12.3.1

Inspect an API call with D3.json()

Roza is gaining proficiency with Plotly and with JavaScript. She can create various types of graphs in Plotly, as well as perform powerful data manipulation operations with JavaScript, such as filtering, sorting, and mapping.

The belly button data exists in JSON format. In order to be able to visualize it with Plotly, she must be able to read the data into her script file. She will use the `D3.js` library to do this. While D3 is primarily a data visualization library, its `d3.json()` method will allow Roza to read external JSON files, as well as place calls to external web APIs for data.

Roza will gain familiarity with using `d3.json()` by first placing an API call to SpaceX, the aerospace manufacturer specializing in space travel.

REWIND

JSON, JavaScript Object Notation, is a data format that sorts and presents data in the form of key-value pairs. It looks much like a Python dictionary and can be traversed through using dot notation.

Let's first take a look at Roza's `index.html` file:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>SpaceX API</title>
```

```
</head>
<body>
  <h1>Open the console!</h1>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/5.9.7/d3.min.js"></script>
  <script src="spaceX.js"></script>
</body>
</html>
```

Like Plotly, the `D3.js` library is downloaded from its CDN link, and loaded into the HTML file. The code Roza writes will be in `spaceX.js`.

Next, open `spaceX.js`.

```
const url = "https://api.spacexdata.com/v2/launchpads";

d3.json(url).then(receivedData => console.log(receivedData));
```

The actual API call to SpaceX is made in these two lines of code:

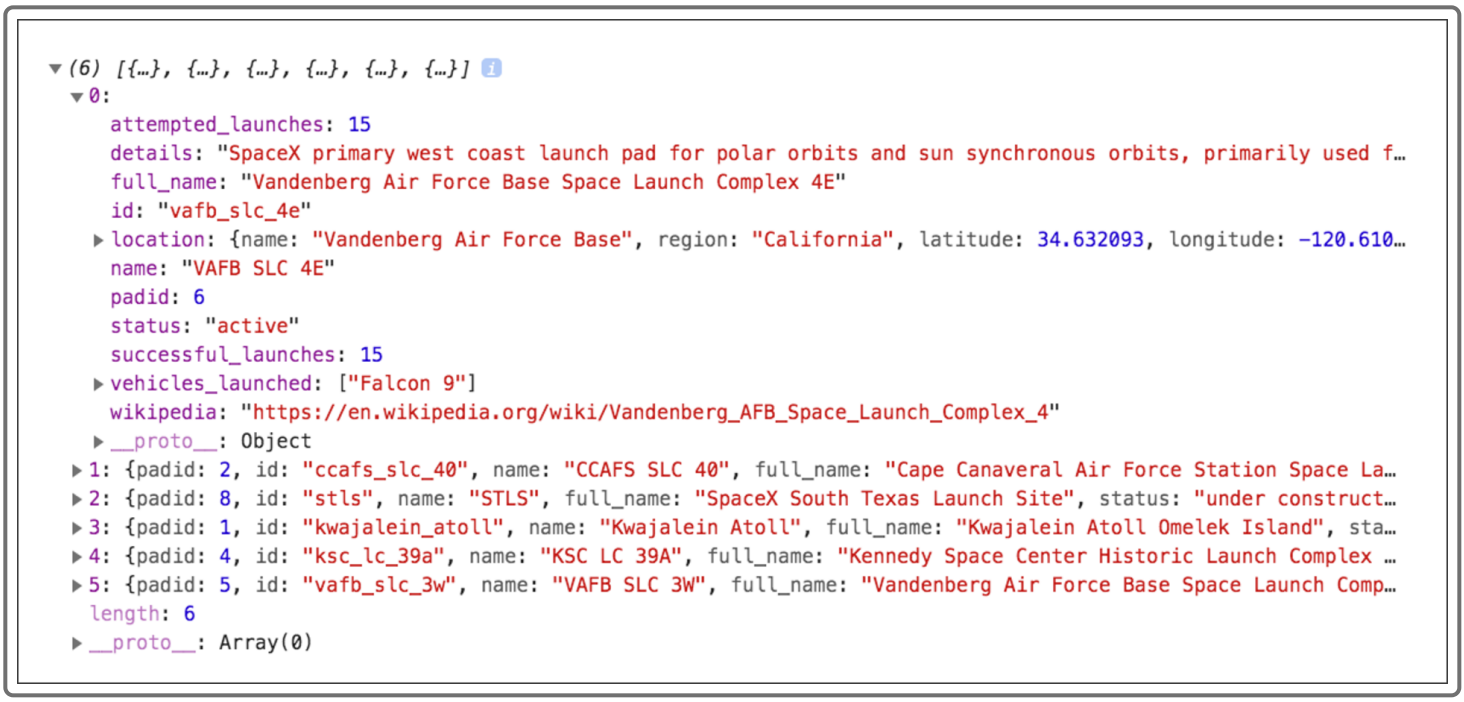
1. In the first line, the URL to the SpaceX is defined.
2. In the second line, `d3.json()` method places a call to SpaceX's API, retrieves the data, and prints it to the browser console.

The `d3.json()` returns a JavaScript promise: it places an API call to the URL and executes subsequent lines of code only when the API call is complete. Once the data is retrieved, it is assigned the arbitrary parameter name `receivedData` by the arrow function and printed in the console. The `d3.json().then() method` ensures that the data is received before executing the arrow function. In summary, a JavaScript promise in this case waits for the data retrieval to finish before moving on to the next code.

Retake

REWIND

JavaScript is an asynchronous language, meaning that a code statement doesn't necessarily wait for the previous statement to finish executing. When a statement involves a task that can take a relatively long time to complete, such as reading large files or retrieving data from an API, the next statement of code can begin executing before the previous task finishes. Using a promise with the `then()` method ensures that all the data requested from the API is received before executing the next part of code.



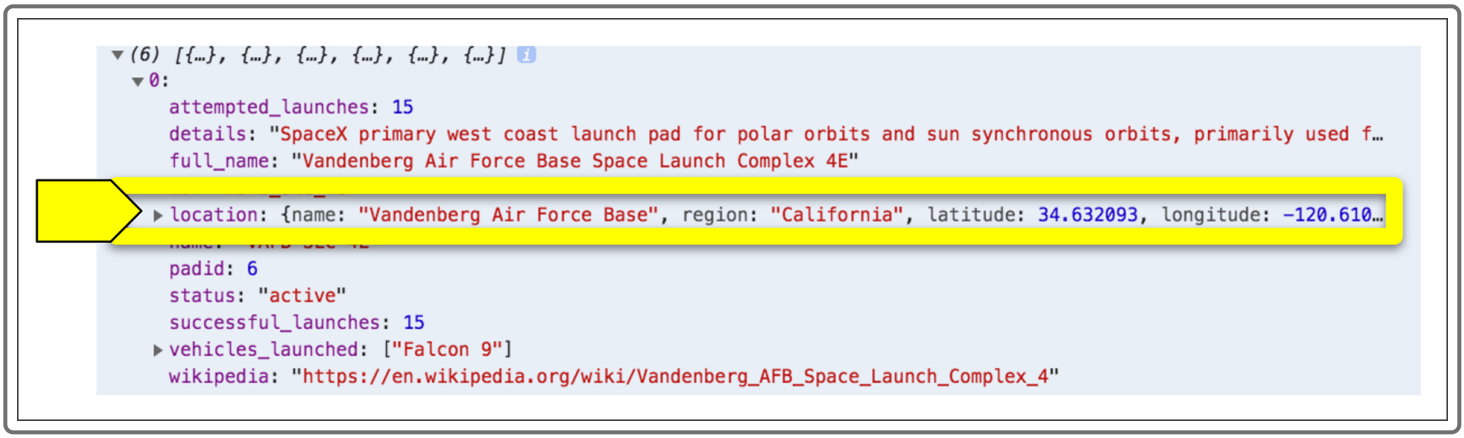
We can see that an array of six objects is returned. Each object contains details about a specific SpaceX launch site, such as the number of space vehicles that have been successfully launched from the site.

Retake

The details of each location, as you have just seen, are enclosed within a JavaScript object. Its properties can therefore be accessed with the dot notation. The code to retrieve the `full_name` of the Vandenberg Air Force Base would look like this:

```
d3.json(url).then(spaceXResults => console.log(spaceXResults[0].full_name));
```

Note also that the value for the location key is an object:



Retake

SKILL DRILL

Use `map()` to print only the latitude and longitude coordinates of each SpaceX launch station.