Overview

Time to begin your Kickstart Frontend solo project! For this and subsequent homework assignments, you will be iteratively tackling data-visualization problems, culminating in publishing a React.js-based data visualization package for others to use on the NPM repositories.[1]

For this homework, you are to pick a data-set or API, and build a single static page using only HTML and CSS. This page will portray a chart that graphically reflects information found from a real data-set or API. No complicated JavaScript or logic coding yet: Purely CSS and HTML.[2] In subsequent homework assignments, you will use JavaScript to make it interactive, and actually respond to the input data in a useful way.

Example Solution

See below for an example solution:

`michaelpb.github.io/sols/`

1 Requirements

- **Must be deployed to GitHub Pages**
- Must be in a separate repo[3]
- Must graphically represent in a clear and pleasant a set of numeric values
- The numeric values must be derived from a real and factual data-set
- Must have a README.md that describes the data being used
- Must use CSS grid and/or flex-box for layout
- Must use CSS `transition` and/or `animation`, such as for a pleasantly gradual hover transition effect.
- Must not use Bootstrap or any other pre-made CSS libraries[4]
- Must have JavaScript onclick events to display an alert with further information when each bar is clicked
- Must use a separate CSS file to contain the style
- Must use at least one of the following concepts (preferably both):
  - BEM naming convention
  - CSS Variables

[1] This final package will be a great portfolio piece. If presented in a polished enough manner, you may even get some other coders to use your package. This sort of open source contribution, beside being a great ego-boost as you "pay it forward", also looks great on resumes and can function as a killer "code sample" when asked during the interview process.

[2] And a simple "onclick" event, see final section

[3] 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.

[4] CSS resets are okay, as is using online examples to learn and build your own CSS. It's only using CSS libraries such as Bootstrap that is prohibited.

1.1 Directory structure

The directory structure should look something like this:

```
- css/
    - reset.css                        (optional)
    - site.css
- img/
    -                                  (optional: any images)
- js/
    -                                  (optional: any future JS files)
- index.html
- README.md
```

1.2 Soft requirements

- The final graph must look pleasant with a nice color scheme
- The topic and data presented must be useful
- Use of CSS grid must be idiomatic. For example, *do not* use the grid to encode the data itself (e.g. 100x100 grid), and *do not* specify the row and column of every single-bar (in the case of a bar graph).
- Must have clean code with a sprinkling of concise comments[5]
- Must be at least mildly responsive[6]

2 Submission

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

3 Steps

3.1 Picking topic & chart type

For your Solo Project for Kickstart Frontend, you must pick a topic. You have two options: Either go with the example one (for which solutions and mockups will be provided), or pick some other topic. **You will continue to build your homeworks based on this topic, so pick well!**

[5] 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. Also, before submission, commented-out code should be removed.

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

3.1.1 Option 1: Currency exchange rate bar-chart

The example solutions will be done with currency exchange rates, using the *Exchange Rates API*. You are welcome to use this topic.

You can get the current exchange rates in JSON format just by visiting `https://api.exchangeratesapi.io/latest` – more information is available at the `exchangeratesapi.io`.

3.1.2 Option 2: Custom Topic & Chart

Alternatively, you can spend the next weeks working on a custom type of chart that is not about currencies. If you go down this route, ensure first you have access to the data of your choice, either via an API, or a static CSV or JSON. A valid topic should have **real, useful things being compared**, and some customizability, such as **the potential for other dimensions to be later included**, e.g. for the currency chart, time. In general, it should be **at least as complicated or realistic as the currency conversion example topic**.

Some example of valid topics:

- Weather graph comparing weather at different cities
- Comparing countries and demographics or statistics (population, Gini coefficient, etc)
- Word count in public domain novels[7]

It's recommended to go with a bar chart. However, if you want to push yourself, feel free to pick any data visualization that is *at least as complicated* as a bar chart. For example, a scatter plot, while more difficult, is doable using absolute positioning. Make sure you can re-create this chart using CSS and HTML (and/or SVGs)! You might find the following Wikipedia article useful:

`en.wikipedia.org/wiki/Data_visualization#Examples_of_diagrams_used_for_data_visualization`

3.1.3 Documenting your topic and bar chart type choice

Once you have made your decision, even if that decision is to go with the "default" of currency exchange rates, then write in the README.md at least a few sentences describing the following:

1. The data you are visualizing. Include any links or credit necessary.
2. The type of chart you are choosing to visualize that data.

3.2 Draw wireframe, pick color scheme

1. As with any task in front-end web development, we must always start with a wireframe. Sketch out, loosely, what your chart should look like in the end. This is essential, as you will be "building toward" this when working on your HTML and CSS.
2. Pick a color scheme. Consider using `colourlovers.com` or `colorhunt.co` for ideas!

[7] Might need to manually gather this data, such as checking word counts at Project Gutenberg.

3.3 Creating the graph

1. Start with an empty or mostly empty HTML file. Link in a CSS file. For example, if you are correctly following the directory structure mentioned at the top, you would include a link in your header like `<link rel="stylesheet" href="./css/style.css" />`.
2. Using HTML and CSS, build your graph or chart. Start with the general layout before tackling the data visualization itself. For this first homework, don't worry about representing all the data, just pick a handful of real data, and correctly represent that (see example solution).

3.3.1 Hint

To make your graphical chart reflect the numbers, consider adding style attributes in your HTML. Within these style attributes, you can set the `height` to create different size elements, thereby creating the "bars" in the bar chart.[8] The rest of the CSS styling, however, should be in a separate file. Structuring your HTML and CSS this way will make it easier as we move to JavaScript and React.js.

For a concrete example, here is a portion of the code used to make the example solution:

```
<div class="BarChart-bar" style="height: 88.5%">EUR</div>
<div class="BarChart-bar" style="height: 75.5%">USD</div>
<div class="BarChart-bar" style="height: 55.9%">AUD</div>
```

The height calculations for this example were the result of a calculation turning the ratios of Euros to the other currencies into a percentage.[9]

3.4 CSS Tips

- You might be tempted to create a grid with, for example, 100 rows to depict percentage height of bars in your bar graph. Don't: It's much simpler and better just to use the suggested `height` CSS property `height: 30px` or `height: 30%` (or for a horizontal bar chart, using `width`).
- It's okay just to use Grid for a minor amount of the layout – some graphs might not have a very complicated Grid layout.
- Absolute positioning might be useful, also, for more complicated graphs, or for labels to be on top of the bars, or for your visualization to have a legend.
- Consider using pseudo-selectors to add striping or even more complicated coloration patterns to your chart.
- Tick-marks can get frustrating to do right. Only try adding tick-marks when you have everything else working.
- To receive full-credit, you must use the CSS `transition` property, and/or `animation`. This is not the same as simply having a `:hover` pseudo-classes. [10]

[8] If you are doing a horizontal bar chart, you might want to use `width` instead. Similarly, if you visualizing data with a scatter plot, you would use instead `position: absolute` with `top` and `bottom` to make the visualization reflect the data.

[9] To turn a decimal number into a percentage, multiply by 100. If there are some that are greater than `1.0`, then you may want to use a number smaller than 100 to turn it into a bar chart. For example, if the largest number is 1.2, you could make that "fit" into a vertical bar chart by using `100 / 1.2 = 83` to ensure all `height` values stay between `0-100%` and no bars "overflow" their container.

[10] You'll probably want to also use a `:hover` pseudo-class to activate that transition or animation, however.

3.5 Following CSS best practices

To get full credit for this assignment, you must follow good CSS practices:

- You must use cutting-edge CSS: Use CSS flexbox layout and/or CSS grid layout somewhere in your application.
- You must research and do at least one of the following cutting-edge CSS best practices: **CSS Variables** and/or **BEM naming convention**
- Think about responsive design: Your graph should at least be somewhat flexible or legible on smaller screens. You may want to use `@media` queries to do this.

3.5.1 CSS Variables

CSS Variables let you DRY out your CSS by putting values you reuse into variables defined at the top of your CSS file, and then re-using those variables throughout other rules. It's great for keeping your color scheme separate from the rest of your file.

As an example:

```
:root {
    --bg-color: #FEAAFE;
    --page-spacing: 50px;
}

.MainContent {
    background-color: var(--bg-color);
    margin-left: var(--page-spacing);
}

.NavBar {
    padding: var(--page-spacing);
    border-color: var(--bg-color);
}
```

Read more here:

`developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_variables`

3.5.2 BEM naming convention

With the advent of React and other sophisticated apps, developers quickly found that chaotically creating and using thousands of CSS classes and IDs made CSS specificity very hard to figure out, and easy to introduce design regressions. Thus, they came up with rules to keep naming more clear and consistent. The most popular is BEM ("Block Element Modifier").

In BEM, we ONLY use classes to style, and we think of 3 different classes: "Block" (highest level, we divide our page into these), "Element" (something more minor within the Blocks), and "Modifier" (for styling variants of Blocks or Elements).

```
.NavBar { }           /* Refers to the highest level ("Block" in BEM) page element */
.NavBar-navLink { }   /* Refers to something less important inside of NavBar ("Element" in BEM)*/
.NavBar--sticky { }   /* Refers to a varient of NavBar ("Modifier" in BEM)*/
```

You have a couple of options to read more on BEM:

1. BEM-based SUIT naming convention (most popular these days, shown above, and used in solutions and activities):

`github.com/suitcss/suit/blob/master/doc/naming-conventions.md`

2. The original BEM naming documentation (best explanation, has diagrams):

`getbem.com/introduction/`

3.6 Adding a simple JavaScript interaction

To receive full credit, you must add a basic JavaScript interaction. All that's necessary here is making parts of your graph interact with an `onclick` event, and display an alert.

3.6.1 Hint

```
<div
    onclick="alert('EUR costs 0.88 pounds')"
    class="BarChart-bar"
    style="height: 88.5%">EUR</div>
```

4 Bonus Assignment: SVG

- **NOTE:** Since there is very no class material supporting the use of SVGs, only attempt this Bonus Assignment if you have time and want the challenge.

This is not required to use, but SVG is typically used for creating more complicated and graphically impressive charts. With SVG, you can create line charts, and more. If you chose a more complicated chart type, you might want to try using SVG instead of traditional HTML and CSS.

4.1 SVG overview

- *Scalable Vector Graphics* (SVG) is a vector image format[11] that can be embedded in HTML documents, and is now supported by all modern browsers
- SVG looks just like HTML and can blend nearly seamlessly within HTML, and even can be styled with CSS just like HTML
- SVG has a wealth of tags for making shapes and lines[12]
- Important to us, SVG plays nicely with JavaScript and React

4.2 Learning SVG Resources

- MDN has a good tutorial here: `developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/SVG_and_CSS`
- A very thorough book and tutorial is free online here: `svgpocketguide.com/book/`

4.3 Using SVG

The best way to get started with SVG is to try an SVG editor such as Inkscape[13], a web-based one like Vectr (`vectr.com`), or a proprietary vector editor like Adobe Illustrator.

1. Using your editor, draw an example of the graph you want, including all lines, shapes, etc.
2. Export to a SVG file.[14]
3. Now, open up the SVG file and see if you can figure out how to tinker with the SVG properties and cause the SVG to change. Can you create a correlation of the SVG appearance with the code?
4. Strip away tags and styling that you think is not useful. Delete or clean up anything that doesn't affect the look of it, until it is only the shapes you need.
5. Using the tutorials above as a guide, embed your cleaned-up SVG into your HTML, then use CSS to re-add or tweak styling.

[11] Vector image formats store lines and shapes as opposed to pixels, and thus can be scaled at any size

[12] Any modern complicated, data visualization you see on the web today was likely accomplished using SVG instead of HTML. HTML is intended for website layout, and thus is mostly limited to squares. Using SVG, any type of shape or line is easily available.

[13] Inkscape is by far the best for our purposes, and free software. It's easy to install in Linux (`sudo apt install inkscape`) but unfortunately a bit clunkier to install on macOS.

[14] If your editor gives you an option, opt for a plain, non-compressed SVG. Use a "simplify path" or "smooth path" features if you are using hand-drawn shapes.