D3 Lab - Week 2

Instructions

Watch the D3 - Part 2 - Dealing with Data playlist on the Media Gallery

***Alert*:** This assignment assumes that you completed and passed in the D3 week 1 assignment, as this assignment will build on the final solution of that assignment. If you didn't finish it, please go back to and complete the assignment before starting this one.

***Important*:** *In the provided code, CHANGE ONLY THE TODOs SECTION, It is important that you KEEP THE OTHER PARTS OF THE CODE UNCHANGED, and do not change names of functions, CSS classes or element IDs.*

Goal

In the previous assignment, we created HTML and CSS basis of our system. In this week we will load the data we need.

Initial Setup

1 - Adding D3.js to the page

Before we start, let's add D3.js to out page. You will use the index.html file you submitted as a solution for last assignment. The file should contain only HTML and CSS code right now. We will add two <script> tags.

The first one loads D3.js and it should be added right before the closing tag </head> you already have in the file.

```
...
<script src="d3.js"></script> <!-- Add this line to your file -->

</head>
...
```

For this tag to work, we have to have the file named d3.js in the same folder of our index.html file. Download the d3.js file using this link, extract it, and save the copy the d3.js file to the same folder of index.html.

2 - Adding a <script> tag for our code

The second <script> tag should go right after </body> and it will be the place where we will add all the code from this assignment, as well as the code for the future assignments.

```
...
</body>
 <!-- Add this section to your file -->
 <script>
    // All your javascript code will go here
 </script>
</html>
...
```

3 - Creating a variable to store our data

Often, in our code, we need to be able to access the data loaded at the beginning of the application. To make this easier, we will add a global variable that will allow us to store any data we may need. Remember, any code from now on should be added to your file inside the <script> tag.

```
let store = {}
```

Loading Data

Every time we have a functionality to implement, we will implement it as a function. To load the data, we will implement the function called `loadData` that will be responsible for loading all data we need.

If you haven't done it yet, download the file below, and save it in the same folder you have your index.html file with the name routes.csv.

routes.csv

Once the file is in the folder, we can write our function as bellow, you will need to complete the places indicated with TODO at the beginning of the comments in order to the function to work.

Copy this code to your index.html file, and complete the TODOs

```
function loadData() {
```

```
    let promise =//TODO 1: Add the code to load the CSV
file named "routes.csv" | 1 Line
    return promise.then(routes => {
        store.routes = //TODO 2: Save the routes into our
store variable;
        return store;
    })
}
```

Preparing The Data

The dataset we are loading contains one line for each route, but what we want to visualize is the list of airlines and we just need the number of routes for each airline. For this reason, we need to create a function that takes as input a list of routes and group these routes by airline. we will call this function "*groupByAirline*".

The function will return a list of objects, where each object represents one airline and contains the following fields:

- AirlineID: The id of the airline
- AirlineName: The name of the airline
- Count: The number of routes the given airline has.

Add this function to your file and complete the TODOs

```
function groupByAirline(data) {
    //Iterate over each route, producing a dictionary where
the keys is are the ailines ids and the values are the
information of the airline.
    let result = data.reduce((result, d) => {
        let currentData = result[d.AirlineID] || {
            "AirlineID": d.AirlineID,
            "AirlineName": d.AirlineName,
            "Count": 0
        }

        currentData.Count += //TODO: Increment the count
(number of routes) of ariline.

        result[d.AirlineID] = //TODO: Save the updated
information in the dictionary using the airline id as key.
```

```
        return result;
    }, {})

    //We use this to convert the dictionary produced by the
code above, into a list, that will make it easier to create
the visualization.
    result = Object.keys(result).map(key => result[key])
    result = //TODO: Sort the data in descending order of
count.

    return result
}
```

Show Data

The last thing we want to do is to create our traditional *showData* function, that will take care of calling the functions responsible for drawing the data.

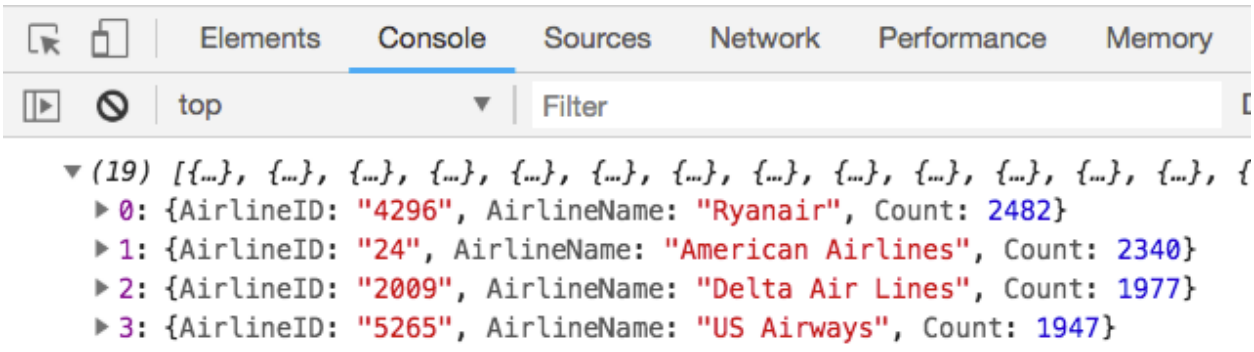
```
function showData() {
  //Get the routes from our store variable
  let routes = store.routes
  // Compute the number of routes per airline.
  let airlines = groupByAirline(store.routes);
  console.log(airlines)
}
```

Running it

We are now ready to run the code. Add the following line at the end of your <script> section to load the data and call *showData*

```
loadData().then(showData);
```

It won't change anything in the screen, but if you open the console (right-click anywhere in the page and select inspect, then the console tab) you should see the counts for the airlines.



Submit

Save your file as index.html and submit it. You don't need to submit the other files in your folder, just the index.html file. All your code should be in this file.