Asynchronous JavaScript Lab

In this lab we'll display a list of persons on the screen with a photo and data about each person. We'll start out by reading this list in real time. Then we'll simulate reading it from a hard-to-get-to source that should not be read synchronously. We'll handle that reading with a promise.

This is a comprehensive lab which uses skills you learned in several previous chapters. You'll need to know about modules (importing and exporting), arrays (especially the map() and join() prototype functions), arrow functions, and string templates (with the backtic characters). Feel free to review those when you need to.

Reading JSON data and displaying it

1. We've provided a starter file for you with a list of persons. It is in the starters folder and is called persons.json. Copy that file into the src folder.
2. Import this array of persons into index.js like so:

```
import jsonPersons from './persons.json';
```

3. In index.js write a function called getPersons(). It should console.log(jsonPersons) and then return jsonPersons.
4. In your code, call getPersons() kind of like this:

```
let persons = [];
persons = getPersons();
console.log(persons);
```

5. Compile this code and test it in the browser. Make sure you're console logging all your persons.
6. Let's get them displayed in the browser. Since this isn't a class on HTML or CSS, we've supplied you with a shape for a customer card. You'll find a file in starters called person-card.html. Go take a look at that file.

We want to display our person data in the format you see in this file. (Feel free to modify the format if you want to).

7. You have a list of persons in a variable called persons. Call the Array.prototype.map function to convert this array into a different kind of array. The output from the map function should be a string using the format of the person-card.html file. Here are some hints:
    - You could simply copy and paste into index.js.
    - Or you could try to use an import statement to read in the html file into a string.
    - Your map function will return an array of strings in that HTML format.
    - There's also an Array.prototype.join function that will take an array of strings and join them into one big string.
8. After you have all of your persons in a huge HTML string, display it on the page like this:

```
document.getElementById("main").innerHTML = personString
```

Once you have a nice list displaying, you can move on.

Now all this happens instantaneously because when we bundle something with webpack it is right in the JavaScript. But in real life most reads happen slowly, like local storage reads, streams, ajax responses, and reading from the file system. If you try to read things like this synchronously, you'll be creating a 'blocking call' which means that nothing else can happen until the read is complete. In the worst case scenario, you'll crash your application because it is overwhelmed with waiting on lots of long-running processes. In the best case scenario, all the other developers will laugh at you behind your back (just kidding).

We're going to simulate reading this file as an asynchronous process by purposely putting a delay in it.

Adding a delay

9. Edit the getPersons function.
10. Put your "return jsonPersons" inside of a setTimeout() with a delay of 3000 milliseconds:

```
setTimeout(() => {
  console.log("finished reading persons", jsonPersons);
  return jsonPersons;
}, 3000);
```

11. If you run it now, you'll see your console.log() run three seconds after the page runs. But you're no longer seeing any persons on the page. Discuss with your partner why not. Make sure you have an understanding of why this is happening before you read on.

Got it? The issue here is that getPersons() now returns nothing -- an undefined -- rather than a list of persons. It returns immediately which is good -- it's no longer a blocking call. But since it returns immediately and we are taking three seconds to read our persons, we're trying to draw up our list of people before they're populated in the array.

We'll fix that by returning a promise.

Creating a promise

12. Edit getPersons() again. At the top, you'll have to instantiate a Promise:

```
const promise = new Promise( (resolve, reject) => {
  // We will do async stuff here in a minute
} );
```

13. Cut your entire setTimeout and paste it inside the promise function body. (HInt: put it where you have that comment.)
14. Remember that a promise represents the future return of some data. You return that data by calling the resolve function and passing the data into it. So we'll need to call *resolve(jsonPersons)* instead of *return jsonPersons;* Go ahead and make that change.
15. Then at the end of the function, you'll return the promise itself:

```
return promise;
```

16. Compile and test. If you're still console.logging the return from getPersons you'll see a Promise object in the console. (You'll also see some errors but we'll fix those in a minute).

We've made progress but we're still trying to draw up our person cards immediately instead of waiting for the promise to resolve. We'll do that in this last section.

Resolving the promise

17. Find where you're calling getPersons(). Since getPersons() now returns a promise, we must call the .then() method to handle a resolving of the promise. Do this:

```
getPersons().then( listOfPeople => {
  console.log("Promise is resolved", listOfPeople);
}
```

18. Compile and test. Oh my gosh! Three seconds pass and then your promise is resolved with your list of people. If that is happening, let's now display the data.
19. All that code where you're drawing things up, from the array.map() to the .join() to the .innerHTML = personsList -- put all of that inside your .then method after the "Promise is resolved" console.log.
20. Compile and test one last time. If you've got everything in the right places, you should see a blank page and three seconds later, boom! All of your persons appear in their cards. Keep discussing with your partner and adjusting until you've got the people appearing in the HTML.

Once you do, you can be finished with this lab.

Bonus! If you get finished early, see if you can refactor your code using async and await to make it more readable.