum. Praesent bl
andit lacinia erat. Vestibulum sed magna at nunc commodo placerat.',
  isbn: '520476104-7',
  language: 'Finnish',
  pageCount: 693,
  price: 25.66,
  format: ['E-Book', 'Hardcover', 'Paperback'],
  authorId: 'f645d28a-670a-457a-b55f-a32876b8511d'
}

await getBookById(-1); // Throws Error
await getBookById(1001); // Throws Error
await getBookById();// Throws Error
await getBookById('7989fa5e-5617-43f7-a931-46036f9dbcff');// Throws book not found Error
```

booksByPageCount(min, max)

This function will return an array of book ids with page counts that fall between the min and max page counts provided. A book is also valid if it is equal to the min or max. So if the min is 200 and max is 400, books with page counts of 200 or 400 should be added to the array.

You must check:

- That both the `min` and `max` parameters exist
- That both the `min` and `max` parameters are numbers
- That both the `min` and `max` parameters are positive whole numbers
- insure that the max parameter is greater than the min parameter.
- Insure that the `max` parameter is greater than 0

If any of these checks fail, you will throw an error.

```
await booksByPageCount(300, 500);
// Returns: ["fe64fc98-95ff-4d47-bac8-93c755b85c95", "04e55bc9-0c7a-47a6-a403-52eabf25c6ef"]
//there are more, these are just a few examples

await booksByPageCount(-1, 100); // Throws Error
await booksByPageCount("ABC", "3"); // Throws Error
await booksByPageCount(); // Throws Error
```

sameYear(year)

For this function, you will return an array of of all of the books published in the year provided to the function. If no books are found in that year, you will return an empty array. For this function, you must insert the full book object into your array.

You must check:

- That the `year` parameter exists
- That the `year` parameter is valid number (positive whole number)
- That the `year` parameter is a valid year

If any of these checks fail, you will throw an error.

The output sample below only shows the first 3results. You must return ALL the book objects from the array of books in books.json that match the `genre` supplied to the function.

```
await sameYear(2000);
// Returns
[  {
    "id": "2a195547-19a8-42bf-9c2a-735b0499f8f2",
    "title": "Simon Magus",
    "genres": [
        "Horror",
        "Humor",
        "History",
        "Travel"
    ],
    "publicationDate": "2/29/2000",
    "publisher": "Rooxo",
    "summary": "Duis bibendum. Morbi non quam nec dui luctus rutrum. Nulla tellus.\n\nIn sag
    ittis dui vel nisl. Duis ac nibh. Fusce lacus purus, aliquet at, feugiat non, pretium qu
    is, lectus.\n\nSuspendisse potenti. In eleifend quam a odio. In hac habitasse platea dic
    tumst.",
    "isbn": "508621237-5",
    "language": "Bengali",
    "pageCount": 143,
    "price": 29.91,
    "format": [
       "Hardcover"
    ],
    "authorId": "6b979044-6c52-4d95-944d-b0f08c724f1c"
 },
 {
    "id": "6e1c66e8-b120-4e22-bc77-1e91b837051e",
    "title": "Sanshiro Sugata Part Two (Judo Saga II) (Zoku Sugata Sanshirô)",
    "genres": [
       "Gothic",
       "Romance"
    ],
    "publicationDate": "8/12/2000",
    "publisher": "Mudo",
    "summary": "In hac habitasse platea dictumst. Morbi vestibulum, velit id pretium iaculi
    s, diam erat fermentum justo, nec condimentum neque sapien placerat ante. Nulla justo.",
    "isbn": "484856959-1",
    "language": "Pashto",
    "pageCount": 533,
    "price": 43.55,
    "format": [
       "Paperback"
    ],
    "authorId": "669c00a3-ff8b-4fb6-a913-f6bd5739a5b1"
 },
   {
    "id": "70b1fcf3-869e-49fd-a200-916ff654542a",
    "title": "Hipsters (Stilyagi)",
    "genres": [
       "Thriller",
       "History",
       "Horror",
       "Art"
    ],
    "publicationDate": "11/12/2000",
    "publisher": "Mudo",
    "summary": "Cras non velit nec nisi vulputate nonummy. Maecenas tincidunt lacus at veli
    t. Vivamus vel nulla eget eros elementum pellentesque.",
    "isbn": "978089876-X",
    "language": "Bengali",
    "pageCount": 262,
    "price": 60.37,
    "format": [
       "E-Book"
    ],
    "authorId": "2579080f-eb74-4ed3-8167-2e376841407c"
 },}
   {
    "id": "dbc85e9f-525d-43bf-bd4e-13f2a581690c",
    "title": "Kundun",
    "genres": [
       "Humor",
       "Art"
    ],
    "publicationDate": "7/23/2000",
    "publisher": "Quatz",
    "summary": "Aliquam quis turpis eget elit sodales scelerisque. Mauris sit amet eros. Sus
    pendisse accumsan tortor quis turpis.\n\nSed ante. Vivamus tortor. Duis mattis egestas m
    etus.\n\nAenean fermentum. Donec ut mauris eget massa tempor convallis. Nulla neque libe
    ro, convallis eget, eleifend luctus, ultricies eu, nibh.",
    "isbn": "225975531-3",
    "language": "Greek",
    "pageCount": 909,
    "price": 94.85,
    "format": [
       "Paperback",
       "E-Book"
    ],
    "authorId": "519bb91e-e170-46e3-96f6-a363ea30ff1d"
}
];
await sameYear(-1); // Throws Error
await sameYear(1001); // Throws Error
await sameYear();// Throws Error
await sameYear(false)// throws error
await sameYear('foo bar');// Throws Error
```

minMaxPrice()

For this function, you will return the id of the cheapest book in the dataset, and the id of the most expensive book. If there is a tie for more than one cheapest/most expensive, you will return all of those books in an array. The result should be in an object formatted as follows:

{cheapest: ["bookId"], mostExpensive: ["bookId"]}, or if there is a tie: {cheapest: ["bookId1", "bookId2"], mostExpensive: ["bookId"]},

We will not provide examples for this one so that students do not hard code answers! Also since it takes in no input, there is no input validation you have to do for this function! (yay!)

searchBooksByPublisher(publisher)

For this function, you will return an array of book ids for all of the books that have the same publisher provided in the publisher argument. If no books are found for the given publisher, you will throw an error.

You must check:

- That the `publisher` parameter exists
- That the `publisher` parameter is a string
- That the `publisher` parameter exists in the data, meaning that publisher name can be found in the dataset.

If any of these checks fail, you will throw an error.

```
await searchBooksByPublisher("Skilith"); // Returns ["519c733a-6a5d-451f-927d-0e860b5d1e3d", "25
4a77b0-f055-4dc1-b9fa-3b23d811c8be"]
await searchBooksByPublisher("A fake publisher"); // Throws Error
await searchBooksByPublisher();// Throws Error
await searchBooksByPublisher(false)// throws error
await searchBooksByPublisher('foo bar');// Throws Error
```

Requirements

1. Write each function in the specified file and export the function so that it may be used in other files.
2. Ensure to properly error check for different cases such as arguments existing and of the proper type as well as **throw (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/throw)** if anything is out of bounds such as invalid array index or negative numbers for different operations.
3. Submit all files (including `package.json`) in a zip with your name in the following format: `LastName_FirstName.zip`.
4. Make sure to save any npm packages you use to your `package.json.`
5. **DO NOT** submit a zip containing your `node_modules` folder.

Lab 3 Rubric Spring 2024

Criteria	Ratings		Pts
authors.js - getAuthorById(id) Test cases used for grading will be different from assignment examples.	**5 to >0.0 pts** **.5 Points/Error Handling Test Case & 1.5 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.	5 pts
authors.js - searchAuthorsByAge(age) Test cases used for grading will be different from assignment examples.	**11.25 to >0.0 pts** **1 Points/Error Handling Test Case & 2.625 Points/Valid Input Test Case** 6 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.	11.25 pts
authors.js - getBooksByState(state) Test cases used for grading will be different from assignment examples.	**11.25 to >0.0 pts** **1 Points/Error Handling Test Case & 2.625 Points/Valid Input Test Case** 6 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.	11.25 pts
authors.js - searchAuthorsByHometown(town, state) Test cases used for grading will be different from assignment examples.	**11.25 to >0.0 pts** **1 Points/Error Handling Test Case & 2.625 Points/Valid Input Test Case** 6 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.	11.25 pts

Criteria	Ratings		Pts
authors.js - getAuthorBooks(authorid) Test cases used for grading will be different from assignment examples.	**11.25 to >0.0 pts** **1 Points/Error Handling Test Case & 2.625 Points/Valid Input Test Case** 6 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.	11.25 pts
books.js - getBookById Test cases used for grading will be different from assignment examples.	**5 to >0.0 pts** **.5 Points/Error Handling Test Case & 1.5 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.	5 pts
books.js -booksByPageCount(min, max) Test cases used for grading will be different from assignment examples.	**11.25 to >0.0 pts** **1 Points/Error Handling Test Case & 2.625 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.	11.25 pts
books.js - sameYear(year) Test cases used for grading will be different from assignment examples.	**11.25 to >0.0 pts** **1 Points/Error Handling Test Case & 2.625 Points/Valid Input Test Case** 6 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.	11.25 pts
books.js - minMaxPrice() Test cases used for grading will be different from assignment examples.	**11.25 to >0.0 pts** **11.25 Points single test case** There are no invalid inputs for this function. It will either produce the correct results or not.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	11.25 pts

Criteria	Ratings		Pts
books.js - ssearchBooksByPublisher(publisher) Test cases used for grading will be different from assignment examples.	**11.25 to >0.0 pts** **11.25 Points single test case** There are no invalid inputs for this function. It will either produce the correct results or not.	**0 pts** **All Test Cases Failed** Incorrect implementation or none of the test cases pass.	11.25 pts
Total Points: 100			