Data Visualization Notes

Patrick Adams

November 1, 2020

Note: This is a draft copy of notes generated by free code camp.
https://www.freecodecamp.org/

Contents

1 Data Visualization With D3

1.1 Add Document Elements with D3

D3 has several methods that let you add and change elements in your document.
The select() method selects one element from the document. It takes an argument for the name of the element you want and returns an HTML node for the first element in the document that matches the name. Here's an example:
const anchor = d3.select("a");
The above example finds the first anchor tag on the page and saves an HTML node for it in the variable anchor. You can use the selection with other methods. The "d3" part of the example is a reference to the D3 object, which is how you access D3 methods.
Two other useful methods are append() and text().
The append() method takes an argument for the element you want to add to the document. It appends an HTML node to a selected item, and returns a handle to that node.
The text() method either sets the text of the selected node, or gets the current text. To set the value, you pass a string as an argument inside the parentheses of the method.
Here's an example that selects an unordered list, appends a list item, and adds text:
“‘js
d3.select("ul")
.append("li")
.text("Very important item");
“‘
D3 allows you to chain several methods together with periods to perform a number of actions in a row.

1.2 Select a Group of Elements with D3

D3 also has the selectAll() method to select a group of elements. It returns an array of HTML nodes for all the items in the document that match the input string. Here's an example to select all the anchor tags in a document:
const anchors = d3.selectAll("a");
Like the select() method, selectAll() supports method chaining, and you can use it with other methods.

1.3 Work with Data in D3

The D3 library focuses on a data-driven approach. When you have a set of data, you can apply D3 methods to display it on the page. Data comes in many formats, but this challenge uses a simple array of numbers.
The first step is to make D3 aware of the data. The data() method is used on a selection of DOM elements to attach the data to those elements. The data set is passed as an argument to the method.
A common workflow pattern is to create a new element in the document for each piece of data in the set. D3 has the enter() method for this purpose.
When enter() is combined with the data() method, it looks at the selected elements from the page and compares them to the number of data items in the set. If there are fewer elements than data items, it creates the missing elements.
Here is an example that selects a ul element and creates a new list item based on the number of entries in the array:
“‘html

“‘
It may seem confusing to select elements that don't exist yet. This code is telling D3 to first select the ul on the page. Next, select all list items, which returns an empty selection. Then the data() method reviews the dataset and runs the following code three times, once for each item in the array. The enter() method sees there are no li elements on the page, but it needs 3 (one for each piece of data in dataset). New li elements

are appended to the ul and have the text "New item".

1.4 Work with Dynamic Data in D3

The last two challenges cover the basics of displaying data dynamically with D3 using the data() and enter() methods. These methods take a data set and, together with the append() method, create a new DOM element for each entry in the data set.
In the previous challenge, you created a new h2 element for each item in the dataset array, but they all contained the same text, "New Title". This is because you have not made use of the data that is bound to each of the h2 elements.
The D3 text() method can take a string or a callback function as an argument:
selection.text((d) => d)
In the example above, the parameter d refers to a single entry in the dataset that a selection is bound to.
Using the current example as context, the first h2 element is bound to 12, the second h2 element is bound to 31, the third h2 element is bound to 22, and so on.

1.5 Add Inline Styling to Elements

D3 lets you add inline CSS styles on dynamic elements with the style() method.
The style() method takes a comma-separated key-value pair as an argument. Here's an example to set the selection's text color to blue:
selection.style("color","blue");

1.6 Change Styles Based on Data

D3 is about visualization and presentation of data. It's likely you'll want to change the styling of elements based on the data. You can use a callback function in the style() method to change the styling for different elements.
For example, you may want to color a data point blue if it has a value less than 20, and red otherwise. You can use a callback function in the style() method and include the conditional logic. The callback function uses the d parameter to represent the data point:
“‘js
selection.style("color", (d) => {
/* Logic that returns the color based on a condition */
});
“‘
The style() method is not limited to setting the color - it can be used with other CSS properties as well.

1.7 Add Classes with D3

Using a lot of inline styles on HTML elements gets hard to manage, even for smaller apps. It's easier to add a class to elements and style that class one time using CSS rules. D3 has the attr() method to add any HTML attribute to an element, including a class name.
The attr() method works the same way that style() does. It takes comma-separated values, and can use a callback function. Here's an example to add a class of "container" to a selection:
selection.attr("class", "container");
Note that the "class" parameter will remain the same whenever you need to add a class and only the "container" parameter will change.

1.8 Update the Height of an Element Dynamically

The previous challenges covered how to display data from an array and how to add CSS classes. You can combine these lessons to create a simple bar chart. There are two steps to this:
1) Create a div for each data point in the array
2) Give each div a dynamic height, using a callback function in the style() method that sets height equal to the data value
Recall the format to set a style using a callback function:
selection.style("cssProperty", (d) => d)

1.9 Change the Presentation of a Bar Chart

The last challenge created a bar chart, but there are a couple of formatting changes that could improve it:
1) Add space between each bar to visually separate them, which is done by adding a margin to the CSS for the bar class
2) Increase the height of the bars to better show the difference in values, which is done by multiplying the value by a number to scale the height

1.10 Learn About SVG in D3

SVG stands for Scalable Vector Graphics.
Here "scalable" means that, if you zoom in or out on an object, it would not appear pixelated. It scales with the display system, whether it's on a small mobile screen or a large TV monitor.
SVG is used to make common geometric shapes. Since D3 maps data into a visual representation, it uses SVG to create the shapes for the visualization. SVG shapes for a web page must go within an HTML svg tag.
CSS can be scalable when styles use relative units (such as vh, vw, or percentages), but using SVG is more flexible to build data visualizations.

1.11 Display Shapes with SVG

The last challenge created an svg element with a given width and height, which was visible because it had a background-color applied to it in the style tag. The code made space for the given width and height.
The next step is to create a shape to put in the svg area. There are a number of supported shapes in SVG, such as rectangles and circles. They are used to display data. For example, a rectangle (<rect>) SVG shape could create a bar in a bar chart.
When you place a shape into the svg area, you can specify where it goes with x and y coordinates. The origin point of (0, 0) is in the upper-left corner. Positive values for x push the shape to the right, and positive values for y push the shape down from the origin point.
To place a shape in the middle of the 500 (width) x 100 (height) svg from last challenge, the x coordinate would be 250 and the y coordinate would be 50.
An SVG rect has four attributes. There are the x and y coordinates for where it is placed in the svg area. It also has a height and width to specify the size.

1.12 Create a Bar for Each Data Point in the Set

The last challenge added only one rectangle to the svg element to represent a bar. Here, you'll combine what you've learned so far about data(), enter(), and SVG shapes to create and append a rectangle for each data point in dataset.
A previous challenge showed the format for how to create and append a div for each item in dataset:
“‘js

```
d3.select("body").selectAll("div")
.data(dataset)
.enter()
.append("div")
```

There are a few differences working with rect elements instead of divs. The rects must be appended to an svg element, not directly to the body. Also, you need to tell D3 where to place each rect within the svg area. The bar placement will be covered in the next challenge.

1.13 Dynamically Set the Coordinates for Each Bar

The last challenge created and appended a rectangle to the svg element for each point in dataset to represent a bar. Unfortunately, they were all stacked on top of each other.
The placement of a rectangle is handled by the x and y attributes. They tell D3 where to start drawing the shape in the svg area. The last challenge set them each to 0, so every bar was placed in the upper-left corner. For a bar chart, all of the bars should sit on the same vertical level, which means the y value stays the same (at 0) for all bars. The x value, however, needs to change as you add new bars. Remember that larger x values push items farther to the right. As you go through the array elements in dataset, the x value should increase.
The attr() method in D3 accepts a callback function to dynamically set that attribute. The callback function takes two arguments, one for the data point itself (usually d) and one for the index of the data point in the array. The second argument for the index is optional. Here's the format:

```js
selection.attr("property", (d, i) => {
/*
* d is the data point value
* i is the index of the data point in the array
*/
})
```

It's important to note that you do NOT need to write a for loop or use forEach() to iterate over the items in the data set. Recall that the data() method parses the data set, and any method that's chained after data() is run once for each item in the data set.

1.14 Dynamically Change the Height of Each Bar

The height of each bar can be set to the value of the data point in the array, similar to how the x value was set dynamically.

```js
selection.attr("property", (d, i) => {
/*
* d is the data point value
* i is the index of the data point in the array
*/
})
```

1.15 Invert SVG Elements

You may have noticed the bar chart looked like it's upside-down, or inverted. This is because of how SVG uses (x, y) coordinates.
In SVG, the origin point for the coordinates is in the upper-left corner. An x coordinate of 0 places a shape

on the left edge of the SVG area. A y coordinate of 0 places a shape on the top edge of the SVG area. Higher x values push the rectangle to the right. Higher y values push the rectangle down.
To make the bars right-side-up, you need to change the way the y coordinate is calculated. It needs to account for both the height of the bar and the total height of the SVG area.
The height of the SVG area is 100. If you have a data point of 0 in the set, you would want the bar to start at the bottom of the SVG area (not the top). To do this, the y coordinate needs a value of 100. If the data point value were 1, you would start with a y coordinate of 100 to set the bar at the bottom. Then you need to account for the height of the bar of 1, so the final y coordinate would be 99.
The y coordinate that is y = heightOfSVG - heightOfBar would place the bars right-side-up.

1.16 Change the Color of an SVG Element

The bars are in the right position, but they are all the same black color. SVG has a way to change the color of the bars.
In SVG, a rect shape is colored with the fill attribute. It supports hex codes, color names, and rgb values, as well as more complex options like gradients and transparency.

1.17 Add Labels to D3 Elements

D3 lets you label a graph element, such as a bar, using the SVG text element.
Like the rect element, a text element needs to have x and y attributes, to place it on the SVG canvas. It also needs to access the data to display those values.
D3 gives you a high level of control over how you label your bars.

1.18 Style D3 Labels

D3 methods can add styles to the bar labels. The fill attribute sets the color of the text for a text node. The style() method sets CSS rules for other styles, such as "font-family" or "font-size".

1.19 Add a Hover Effect to a D3 Element

It's possible to add effects that highlight a bar when the user hovers over it with the mouse. So far, the styling for the rectangles is applied with the built-in D3 and SVG methods, but you can use CSS as well.
You set the CSS class on the SVG elements with the attr() method. Then the :hover pseudo-class for your new class holds the style rules for any hover effects.

1.20 Add a Tooltip to a D3 Element

A tooltip shows more information about an item on a page when the user hovers over that item. There are several ways to add a tooltip to a visualization, this challenge uses the SVG title element.
title pairs with the text() method to dynamically add data to the bars.

1.21 Create a Scatterplot with SVG Circles

A scatter plot is another type of visualization. It usually uses circles to map data points, which have two values each. These values tie to the x and y axes, and are used to position the circle in the visualization.
SVG has a circle tag to create the circle shape. It works a lot like the rect elements you used for the bar chart.

1.22 Add Attributes to the Circle Elements

The last challenge created the circle elements for each point in the dataset, and appended them to the SVG canvas. But D3 needs more information about the position and size of each circle to display them correctly.
A circle in SVG has three main attributes. The cx and cy attributes are the coordinates. They tell D3 where to position the center of the shape on the SVG canvas. The radius (r attribute) gives the size of the circle.
Just like the rect y coordinate, the cy attribute for a circle is measured from the top of the SVG canvas, not from the bottom.
All three attributes can use a callback function to set their values dynamically. Remember that all methods chained after data(dataset) run once per item in dataset. The d parameter in the callback function refers to the current item in dataset, which is an array for each point. You use bracket notation, like d[0], to access the values in that array.

1.23 Add Labels to Scatter Plot Circles

You can add text to create labels for the points in a scatter plot.
The goal is to display the comma-separated values for the first (x) and second (y) fields of each item in dataset.
The text nodes need x and y attributes to position it on the SVG canvas. In this challenge, the y value (which determines height) can use the same value that the circle uses for its cy attribute. The x value can be slightly larger than the cx value of the circle, so the label is visible. This will push the label to the right of the plotted point.

1.24 Create a Linear Scale with D3

The bar and scatter plot charts both plotted data directly onto the SVG canvas. However, if the height of a bar or one of the data points were larger than the SVG height or width values, it would go outside the SVG area.
In D3, there are scales to help plot data. Scales are functions that tell the program how to map a set of raw data points onto the pixels of the SVG canvas.
For example, say you have a 100x500-sized SVG canvas and you want to plot Gross Domestic Product (GDP) for a number of countries. The set of numbers would be in the billion or trillion-dollar range. You provide D3 a type of scale to tell it how to place the large GDP values into that 100x500-sized area.
It's unlikely you would plot raw data as-is. Before plotting it, you set the scale for your entire data set, so that the x and y values fit your canvas width and height.
D3 has several scale types. For a linear scale (usually used with quantitative data), there is the D3 method scaleLinear():
const scale = d3.scaleLinear()
By default, a scale uses the identity relationship. The value of the input is the same as the value of the output. A separate challenge covers how to change this.

1.25 Set a Domain and a Range on a Scale

By default, scales use the identity relationship. This means the input value maps to the output value. However, scales can be much more flexible and interesting.
Say a dataset has values ranging from 50 to 480. This is the input information for a scale, also known as the domain.
You want to map those points along the x axis on the SVG canvas, between 10 units and 500 units. This is the output information, also known as the range.
The domain() and range() methods set these values for the scale. Both methods take an array of at least two elements as an argument. Here's an example:
"'js

// Set a domain
// The domain covers the set of input values
scale.domain([50, 480]);
// Set a range
// The range covers the set of output values
scale.range([10, 500]);
scale(50) // Returns 10
scale(480) // Returns 500
scale(325) // Returns 323.37
scale(750) // Returns 807.67
d3.scaleLinear()
‘‘‘

Notice that the scale uses the linear relationship between the domain and range values to figure out what the output should be for a given number. The minimum value in the domain (50) maps to the minimum value (10) in the range.

1.26 Use the d3.max and d3.min Functions to Find Minimum and Maximum Values in a Dataset

The D3 methods domain() and range() set that information for your scale based on the data. There are a couple methods to make that easier.
Often when you set the domain, you'll want to use the minimum and maximum values within the data set. Trying to find these values manually, especially in a large data set, may cause errors.
D3 has two methods - min() and max() to return this information. Here's an example:
“‘js
const exampleData = [34, 234, 73, 90, 6, 52];
d3.min(exampleData) // Returns 6
d3.max(exampleData) // Returns 234
‘‘‘

A dataset may have nested arrays, like the [x, y] coordinate pairs that were in the scatter plot example. In that case, you need to tell D3 how to calculate the maximum and minimum.
Fortunately, both the min() and max() methods take a callback function.
In this example, the callback function's argument d is for the current inner array. The callback needs to return the element from the inner array (the x or y value) over which you want to compute the maximum or minimum. Here's an example for how to find the min and max values with an array of arrays:
“‘js
const locationData = [[1, 7],[6, 3],[8, 3]];
// Returns the smallest number out of the first elements
const minX = d3.min(locationData, (d) => d[0]);
// minX compared 1, 6, and 8 and is set to 1
‘‘‘

1.27 Use Dynamic Scales

The D3 min() and max() methods are useful to help set the scale.
Given a complex data set, one priority is to set the scale so the visualization fits the SVG container's width and height. You want all the data plotted inside the SVG canvas so it's visible on the web page.
The example below sets the x-axis scale for scatter plot data. The domain() method passes information to the scale about the raw data values for the plot. The range() method gives it information about the actual space on the web page for the visualization.
In the example, the domain goes from 0 to the maximum in the set. It uses the max() method with a callback function based on the x values in the arrays. The range uses the SVG canvas' width (w), but it includes

some padding, too. This puts space between the scatter plot dots and the edge of the SVG canvas.
“‘js
const dataset = [
[34, 78],
[109, 280],
[310, 120],
[79, 411],
[420, 220],
[233, 145],
[333, 96],
[222, 333],
[78, 320],
[21, 123]
];
const w = 500;
const h = 500;
// Padding between the SVG canvas boundary and the plot
const padding = 30;
const xScale = d3.scaleLinear()
.domain([0, d3.max(dataset, (d) => d[0])])
.range([padding, w - padding]);
“‘

The padding may be confusing at first. Picture the x-axis as a horizontal line from 0 to 500 (the width value for the SVG canvas). Including the padding in the range() method forces the plot to start at 30 along that line (instead of 0), and end at 470 (instead of 500).

1.28 Use a Pre-Defined Scale to Place Elements

With the scales set up, it’s time to map the scatter plot again. The scales are like processing functions that turn the x and y raw data into values that fit and render correctly on the SVG canvas. They keep the data within the screen’s plotting area.
You set the coordinate attribute values for an SVG shape with the scaling function. This includes x and y attributes for rect or text elements, or cx and cy for circles. Here’s an example:
“‘js
shape
.attr("x", (d) => xScale(d[0]))
“‘

Scales set shape coordinate attributes to place the data points onto the SVG canvas. You don’t need to apply scales when you display the actual data value, for example, in the text() method for a tooltip or label.

1.29 Add Axes to a Visualization

Another way to improve the scatter plot is to add an x-axis and a y-axis.
D3 has two methods, axisLeft() and axisBottom(), to render the y-axis and x-axis, respectively. Here’s an example to create the x-axis based on the xScale in the previous challenges:
“‘js
const xAxis = d3.axisBottom(xScale);
“‘

The next step is to render the axis on the SVG canvas. To do so, you can use a general SVG component, the g element. The g stands for group.
Unlike rect, circle, and text, an axis is just a straight line when it’s rendered. Because it is a simple shape, using g works.

The last step is to apply a transform attribute to position the axis on the SVG canvas in the right place. Otherwise, the line would render along the border of SVG canvas and wouldn't be visible.
SVG supports different types of transforms, but positioning an axis needs translate. When it's applied to the g element, it moves the whole group over and down by the given amounts. Here's an example:
“‘js
const xAxis = d3.axisBottom(xScale);
svg.append("g")
.attr("transform", "translate(0, " + (h - padding) + ")")
.call(xAxis);
“‘
The above code places the x-axis at the bottom of the SVG canvas. Then it's passed as an argument to the call() method.
The y-axis works in the same way, except the translate argument is in the form (x, 0). Because translate is a string in the attr() method above, you can use concatenation to include variable values for its arguments.

2 Json Apis And Ajax

2.1 Handle Click Events with JavaScript using the onclick property

You want your code to execute only once your page has finished loading. For that purpose, you can attach a JavaScript event to the document called DOMContentLoaded. Here's the code that does this:
“‘js
document.addEventListener('DOMContentLoaded', function() {
});
“‘
You can implement event handlers that go inside of the DOMContentLoaded function. You can implement an onclick event handler which triggers when the user clicks on the element with id getMessage, by adding the following code:
“‘js
document.getElementById('getMessage').onclick = function(){};
“‘

2.2 Change Text with click Events

When the click event happens, you can use JavaScript to update an HTML element.
For example, when a user clicks the "Get Message" button, it changes the text of the element with the class message to say "Here is the message".
This works by adding the following code within the click event:
document.getElementsByClassName('message')[0].textContent="Here is the message";

2.3 Get JSON with the JavaScript XMLHttpRequest Method

You can also request data from an external source. This is where APIs come into play.
Remember that APIs - or Application Programming Interfaces - are tools that computers use to communicate with one another. You'll learn how to update HTML with the data we get from APIs using a technology called AJAX.
Most web APIs transfer data in a format called JSON. JSON stands for JavaScript Object Notation.
JSON syntax looks very similar to JavaScript object literal notation. JSON has object properties and their current values, sandwiched between a { and a }.
These properties and their values are often referred to as "key-value pairs".
However, JSON transmitted by APIs are sent as bytes, and your application receives it as a string. These can be converted into JavaScript objects, but they are not JavaScript objects by default. The JSON.parse method parses the string and constructs the JavaScript object described by it.
You can request the JSON from freeCodeCamp's Cat Photo API. Here's the code you can put in your click event to do this:
“‘js
const req = new XMLHttpRequest();
req.open("GET",'/json/cats.json',true);
req.send();
req.onload = function(){
const json = JSON.parse(req.responseText);
document.getElementsByClassName('message')[0].innerHTML = JSON.stringify(json);
};
“‘
Here's a review of what each piece is doing. The JavaScript XMLHttpRequest object has a number of properties and methods that are used to transfer data. First, an instance of the XMLHttpRequest object is created and saved in the req variable.
Next, the open method initializes a request - this example is requesting data from an API, therefore is a

"GET" request. The second argument for open is the URL of the API you are requesting data from. The third argument is a Boolean value where true makes it an asynchronous request.
The send method sends the request. Finally, the onload event handler parses the returned data and applies the JSON.stringify method to convert the JavaScript object into a string. This string is then inserted as the message text.

2.4 Get JSON with the JavaScript fetch method

Another way to request external data is to use the JavaScript fetch() method. It is equivalent to XMLHttpRequest, but the syntax is considered easier to understand.
Here is the code for making a GET request to /json/cats.json
“‘js
fetch('/json/cats.json')
.then(response => response.json())
.then(data => {
document.getElementById('message').innerHTML = JSON.stringify(data);
})
“‘
Take a look at each piece of this code.
The first line is the one that makes the request. So, fetch(URL) makes a GET request to the URL specified. The method returns a Promise.
After a Promise is returned, if the request was successful, the then method is executed, which takes the response and converts it to JSON format.
The then method also returns a Promise, which is handled by the next then method. The argument in the second then is the JSON object you are looking for!
Now, it selects the element that will receive the data by using document.getElementById(). Then it modifies the HTML code of the element by inserting a string created from the JSON object returned from the request.

2.5 Access the JSON Data from an API

In the previous challenge, you saw how to get JSON data from the freeCodeCamp Cat Photo API.
Now you'll take a closer look at the returned data to better understand the JSON format. Recall some notation in JavaScript:
[ ] -> Square brackets represent an array{ } -> Curly brackets represent an object" " -> Double quotes represent a string. They are also used for key names in JSON
Understanding the structure of the data that an API returns is important because it influences how you retrieve the values you need.
On the right, click the "Get Message" button to load the freeCodeCamp Cat Photo API JSON into the HTML.
The first and last character you see in the JSON data are square brackets [ ]. This means that the returned data is an array. The second character in the JSON data is a curly { bracket, which starts an object. Looking closely, you can see that there are three separate objects. The JSON data is an array of three objects, where each object contains information about a cat photo.
You learned earlier that objects contain "key-value pairs" that are separated by commas. In the Cat Photo example, the first object has "id":0 where "id" is a key and 0 is its corresponding value. Similarly, there are keys for "imageLink", "altText", and "codeNames". Each cat photo object has these same keys, but with different values.
Another interesting "key-value pair" in the first object is "codeNames":["Juggernaut","Mrs. Wallace","ButterCup"]. Here "codeNames" is the key and its value is an array of three strings. It's possible to have arrays of objects as well as a key with an array as a value.
Remember how to access data in arrays and objects. Arrays use bracket notation to access a specific index of an item. Objects use either bracket or dot notation to access the value of a given property. Here's an

example that prints the "altText" of the first cat photo - note that the parsed JSON data in the editor is saved in a variable called json:
“‘js
console.log(json[0].altText);
// Prints "A white cat wearing a green helmet shaped melon on its head."
“‘

2.6 Convert JSON Data to HTML

Now that you're getting data from a JSON API, you can display it in the HTML.
You can use a forEach method to loop through the data since the cat photo objects are held in an array. As you get to each item, you can modify the HTML elements.
First, declare an html variable with let html = "";.
Then, loop through the JSON, adding HTML to the variable that wraps the key names in strong tags, followed by the value. When the loop is finished, you render it.
Here's the code that does this:
“‘js
let html = "";
json.forEach(function(val) {
const keys = Object.keys(val);
html += "";
keys.forEach(function(key) {
html += "" + key + ": " + val[key] + "";
});
html += "";
});
“‘
Note: For this challenge, you need to add new HTML elements to the page, so you cannot rely on ‘textContent‘. Instead, you need to use ‘innerHTML‘, which can make a site vulnerable to Cross-site scripting attacks.

2.7 Render Images from Data Sources

The last few challenges showed that each object in the JSON array contains an imageLink key with a value that is the URL of a cat's image.
When you're looping through these objects, you can use this imageLink property to display this image in an img element.
Here's the code that does this:
html += "<img src = '" + val.imageLink + "' " + "alt='" + val.altText + "'>";

2.8 Pre-filter JSON to Get the Data You Need

If you don't want to render every cat photo you get from the freeCodeCamp Cat Photo API, you can pre-filter the JSON before looping through it.
Given that the JSON data is stored in an array, you can use the filter method to filter out the cat whose "id" key has a value of 1.
Here's the code to do this:
“‘js
json = json.filter(function(val) {
return (val.id !== 1);
});
“‘

2.9 Get Geolocation Data to Find A User's GPS Coordinates

Another cool thing you can do is access your user's current location. Every browser has a built in navigator that can give you this information.
The navigator will get the user's current longitude and latitude.
You will see a prompt to allow or block this site from knowing your current location. The challenge can be completed either way, as long as the code is correct.
By selecting allow, you will see the text on the output phone change to your latitude and longitude.
Here's code that does this:
“‘js
if (navigator.geolocation){
navigator.geolocation.getCurrentPosition(function(position) {
document.getElementById('data').innerHTML="latitude: " + position.coords.latitude + "longitude: " + position.coords.longitude;
});
}
“‘
First, it checks if the navigator.geolocation object exists. If it does, the getCurrentPosition method on that object is called, which initiates an asynchronous request for the user's position. If the request is successful, the callback function in the method runs. This function accesses the position object's values for latitude and longitude using dot notation and updates the HTML.

2.10 Post Data with the JavaScript XMLHttpRequest Method

In the previous examples, you received data from an external resource. You can also send data to an external resource, as long as that resource supports AJAX requests and you know the URL.
JavaScript's XMLHttpRequest method is also used to post data to a server. Here's an example:
“‘js
const xhr = new XMLHttpRequest();
xhr.open('POST', url, true);
xhr.setRequestHeader('Content-Type', 'application/json; charset=UTF-8');
xhr.onreadystatechange = function () {
if (xhr.readyState === 4 && xhr.status === 201){
const serverResponse = JSON.parse(xhr.response);
document.getElementsByClassName('message')[0].textContent = serverResponse.userName + serverResponse.suffix;
}
};
const body = JSON.stringify({ userName: userName, suffix: ' loves cats!' });
xhr.send(body);
“‘
You've seen several of these methods before. Here the open method initializes the request as a "POST" to the given URL of the external resource, and uses the true Boolean to make it asynchronous.
The setRequestHeader method sets the value of an HTTP request header, which contains information about the sender and the request. It must be called after the open method, but before the send method. The two parameters are the name of the header and the value to set as the body of that header.
Next, the onreadystatechange event listener handles a change in the state of the request. A readyState of 4 means the operation is complete, and a status of 201 means it was a successful request. The document's HTML can be updated.
Finally, the send method sends the request with the body value, which the userName key was given by the user in the input field.

3 Data Visualization Projects

3.1 Visualize Data with a Bar Chart

Objective: Build a CodePen.io app that is functionally similar to this: https://codepen.io/freeCodeCamp/full/GrZVaM.
Fulfill the below user stories and get all of the tests to pass. Give it your own personal style.
You can use HTML, JavaScript, CSS, and the D3 svg-based visualization library. The tests require axes to be generated using the D3 axis property, which automatically generates ticks along the axis. These ticks are required for passing the D3 tests because their positions are used to determine alignment of graphed elements. You will find information about generating axes at https://github.com/d3/d3/blob/master/API.md#axes-d3-axis. Required (non-virtual) DOM elements are queried on the moment of each test. If you use a frontend framework (like Vue for example), the test results may be inaccurate for dynamic content. We hope to accommodate them eventually, but these frameworks are not currently supported for D3 projects.
User Story #1: My chart should have a title with a corresponding id="title".
User Story #2: My chart should have a g element x-axis with a corresponding id="x-axis".
User Story #3: My chart should have a g element y-axis with a corresponding id="y-axis".
User Story #4: Both axes should contain multiple tick labels, each with the corresponding class="tick".
User Story #5: My chart should have a rect element for each data point with a corresponding class="bar" displaying the data.
User Story #6: Each bar should have the properties data-date and data-gdp containing date and GDP values.
User Story #7: The bar elements' data-date properties should match the order of the provided data.
User Story #8: The bar elements' data-gdp properties should match the order of the provided data.
User Story #9: Each bar element's height should accurately represent the data's corresponding GDP.
User Story #10: The data-date attribute and its corresponding bar element should align with the corresponding value on the x-axis.
User Story #11: The data-gdp attribute and its corresponding bar element should align with the corresponding value on the y-axis.
User Story #12: I can mouse over an area and see a tooltip with a corresponding id="tooltip" which displays more information about the area.
User Story #13: My tooltip should have a data-date property that corresponds to the data-date of the active area.
Here is the dataset you will need to complete this project: https://raw.githubusercontent.com/freeCodeCamp/ProjectReferenc
data.json
You can build your project by forking this CodePen pen. Or you can use this CDN link to run the tests in any environment you like: https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js.
Once you're done, submit the URL to your working project with all its tests passing.

3.2 Visualize Data with a Scatterplot Graph

Objective: Build a CodePen.io app that is functionally similar to this: https://codepen.io/freeCodeCamp/full/bgpXyK.
Fulfill the below user stories and get all of the tests to pass. Give it your own personal style.
You can use HTML, JavaScript, CSS, and the D3 svg-based visualization library. The tests require axes to be generated using the D3 axis property, which automatically generates ticks along the axis. These ticks are required for passing the D3 tests because their positions are used to determine alignment of graphed elements. You will find information about generating axes at https://github.com/d3/d3/blob/master/API.md#axes-d3-axis. Required (non-virtual) DOM elements are queried on the moment of each test. If you use a frontend framework (like Vue for example), the test results may be inaccurate for dynamic content. We hope to accommodate them eventually, but these frameworks are not currently supported for D3 projects.
User Story #1: I can see a title element that has a corresponding id="title".
User Story #2: I can see an x-axis that has a corresponding id="x-axis".
User Story #3: I can see a y-axis that has a corresponding id="y-axis".
User Story #4: I can see dots, that each have a class of dot, which represent the data being plotted.
User Story #5: Each dot should have the properties data-xvalue and data-yvalue containing their corresponding x and y values.

User Story #6: The data-xvalue and data-yvalue of each dot should be within the range of the actual data and in the correct data format. For data-xvalue, integers (full years) or Date objects are acceptable for test evaluation. For data-yvalue (minutes), use Date objects.
User Story #7: The data-xvalue and its corresponding dot should align with the corresponding point/value on the x-axis.
User Story #8: The data-yvalue and its corresponding dot should align with the corresponding point/value on the y-axis.
User Story #9: I can see multiple tick labels on the y-axis with %M:%S time format.
User Story #10: I can see multiple tick labels on the x-axis that show the year.
User Story #11: I can see that the range of the x-axis labels are within the range of the actual x-axis data.
User Story #12: I can see that the range of the y-axis labels are within the range of the actual y-axis data.
User Story #13: I can see a legend containing descriptive text that has id="legend".
User Story #14: I can mouse over an area and see a tooltip with a corresponding id="tooltip" which displays more information about the area.
User Story #15: My tooltip should have a data-year property that corresponds to the data-xvalue of the active area.
Here is the dataset you will need to complete this project: https://raw.githubusercontent.com/freeCodeCamp/ProjectReferenc
data.json
You can build your project by forking this CodePen pen. Or you can use this CDN link to run the tests in any environment you like: https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js
Once you're done, submit the URL to your working project with all its tests passing.

3.3 Visualize Data with a Heat Map

Objective: Build a CodePen.io app that is functionally similar to this: https://codepen.io/freeCodeCamp/full/JEXgeY.
Fulfill the below user stories and get all of the tests to pass. Give it your own personal style.
You can use HTML, JavaScript, CSS, and the D3 svg-based visualization library. Required (non-virtual) DOM elements are queried on the moment of each test. If you use a frontend framework (like Vue for example), the test results may be inaccurate for dynamic content. We hope to accommodate them eventually, but these frameworks are not currently supported for D3 projects.
User Story #1: My heat map should have a title with a corresponding id="title".
User Story #2: My heat map should have a description with a corresponding id="description".
User Story #3: My heat map should have an x-axis with a corresponding id="x-axis".
User Story #4: My heat map should have a y-axis with a corresponding id="y-axis".
User Story #5: My heat map should have rect elements with a class="cell" that represent the data.
User Story #6: There should be at least 4 different fill colors used for the cells.
User Story #7: Each cell will have the properties data-month, data-year, data-temp containing their corresponding month, year, and temperature values.
User Story #8: The data-month, data-year of each cell should be within the range of the data.
User Story #9: My heat map should have cells that align with the corresponding month on the y-axis.
User Story #10: My heat map should have cells that align with the corresponding year on the x-axis.
User Story #11: My heat map should have multiple tick labels on the y-axis with the full month name.
User Story #12: My heat map should have multiple tick labels on the x-axis with the years between 1754 and 2015.
User Story #13: My heat map should have a legend with a corresponding id="legend".
User Story #14: My legend should contain rect elements.
User Story #15: The rect elements in the legend should use at least 4 different fill colors.
User Story #16: I can mouse over an area and see a tooltip with a corresponding id="tooltip" which displays more information about the area.
User Story #17: My tooltip should have a data-year property that corresponds to the data-year of the active area.
Here is the dataset you will need to complete this project: https://raw.githubusercontent.com/freeCodeCamp/ProjectReferenc
temperature.json

You can build your project by forking this CodePen pen. Or you can use this CDN link to run the tests in any environment you like: https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js
Once you're done, submit the URL to your working project with all its tests passing.

3.4 Visualize Data with a Choropleth Map

Objective: Build a CodePen.io app that is functionally similar to this: https://codepen.io/freeCodeCamp/full/EZKqza.
Fulfill the below user stories and get all of the tests to pass. Give it your own personal style.
You can use HTML, JavaScript, CSS, and the D3 svg-based visualization library. Required (non-virtual) DOM elements are queried on the moment of each test. If you use a frontend framework (like Vue for example), the test results may be inaccurate for dynamic content. We hope to accommodate them eventually, but these frameworks are not currently supported for D3 projects.
User Story #1: My choropleth should have a title with a corresponding id="title".
User Story #2: My choropleth should have a description element with a corresponding id="description".
User Story #3: My choropleth should have counties with a corresponding class="county" that represent the data.
User Story #4: There should be at least 4 different fill colors used for the counties.
User Story #5: My counties should each have data-fips and data-education properties containing their corresponding fips and education values.
User Story #6: My choropleth should have a county for each provided data point.
User Story #7: The counties should have data-fips and data-education values that match the sample data.
User Story #8: My choropleth should have a legend with a corresponding id="legend".
User Story #9: There should be at least 4 different fill colors used for the legend.
User Story #10: I can mouse over an area and see a tooltip with a corresponding id="tooltip" which displays more information about the area.
User Story #11: My tooltip should have a data-education property that corresponds to the data-education of the active area.
Here are the datasets you will need to complete this project:US Education Data: https://cdn.freecodecamp.org/testable-projects-fcc/data/choropleth_map/for_user_education.jsonUS County Data: https://cdn.freecodecamp.org/testable-projects-fcc/data/choropleth_map/counties.json
You can build your project by forking this CodePen pen. Or you can use this CDN link to run the tests in any environment you like: https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js
Once you're done, submit the URL to your working project with all its tests passing.

3.5 Visualize Data with a Treemap Diagram

Objective: Build a CodePen.io app that is functionally similar to this: https://codepen.io/freeCodeCamp/full/KaNGNR.
Fulfill the below user stories and get all of the tests to pass. Give it your own personal style.
You can use HTML, JavaScript, CSS, and the D3 svg-based visualization library. The tests require axes to be generated using the D3 axis property, which automatically generates ticks along the axis. These ticks are required for passing the D3 tests because their positions are used to determine alignment of graphed elements. You will find information about generating axes at https://github.com/d3/d3/blob/master/API.md#axes-d3-axis. Required (non-virtual) DOM elements are queried on the moment of each test. If you use a frontend framework (like Vue for example), the test results may be inaccurate for dynamic content. We hope to accommodate them eventually, but these frameworks are not currently supported for D3 projects.
User Story #1: My tree map should have a title with a corresponding id="title".
User Story #2: My tree map should have a description with a corresponding id="description".
User Story #3: My tree map should have rect elements with a corresponding class="tile" that represent the data.
User Story #4: There should be at least 2 different fill colors used for the tiles.
User Story #5: Each tile should have the properties data-name, data-category, and data-value containing their corresponding name, category, and value.

User Story #6: The area of each tile should correspond to the data-value amount: tiles with a larger data-value should have a bigger area.
User Story #7: My tree map should have a legend with corresponding id="legend".
User Story #8: My legend should have rect elements with a corresponding class="legend-item".
User Story #9: The rect elements in the legend should use at least 2 different fill colors.
User Story #10: I can mouse over an area and see a tooltip with a corresponding id="tooltip" which displays more information about the area.
User Story #11: My tooltip should have a data-value property that corresponds to the data-value of the active area.
For this project you can use any of the following datasets:Kickstarter Pledges: https://cdn.freecodecamp.org/testable-projects-fcc/data/tree_map/kickstarter-funding-data.jsonMovie Sales: https://cdn.freecodecamp.org/testable-projects-fcc/data/tree_map/movie-data.jsonVideo Game Sales: https://cdn.freecodecamp.org/testable-projects-fcc/data/tree_map/video-game-sales-data.json
You can build your project by forking this CodePen pen. Or you can use this CDN link to run the tests in any environment you like: https://cdn.freecodecamp.org/testable-projects-fcc/v1/bundle.js
Once you're done, submit the URL to your working project with all its tests passing.