Lab 3

Re-submit Assignment

Due Sep 29 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`, `work.js` and `app.js`.

Network JSON Data

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

- people.json (https://gist.githubusercontent.com/graffixnyc/31e9ef8b7d7caa742f56dc5f5649a57f/raw/43356c676c2cdc8
- work.json (https://gist.githubusercontent.com/graffixnyc/febcdd2ca91ddc685c163158ee126b4f/raw/c9494f59261f655

Note: The data in the json files have quotes around the key names where the output that your functions output do not. Axios should parse this for you, however, if you have quotes in your output for whatever reason, you will need to use the `JSON.parse()` function to parse the json into a JS object.

For every function you write, you will download the necessary JSONs with `axios`. Here is an example of how to do so:

```
async function getPeople(){
  const { data } = await axios.get('https://.../people.json')
  const parsedData = JSON.parse(data) // parse the data from JSON into a normal JS Object
  return parsedData // this will be the array of people objects
}
```

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.

You must check:

- That the `id` parameter exists and is of proper type (number). If not, throw an error.
- The `id` is within bounds in respect to id values in the array of people. If not, throw an error.

```
await getPersonById(43);
\\ Returns:
{
  id: 43,
  first_name: "Tades",
  last_name: "Paskell",
  date_of_birth: "01/02/1987",
  ssn: "648-97-2273",
  email: "tpaskell16@cam.ac.uk",
  ip_address: "72.63.45.5",
  address: {
    street_address: "6 South Plaza",
    city: "Huntsville",
    state: "AL",
    zip_code: "35810"
  }
}
await getPersonById(-1); \\ Throws Error
await getPersonById(1001); \\ Throws Error
await getPersonById();\\ Throws Error
```

howManyPerState(stateAbbrv)

For this function, you will return the count of how many people live in the state passed as `stateAbbrv` from `people.json`

You must check:

- That the `stateAbbrv` parameter exists and is of proper type (string). If not, throw an error.
- If there are no people that live in the state that was provided in `stateAbbrv`, you will throw an error.

```
await howManyPerState("NY"); \\ Returns: 64
await howManyPerState("CO"); \\ Returns: 27
await howManyPerState(-1); \\ Throws Error
await howManyPerState("WY"); \\ Throws Error since there are no people in WY
await howManyPerState(); \\ Throws Error
```

personByAge(index)

For this function, you must sort all the people by their date_of_birth from oldest to youngest (in `people.json`) Then, return the person for the index provided with the fields below. Notice, that you need to compute their age (in full years) and return that in the return object

You must check:

- That the `index` parameter exists and is of proper type (number). If not, throw an error.
- The `index` is within bounds in respect to the array. If not, throw an error.

```
await personByAge(0);
\\ Returns:
{
  first_name: 'Chaim',
  last_name: 'Giovannacci',
  date_of_birth: '01/30/1930',
  age: 90
}

await personByAge(43);
\\ Returns:
{
  first_name: 'Chen',
  last_name: 'Sowthcote',
  date_of_birth: '10/01/1934',
  age: 85
}

await personByAge(500);
\\ Returns:
{
  first_name: 'Guglielmo',
  last_name: 'Kubera',
  date_of_birth: '05/07/1974',
  age: 46,

}

await personByAge(999);
\\ Returns:
{

  first_name: 'Jacenta',
  last_name: 'Gowrich',
  date_of_birth: '08/21/2020',
  age: 0,
}

await personByAge(-1); \\ Throws Error
await personByAge(1000); \\ Throws Error
await personByAge(); \\ Throws Error
```

peopleMetrics()

Using all the people in `people.json`, collect and return the following metrics:

```
{
  totalLetters: sum of all the letters in all the first and last names combined (full names),
  totalVowels: sum of all the vowels in all the first and last names combined (full names),
  totalConsonants: sum of all the consonants in all the first and last names combined (full names),
  longestName: The longest name in the list (full name, so: first and last name combined),
  shortestName: The shortest name in the in the list (full name, so: first and last name combined)
  mostRepeatingCity: the name of the city that appears most in the list,
  averageAge: The average age of everyone in the list
}
```

This function does not take any arguments. We will only count: `a,e,i,o,u` as vowels.

work.js

This file will export three functions:

listEmployees()

For this function, you will return an array of objects. You will have the following keys: `companyName` the value for this key will be the name of the company from `work.json.` The list of companies has an array that contains the IDs of the employees that work there. You will look up each employee in `people.json` based on the ID stored and then return an array of objects with the first and last name of each employee. Here is a sample of what you will be returning for each company in the list.

```
\\ Returns:
[
  {
    company_name": "Raynor-Grimes",
    employees: [
      {
        first_name: 'Konstantine',
        last_name: 'Edlyn'
      },
      {

        first_name: 'Montague',
        last_name: 'Harrop'
      },
      {
        first_name: 'Barbette',
        last_name: 'Bacher'
      },
      {
        first_name: 'Kermie',
        last_name: 'Farmer'
      }
    ]
  },
  {
    company_name: "Hilll, Waters and Bins",
    employees: [
      {
        first_name: 'Shelby',
        last_name: 'Elcum'
      },
      {
        first_name: 'Dusty',
        last_name: 'Leipnik'
      },
      {
        first_name: 'Adan',
        last_name: 'Marson'
      },
      {
        first_name: 'Lindsey',
        last_name: 'Hubeaux'
      }
    ]
  }, ...
]
```

fourOneOne(phoneNumber)

Given the phone number provided, you will find the company from `work.json` for that phone number and will return the following: the company name, and the full company address

You must check:

- That `phoneNumber` parameter exists and is of the proper type (string). If not, throw an error.
- You must check to make sure the `phoneNumber` parameter is in the same format as the data: `"###-###-####"`. If not, throw an error.
- If the company cannot be found for the supplied phone number, then throw an error.

```
await fourOneOne('240-144-7553');
\\ Returns:
{

  company_name: "Kassulke, Towne and Davis",
  company_address:
  {
    street_address: "1 Claremont Plaza",
    city: "Frederick",
    state: "MD",
    zip_code: "21705"
  }
}
```

```
await fourOneOne(43); \\ Throws Error
await fourOneOne('212-208-8374'); \\ Throws error as nothing exists for that phone number
await fourOneOne('5045890047'); \\ Throws error not in proper ###-###-#### format
await fourOneOne(); \\ Throws Error
```

whereDoTheyWork(ssn)

Given a `ssn`, find the person in the `people.json` array, then, if they are found, get their ID and find out where they work from `work.json`. Then, print their full name and the name of the company they work for.

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

You must check:

- That `ssn` parameter exists and is of the proper type. (string). If not, throw an error.
- You must check to make sure the `ssn` parameter is in the same format as the data: `"###-##-####".` If not, throw an error.
- That the person specified exists in the `people.json` array. If not, throw an error.

```
whereDoTheyWork('299-63-8866'); // Returns: "Marga Dawidowitsch works at Durgan LLC."
whereDoTheyWork('277-85-0056'); // Returns: "Toby Ginsie works at Hirthe, Adams and Reilly."
whereDoTheyWork(); // Throws Error
whereDoTheyWork("123456789"); // Throws Error, not in proper format
whereDoTheyWork("264-67-0084"); // Throws Error as no one exists with that SSN.
```

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: `LastName_FirstName.zip`.
4. Make sure to save any npm packages you use to you `package.json.`
5. **DO NOT** submit a zip containing your `node_modules` folder.