Overview

Your previous homework was simply a static page, with heights manually calculated and hard-coded. In this iteration, you'll make that page "come alive", by adding JavaScript which will construct the DOM when the page is loaded to display the data. You will also add another dimension to explore with the data, transforming your static graph into an interactive data visualization project. The data will be fetched either from an API, or from a static JSON file.

Example Solution

See below for an example solution:

`michaelpb.github.io/sols/hw2/`

1 Requirements

- **Must be deployed to GitHub Pages**
- Must be in a brand new, separate repo[1]
- Must use JavaScript to add DOM elements to the page, adjusting their size or location to match data
- Must retrieve the data from either a JSON file, or an API
- Must be able to browse or transform (at least) one more dimension to the data[2]
- JavaScript code should be in separate file[3]
- Must present the data in a clear and graphically pleasing way
- Must have a README.md that describes the data being used
- Must *NOT* use a pre-made charting library – you have to write your own!

1.1 Directory structure

The directory structure should look something like this:

```
- css/
    - reset.css                        (optional)
    - site.css
- img/
    -                                  (optional: any images)
- js/
    - main.js
- data/
    - data.json                        (optional: if not API)
- index.html
- README.md
```

[1]Going forward, we will be creating a new repo each time, to flesh out your GitHub profile. Do not name your GitHub repo "homework" or something like that. Instead, give it a relevant name about its contents. This is especially important for those seeking a career change: Your GitHub profile will look less professional if all your projects seem only related to homework.

[2]You must add some form elements such as a `<select>` drop-down, check-boxes, buttons, inputs, or anything else, to allow searching, limiting, or in some other way exploring or transforming your data.

[3]As opposed to embedded directly in the HTML page in `<script>` tags.

1.2 Soft requirements

While these are not hard requirements, extremely messy repos might be docked in grade. Keeping your code clean and clear will look the most professional.

- *Comenting:* Your code should have a light sprinkling of useful comments explaining what you are doing.[4]
- *Commented out code:* While its fine (and encouraged!) to comment out code while debugging, generally speaking "commented out" code should be deleted before submission.
- *Variable and function names:* Your variables, functions, and classes if applicable should have concise and descriptive names. Avoid single letter names.
- *console.log:* Extraneous `console.log`s should be removed before submission.
- The topic and data presented must be useful
- Must be at least mildly responsive[5]

2 Submission

1. Include a link to both the code view of your repo, and the published GitHub page
2. Include a screenshot of it working locally on your computer

3 Tips

3.1 Develop using a static server

You should avoid developing directly by accessing the HTML file via the `file://` protocol — that is, the default when double clicking on the file and seeing it in your browser. The reason is that you will run into problems when trying to load data using `fetch`.[6]

Instead, you should run a testing static server. We suggest two options for test servers: Pick either a Node.js based one which you will have to download and install, or a Python-based one. [7]

These test servers will run a little web-server that will serve up that directory. Then, in your browser, you will be able to go to `localhost:8000` or `localhost:8080`. Leave this running while you develop.

3.1.1 Python-based static test server

Python3 should already be installed on your system. To run the test server, run the following command in your homework directory:

```
python3 -m http.server 8000
```

[4]Good comments are written in the present tense, imperative mood, omitting most articles, as though you were giving instructions to the computer what it is doing. For example, "Fetch currency data from API and then display" is a good comment.

[5]That is to say, it shouldn't be stuck in one set of dimensions, but be flexible when the window is resized

[6]Browsers by default forbid doing "fetch" when viewing a local file as a security precaution.

[7]The Node.js one is preferred. While the Python server is much easier to use since it is probably already pre-installed, some students have reported frustrations with the Python one being occasionally glitchy, with it occasionally failing, or caching data and not fully refreshing.

3.1.2 Node.js static test server

If you do not have NPM already installed for class (first covered in 3.1), then see *Appendix: Installing and Configuring Node and NPM*. Once Node & NPM are installed, you can then install `node-static`:

```
npm install -g node-static
```

Once `node-static` is installed, start it with:

```
static
```

3.1.3 Gotcha: Force refreshing

Sometimes while testing, even if you hit the refresh button, your browser doesn't truly "refresh" and load changes you make to JavaScript or other files, even if you hit the refresh button. If you suspect that is the case, use `Ctrl+Shift+R` or `Command+Shift+R` to force Chrome(ium) to reload everything.

3.2 Creating a JS file

Your JavaScript code must be in a separate `main.js` file, as described in the file structure mentioned above.

```
<script src="js/main.js"></script>
```

- Your CSS can remain mostly unchanged from the previous exercise
- You will have to also change your HTML significantly, notably removing the "hard-coded" bars, and adding things like IDs or unique classes to give the JavaScript "easy access"

3.3 Picking data source

- **API:** If your data source is a frontend friendly API[8], such as the Exchange Rates API `exchangeratesapi.io` used in the example, then you may wish to go down the API route.
- **JSON:** For all other cases, your data should be stored in a separate JSON file. This makes it easily "swappable" with an API later on.[9]

3.4 Creating a JSON file

If your data source is not an API, you should store your data in a (at least one) separate JSON file. A JSON file is a data file-format that looks just like JavaScript objects & arrays (aka Python dictionaries & lists).

[8]This would mean APIs that do not require API secrets, and also are CORS-friendly ("cross-origin request"), such as the Exchange Rates. The reason being browsers forbid suspicious requests made by JS. Only if the Cross Origin header is sent by the server that is serving up the API (not in your control) then will browsers allow JS to send requests to remote servers.

[9]You may even want to go down the JSON route if you are having difficulty with fetching the API due to, for example, a Cross-Origin Request related error.

3.4.1 Creating your JSON file

If the data you have is not already in JSON format, you have a few options:

1. Creating the JSON file by hand (easiest for very small data-sets). You can do this in your preferred text editor (just be careful of syntax), or with an online tool, such as jsoneditoronline.org
2. If your data is already in CSV format (such as from a spreadsheet), you can convert it to JSON format with an online converter, such as this: `convertcsv.com/csv-to-json.htm`

3.4.2 How to start

A good first step is start by hardcoding your data in your source code, for example:

```
let data = {
    "test": "data"
};
```

Once you confirm the rest of the application is working, then move that out of your source code and store it in a separate JSON formatted file. Then, use `fetch` as described below to retrieve it.

3.4.3 Syntax Gotchas

- Only use double quotes: `{"test": "data"}` is okay, not `{test: "data"}`
- Be careful of trailing commas: `{"test": "data", }` is not valid JSON

3.5 Accessing the data source

To load a JSON file, you can do something like this:

```
fetch("./data/data.json")
    .then(response => response.json())
    .then(data => {
        console.log("Got the data!");
        console.log(data);

        // TODO: Call a function to do something with this data.
    });
```

To use an API, you will have to replace the top line with something like:

```
fetch("https://api.exchangeratesapi.io/latest")
```

3.6 Creating the graph

- You will likely need to write some sort of "render" function that will refresh the graph based on the data received.[10]
- You will likely have to loop through the data, creating a new element in each iteration of the loop.
- The two main forms of DOM manipulation we learned in class for plain JavaScript use either `document.createElement` and `innerHTML` with templating. Read below for more tips on these two methods.

3.6.1 DOM Manipulation Tip: Using document.createElement

One route might be to use `document.createElement` function to create the bar chart elements. You might do something like this:

```
let chart = document.querySelector("#chart-location");
let height = 70;
let bar = document.createElement("div");
bar.classList.add("Bar");
bar.style.height = height + "%";
chart.appendChild(bar);
```

- `developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style`

3.6.2 DOM Manipulation Tip: Using innerHTML and template literals

A different route might be using the innerHTML property to set HTML, which works well with template literals. JavaScript ES6 supports multi-line strings using the "backtick" character (on your keyboard to the left of the `1`) This syntax also supports insertion of variables into it, for a simple templating syntax.

`developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals`

```
let chart = document.querySelector("#chart-location");
let height = 70;
chart.innerHTML += `
    <div style="height: ${height}%">
        Test
    </div>
`;
```

3.7 Adding another dimension to the data

Once you have the graph rendering with JavaScript, the main next requirement to tackle is to have at least one other dimension to explore with your data.[11]

You must add some form elements such as a `<select>` drop-down, check-boxes, buttons, inputs, or anything else, to allow searching, limiting, or in some other way exploring or transforming your data.

[10]For practice with this, try out the activities that have a "render" function and create the DOM, such as the ToDo list or shopping list activities.

[11]For the example solution it sports a select showing which base currency to use, and a set of check boxes for which currencies are enabled.

1. If you are using an API, and your API already supports this, then it would be a matter of passing these as arguments to the API.
2. If you are using JSON data, or your API does not support this:
   - One route to write code to filter or process the data in your JavaScript code
     - This could be done via a new for-loop[12], or within existing for-loops you already created.
     - For exaple, selecting relevant bits based on user input, or transforming it in some way
     - For example, reporting average values, or sums or totals, might be useful in certain circumstances.
   - Another route would be to make multiple JSON files for different aspects to your data[13]

3.8 How to start

- Blank page syndrome? Try starting with code from in-class activities.
- As with anything complicated, be sure to start with pseudocode.
- Remember to always start with `console.log`.

Broadly, your code will have to involve:

1. Getting the parent element to each individual bar using `document.querySelector`
2. Fetching data from API or JSON
3. Assuming you are using a Bar Chart: Looping through the data fetched, creating each bar using `document.createElement` or `.innerHTML` (see above)

4 Bonus - Two dimensions

If you want to go above and beyond, add yet another extra dimension to explore with your graph, one that interacts in some way with your existing one.

[12]The syntax is tricky, but you might also consider `map` or `filter` here.

[13]As an example, if your other dimension is geographical such as "city" or "country", then you could generate a separate JSON file for each city in the dataset.

Appendix: Installing and Configuring Node & NPM

Background: Node and NPM are application that we will use later for developing React.js based applications. This homework assignment **does not use React.js**, but instead is in “plain vanilla JS”, however, Node & NPM do have a convenient test-server to use during development.

4.1 Installing Node and NPM

Ensure you have Node and NPM by running each of the following commands:

```
node --version
npm --version
```

If you get “command not found”, that means you don’t have node or npm installed.

- Ubuntu Linux:

  ```
  sudo apt-get install nodejs
  ```

- macOS:

  ```
  brew install node
  ```

4.2 Configuring NPM for global installs

1. Make the local home directory (this will serve as a place to store NPM installed applications):

   ```
   mkdir ~/.local
   ```

2. Configure NPM:

   ```
   npm config set prefix ~/.local
   ```

3. Open ~/.bashrc (Ubuntu Linux) or ~/.profile (macOS) with your editor.

   - Ubuntu Linux: `gedit ~/.bashrc`
   - macOS: `open ~/.profile`

4. Add the following code at the very end of this file:

   ```
   PATH=~/.local/bin/:$PATH
   ```

5. Close and restart your terminal window.