Lab 3

Start Assignment

Due Tuesday by 11:59pm **Points** 100 **Submitting** a file upload

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 Promises). 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`.

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: `people.js`, `stocks.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"}`

Network JSON Data

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

- people.json (https://gist.githubusercontent.com/graffixnyc/a1196cbf008e85a8e808dc60d4db7261/raw/9fd0d1a4d7846b
- stocks.json (Links to an external site.) (https://gist.githubusercontent.com/graffixnyc/8c363d85e61863ac044097c0d199dbcc/raw/7d79752a9342a

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 getPeople(){
  const { data } = await axios.get('https://.../people.json')
  return data // this will be the array of people objects
}
```

Instead of making the request in every single function, remember that code reuse is key, so if you see

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 calling that function in all 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.

people.js

This file will export the following four functions:

getPersonById(id)

This will return the person for the specified id within the `people.json` array. Note: The `id` is case sensitive.

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 people, throw a 'person not found' error.
- if the `id` parameter is just empty spaces, throw an error.

```
await getPersonById("7989fa5e-8f3f-458d-ad58-23c8d9ef5a10");
\\ Returns:

{

  id: "7989fa5e-8f3f-458d-ad58-23c8d9ef5a10",

  first_name: "Val",

  last_name: "Kinsell",

  email: "vkinsell4@icq.com",

  ip_address: "169.162.241.22",

  ssn: "578-08-2277",

  date_of_birth: "11/30/1979",


  address: {

    home: {

      street_number: "0",

      street_name: "Schiller",

     street_suffix: "Junction",

     city: "Houston",
```

```
    state: "TX",
    zip: "77090"
  },
  work: {
    street_number: "21",
    street_name: "Dryden",
    street_suffix: "Trail",
    city: "New York City",
    state: "NY",
    zip: "10034"
    }
  }
}


await getPersonById(-1); \\ Throws Error
await getPersonById(1001); \\ Throws Error
await getPersonById();\\ Throws Error
await getPersonById('7989fa5e-5617-43f7-a931-46036f9dbcff');\\ Throws person not found Error
```

sameEmail(emailDomain)

For this function, you will return an array of people `objects` who have the same email address domain from `people.json`. B**oth the domain name and extension (.com, .edu etc) must match**. You must return at least two people, so if there are not 2 or more people that have the same email domain for the `emailDomain` provided you will throw an error.

You must check:

- That the `emailDomain` parameter exists and is of proper type (string). If not, throw an error.
- If there are not at least two people that have the same email domain provided, you will throw an error.
- if `emailDomain` is just empty spaces, throw an error.
- if `emailDomain` does not contain a dot `.` , throw an error

- if `emailDomain` does not have at LEAST 2 LETTERS after the dot, throw an error Note: Some of the email addresses have multiple dots like `@google.co.jp` you will need to take that into account and check the letters after the last dot in the email domain.
- The `emailDomain` parameter must be case in-sensitive i.e. `sameEmail("harvard.edu")` should return the same results as passing `sameEmail("HARVARD.EDU")`

```
await sameEmail("harvard.edu");
\\ Returns:
[
```

```
{
  "id": "1380f2af-e0d8-4231-a9e7-f09650afc0bb",
  "first_name": "Vonnie",
  "last_name": "Skoate",
  "email": "vskoatend@harvard.edu",
  "ip_address": "136.44.63.233",
  "ssn": "752-11-3294",
  "date_of_birth": "09/30/1999",
  "address": {
    "home": {
      "street_number": "43",
      "street_name": "Hermina",
      "street_suffix": "Street",
      "city": "Port Washington",
      "state": "NY",
      "zip": "11054"
    },
    "work": {
      "street_number": "65522",
      "street_name": "Hayes",
      "street_suffix": "Avenue",
      "city": "Mobile",
      "state": "AL",
      "zip": "36616"
    }
}
},

{
  "id": "3c9e028d-c739-4c70-b81a-cfcf8c094cb0",
  "first_name": "Arlin",
  "last_name": "Awdry",
  "email": "aawdryql@harvard.edu",
  "ip_address": "220.107.136.220",
  "ssn": "291-66-6045"
```

```
    "ssn": "291-66-6045",
    "date_of_birth": "06/30/1985",
    "address": {
      "home": {
        "street_number": "9",
        "street_name": "Columbus",
        "street_suffix": "Center",
        "city": "Rochester",
        "state": "NY",
        "zip": "14609"
      },
     "work": {
        "street_number": "69431",
        "street_name": "Scofield",
        "street_suffix": "Alley",
        "city": "Green Bay",
        "state": "WI",
        "zip": "54305"
      }
    }
  },
  {
    "id": "3c9e028d-c739-4c70-b81a-cfcf8c094cb0",
    "first_name": "Arlin",
    "last_name": "Awdry",
    "email": "aawdryql@harvard.edu",
    "ip_address": "220.107.136.220",
    "ssn": "291-66-6045",
    "date_of_birth": "06/30/1985",
    "address": {
      "home": {
        "street_number": "9",
        "street_name": "Columbus",
        "street_suffix": "Center",
        "city": "Rochester"
```

```
        "city": "Rochester",
        "state": "NY",
        "zip": "14609"
      },
    "work": {
        "street_number": "69431",
        "street_name": "Scofield",
        "street_suffix": "Alley",
        "city": "Green Bay",
       "state": "WI",
       "zip": "54305"
      }
    }
  }
]
```

```
await sameEmail("foobar"); \\ Throws Error
await sameEmail("foobar."); \\ Throws Error
await sameEmail("foobar.123"); \\ Throws Error
await sameEmail(".com"); \\ Throws Error
await sameEmail(); \\ Throws Error
await sameEmail("google.com.hk"); \\ Throws Error since there are not at least two people that have the email
domain google.com.hk
```

manipulateIp()

For this function, you must convert all the IP addresses to numbers (removing the dots) then sort each person's IP field from lowest to highest digit. For example, an IP of: `"111.26.173.248"` would become the number `11112234678`. Note: If the sorted IP's first number(s) is/are 0, Javascript will drop the leading 0's. This is expected. So the IP of: `"143.90.193.247"` would become: `1123344799` you will then return an

object that contains the person's name (first and last name) with the highest number, the person's name (first and last name) with the lowest number and the Math.floor of the average from all people:

Note: If there is more than one person who has the same highest/lowest sorted IP value return the first person.

You will return an object in the following format:

```
{
```

```
highest: { firstName: 'Patrick', lastName: 'Hill' },

lowest: { firstName: 'Christina', lastName: 'Li' },

average: 88383773  // just an example, this will be computed average from all converted SSNS  make sure this
 value is a number, NOT a string

}
```

sameBirthday(month, day)

This function will take in the month and day of a birthday: `sameBirthday(9, 25)` and it will return an array of strings with all the people who were born in that month, on that day. You will show each person's first and last name as one string for each person as shown in the output below.

You must check:

- That the `month` and `day` parameters exists and are of proper type (numbers). If not, throw an error. (If a string is passed in, you can try to parse that into a valid number first, if it's a valid number, then you can check the following checks, if it cannot be parsed into a number, throw an error.
- That the `month` parameter is 1-12 , if not, throw an error
- That the `day` parameter is valid for that month. i.e. if `month` = 9 then `day` cannot be >= 31 since there are only 30 days in September. If `month` = 2 then `day` cannot be >= 29 since there are only 28 days in February. (You do not have to take leap year into account).
- If there are no people with that birthday, throw an error

```
await sameBirthday(9, 25); \\ Returns: ['Khalil Ovitts',  'Erny Van Merwe', 'Emanuel Saben', 'Iorgos Tembridg
e']
await sameBirthday("09", "25"); \\ Returns: ['Khalil Ovitts',  'Erny Van Merwe', 'Emanuel Saben', 'Iorgos Tem
bridge'] because the parameters can be parsed into valid numbers.
await sameBirthday(9, 31); \\ Throws Error: There are not 31 days in Sept
await sameBirthday(13, 25); \\ Throws Error: Month > 12
await sameBirthday(2, 29); \\ Throws Error: There are not 29 days in Feb
await sameBirthday("09", "31"); \\ Throws Error: There are not 31 days in Sept
await sameBirthday("      ", "25"); \\ Throws Error

await sameBirthday(); \\ Throws Error:
```

stocks.js

This file will export four functions:

listShareholders(stockName)

For this function, you will return an `object` for the `stockName` provided. You will return an object that has the stock data for the stockName provided, but instead of displaying the userid's of the shareholders, you will look up that userid in `people.json` and return that user's first name and last name in the return object instead:

You must check:

- That `stockName` parameter exists and is of the proper type (string). If not, throw an error.
- You must check to make sure the `stockName` parameter is not just empty spaces: If it is, throw an error.
- If the stock name cannot be found in `stocks.json` for the supplied `stockName` parameter, then throw an error.

Note: If a stock has no shareholders, you will return the stock object as is with the empty array (shown below).

```
Original Stock object:

{

  "id": "7283e5d6-7481-41cb-83b3-5a4a2da34717",

  "stock_name": "Aeglea BioTherapeutics, Inc.",

  "shareholders": [
    {"userId": "d8867404-3006-4b09-bb13-1836f88217e2","number_of_shares": 55},
    {"userId": "2a9896a4-9f83-4478-bdc4-bc528fde97a4","number_of_shares": 449},
    {"userId": "77f18859-4f6e-451d-ad4d-137e10084bf7","number_of_shares": 120},
    {"userId": "76ed376e-d673-422d-912d-aefba645bae7","number_of_shares": 14},
    {"userId": "e9db7f4e-d1e1-4eac-a329-16fc7e9e8b82","number_of_shares": 25}
  ]
}



listShareholders("Aeglea BioTherapeutics, Inc.") Would return:

{

  id: '7283e5d6-7481-41cb-83b3-5a4a2da34717',

  stock_name: 'Aeglea BioTherapeutics, Inc.',

  shareholders: [

     { first_name: "Paolo", last_name: "Victoria", number_of_shares: 55 },

     { first_name: "Caresse",last_name: "Clissett", number_of_shares: 449 },

     { first_name: "Benedikta",last_name: "Meller", number_of_shares: 120 },

     { first_name: "Kristy",last_name: "Goady", number_of_shares: 14 },

     { first_name: "Balduin",last_name: "Blackmuir", number_of_shares: 25 }

   ]

}

listShareholders("Powell Industries, Inc.") Would return:



{
        "id": "929686a2-dd3a-42c7-a88d-b170e2590252",
        "stock_name": "Powell Industries, Inc.",
        "shareholders": []
}
```

```
await listShareholders('foobar') // Throws Error
await listShareholders() // Throws Error
```

totalShares(stockName)

Given the `stockName` provided, you will find the company from `stocks.json` and will then calculate how many shareholders that company has and how many total shares people own of that stock.

Make sure to follow the example output string below **exactly**.

You must check:

- That `stockName` parameter exists and is of the proper type (string). If not, throw an error.
- You must check to make sure the `stockName` parameter is not just empty spaces: If it is, throw an error.
- If the stock name cannot be found in `stocks.json` for the supplied `stockName` parameter, then throw an error.

NOTE: If the stock name is found in `stocks.json` but there are no shareholders you will return: "stockName currently has no shareholders." (replacing stockName with the name of the stock as shown in the example below). NOTE: That the output is a little different if there is just one shareholder. Take note of the plural/singular examples below:

```
await totalShares('Aeglea BioTherapeutics, Inc.');
\\ Returns: "Aeglea BioTherapeutics, Inc., has 5 shareholders that own a total of 663 shares."

await topShareholder('Nuveen Preferred and Income 2022 Term Fund');
\\ Returns: "Nuveen Preferred and Income 2022 Term Fund, has 1 shareholder that owns a total of 285 shares."

await totalShares('Aeglea BioTherapeutics, Inc.');
\\ Returns: "Aeglea BioTherapeutics, Inc., has 5 shareholders that own a total of 663 shares."

await topShareholder('Powell Industries, Inc.');
\\ Returns: "Powell Industries, Inc. currently has no shareholders."

await totalShares(43); \\ Throws Error
await totalShares(' '); \\ Throws error
await totalShares('Foobar Inc'); \\ Throws error No stock with that name
await totalShares(); \\ Throws Error
```

NOTE: The quotes are just to denote that your function returns a string do not return it with extra quotes around it.
However, the exact format of the string needs to be precise including case, spacing and punctuation.

listStocks(firstName, lastName)

Given a `firstName` and `lastName`, find the person in the `people.json` array, then, if they are found, get their ID and find out how many company stocks they own and the number of shares for each from `stocks.json`. You will return an array of objects with the stock name and the number of shares they own in each company.

You must check:

- That `firstName` and `lastName` parameters exists and are of the proper type. (strings). If not, throw an error.
- That `firstName` and `lastName` parameters are not just empty strings
- That the person specified exists in the `people.json` array. If not, throw an error.
- That the person specified owns shares in at least one company. If not, throw an error

```
await listStocks("Grenville", "Pawelke" );
// Returns:
[
  {stock_name: "PAREXEL International Corporation", number_of_shares: 443},
  {stock_name: "Vanguard Russell 2000 ETF", number_of_shares: 59},
  {stock_name: "National CineMedia, Inc.", number_of_shares: 320},
  {stock_name: "CombiMatrix Corporation", number_of_shares: 434}

]
await listStocks('Patrick', "Hill"); // Throws Error because Patrick Hill is not in people.json
await listStocks(); // Throws Error
await listStocks("foo"); // Throws Error
await listStocks("      ", "        "); // Throws Error
await listStocks(1,2); // Throws Error
```

getStockById(id)

This will return the Stock for the specified id within the `stocks.json` array. Note: The `id` is case sensitive.

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 stocks, throw a 'stock not found' error.
- if the `id` parameter is just empty spaces, throw an error.

```
await getStockById("f652f797-7ca0-4382-befb-2ab8be914ff0");
\\ Returns:
{
  id: 'f652f797-7ca0-4382-befb-2ab8be914ff0',

  stock_name: 'Transcat, Inc.',

  shareholders: [

    {userId: '55ce26c4-915c-4a99-afe9-544e57227fcd',number_of_shares: 155},

    {userId: 'b9245e24-0ac7-49fc-a487-af4b7142a7e4',number_of_shares: 307},
```

```
    {userId: 'ed0ec3bf-3ec4-4025-8b26-ebd17487cb22',number_of_shares: 44},

    {userId: '56b285ff-ff97-417b-a9c7-d423b13c25a6',number_of_shares: 396},

    {userId: 'bf8b066a-3f06-4987-9e61-0388f6374a4d',number_of_shares: 335},

    {userId: '9369a0eb-9d4c-434a-901b-b29a92da91ed',number_of_shares: 399},

    {userId: '74fd73cb-1ca9-443d-9c8a-b263fe4e0ce3',number_of_shares: 139}

  ]

} await getStockById(-1); \\ Throws Error
await getStockById(1001); \\ Throws Error
await getStockById();\\ Throws Error
await getStockById('7989fa5e-5617-43f7-a931-46036f9dbcff');\\ Throws stock not found Error
```

app.js

This file is where you will import your functions from the two other files and run test cases on your functions by calling them with various inputs. We will not use this file for grading and is only for your testing purposes to make sure:

1. Your functions in your 2 files are exporting correctly.

2. They are returning the correct output based on the input supplied (throwing errors when you're supposed to, returning the right results etc..).

Note: You will need an `async` function in your app.js file that `awaits` the calls to your function like the example below. You put all of your function calls within `main` each in its own `try/catch` block. and then you just call `main()`.

```
const people = require("./people");

async function main(){
    try{
        const peopledata = await people.getPeople();
        console.log (peopledata);
    }catch(e){
        console.log (e);
    }
}

//call main
main();
```

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:

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.