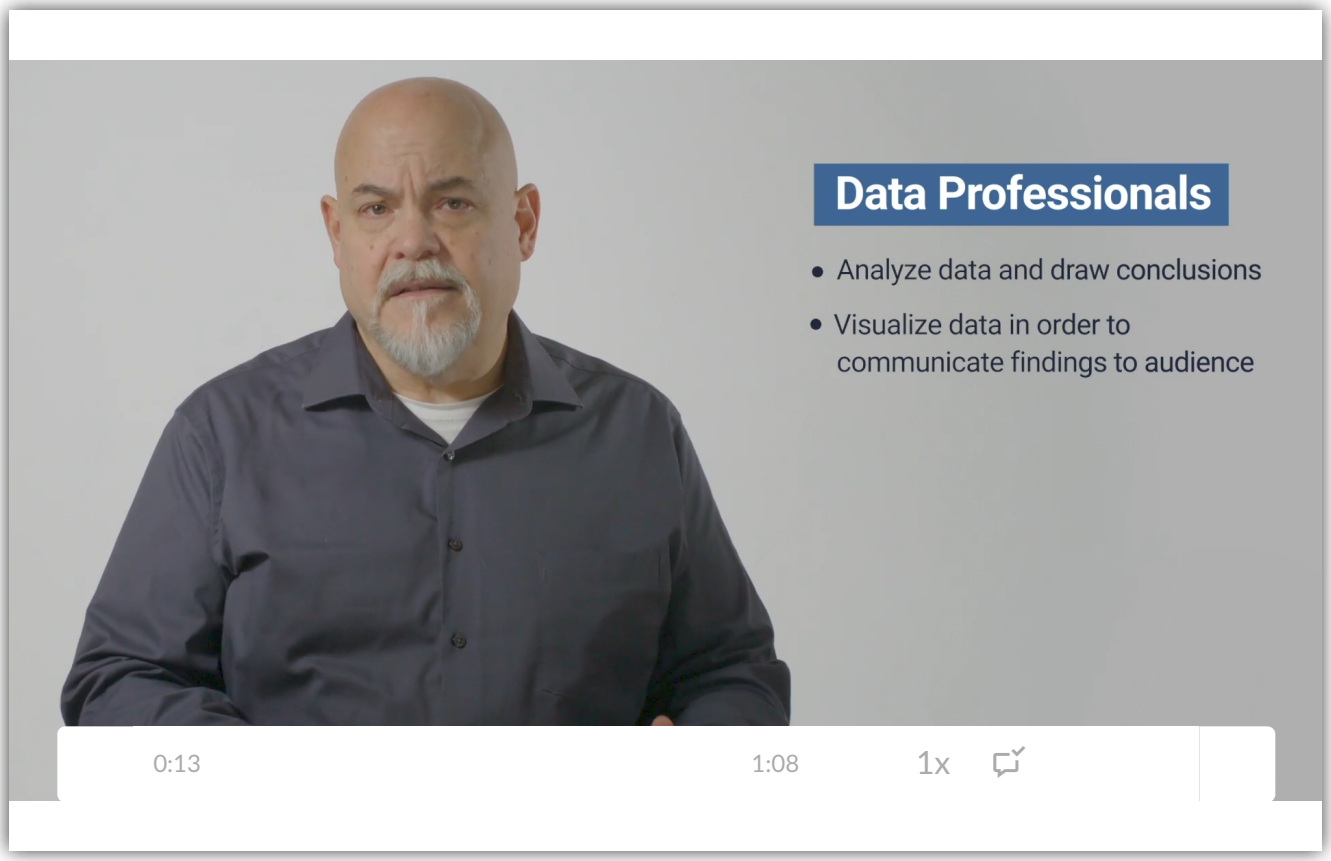
12.0.1: Data Visualization in JavaScript

Welcome to Module 12: Belly Button Biodiversity! In this video, we will explain the relationship between data analytics and visualization.



12.0.2: Module 12 Roadmap

Looking Ahead

In this module, you will use Plotly.js, a JavaScript data visualization library, to create an interactive data visualization for the web. The completed work will be displayed in a portfolio you create.

To be successful in this module, you'll need to be familiar with HTML and basic JavaScript.

What You Will Learn

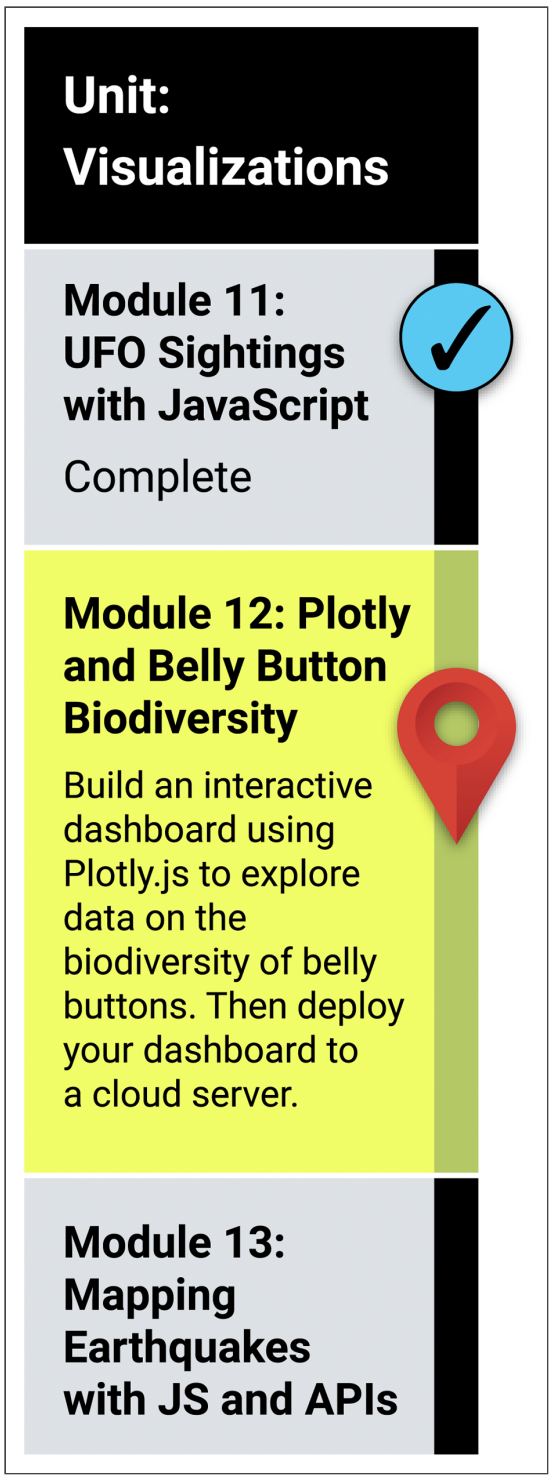
By the end of this module, you will be able to:

- Create basic plots with Plotly, including bar charts, line charts, and pie charts.
- Use D3.json() to fetch external data, such as CSV files and web APIs.
- Parse data in JSON format.
- Use functional programming in JavaScript to manipulate data.
- Use JavaScript's Math library to manipulate numbers.
- Use event handlers in JavaScript to add interactivity to a data visualization.
- How to use interactivity to enhance your visualizations.
- Deploy an interactive chart to GitHub Pages.

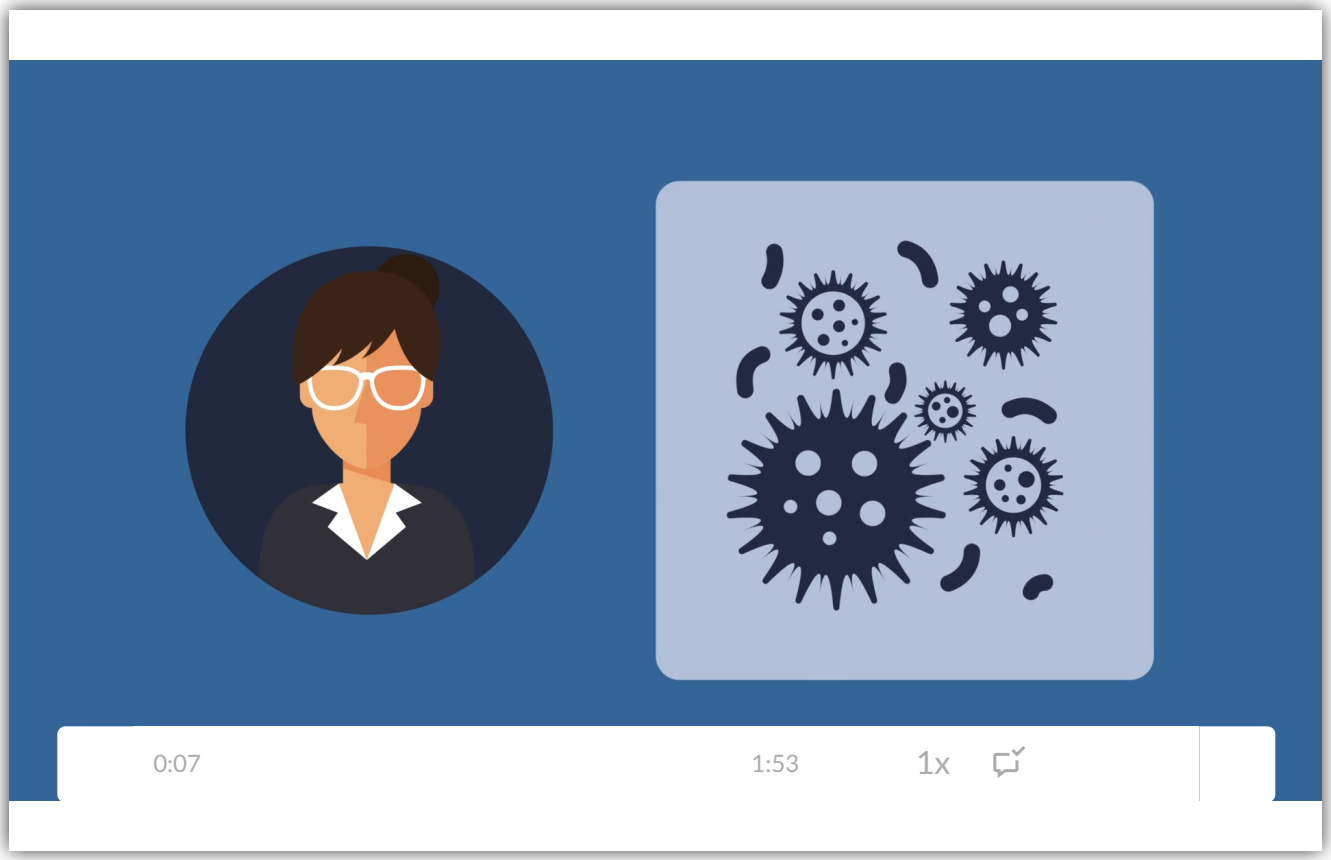
Planning Your Schedule

Here's a quick look at the lessons and assignments you'll cover in this module. You can use the time estimates to help pace your learning and plan your schedule.

- Introduction to Module 12 (15 mins)
- Getting Started with Plotly (1 hour)
- Transform Data with JavaScript (4 hours)
- Retrieve External Data (4 hours)
- JavaScript Events (3 hours)
- Deployment (15 mins)
- Application (5 hours)



12.0.3: Tools for Scraping



12.1.1: Organizing Your Tools

Roza is excited to get started. She now needs to take stock of the tools that she'll need to use to complete her project.

One of the advantages of programming in JavaScript is that the web browser is a tool both for development and debugging. So fortunately, Roza does not need to install any software to complete this project.

You are not required to install new tools in this module. Here is a rundown of what you will need:

- **VS Code:** You will use a text editor to create and edit HTML and JavaScript files.
- **Web browser:** You will use a web browser, such as Chrome, to view, inspect, and debug your visualizations. We will use Chrome in our examples.
- **Command-line interface:** You will use the command-line interface to run a local server. If you're a Mac user, this means Terminal. If you're a Windows user, you'll be using Git Bash.
- **GitHub:** As you will deploy your final data visualization on GitHub Pages, you will need your GitHub account.

12.1.2: Inspect a Plotly.js Chart

Roza is raring to start her project. She knows that she wants to create engaging and dynamic charts. It's her task to identify the best way to share her information with her audience.

Her first step is to master the basics of Plotly. For example, if she can create a basic chart in Plotly, she can build on her knowledge to create more advanced ones. Furthermore, she needs to identify the types of charts she can create with Plotly. Knowing the available options will enable her to select an appropriate visualization type for a dataset.

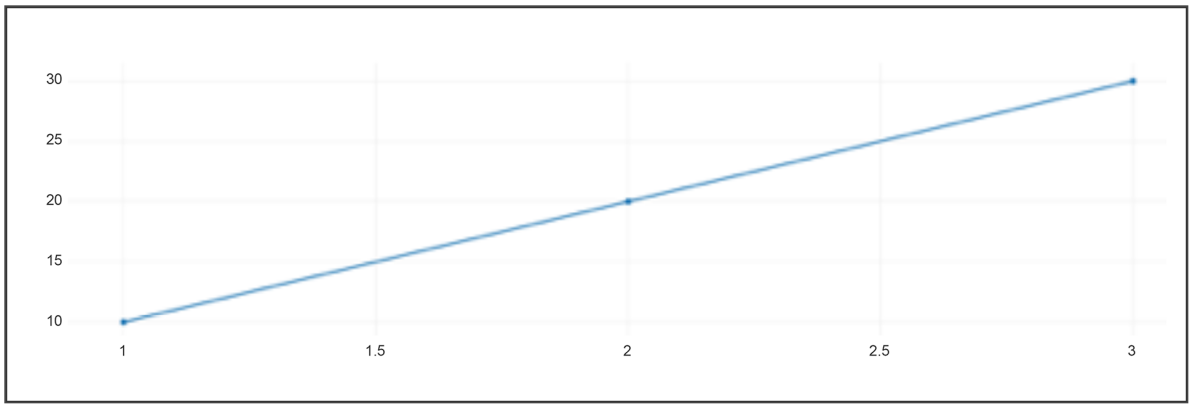
Throughout her journey, Roza will keep in mind how to best convey data to her volunteers and other researchers. For volunteers who are interested in selling their bacteria to Improbable Beef, what is the best way to visualize the types of bacteria that colonize their bellies? Some of them will be suitable for synthetic beef production, while others will not. Learning to use Plotly will be an important first step.

Let's begin!

To get started, download the data you'll need for this project. Click the following link to download `index.html` and `plot.js`.

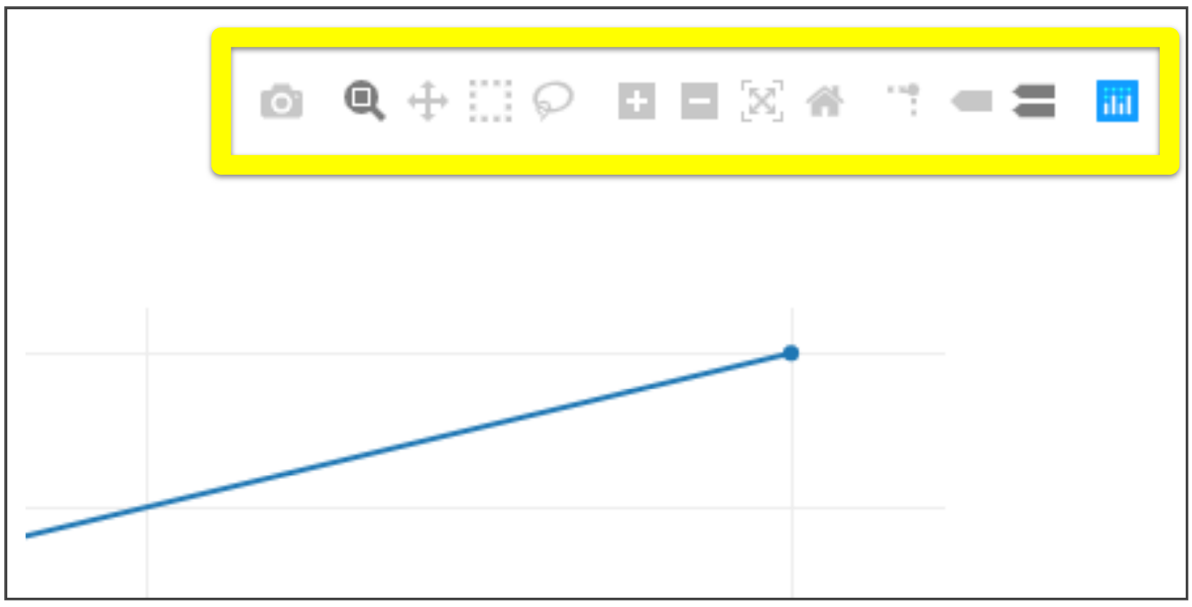
Download the data files
(https://courses.bootcampspot.com/courses/138/files/13732/download?wrap=1)

Open `index.html` in your browser. You should see the following chart:



This is a simple line chart with three data points. Notice that gridlines are supplied automatically.

Next, place your mouse over the top right corner of the chart, and you'll see the following menu appear. It has options such as zoom, zoom out, and pan. This menu also comes with the Plotly library and will be available for every visualization you create.



Next, open `index.html` in VS Code. You should see the following code.

```
<!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>Basic Charts</title>
   <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
<body>
   <div id="plotArea"></div>
   <script src="plots.js"></script>
</body>
</html>
```

Let's break the code down:

- The first `<script>` tag links to a file called `plotly-latest.min.js`, which is downloaded from an online host. This file contains the JavaScript code that makes up the Plotly library. Such a link is called a CDN.
- CDN stands for Content Delivery Network. In short, a CDN provides links to downloadable resources, such as Plotly library code. An alternative to using CDNs is to download the file and use it locally, but using this particular CDN ensures that the most up-to-date version of Plotly is always delivered.
- The `<div>` with the ID `plotArea` refers to the `div` in which the chart will be displayed. You can also give the `div` another `id` of your choice.
- The second `<script>` tag refers to the file named `plots.js,` which contains the JavaScript code.

Open `plots.js`. You will see the following line of code.

```
Plotly.newPlot("plotArea", [{x: [1,2,3], y:[10,20,30]}]);
```

A chart was created with a single line of code! The `Plotly.newPlot()` method creates a new chart, as its name indicates. This method has two arguments:

1. The first argument in Plotly, `newPlot()` is `"plotArea"`. Recall that this corresponds to the ID of the `<div>` tag in the HTML document.
2. The second argument is an array, as indicated by the square brackets. Inside the array is an object, as notated by the curly brackets, in which values of `x` and `y` are specified. The `x` and `y` values are contained inside arrays as well.

You want to create a line chart of Cartesian coordinates. The x-coordinates are 5, 10, 15 and 20. The y-coordinates are 3, 6, 9 and 12. How would the data be formatted for Plotly?

- `[{x: [5,10,15,20], y:[3,6,9,12]}]`
- `{x: [5,10,15,20], y:[3,6,9,12]}`
- `[x: [5,10,15,20], y:[3,6,9,12]]`

Check Answer

Finish ▸

IMPORTANT

The formatting matters in Plotly! Make sure that your data is enclosed inside an outer array.

In VS Code, create a JavaScript file named `plots.js`. In this file, write the code for a line chart with the following values. `X: 5,10,15. Y: 10,20,30.` The chart will be displayed in a `div` named `plotArea`. What will your code look like?

- `(“plotArea”, [{x:[5,10,15],y:[10,20,30]}]);`
- `Plotly.newPlot(“plotArea”, [{x:[5,10,15],y:[10,20,30]}]);`
- `Plotly.newPlot({x:[5,10,15],y:[10,20,30]});`

Check Answer

Finish ▸

Skill Drill

Open VS Code and use Plotly to create a line chart of your own.

Play around with the `div` ID, the JavaScript file name, and the data array. Then verify the results by opening `index.html` with your browser. You should see a chart. If the chart doesn’t render, try opening the developer/debugging console.

Make sure that you are comfortable with creating a line chart with Plotly before moving on.

12.1.3: Create a Bar Chart

Roza can now create a basic line chart in Plotly. She knows how to create a new graph, place it in an HTML document, and format the data for Plotly.

A line chart isn’t suitable for all data visualizations, however. For example, a line chart will generally do a better job of displaying trends, but a bar chart may be more appropriate to visualize how data is distributed across a number of categories.

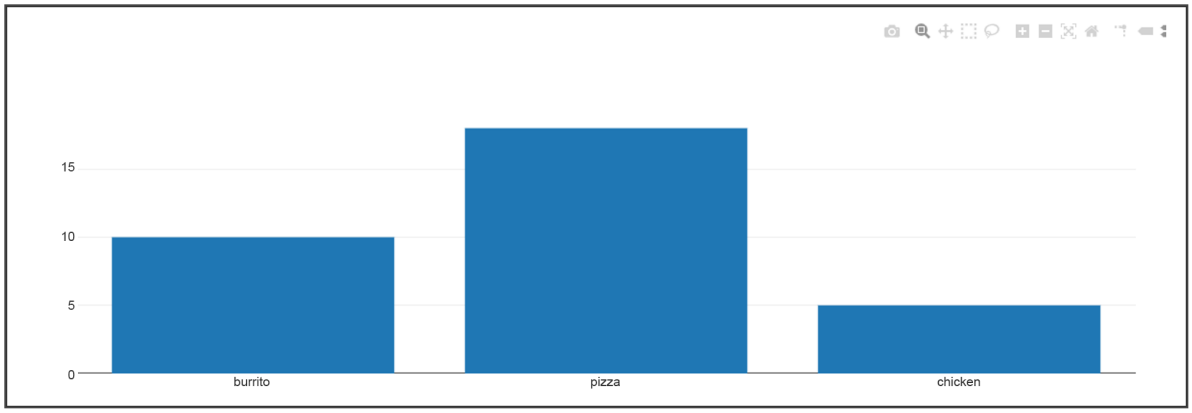
Roza has decided to build on her foundations and explore other types of graphs available in Plotly. She’ll begin with some other charts, such as a bar chart, a scatter plot, and a pie chart.

A chart is hard to decipher without appropriate labels. Roza will therefore also learn to customize other elements of her charts, such as axis labels and titles.

To help Roza improve her visualization skills using Plotly, we'll use an example of ordering food for a department luncheon.

The 33 people in the department are surveyed about their lunch preferences. They were asked to choose burritos, pizza, or chicken for the department lunch. 10 voted for burritos, 18 for pizza, and 5 for chicken. Let's help Roza plot the results.

> **Note**
> The terms *graph*, *chart*, and *plot* are used interchangeably here.



Now look at the code used to create this bar chart.

```
var trace = {
   x: ["burrito", "pizza", "chicken"],
   y: [10, 18, 5],
   type: "bar"
};
Plotly.newPlot("plotArea", [trace]);
```

There are several differences between the code seen here and the code from the first chart we created.

```
Plotly.newPlot("plotArea", [{x:[1,2,3], y:[10,20,30]}]);
```

- The data is no longer contained inside the `Plotly.newPlot()` method. The object that contains the `x` and `y` arrays is instead assigned to the variable "`trace`." This variable, `trace`, is the second argument of the `newPlot()`. Note, however, that the contents of the variable have been enclosed inside an array. The effect is still the same: an object contained inside an array.

- The `trace` object now specifies the chart as a bar chart with `type: "bar"`.

Note that it makes sense to assign the data to a variable, as it would be very unwieldy to place an entire dataset inside the `Plotly.newPlot()` function call.

Take a look at the following code. Will it run correctly in the browser?

Hint: Feel free to test this code in your browser.

```
var trace = [{
    x: ["burrito", "pizza", "chicken"],
    y: [10, 18, 5],
    type: "bar"
}];

Plotly.newPlot("plotArea", trace);
```

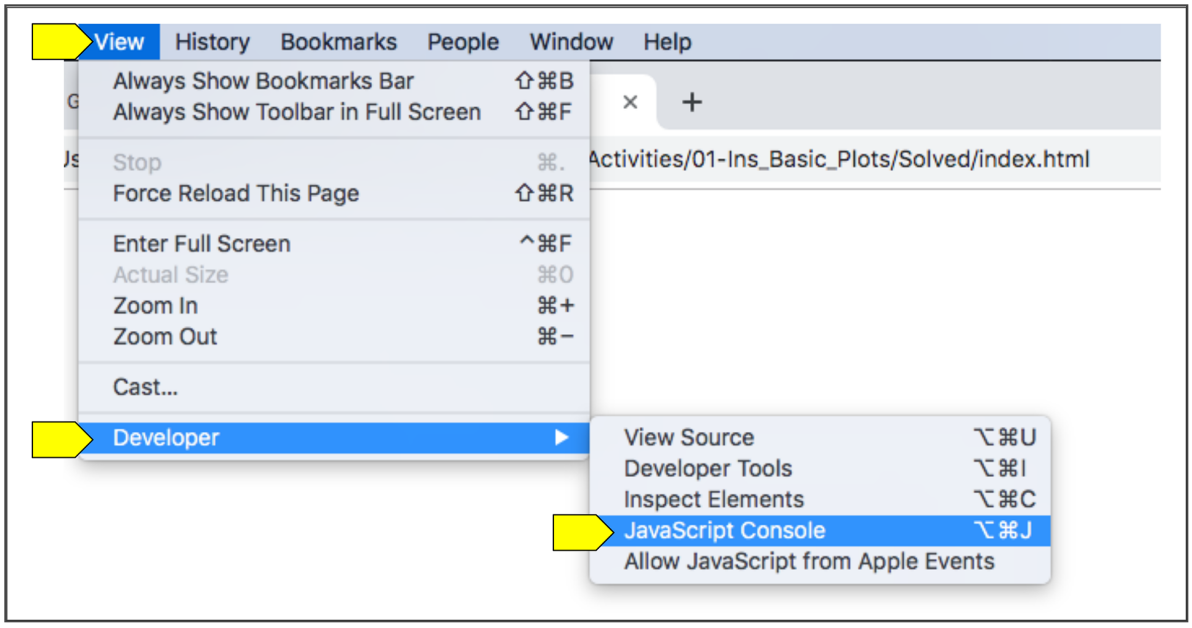
- ○ Yes. The data object is no longer enclosed in an array in the function call, but it is still enclosed inside an array in the variable assignment.
- ○ No. The data object is not enclosed in an array in the function call.
- ○ No. The data object uses incorrect key-value pairs.

Check Answer

Finish ▸

Troubleshooting

Every programmer runs into errors: troubleshooting is part of the job! If a plot doesn't render in the browser, try inspecting the browser console. In Chrome, you would click on View in the menu, followed by Developer, then JavaScript Console:



You code a chart in Plotly and your browser is blank. What might have caused this error?

- [ ] This is a simple error message due to a misspelling.
- [ ] `myPlot` does not exist in the HTML document.
- [ ] Either A or B could have caused this error.

Check Answer

Finish ▸

Choose Layout Options

So far, so good! But something is still missing. The chart would be vastly improved by including a title and axis labels. A good data visualization practice is to make graphs as clear as possible to viewers. Adding a title and labeling the axes will help them understand what they see. Let's first add a title to the graph.

Take a look at our code.

```
var trace = {
    x: ["burrito", "pizza", "chicken"],
    y: [10, 18, 5],
    type: "bar"
};

var layout = {
    title: "Luncheon Survey"
};

Plotly.newPlot("plotArea", [trace], layout);
```

What do you notice that's different?

1. There is an object assigned to the variable `layout`. It contains the key `title` and the value `"Luncheon Survey"`.
2. `Plotly.newPlot()` now has a third argument: `layout`, which refers to the object we just discussed.

What do the three arguments in the `Plotly.newPlot()` function call seen above refer to?

Argument #1: ____
Argument #2: ____
Argument #3: ____

- Argument #1, `"plotArea"`, refers to the HTML div that contains the chart.
  Argument #2, `[trace]`, refers to the array that contains the data object.
  Argument #3, `layout`, refers to the object that contains the layout details.

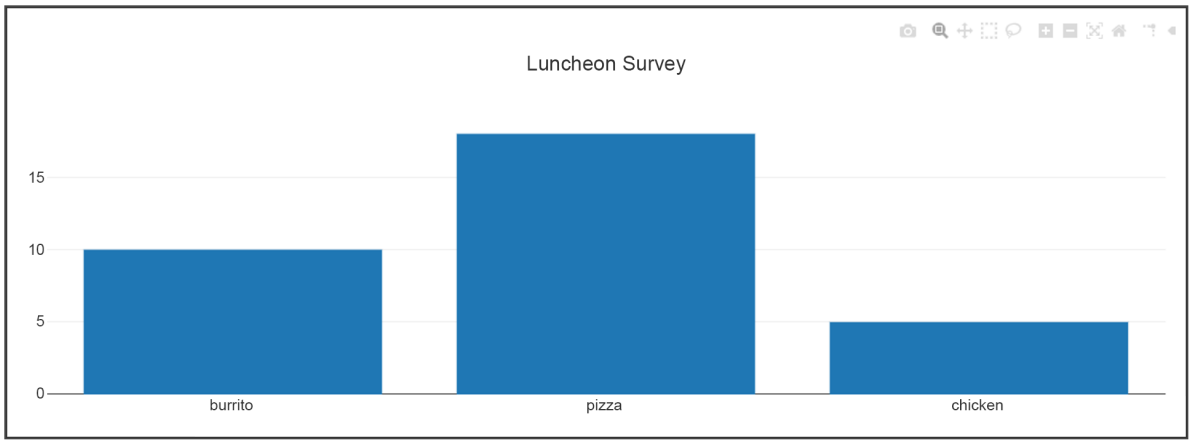
- Argument #1, `"plotArea"`, refers to the array that contains the data object.
  Argument #2, `[trace]`, refers to the HTML div that contains the chart.
  Argument #3, `layout`, refers to the object that contains the layout details.

- Argument #1, `"plotArea"`, refers to the object that contains the layout details.
  Argument #2, `[trace]`, refers to the array that contains the data object.
  Argument #3, `layout`, refers to the HTML div that contains the chart.

Check Answer

Finish ▸

The chart now has a title:



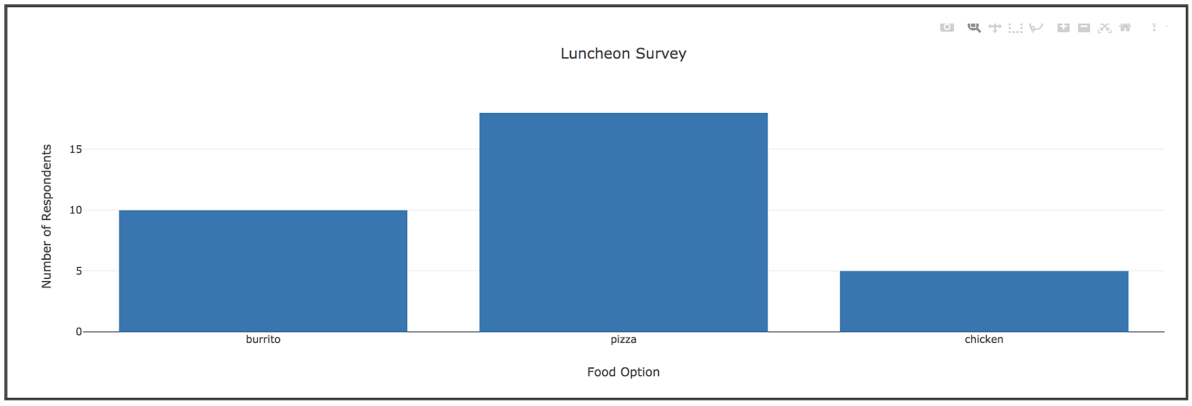
This is a great addition, but our chart is still missing axis labels. From glancing at this chart, we know that the label on the x-axis for each column is the food item. What about the y-axis, though? It's difficult to determine at a glance what the

numbers mean. For example, does “10” for burrito indicate that 10 types of burritos were ordered? Labeling the axes on a chart helps avoid this ambiguity.

Take a look at the code. What do you notice?

```
var layout = {
    title: "Luncheon Survey",
    xaxis: {title: "Food Option"},
    yaxis: {title: "Number of Respondents"}
};
Plotly.newPlot("plotArea", trace, layout);
```

In the same `layout` object, two key-value pairs have been added. The first key, `xaxis`, denotes the axis label for the x-axis. Its value, `{title: "Food Option"}`, is itself an object whose key is `title` and whose value is `Food Option`. The same format holds for the y-axis label.



Skill Drill

Open VS Code and create a new bar chart with Plotly. This graph will chart several beverages and the percentage of the total number of orders they comprise in a popular nonalcoholic bar. Here is your data:

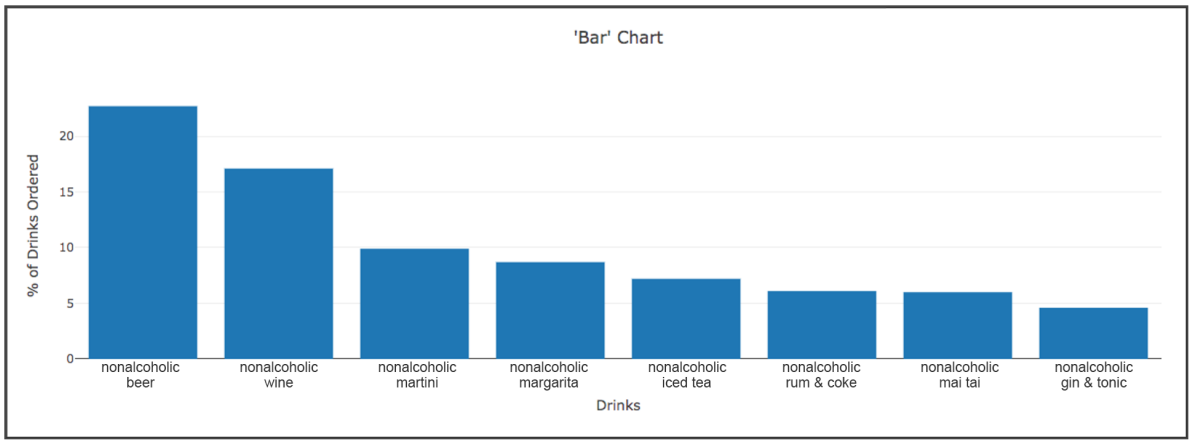
`Drinks: ["nonalcoholic beer", "nonalcoholic wine", "nonalcoholic martini", "nonalcoholic margarita", "ice tea", "nonalcoholic rum & coke", "nonalcoholic mai tai", "nonalcoholic gin & tonic"]`

Percent of Drinks Ordered: `[22.7, 17.1, 9.9, 8.7, 7.2, 6.1, 6.0, 4.6]`

> Be sure to give the plot a title and label the axes.

Roza decides to create a “bar” chart of popular beverages in a nonalcoholic bar, as well. This is her code:

```
var trace = {
 x: ["nonalcoholic beer", "nonalcoholic wine", "nonalcoholic martini", "r
 y: [22.7, 17.1, 9.9, 8.7, 7.2, 6.1, 6.0, 4.6],
 type: "bar"
};
var data = [trace];
var layout = {
 title: "'Bar' Chart",
 xaxis: { title: "Drinks"},
 yaxis: { title: "% of Drinks Ordered"}
};
Plotly.newPlot("plotArea", data, layout);
```



12.1.4: Create a Pie Chart

Let's summarize what we've done so far:

1. Create a new line chart in Plotly with a single line of code.
2. Create a bar chart.
3. Add layout options, such as title and axis labels.

Roza's joy is uncontained. She can create a beautiful chart in JavaScript with only a few lines of code. She would like to learn to create other types of charts as well. Let's help her do that.

Since the bar dataset is the percentage of total sales comprised by each drink, Roza believes that a pie chart may be helpful in providing an instant view of the relative popularity of each drink. So she creates a pie chart of the bar data:

```
var trace = {
  x: ["nonalcoholic beer", "nonalcoholic wine", "nonalcoholic martini", '
     "ice tea", "nonalcoholic rum & coke", "nonalcoholic mai tai", "nonal
  y: [22.7, 17.1, 9.9, 8.7, 7.2, 6.1, 6.0, 4.6],
  type: 'pie'
};
var data = [trace];

var layout = {
  title: "'Bar' Chart",
};
Plotly.newPlot("plotArea", data, layout);
```

But it doesn't render.

Why doesn't the pie chart display on the browser?

```
var trace = {
  x: ["nonalcoholic beer", "nonalcoholic wine", "nonalcoholic martini", "non
alcoholic margarita",
     "ice tea", "nonalcoholic rum & coke", "nonalcoholic mai tai", "nonalcoh
olic gin & tonic"],

  y: [22.7, 17.1, 9.9, 8.7, 7.2, 6.1, 6.0, 4.6],
  type: 'pie'
};
```

- When specifying the chart type, `'pie'` should be capitalized.
- In `trace`, the data arrays have x and y keys: This causes a problem because a pie chart does not have x and y axes. Different keys should be used.
- In `trace`, the data arrays have x and y keys: They need to be wrapped inside an object.

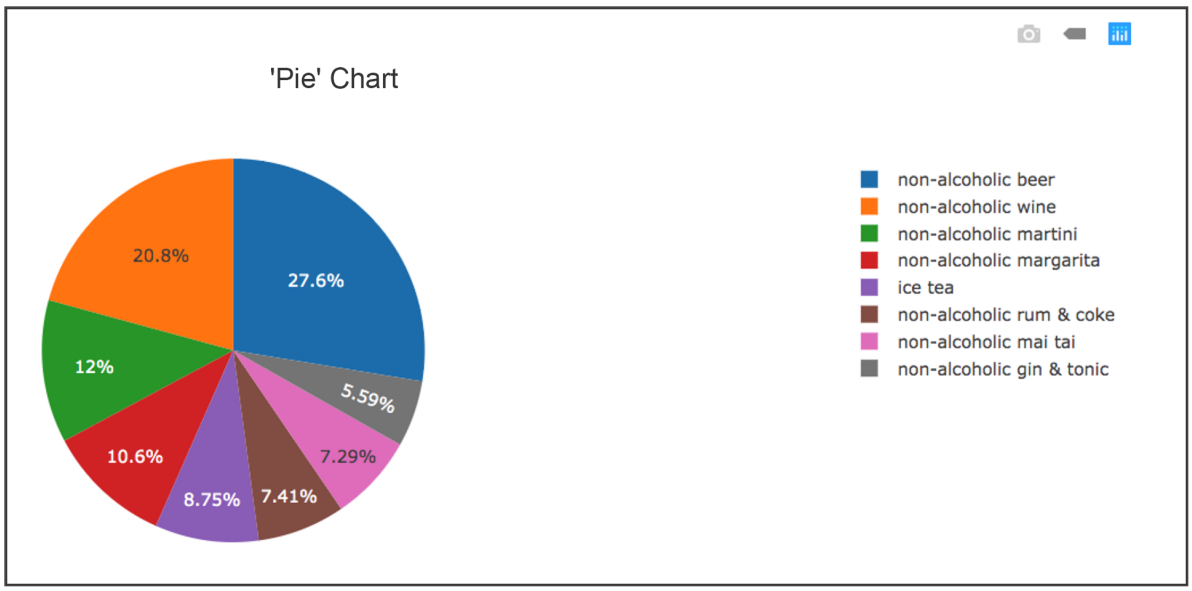
Check Answer

Finish ▸

Roza modifies her code:

```
var trace = {
 labels: ["nonalcoholic beer", "nonalcoholic wine", "nonalcoholic martini
 "ice tea", "nonalcoholic rum & coke", "nonalcoholic mai tai", "nonalcoho
 values: [22.7, 17.1, 9.9, 8.7, 7.2, 6.1, 6.0, 4.6],
 type: 'pie'
};

var data = [trace];
var layout = {
 title: "'Bar' Chart",
};
Plotly.newPlot("plotArea", data, layout);
```

What did she do differently? In `trace`, the keys for the data are now assigned appropriately. Instead of `x` and `y`, they are `labels` and `values`. This is what the pie chart looks like:



She now has a clearer view of the relative contribution of each drink to total sales. Encouraged and empowered, Roza decides to consult the `Plotly.js` documentation to see what other visualizations are possible with Plotly. Fortunately, Plotly is a well-documented library, and the documentation is easy to read.

> **Skill Drill**
>
> Use the **Plotly documentation (https://plot.ly/javascript/basic-charts/)** to create a scatter plot from scratch. Be sure to label your axes.

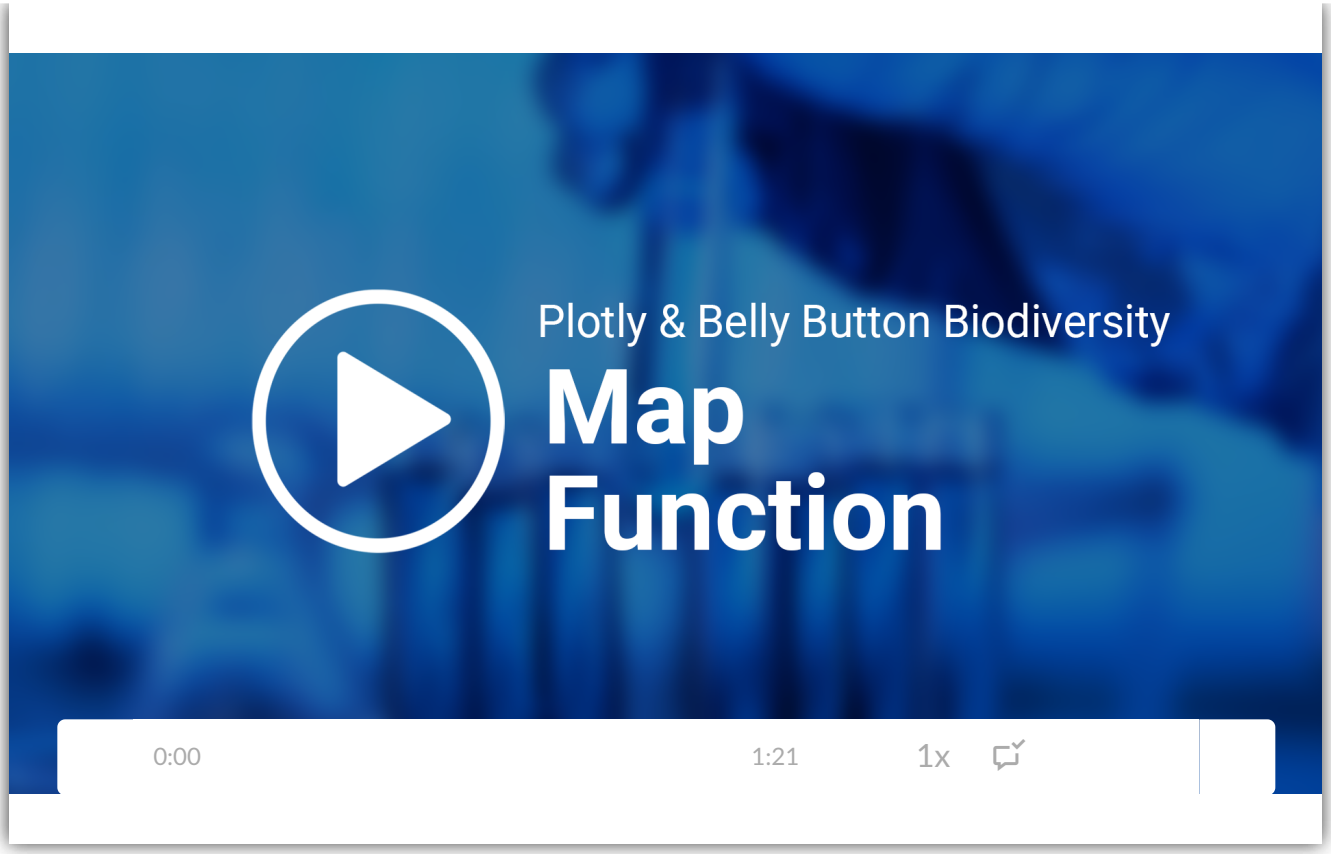
Roza now has a good grasp of creating basic charts with Plotly. This will help her decide which type of chart to use when visualizing data of belly button critters. For example, which type of chart might she use if she were to display the top ten most commonly found bacteria in a person's belly button?

12.2.1: Functional JavaScript

Roza is now familiar with creating various types of charts in Plotly. In order to visualize the data effectively, she needs to be able to manipulate the data. For example, out of the entire belly button dataset, she may wish to select only the data that pertain to one individual. She may also wish to sort data, so that she can display a data subset in ascending or descending order.

Roza's next step is to delve into JavaScript to take advantage of its power in manipulating datasets. Roza will be able to do things like filter the data that meet her specific criteria and to apply a transformation to each element in the dataset, often with just a few lines of code.

The first technique we'll use to transform data is the `map()` method. The `map()` method in JavaScript applies a transformation to each element in an array. Like a `for` loop, it can perform an operation to every element of an array.



Here is an example in which all the numbers of an array are doubled:

```
var numbers = [1,2,3,4,5];
var doubled = numbers.map(function(num){
    return num * 2;
});
console.log(doubled);
```

In this code, an array named `numbers` contains five integers: `var numbers = [1,2,3,4,5];`. Let's break down the rest of the code in more detail:

- The `numbers` array calls the `map()` method.
- Inside the `map()` method, there is another function. This function is anonymous, meaning that the function does not have a name. When `map()` is called, it in turn calls this anonymous function.
- The anonymous function takes a parameter, named `num`, and returns the number multiplied by 2. Its sole task is to perform this single action.
- For every element in the array, the `map()` method calls the anonymous function, which doubles the value of the element.

- The `map()` method returns an array of doubled values, which is assigned the variable `doubled`.

Here, the `map()` function becomes a method of the `numbers` array. It then takes in an anonymous function whose sole task is to double the value of `num`, its argument.

Behind the scenes, an iterative process similar to a `for` loop takes place. The anonymous function takes in each integer of the `numbers` array and doubles it. Finally, the variable `doubled` is an array of integers whose values are twice their original values.

Try running the code in your browser console and view the results for `doubled`. You should see the following:

```
> var numbers = [1,2,3,4,5];

  var doubled = numbers.map(function(num){
      return num * 2;
  });
  doubled
< ▶ (5) [2, 4, 6, 8, 10]
```

IMPORTANT

In the anonymous function inside the `map()` method, the parameter name `num` is arbitrary. It could have been named anything else, such as `integer` or `carPrice`. For example, the following two examples would be equally valid:

```
var doubled = numbers.map(function(integer) {
return integer * 2;
});
```

As would this:

```
var doubled = numbers.map(function(carPrice) {
return carPrice * 2;
});
```

Which of the following are true? Select all that apply.

- [ ] The `map()` method must take in a function as its parameter.
- [ ] The parameter name is arbitrary inside the anonymous function.
- [ ] The `map()` method calls the anonymous function on each element in the array.
- [ ] The anonymous method called by the `map()` method must perform a numerical operation.

Check Answer

Finish ▸

Skill Drill

Open VS Code and use `map()` to add 5 to each number in the following array:

```
var numbers = [0,2,4,6,8];
```

Verify your results in your browser console.

Roza now is able to perform a transformation, such as addition or multiplication, to every element of an array with `map()`. While a `for` loop could have done the same job, the `map()` syntax is cleaner and less cumbersome. Additionally, there are fewer variables involved in a map than in a `for` loop, meaning fewer opportunities for the coder to make a mistake. How would this be useful for Roza's project? Imagine that part of her dataset is an array of decimal numbers. If she wanted to round down each number in the array to the nearest whole number,

she could use `map()` to create and display an array of the whole numbers on the page.

Here's another way to use `map()`. In this example, `map()` is used to extract a specific property from each object in an array.

```
cities = [
    {
      "Rank": 1,
      "City": "San Antonio ",
      "State": "Texas",
      "Increase_from_2016": "24208",
      "population": "1511946"
    },
    {
      "Rank": 2,
      "City": "Phoenix ",
      "State": "Arizona",
      "Increase_from_2016": "24036",
      "population": "1626078"
    },
    {
      "Rank": 3,
      "City": "Dallas",
      "State": "Texas",
      "Increase_from_2016": "18935",
      "population": "1341075"
    }
];

cityNames = cities.map(function(city){
    return city.City;
});
console.log(cityNames);
```

Note the following:

1. `cities` is an array of objects. Each object has multiple properties, such as `Rank`, `City`, and `State`.

2. The `map()` method is used to extract only the `City` property of each object, i.e., San Antonio, Phoenix, and Dallas. During each iteration, the anonymous function inside `map()` returns only that property of each object.
3. `cityNames` is an array of only these city names.

> **Skill Drill**
> Open your text editor and browser. Modify the code in the previous example to extract the population of each city, instead of the city name.

Have you noticed that the syntax for the `map()` method is far cleaner and involves fewer variables than a `for` loop? Additionally, whereas a `for` loop gives specific instructions on start and end points of the loop, `map()` does not. A `for` loop is imperative, meaning that its code is more detailed on the specific operations involved in it. The `map()` method, on the other hand, is more abstract, and does not specify things such as stop and end points.

However, the `map()` method does something that a `for` loop does not always do: it calls another function. These are some of the features of the functional programming paradigm, which as we have seen, may lead to fewer errors and complications.

The filter() Method

Another functional programming technique is the `filter()` method. Like the `map()` method, it accepts another function as its parameter. Like `map()`, `filter()` performs an operation on every element in the original array. Unlike `map()`, however, `filter()` does not necessarily return an array whose length is the same as the original array.

Let's see what this means in an example. Run the following code in your console. What does `larger` return?

```
var numbers = [1,2,3,4,5];

var larger = numbers.filter(function(num){
    return num > 1;
});

console.log(larger);
```

It returns an array of integers that are larger than 1: `[2,3,4,5]`.This example is remarkably similar to the last one, with one major difference.

First, the similarities:

- The `numbers` array uses the `filter()` method.
- The `filter()` method, in turn, takes an anonymous function as its argument. The anonymous function's sole task is to take in a parameter, called `num`.

The `filter()` method operates on each element of the `numbers` array. So how does it differ from `map()`?

The `map()` method transforms every element of the original array, and so the size of the transformed array is the same as that of the original array.

The `filter()` method, on the other hand, returns an array of values that meet certain criteria. Values in the original array that do not fulfill the condition are filtered out. In this case, specifically, the anonymous function called by `filter()` returns `true` if an argument is larger than 1, and `false` if it does not. The `filter()` method runs the anonymous function on every element of the original `numbers` array. Only numbers that are larger than 1 are returned: `[2,3,4,5]`. So whereas applying `map()` to the numbers array would have returned an array with five elements, applying this specific filter returned an array of only four elements.

Inspect the code below. An unsorted array of the age of each person in a household is filtered to include only those who are older than five. How many elements will olderThanFive contain?

```
var familyAge = [2,3,39,37,9];

var olderThanFive = familyAge.filter(function(age){
    return age > 5;
});
```

- Two: [2,3]
- Three: [37,39,9]
- One: [9]

Check Answer

Finish ▸

Now test your skills in the following Skill Drill.

Skill Drill

> You are given the following array:
>
> `var words = ['seal', 'dog', 'scorpion', 'orangutan', 'salamander'];`
>
> Filter the results to include only animals whose species name starts with the letter “s.”

Arrow Functions

Let’s do a quick review of arrow functions.

> **Rewind**
> An arrow function in JavaScript is syntactic sugar. That is, an arrow function does the same thing as a standard JavaScript function, but it streamlines the syntax used to accomplish the same task.

The anonymous function inside `map()` and `filter()` can be simplified as an arrow function. Here’s an example:

```
var numbers = [1,2,3,4,5];


var doubled = numbers.map(num => num * 2);
console.log(doubled);
```

The `map()` method performs the identical operation as before: it doubles each element in the `numbers` array. However, the anonymous function inside `map()` has been replaced by an arrow function. Contrast the two:

```
var doubled = numbers.map(function(num){
    return num * 2;
});
var doubled = numbers.map(num => num * 2);
```

In the first, `function(num){return num * 2;};` doubles each array element and returns it to `map()`. In the second, the code used to perform the same task is `num => num * 2`. The arrow, `=>`, is a shorthand for an anonymous function that takes the parameter `num` and returns `num * 2`.

> **Skill Drill**
>
> Modify the code below to use an arrow function inside `filter()`:
>
> ```
> var words = ['seal', 'dog', 'scorpion', 'orangutan', 'salamander'];
> var animalsWithS = words.filter(function(word){
> return word[0] == 's';
> });
> ```

The sort() and reverse() Methods

Roza is now able to transform or filter data in one swoop. It would be helpful to be able to sort a dataset as well.

For example, suppose that she wants to use her data to rank the most commonly found inhabitants of the human navel. She could collect the frequency of each species among her volunteers, then use the `sort()` method.

Let's take a look at a simple example.

```
var familyAge = [3,2,39,37,9];
sortedAge = familyAge.sort((a,b) => a - b);
console.log(sortedAge);
```

`sortedAge` returns the array `[2,3,9,37,39]`. Like `map()` and `filter()`, `sort()` takes in an anonymous function. During each iteration, the anonymous function, an arrow function in this case, compares one element of the array (`a`) with the next element in the array (`b`). From `a`, it subtracts `b`. If the result is negative (i.e.,

`b` is larger than `a`) then it stays put. If the result of the subtraction is positive, the order of the two elements is reversed. Look at a modified version of this example.

```
var familyAge = [3,2,39,37,9];

sortedAge = familyAge.sort((anElement,nextElement) => anElement - nextEle
```

Let's break down this code:

- The variables `a` and `b` are replaced by `anElement` and `nextElement`.
- The first two elements that are compared are 3 and 2. The variable `anElement` is assigned to 3, and `nextElement` to 2.
- The arrow function performs the subtraction `anElement - nextElement`, or 3 - 2.
- Since the result is positive (3 - 2 = 1), the order of the two numbers is reversed.
- The `sort()` method compares the second and third elements in the array: 3 and 39.
- Since 3 - 39 is a negative number, their ordering is kept.
- The process is repeated for all remaining elements in the array.

Appending `reverse()` to the above sorts the array in descending order. Try it in your browser console.

Will the following code sort the familyAge array in ascending or descending order?

```
var familyAge = [3,2,39,37,9];
sortedAge = familyAge.sort((a,b) => b - a);
```

- ○ Ascending order.
- ○ Descending order.

Check Answer

Finish ▸

Let's reflect briefly on how Roza might apply the `sort()` function. Each of her volunteers carries a variety of bacterial species in his or her belly button. There is also information on the number of bacteria found for each species. The dashboard Roza has in mind will display the most common bacterial species, by count, in the navel. If Improbable Beef is looking for people who carry a large number of a certain bacterial species, Roza's volunteers should be able to quickly use the dashboard to figure out whether they are eligible to sell their bacteria to the company. To accomplish this goal, she should sort the most common species of bacteria with `sort()`, and then display the results on the webpage.

The slice() Method

Roza also needs to be able to select a subset of the data. In her project, for example, she might perform a transformation on an array, filter it, sort it, and then display only the top five results.

```
var integers = [0,1,2,3,4,5];
slice1 = integers.slice(0,2);
```

In this example, the `slice()` method returns the first two elements of the `integer` array: `[0,1]`. The first argument is the position of where to begin the selection. Here, it is at index position 0. The next argument, 2, denotes the position of the array where the slicing ceases. In other words, the `slice()` method begins selecting the array at index position 0, and stops right before reaching index position 2. So here, it returns elements at index positions 0 and 1, but not 2.

> ### Skill Drill
>
> `var words = ['seal', 'dog', 'scorpion', 'orangutan', 'salamander'];`
>
> Use `slice()` to select the first three elements of the `words` array.

To slice to the end of an array, you can omit the second argument:

```
var words = ['seal', 'dog', 'scorpion', 'orangutan', 'salamander'];
words.slice(3, );
```

The elements sliced here are `['orangutan', 'salamander']`. Great job!

Roza is now able to create charts with Plotly as well as manipulate datasets with JavaScript. She's now ready to combine these skills in order to create a Plotly chart whose underlying data has been filtered, sliced, mapped, or sorted by JavaScript.

12.2.2: Practicing JavaScript Methods

Roza feels more comfortable with using JavaScript methods to manipulate data, but would like to consolidate her skills in building an actual plot. Instead of jumping into the belly button dataset, she would like to practice creating a simpler graph in Plotly that requires some behind-the-scenes JavaScript action. She will create a bar chart of five cities whose populations have seen the greatest increase in the period 2016–2017.

We will help her from beginning to end. Are you ready?

The first thing we need to do is to obtain the dataset, which can be downloaded using the link below.

Download the dataset (https://courses.bootcampspot.com/courses/138/files/14444/download?wrap=1)

Now let's set up our `index.html` to use Plotly. Create `index.html` and paste the following code into it.

```
<!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">
```

```
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <title>Fast-Growing Cities 2016-2017</title>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
<body>
  <div id="bar-plot"></div>
  <script src="data.js"></script>
  <script src="plot.js"></script>
</body>
</html>
```

Note the following:

- The `<title>` tag is appropriate to the chart Roza is creating: “Fast-Growing Cities 2016–2017.”
- The first `<script>` tag links to the Plotly CDN.
- The `<div>` tag, where the Plotly chart will be displayed, is titled “bar-plot.”
- The second `<script>` tag links to the data file, `data.js.`
- The third `<script>` tag links to the script file for the chart, i.e., the JavaScript code.

> **Important**
> Always make sure that your files are linked correctly and in the right order. Since Roza needs the Plotly CDN to use Plotly, the CDN `<script>` tag should come first. Since she will also need the data to load before using Plotly, the data file should be linked next. Finally, the script file is linked.

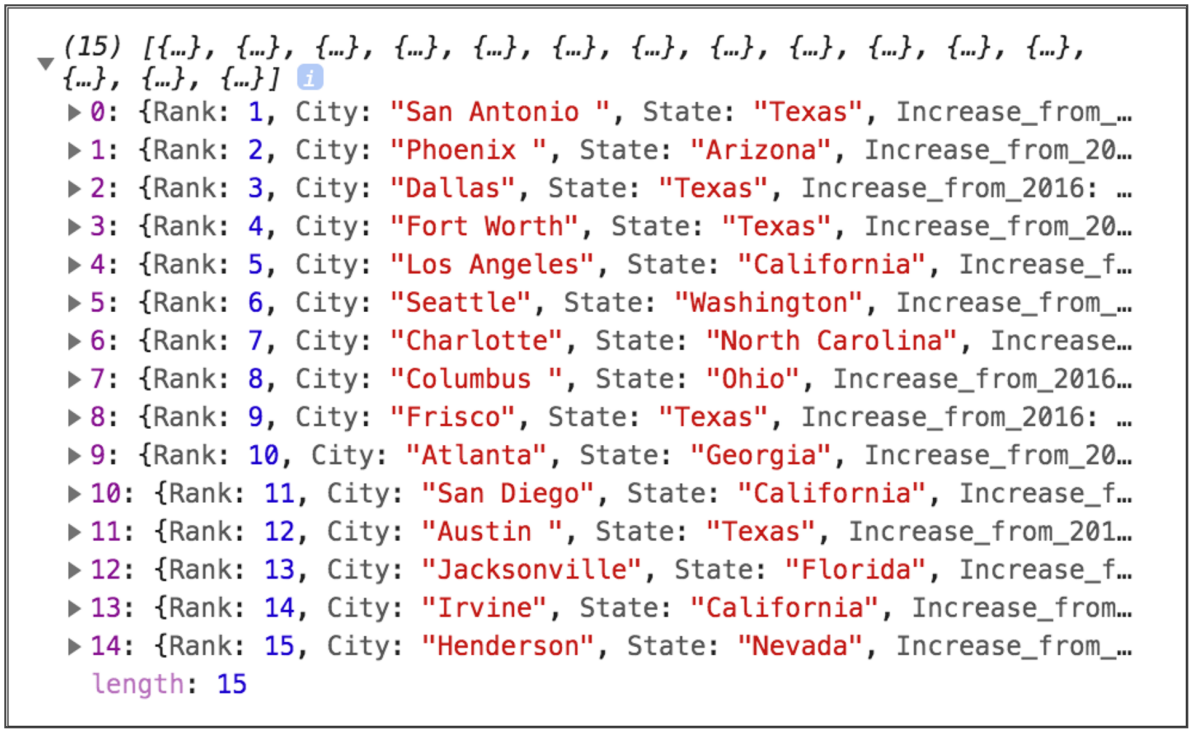
Now take a moment to examine the dataset, contained in `data.js`, of which you have seen a sample.

```
var cityGrowths = [
  {
    "Rank": 1,
    "City": "San Antonio ",
    "State": "Texas",
```

```
    "Increase_from_2016": "39208",
    "population": "1511946"
  },
```

The variable `cityGrowths` is assigned to an array of objects, each of which contains a city and some associated facts, such as its population. This array, although contained in a separate file, can be accessed from `plot.js` so create a blank file named `plot.js` now. To verify that the data is correctly read in, Roza should call `console.log(cityGrowths);` in `plot.js`.

Here's the result in the browser console:



Now take a moment to plan our next steps. We will need to:

1. Sort the cities in descending order of population growth.
2. Select only the top five cities in terms of growth.
3. Create separate arrays for the city names and their population growths.
4. Use Plotly to create a bar chart with these arrays.

Sort and Select the Cities

The first step is to sort the cities by population growth. For this she will use the `sort()` method, which in turn will call an anonymous function to sort objects by the `Increase_from_2016` property.

What would the code to sort the cities by growth look like?

- `var sortedCities = cityGrowths.sort( => a.Increase_from_2016 - b.Increase_from_2016);`
- `var sortedCities = cityGrowths.filter((a,b) => a.Increase_from_2016 - b.Increase_from_2016).reverse();`
- `var sortedCities = cityGrowths.sort((a,b) => a.Increase_from_2016 - b.Increase_from_2016).reverse();`

Check Answer

Finish ▸

The next step is to select only the top five cities by population growth. We'll use `slice()` to perform this task.

How would Roza slice only the top five cities? Assign the results to a variable named topFiveCities.

- `var topFiveCities = sortedCities.slice(1,5);`
- `var topFiveCities = sortedCities.slice(0,5);`
- `var topFiveCities = sortedCities.slice(0,4);`

Check Answer

Finish ▸

Create Arrays of City Names and Growth Figures

Now we need to create two arrays: an array of city names, and an array of corresponding population growths. We'll use the `map()` method to extract these properties. These arrays will be the `x` and `y` axis data of the Plotly chart.

Suppose that Roza wants to use `map()` to create a separate array of the top five city names, as well as the top five growth figures. How would we help her do this?

Add the following JavaScript code from `plot.js`.

```
var topFiveCityNames = topFiveCities.map(city => city.City);
var topFiveCityGrowths = topFiveCities.map(city => parseInt(city.Increase
```

Did you notice the `parseInt()` method used above? You might have also noticed that all values in the dataset are strings. For example, in this object, the numerical values are formatted as strings, as indicated by the quotation marks, rather than integers.

```
{
    "Rank": 2,
    "City": "Phoenix",
    "State": "Arizona",
    "Increase_from_2016": "34036",
    "population": "1626078"
  },
```

That is why `parseInt(city.Increase_from_2016)` converts strings into integers. While JavaScript is flexible enough to interpret numbers enclosed in quotation marks as numbers, it's good practice to explicitly transform, or cast, strings into numbers. Below is the array that is returned when `parseInt()` is called:

And this is the array that is returned when `parseInt()` is not called:

Create a Bar Chart with the Arrays

The final task is to render these arrays in Plotly. Add the following code to `plot.js`.

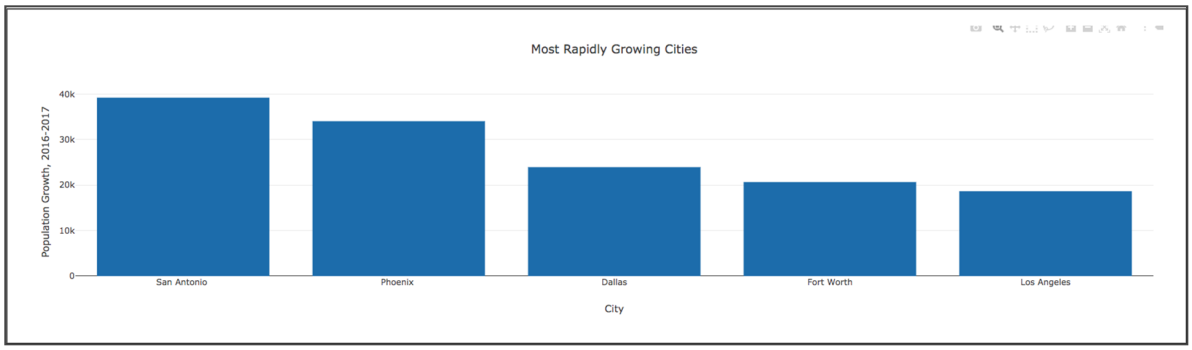
```
var trace = {
  x: topFiveCityNames,
  y: topFiveCityGrowths,
  type: "bar"
};
var data = [trace];
var layout = {
  title: "Most Rapidly Growing Cities",
  xaxis: { title: "City" },
  yaxis: { title: "Population Growth, 2016-2017"}
};
Plotly.newPlot("bar-plot", data, layout);
```

Here's what's happening in this code:

- The `trace` specifies the type of graph as a bar chart as well as defines the `x-` and `y`-axis data.

- The variable `data` encloses `trace` in an array to meet Plotly's format requirement.
- The variable `layout` is assigned to an object that specifies the chart's title and axis labels.
- Finally, the graph is rendered with `Plotly.newPlot()`.

This is the resulting chart:



Well done!

> **Skill Drill**
>
> Use the same dataset to create a bar chart of the seven largest cities by population.

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 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.

Will the following lines of code run?

```
const url = "https://api.spacexdata.com/v2/launchpads";
d3.json(url).then(data => console.log(data));
```

- ○ Yes
- ○ No

Check Answer

Finish ▸

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.

How would Roza rewrite the code to retrieve the details only from the Vandenberg Air Force Base?

She would use the built-in property only to retrieve its first element.

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

She would use indexing to retrieve only the first element in the array returned from the API call.

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

She would use indexing to retrieve the first element of the array returned from the API call.

```
d3.json(url).then(spaceXResults =>
console.log(spaceXResults[-1]));
```

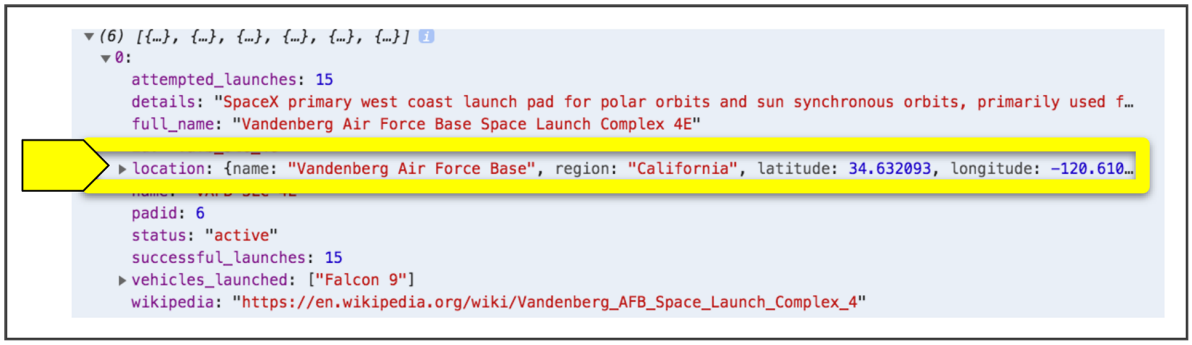
Check Answer

Finish ▸

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:



How would Roza access the latitude of the Vandenberg Airforce Base?

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

Check Answer

Finish ▸

Skill Drill

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

12.3.2: Load a JSON file with D3.json()

Roza is now ready to load the belly button data into her script. First, she needs to download the data file to her computer. She'll then use two things to read the data file: the D3.js library and a local server.

To download the zip file containing the data file, use the following link.

Download the data (https://courses.bootcampspot.com/courses/138/files/13727/download?wrap=1)

Recall the previous syntax used to place an API call:

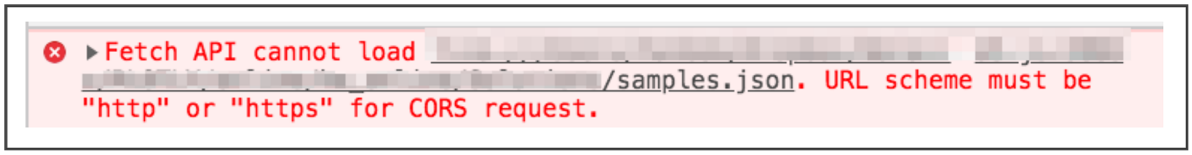
```
const url = "https://api.spacexdata.com/v2/launchpads";
d3.json(url).then();
```

Here, the URL string is received by `d3.json()` as an argument. The `d3.json()` method then retrieves the data from the address specified by the URL. After the data is fully retrieved, the function inside the `then()` method is executed.

The syntax used to retrieve data from an external data file, instead of a web API, is the same:

```
d3.json("samples.json").then(function(data){
    console.log("hello");
});
```

When we open the browser, however, nothing is printed to the console. We get this error message:



What gives? What is a CORS request?

The short explanation is that, for security reasons, a local server must be run when loading an external file into a JavaScript script file. If you don't understand these security issues right now, don't worry. We will come back to it later.

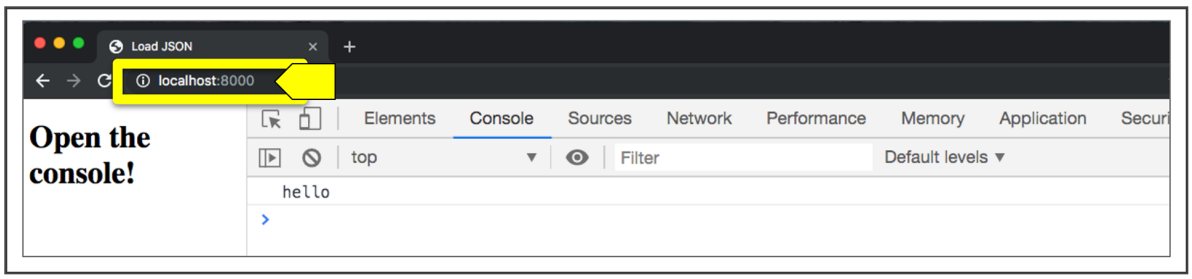
To load the page, navigate to the directory where `samples.json` and `index.html`, as well as the script file, `plots.js`, are located. Open the command line (Terminal or Git Bash) and type the following:

```
python -m http.server
```

You should see the following message in the command line:

```
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [30/Oct/2019 13:23:53] "GET / HTTP/1.1" 304 -
```

Navigate to the given port number in your browser: `localhost:8000`. When you do this, you'll see the following screen:



IMPORTANT

When reading an external data file such as a CSV or JSON file into a script, you must run a server. You cannot directly open `index.html` with your browser.

Skill Drill

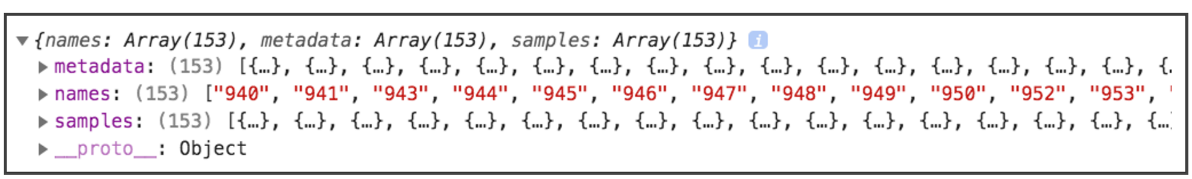
Repeat the following steps:

1. Create a directory on your computer and copy the file `samples.json` into it.
2. Create `index.html` and `plots.js` files.
3. Be sure to import `D3.js` via a CDN in your HTML file.
4. Use `d3.json()` to read `samples.json` into your script.
5. Print a simple message to your browser console.
6. Run `python -m http.server` and open your browser to the port address (likely 8000).

You're now prepared to help Roza read and parse the actual data. Modify the code to change the printed console message from a simple "hello" to the entire dataset:

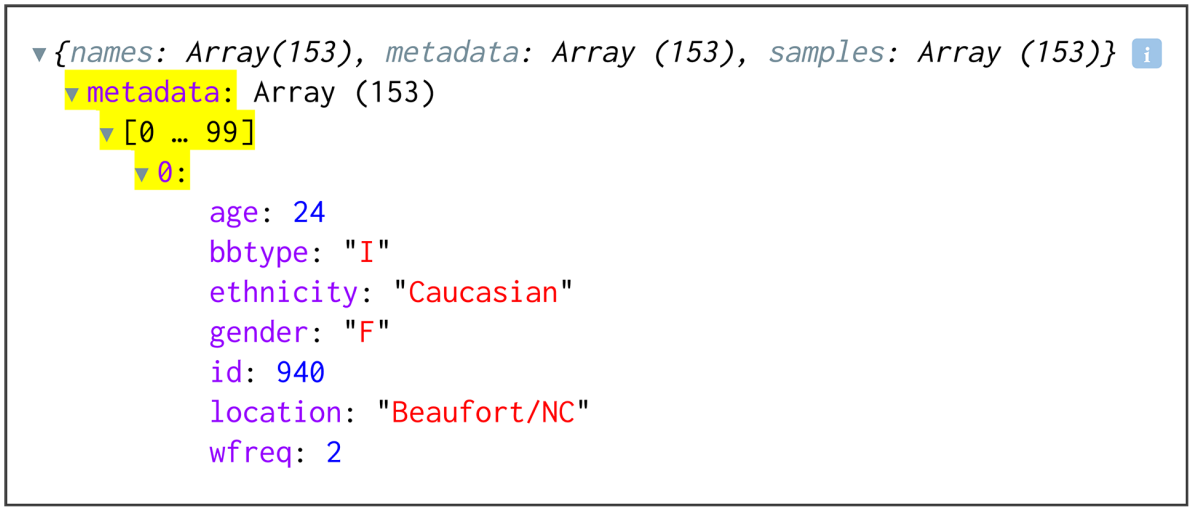
```
d3.json("samples.json").then(function(data){
    console.log(data);
});
```

Then examine the results in the browser console:



The data is structured as an object that contains three keys at the top level: `metadata`, `names`, and `samples`. Each of these keys is associated with an array that contains 153 elements.

Let’s look at each array in more detail. Click on the arrows successively to list the details of the first person:



The `metadata` array contains objects, each of which contains details of a volunteer, such as age, location, ethnicity, ID number, and weekly washing frequency of the belly button.

Roza wants to extract only the `wfreq`, or the weekly belly button washing frequency, of each person into a new array. What would her code look like?

```
d3.json("samples.json").then(function(APIresult){
    wfreq = data.metadata.map(person => person.wfreq);
    console.log(wfreq);
});
```

```
d3.json("samples.json").then(function(data){
    wfreq = data.metadata.map(person => person.wfreq);
    console.log(wfreq);
});
```

```
d3.json("samples.json").then(function(data){
    wfreq = data.metadata(person => person.wfreq);
    console.log(wfreq);
});
```

Check Answer

Finish ▸

Roza now wants to sort the `wfreq` array in descending order. What would her code look like?

```
d3.json("samples.json").then(function(data){
    wfreq = data.sort((a,b) => b - a);
    console.log(wfreq);
});
```

```
d3.json("samples.json").then(function(data){
    wfreq = data.metadata.map(person =>
person.wfreq).sort((a,b) => a - b);
    console.log(wfreq);
});
```

```
d3.json("samples.json").then(function(data){
    wfreq = data.metadata.map(person =>
person.wfreq).sort((a,b) => b - a);
    console.log(wfreq);
});
```

Check Answer

Finish ▸

Roza now wants to delete null values from the sorted `wfreq` array. Which of the following methods would she use?

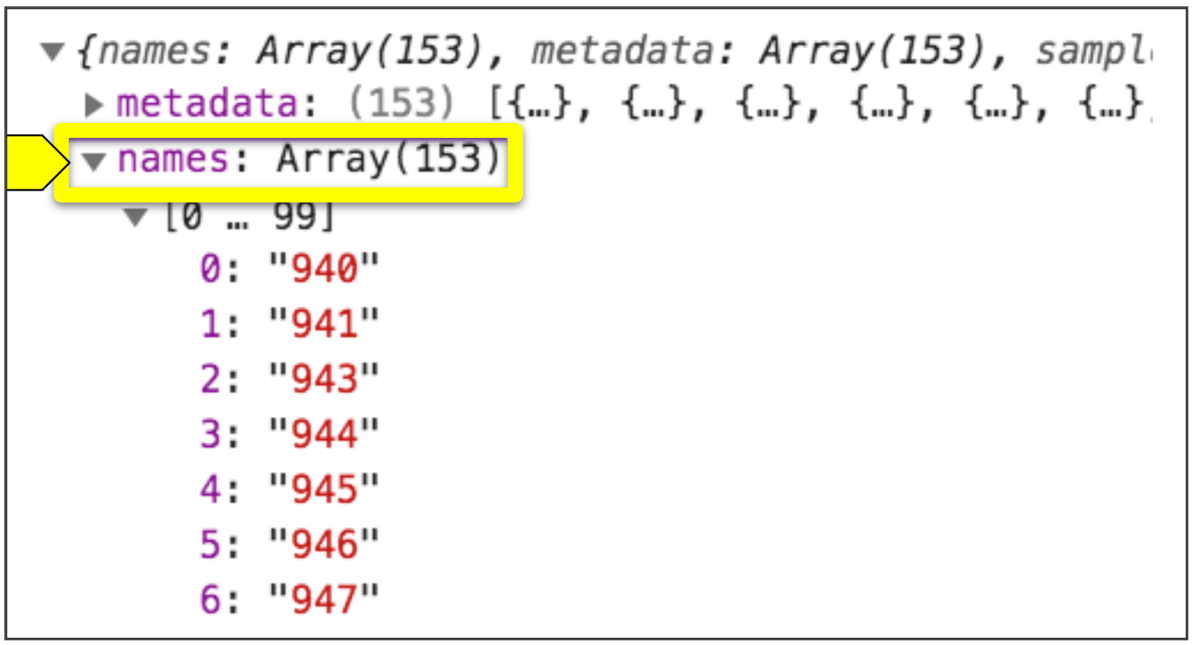
- `map()`
- `sort()`
- `filter()`

Check Answer

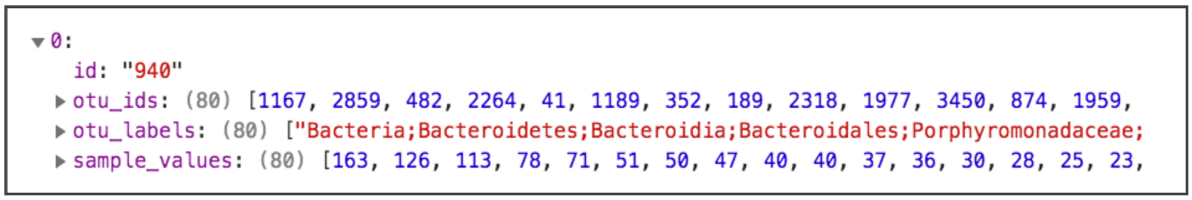
Finish ▸

Now let's examine the rest of the dataset. Looking at the image below, we can see that `names` is simply an array of the ID numbers of the volunteers. Even though

this information is included in the `metadata` array, the `names` array may be useful in rapidly retrieving an ID number when creating a plot.



Next, check out the `samples` array and inspect the first element.



The array's first element is an object, with four key-value pairs. Note the following:

- The `id` key identifies the ID number.
- The `otu_ids` property is an array of the ID numbers of all the bacteria found in this person's navel. OTU stands for Operational Taxonomic Unit, and here it means species or bacterial type. In this instance, there were 80 bacterial types with distinct ID numbers.
- The `sample_values` array contains the corresponding species name for each bacterial ID number. Some bacterial species have different ID numbers, but are clumped together under the same `otu_label`.

In her final visualization of the belly button data, Roza would like to be able to select an individual from a dropdown menu. Once a person's ID number is selected, she would like to display the demographic information of that individual.

Since each individual is represented by an object, she'll need to access both keys and values inside an object in order to do this.

Rewind

The `Object.entries()` method allows access to both an object's keys and values. It returns each key-value pair as an array.

You are given the following object. Use `Object.entries()` to print each key-value pair inside an array.

```
researcher1 = {
    name: 'Roza',
    age: 34,
    hobby: 'Hiking'
};
```

- `console.log(researcher.map(researcher1));`
- `console.log(Object.entries(researcher1));`
- `console.log(Object.keys(researcher1));`

Check Answer

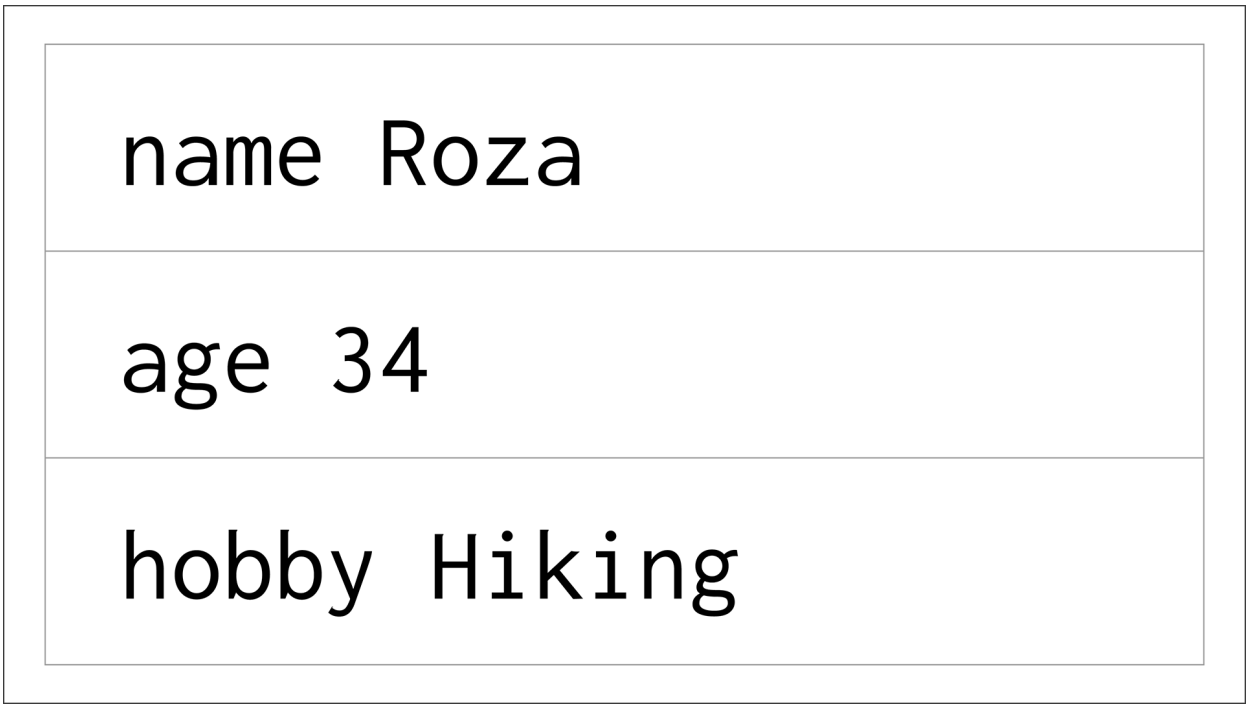
Finish ▸

Rewind

The `forEach()` method allows access to each element of an array.

Which of the following methods would you use to print to the console each trait and corresponding property?

```
researcher1 = [['name', 'Roza'], ['age', 34], ['hobby',
'Hiking']];
```



- `map()`
- `sort()`
- `forEach()`

Check Answer

Finish ▸

Now test your skills in the following Skill Drill:

Skill Drill

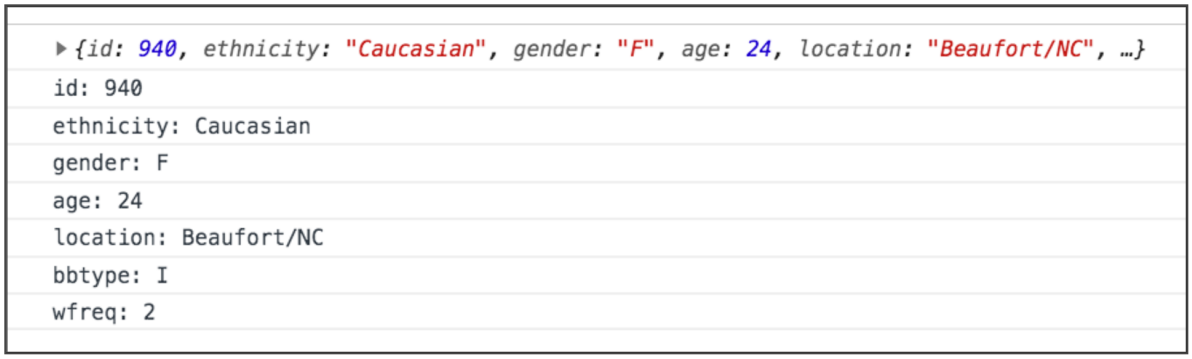
> Use `Object.entries()` and `forEach()` to print all the metadata of the first person in the `samples.json()` dataset (ID `940`).

Roza has made a definite step forward. With the following code, we can display the metadata of any individual from the dataset:

```
d3.json("samples.json").then(function(data){
    firstPerson = data.metadata[0];
    Object.entries(firstPerson).forEach(([key, value]) =>
      {console.log(key + ': ' + value);});
});
```

In this case, we are extracting the metadata of the first person in the dataset, as indicated by the zero index position in `metadata[0]`. We then use the `Object.entries()` method to return each key-value pair in an array, and the `forEach()` method to access each element of these pairs.

Open the browser console to see the results:



Here, we manually specify the individual by the index position. Ultimately, we need to be able to choose an ID number from a dropdown menu and then display the metadata associated with that ID.

12.3.3: Handle CORS Errors

Earlier, Roza discovered that she couldn't simply read in an external JSON dataset into her JavaScript file. Instead, she was required to run a local server. She wants to understand more about the CORS error message she came across, and why she must run a server to bypass it.

When Roza first began to learn to use Plotly, the data used to create charts were contained in the JavaScript file. She was able to open `index.html` directly in her browser and see the graphs. Likewise, when she made a call to the SpaceX API, she was able to open `index.html` directly in her browser.

However, when she attempted to read data from an external JSON file, she encountered an error:

To bypass this CORS error message, we navigated to the directory where `index.html` is located and ran `python -m http.server` in the CLI.

CORS stands for Cross-Origin Resource Sharing. In short, browsers by default do not permit reading of resources from multiple sources. This restriction is in place because of security concerns.

To better understand the issue, we first need to define servers. A **server** is a program or device that performs actions such as processing and sharing data. Our discussion will be limited to servers in the sense of software programs.

A Flask app, for example, is a server program that processes and shares data. Likewise, when we place a call to the SpaceX API, there is a server behind the scenes that processes and shares the requested data. Another example is when a user logs in on a website, the server receives the user's information, compares it against information in its database, and approves or denies the login attempt.

This is called a **request-response model**. The user (also known as the **client**) sends a request to the webpage server. The server, in turn, sends the requested data in response.

Web browsers, for security reasons, heavily restrict reading from, and writing to, local files. If access to local files was allowed, remote sites would be allowed to read and manipulate your private data. Or simply opening a local file with the browser could trigger a malicious script that transmits your data across the internet. This is why we're unable to read a JSON file directly.

However, running a static server, or `python -m http.server` in our case, allows us to skirt this restriction. Python's HTTP server provides a web address for both the JSON and HTML files to avoid these security issues.

Now let's return to the browser error message: `URL scheme must be 'http' or 'https' for CORS request.` This means that the browser can request external data only through the HTTP/HTTPS protocols. In other words, the CORS policy, by default, does not allow data to come in through channels other than through HTTP or HTTPS. Furthermore, the origin of the data must be from a single source, unless specified by CORS.

Here is a concrete example of how CORS works. Suppose that you navigate to a news website, and you are served an ad from adspamnetwork.com. If you happen to be logged into PayPal, and if these browser restrictions weren't in place, the JavaScript code in the ad might make an API call to PayPal and make unauthorized transactions. For this reason, browsers restrict a server from one

site (adspamnetwork.com in this case) from making a request to a server from a different site (paypal.com) unless it has been given explicit permission.

How, then, does a website such as ebay.com make API calls to PayPal? The browser generally makes a preflight request to the server, which verifies whether the browser's origin is allowed to make a request to it. The preflight request also includes other details, such as the types of requests permitted to be made, and the types of files permitted to be transferred. Then a request is made. The code on PayPal's server contains a CORS header that explicitly permits ebay.com to make API requests.

> **Note**
>
> For more information about CORS, see the **MDN documentation (https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS)** .

Which of the following files is the browser restricted from accessing by default?

- `index.html`
- `style.css`
- `script.js`
- `citydata.csv`

Check Answer

Finish ▸

12.4.1: JavaScript Event Listeners

Roza is getting closer to her goal. Let's summarize what she has learned so far.

She is able to create various types of static visualizations with Plotly, such as bar and line charts.

She is also able to perform sophisticated data manipulations under the hood. She can retrieve data from an external JSON file, iterate through objects and retrieve necessary data from them, whether they are object keys or object values. She can also iterate through arrays with methods such as `map()` and `filter()`.

The missing link between the static visualizations and under-the-hood JavaScript data operations is interactivity. It is interactivity that will enable Roza to generate customizable charts dynamically. In order to make interactive visualizations, she'll first need to create JavaScript event listeners.

First we'll help Roza create a very simple dropdown menu, and then build on our skills. Let's look at the contents of the `index.html`:

```
<!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>Events</title>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/5.9.7/d3.min.js"
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
<body>
  <div id="menu"></div>
  <select id="selectOption">
    <option value="option1">First Option</option>
    <option value="option2">Second Option</option>
  </select>
<script src="script.js"></script>
</body>
</html>
```

Note the following:

- This time, there are links to two CDNs: D3 and Plotly.
- The `<select>` tag indicates a dropdown menu. Its `id` is `"selectOption"`.
- The dropdown menu has two options, as indicated by the two `<option>` tags.
- The option values `"option1"` and `"option2"` are internal names for each dropdown menu option.
- `First Option` and `Second Option` are the text displayed in the browser for each menu option.
- A plc tag links to `script.js`, a JavaScript file.

> **Rewind**
>
> The `<select>` tag is used to create a dropdown menu. The `<option>` tag is used to create each menu option.

Now open `script.js`:

```
d3.selectAll("body").on("change", updatePage);

function updatePage() {
  var dropdownMenu = d3.selectAll("#selectOption").node();
  var dropdownMenuID = dropdownMenu.id;
  var selectedOption = dropdownMenu.value;

  console.log(dropdownMenuID);
  console.log(selectedOption);
};
```

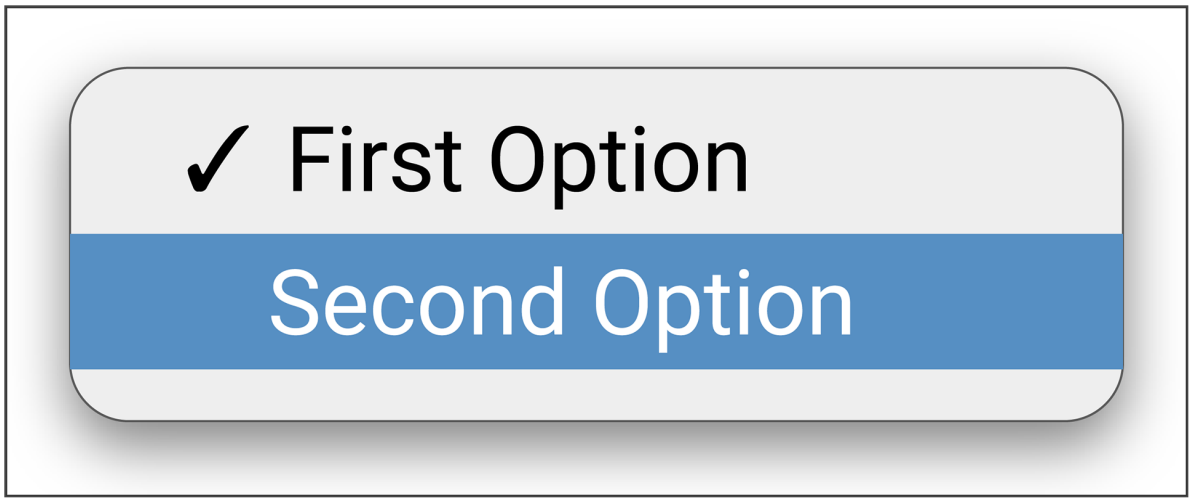
The first line uses the `d3.selectAll()` method to create an event listener. Whenever there is a change to the HTML body, the `updatePage()` function is called. That is, when an event occurs on the page, such as selection of a dropdown menu option, the `updatePage()` function is triggered.

> **Rewind**
>
> `d3.selectAll().on("change", );` creates an event listener that triggers the custom function every time a change takes place to the HTML element specified by `selectAll().`

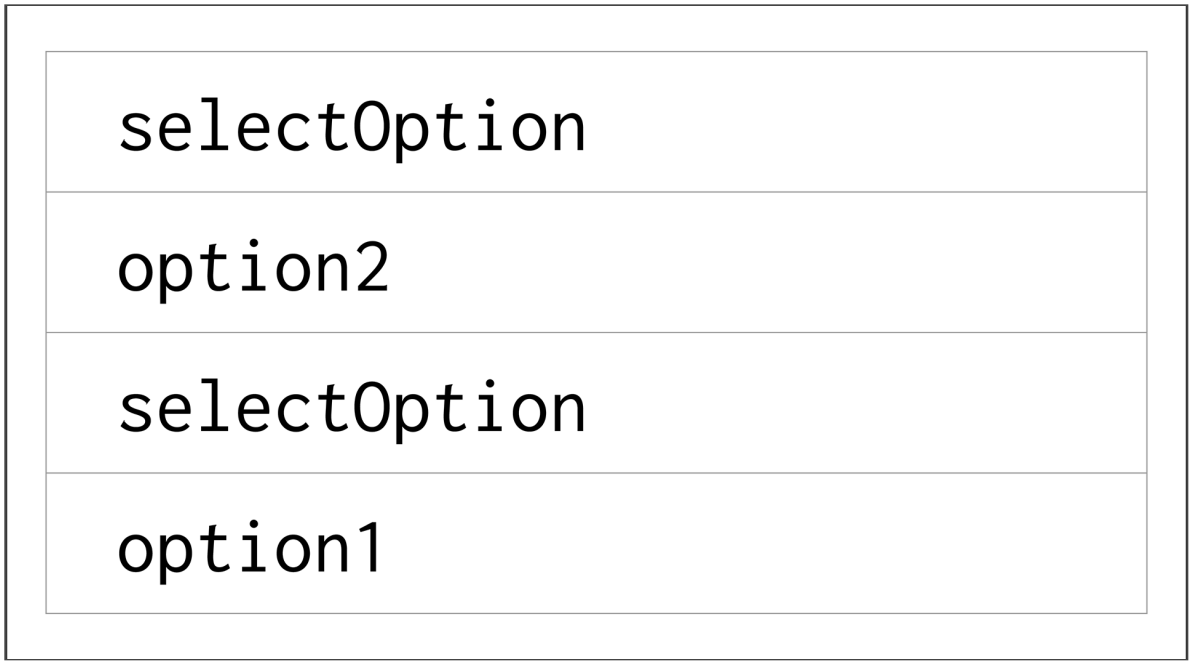
Let's look at the `updatePage()` function in greater detail:

1. The function uses `d3.selectAll()` to create its own event listener. This time, however, it is limited to listening to events that take place to the `<select>` tag, which has an `id` of `selectOption`.
2. The `id` of the dropdown menu, `selectOption`, is assigned the variable `dropdownMenuID`.
3. Whenever a dropdown menu option is selected, its value is assigned the variable `selectedOption`. Note that `selectOption` is the `id` value of the dropdown menu, while `selectedOption` is the option that is chosen by the user.
4. Each time `updatePage()` is triggered, the `id` value of the dropdown menu, as well as the value of the chosen menu option, are printed to the browser console.

When we open the browser, we'll see a dropdown menu with two options.



If we open the console, we'll see that every time we toggle between the two menu options, `selectOption` and the option value are printed to the console:



Suppose that you add the following line of code to the `updatePage()` function. What will it print to the browser console when a dropdown menu item is selected?

```
console.log(d3.selectAll("#menu").node().id);
```

- `menu`
- `id`
- `node()`

Check Answer

Finish ▸

Now test your skills in the following Skill Drill.

Skill Drill

Create a new directory, containing new `index.html` and `script.js` files. Use the `D3.js` library to create an event listener for a dropdown menu.

Your dropdown menu should contain the following names: Mickey, Minnie, Donald, Goofy. When a character (e.g., Minnie) is chosen from the dropdown

menu by a user, the character's name should be printed to the browser console.

12.4.2: Create a Dynamic Plotly Chart

Having become more comfortable with using an event listener with a dropdown menu, Roza is now ready to create a dynamic chart in Plotly.

She'll now create a dynamic line chart: there will be a dropdown menu in the browser with two options. When an option is selected, the browser will display the graph for the dataset associated with that option.

This skill will help Roza with a major task in her project: to create a dashboard in which her volunteers can select their anonymized ID from a dropdown menu in the browser in order to display information about their belly button critters.

We'll first help Roza create a simple dynamic line chart in Plotly. As before, we'll create an `index.html` page with the appropriate links to CDNs and a JavaScript file `(plots.js)`. The page also has a dropdown menu with an `id` of `dropdownMenu`:

```
<!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>Events</title>
  <script src="https://d3js.org/d3.v5.min.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
<body>
  <div id="plot"></div>
  <select id="dropdownMenu">
    <option value="dataset1">DataSet1</option>
    <option value="dataset2">DataSet2</option>
  </select>
  <script src="plots.js"></script>
</body>
</html>
```

Take a moment to examine `plots.js` in detail:

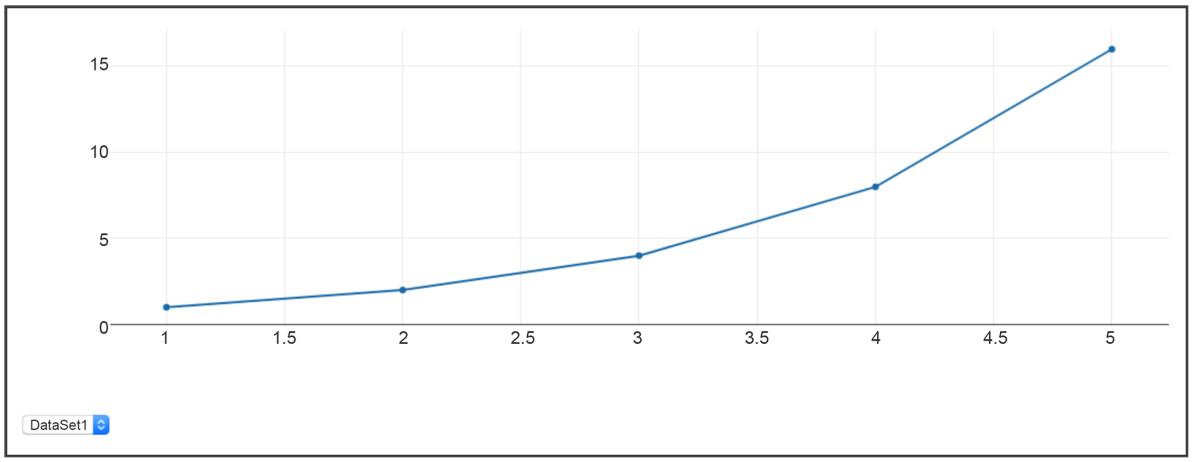
```
function init() {
  data = [{
    x: [1, 2, 3, 4, 5],
    y: [1, 2, 4, 8, 16] }];
  Plotly.newPlot("plot", data);
};

d3.selectAll("#dropdownMenu").on("change", updatePlotly);
function updatePlotly() {
  var dropdownMenu = d3.select("#dropdownMenu");
  var dataset = dropdownMenu.property("value");

  var xData = [1, 2, 3, 4, 5];
  var yData = [];

  if (dataset === 'dataset1') {
    yData = [1, 2, 4, 8, 16];
  };

  if (dataset === 'dataset2') {
    yData = [1, 10, 100, 1000, 10000];
```

```
  };

  var trace = {
    x: [xData],
    y: [yData],
  };
  Plotly.restyle("plot", trace);
};

init();
```

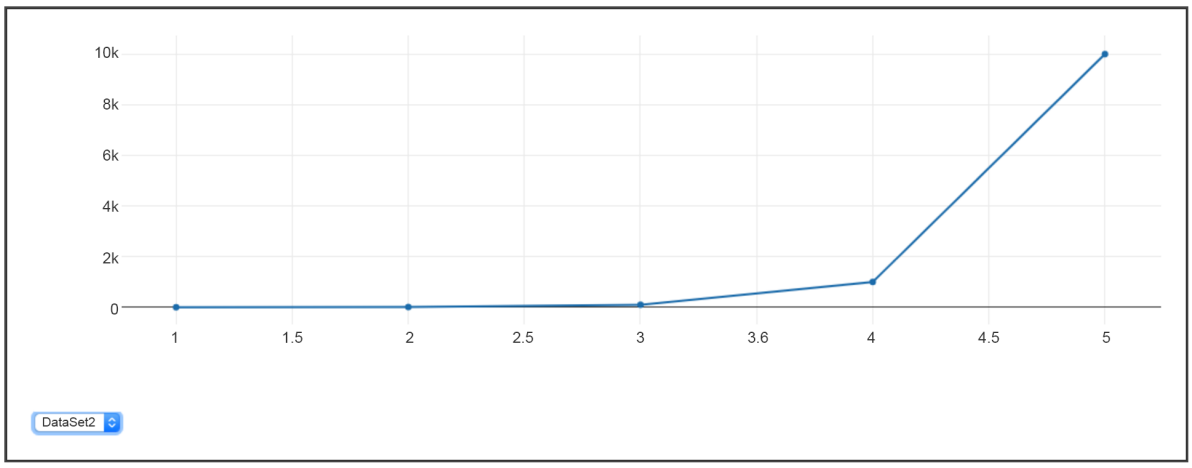
The first function to execute is `init()`, which renders the initial visualization:

```
function init() {
  data = [{
    x: [1, 2, 3, 4, 5],
    y: [1, 2, 4, 8, 16] }];
  Plotly.newPlot("plot", data);
};
init();
```

In this part of the code, a simple line chart, with `x` and `y` axes, is rendered with `Plotly.newPlot()`. By default, when a user opens `index.html` in a browser, this is the chart that is displayed:



Notice, however, that there is a dropdown menu, and that the visualization changes when the `DataSet2` option is selected:



Which function is triggered when a dropdown menu option is selected?

- `init()`
- `Plotly.newPlot()`
- `updatePlotly()`

Check Answer

Finish ▸

When the user first loads the page, `init()` is called, and the initial plot is rendered. However, when the user selects a dropdown menu option, the `updatePlotly()` function is called:

```
d3.selectAll("#dropdownMenu").on("change", updatePlotly);
```

Specifically, through the `d3.selectAll()` function, when a change takes place to the HTML DOM element with the `id` of `dropdownMenu`, the `upDatePlotly()` function is triggered.

Let's now dissect the `updatePlotly()` function:

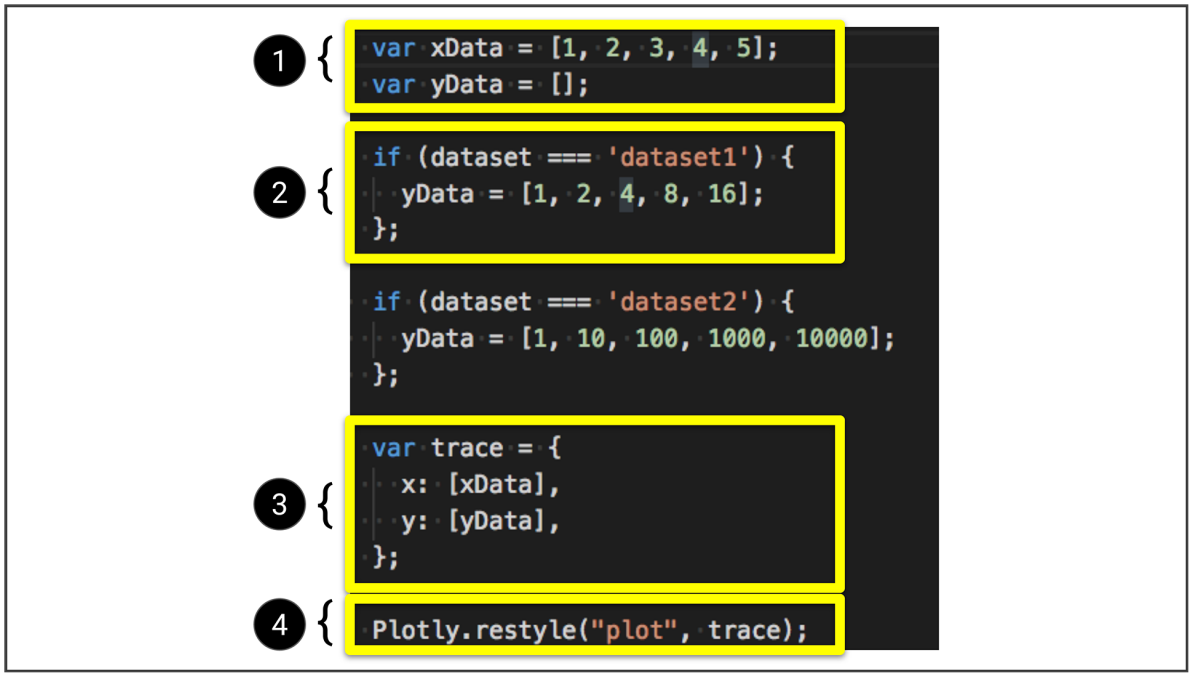
```
function updatePlotly() {
  var dropdownMenu = d3.select("#dropdownMenu");
  var dataset = dropdownMenu.property("value");

  var xData = [1, 2, 3, 4, 5];
  var yData = [];

  if (dataset === 'dataset1') {
    yData = [1, 2, 4, 8, 16];
  };

  if (dataset === 'dataset2') {
    yData = [1, 10, 100, 1000, 10000];
  };

  var trace = {
    x: [xData],
    y: [yData],
  };

  Plotly.restyle("plot", trace);
};
```

The variable `dropdownMenu` is assigned to the DOM element with the `id` of `dropdownMenu`. Recall that it's the dropdown menu from `index.html`:

```
<select id="dropdownMenu">
  <option value="dataset1">DataSet1</option>
  <option value="dataset2">DataSet2</option>
</select>
```

The variable `dataset` is assigned to the value of the dropdown menu option selected by the user. Here, it is either `"dataset1"` or `"dataset2"`.

The rest of `updatePlotly()` function is concerned with switching between two datasets:



Let's break down this code:

1. The x-axis values, or `xData`, remain the same. However, the y-axis values, or `yData`, depend on which dropdown menu option was selected. `yData` is initially a blank array.
2. If the value of the dropdown menu option is `'dataset1'`, `yData` is assigned an array of integers. If `'dataset2'` is chosen, another array of integers is assigned to `yData`.
3. The `xData` and `yData` arrays are assembled inside the trace object. Unlike the `Plotly.newPlot()` method, the `Plotly.restyle()` method defaults to accepting an object (`trace` in this case) as its data argument, rather than an array.
4. The `Plotly.restyle()` method is used to re-render the page on the browser. This method is more efficient than calling the `Plotly.newPlot()` method, as it does not create a brand new chart from scratch, but instead modifies the previously displayed chart with the updated information.

In the line `Plotly.restyle("plot" trace);` , what do the two arguments refer to, respectively?

- A dropdown menu and a JavaScript array.
- A `div` containing a chart and a JavaScript array.
- A `div` containing a chart and a JavaScript object.

Check Answer

Finish ▸

12.4.3: Belly Button Demographics Panel

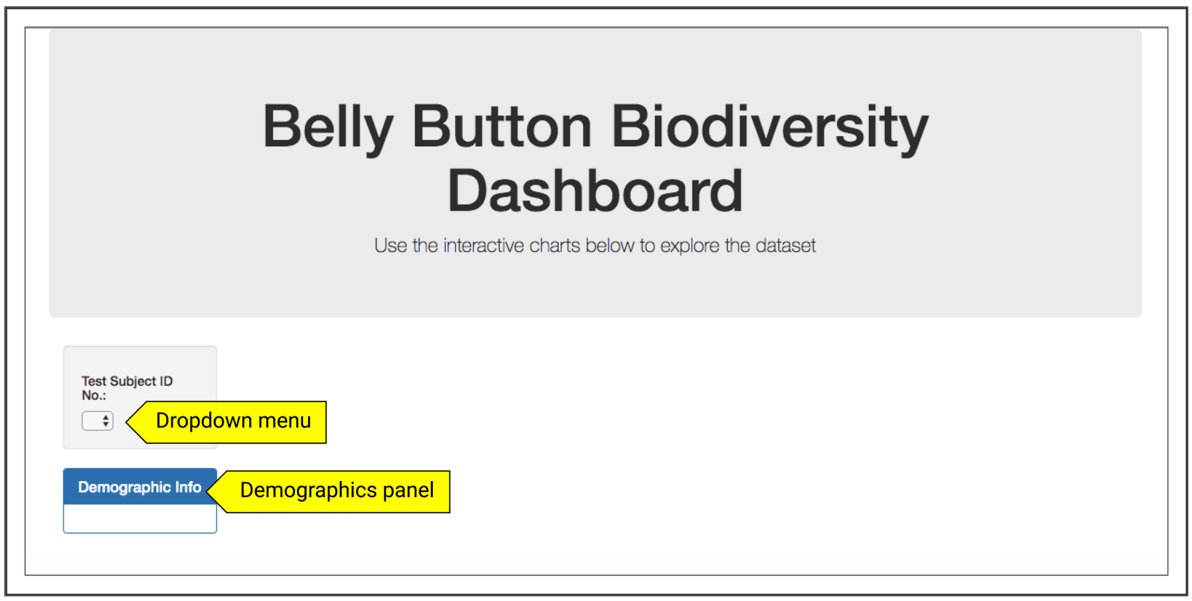
Block by block, Roza has been building her skills to create a visually compelling dashboard. So far, she has learned to create basic static charts with Plotly. She has also learned to read and parse external data files, and to create dynamic charts that respond to user input.

Roza is now ready to consolidate her knowledge in order to build the first part of her dashboard: the demographics panel. A dropdown menu will list the ID numbers of all the volunteers. When a volunteer ID is chosen from the dropdown menu, that person's demographics information, such as location, sex, and age, will be displayed.

Let's first look at the `index.html` document Roza will need by downloading a zip file containing the HTML file.

Download the file (https://courses.bootcampspot.com/courses/138/files/13720/download?wrap=1)

Set Up HTML Elements



This is a Bootstrap page with a dropdown menu and an info panel. Selecting an ID number from the dropdown menu will populate the panel with that person's information.

Open `index.html` to examine the code. As expected, the header links to the Bootstrap CDN.

```
<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>Belly Button Biodiversity</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/
</head>
```

The next block of code creates two HTML elements: the dropdown menu and the panel for the demographic information.

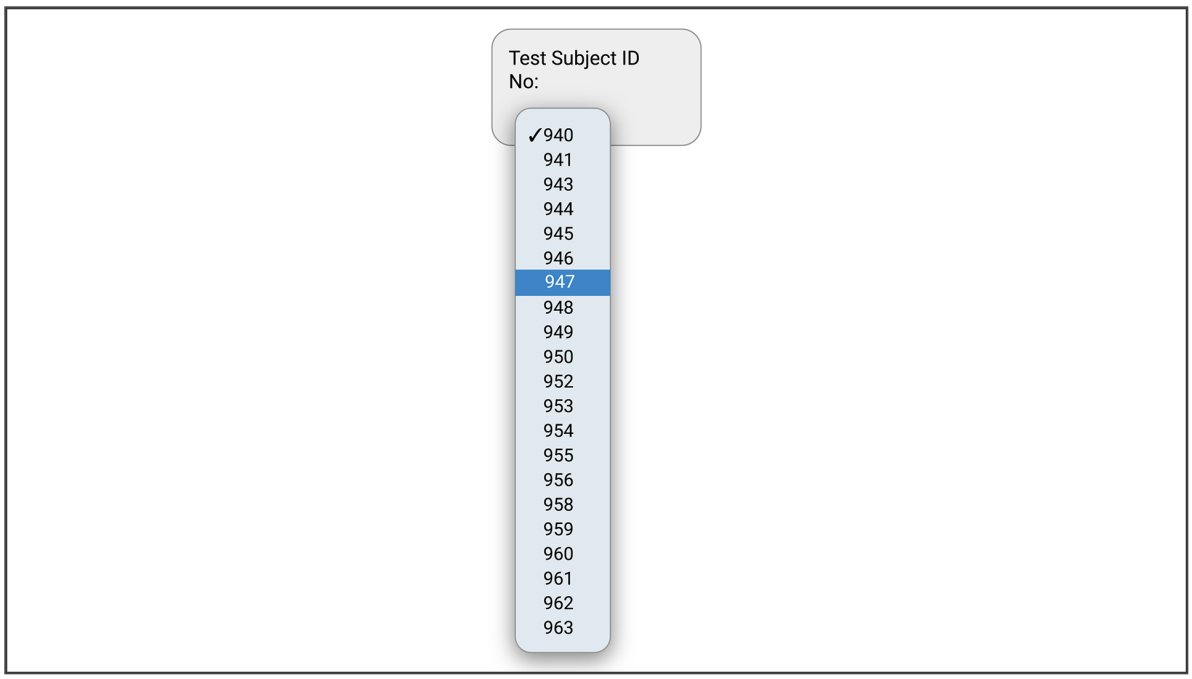
```
<div class="well">
  <h5>Test Subject ID NO.:</h5>
1 {  <select id="selDataset"></select>
</div>
<div class="panel panel-primary">
  <div class="panel-heading">
    <h3 class="panel-title">Demographic Info</h3>
  </div>
2 {  <div id="sample-metadata" class="panel-body"></div>
</div>
```

In this code, there are two key things to highlight:

1. A `<div>` belonging to the `well` class contains a dropdown menu with an `id` of `selDataset`.
2. A `<div>` with an `id` of `sample-metadata` is the information panel.

The dropdown menu doesn't have any options yet, and the panel doesn't contain any demographic information. These will be dynamically generated with information from the dataset.

We'll use the data from `samples.json` to create a dropdown menu of volunteer (test subject) ID numbers dynamically:



Dynamically Generate Dropdown Menu Items

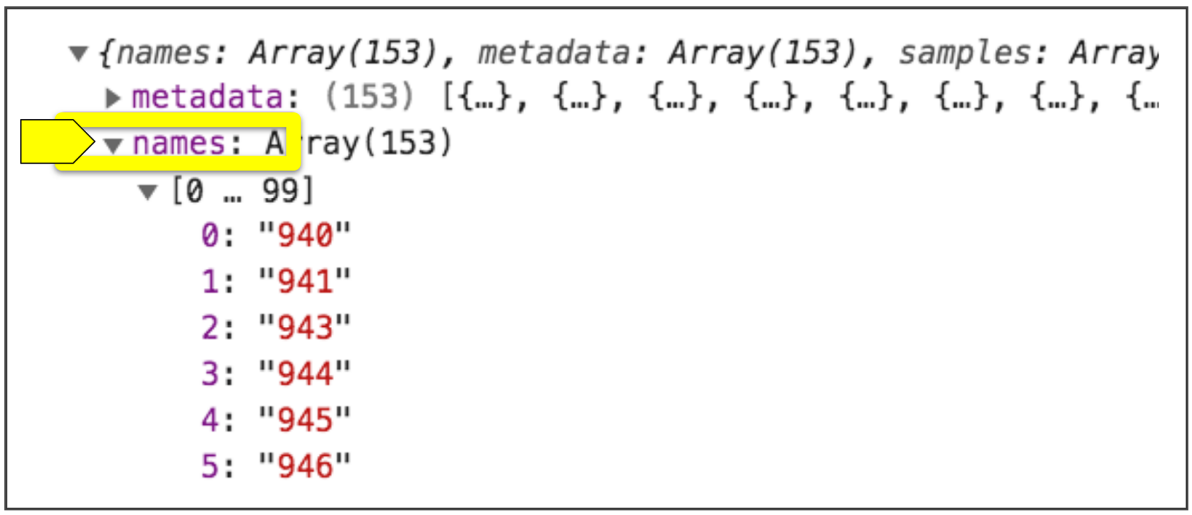
Let's examine the code that creates a dropdown menu of ID numbers dynamically.

```
function init() {
  var selector = d3.select("#selDataset");

  d3.json("samples.json").then((data) => {
    console.log(data);
    var sampleNames = data.names;
    sampleNames.forEach((sample) => {
      selector
        .append("option")
        .text(sample)
        .property("value", sample);
    });
})}

init();
```

Let's break down what this code is doing:

- All the code is enclosed inside the `init()` function, which is called on the last line.
- Inside `init()`, the `d3.select()` method is used to select the dropdown menu, which has an `id` of `#selDataset`. The dropdown menu is assigned to the variable `selector`.
- The `d3.json()` method is used to read the data from `samples.json`. The data from the entire JSON file is assigned the (arbitrary) argument name `data.`
- Inside the `data` object, the `names` array, as seen from `console.log(data)`, contains the ID numbers of all the study participants. The variable `sampleNames` is assigned to this array.

Open the browser console to examine the `names` array. As expected, it is an array of ID numbers.



In this code, note that the `forEach()` method is called on the `sampleNames` array. For each element in the array, a dropdown menu `option` is appended. The `text` of each dropdown menu option is the ID. Its `value` property is also assigned the ID.

For example, ID `"940"` is the first element of the `sampleNames` array. As the `forEach()` method iterates over the first element of the array, a menu option is appended to the dropdown menu. It is then given the text (the text seen in the dropdown menu) `"940"`, and its property is also assigned `"940"`. The `forEach()` method will perform the same tasks for the next element of the array, `"941"`.

Up to this point, the dropdown menu examples we have seen had hard-coded menu options. For example, in the code below, there are two dropdown menu options, for which each `value` property can be selected with JavaScript:

```
<select id="dropdownMenu">
  <option value="dataset1">DataSet1</option>
  <option value="dataset2">DataSet2</option>
</select>
```

With the belly button data, the dropdown menu options are generated dynamically. However, the HTML code can be modified to call a JavaScript function:

```
<select id="selDataset" onchange="optionChanged(this.value)"></select>
```

In this code, note the following:

- The `<select>` tag indicates a dropdown menu. Its `id` is `selDataset`.
- The `<select>` tag now has an additional attribute, called `onchange`, which is associated with the `optionChanged()` function.
- When a change takes place in the dropdown menu, the `optionChanged()` function is called.
- The argument for the `optionChanged()` function is `this.value`.
- Here, `this` refers to the dropdown menu. `this.value` therefore returns to the value attribute of the current dropdown menu selection.

> **Rewind**
>
> The JavaScript keyword `this` is used to access the object in question. In the context of an event, it refers to the HTML element that received the event. In this case, `this` refers to the dropdown menu.

In the following code example, what is doSomething?

```
<select id="myDropdownMenu"
onchange="doSomething(this.value)"></select>
```

- ○ The name of a JavaScript function that is defined in a separate JavaScript file.
- ○ An HTML function that is triggered by an event.
- ○ A JSON attribute.

Check Answer

Finish ▸

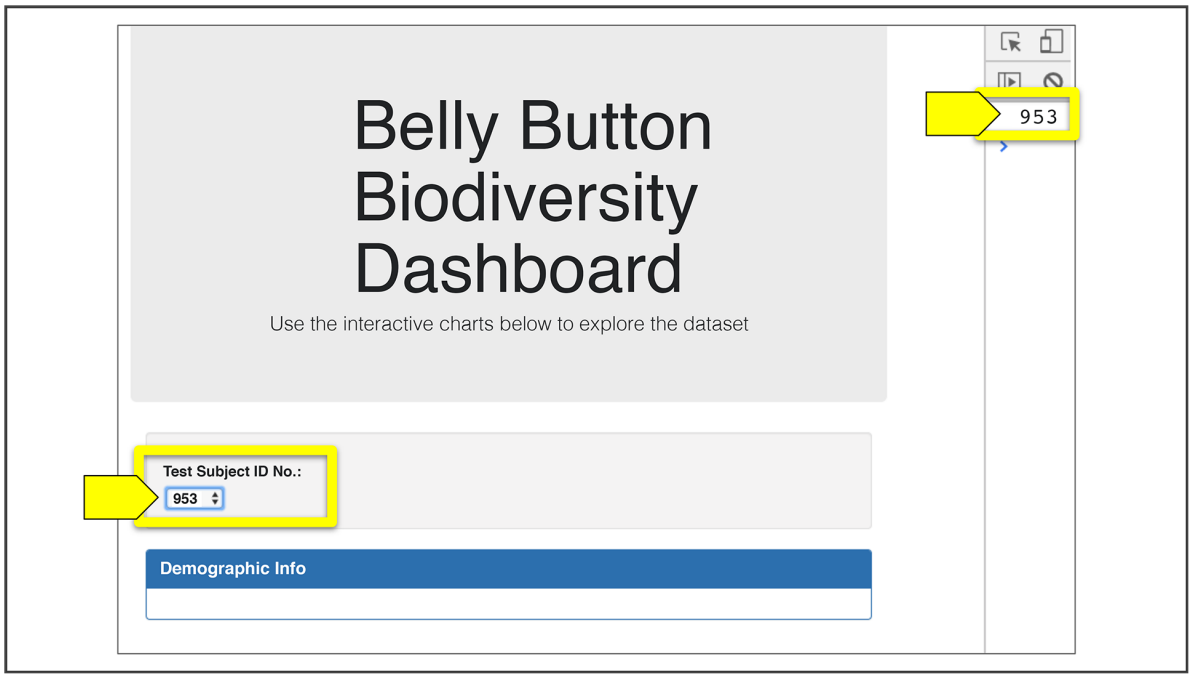
Since the function `optionChanged()` does not yet exist in `plots.js`, let's add it:

```
function optionChanged(newSample) {
  console.log(newSample);
}
```

It takes in an argument, named `newSample`, and logs it to the browser console. Note the following:

- This function is declared in `plots.js`, but it is never called in `plots.js`. It's instead called by the `onchange` attribute of the dropdown menu in `index.html`. Compare this to the `init()` function, which is both declared and called in `plots.js`.
- The argument name `newSample` refers to the value of the selected menu option. In `index.html`, `onchange=optionChanged(this.value)` passes the selected menu option's value to the `optionChanged()` function. This function gives this information the argument name `newSample`. In other words, `this.value` and `newSample` are equivalent.

When an option is selected from the menu, its value is printed to the browser console:



The next task is to print information to the `Demographic Info` panel: once a user selects an ID number, the associated volunteer's demographic information needs to be filtered from `samples.json` and placed in the panel.

When a change takes place to the dropdown menu, two things will need to occur:

1. The demographic information panel is populated with a specific volunteer's information.
2. The volunteer's data is visualized in a separate `div`.

These tasks should be modularized—that is, the code statements required to perform each task should be packaged as a separate function. When a change takes place to the dropdown menu, each function should be called. Notice that in the `optionChanged()` function, `console.log(newSample);` has been replaced with two function calls:

```
function optionChanged(newSample) {
  buildMetadata(newSample);
  buildCharts(newSample);
}
```

Recall that `optionChanged()` is called from the HTML document and, in turn, calls `buildMetadata()` and `buildCharts()`. The argument, `newSample`, is the volunteer ID number that is passed to both of these functions. These two functions will use the ID number to create that specific individual's information panel and charts, respectively.

Let's now declare the first of these functions: `buildMetadata()`.

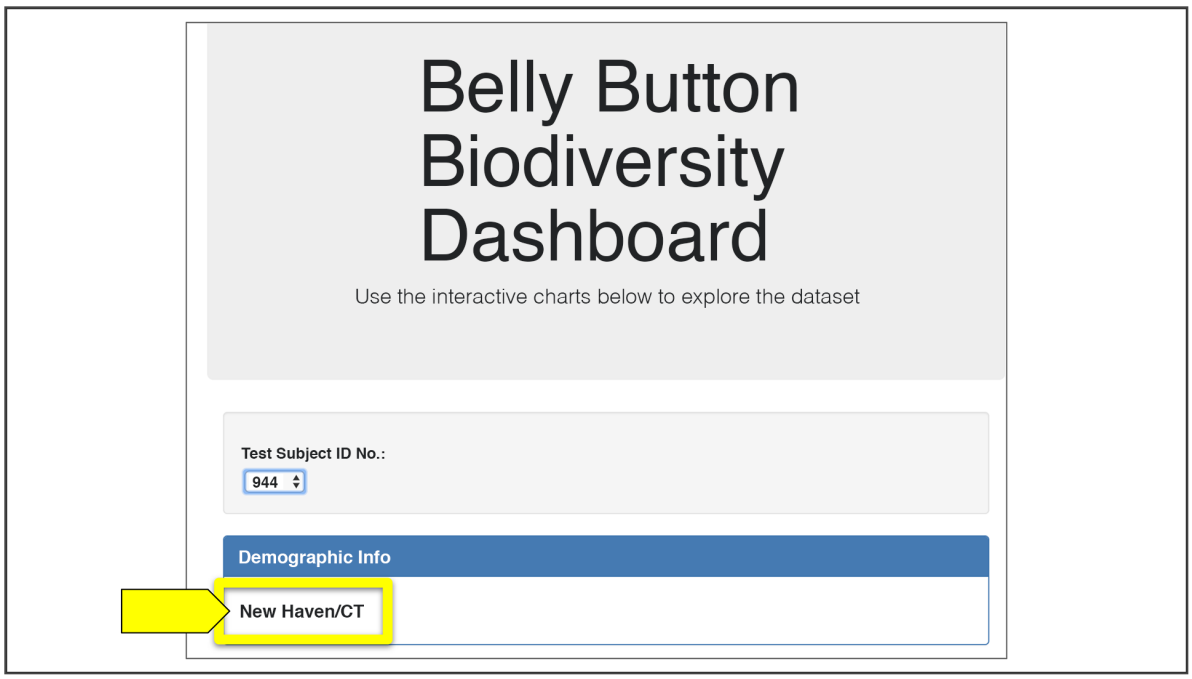
```
function buildMetadata(sample) {
  d3.json("samples.json").then((data) => {
    var metadata = data.metadata;
    var resultArray = metadata.filter(sampleObj => sampleObj.id == sample
    var result = resultArray[0];
```

```
    var PANEL = d3.select("#sample-metadata");

    PANEL.html("");
    PANEL.append("h6").text(result.location);
  });
}
```

Here's a breakdown of what's happening in this code:

- The function `buildMetadata()` takes in sample, or an ID number, as its argument. That is, when a dropdown menu option is selected, the ID number is passed in as `sample`.
- Then `d3.json()` pulls in the entire dataset contained in `samples.json`. Once the dataset is read in, it is referred to as `data`.
- The `metadata` array in the dataset (`data.metadata`) is assigned the variable `metadata`.
- Then the `filter()` method is called on the `metadata` array to filter for an object in the array whose `id` property matches the ID number passed into `buildMetadata()` as `sample`. Recall that each object in the `metadata` array contains information about one person.
- Because the results of the `filter()` method are returned as an array, the first item in the array (`resultArray[0]`) is selected and assigned the variable `result`.
- The `id` of the `Demographic Info` panel is `sample-metadata`. The `d3.select()` method is used to select this `<div>`, and the variable `PANEL` is assigned to it.
- `PANEL.html("")` ensures that the contents of the panel are cleared when another ID number is chosen from the dropdown menu.
- Finally, the `append()` and `text()` methods are chained to append a H6 heading to the panel and print the location of the volunteer to the panel, respectively.

This is what the result looks like:



Which JavaScript method is used to iterate through the keys and values of an object?

- `Object.entries()`
- `map()`
- `filter()`

Check Answer

Finish ▸

Which data structure type is returned by the `Object.entries()` method?

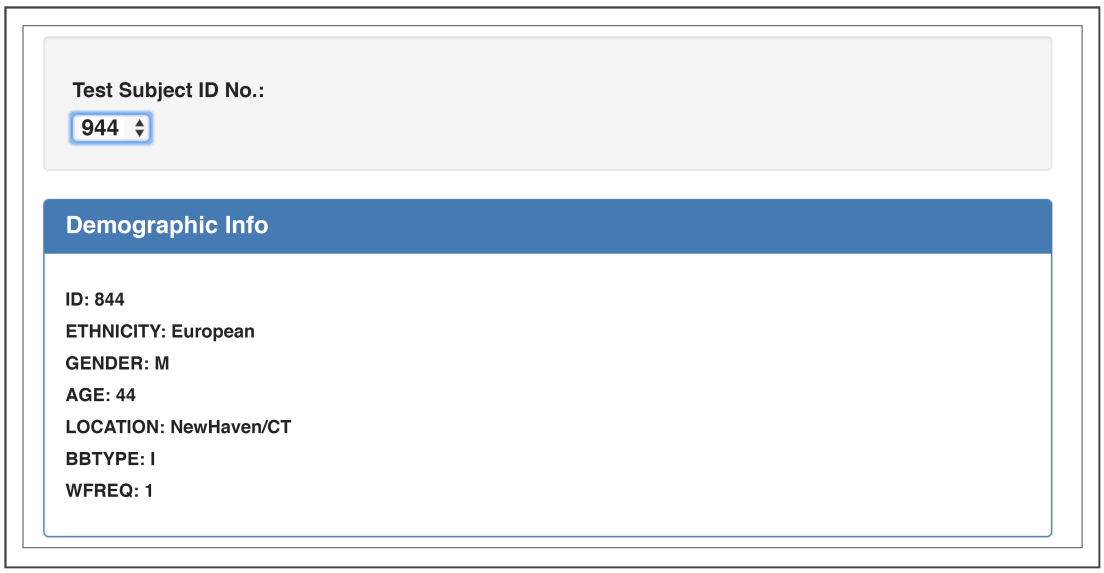
- Dictionary
- Array
- Object
- JSON

Check Answer

Finish ▸

Skill Drill

Open VS Code and modify the `buildMetadata()` function to populate the `Demographic Info` panel with the rest of the demographic data when a menu option is selected:



Great job! Roza is now well-placed to complete the rest of the dashboard.

12.5.1: Deploy the Project to GitHub Pages

Roza has come a long way. She's learned to create attractive charts in Plotly, as well as use JavaScript to make these charts dynamic and customizable. Furthermore, Roza has learned how to filter the belly button dataset and to display information specific to each volunteer.

Roza also knows about cross-origin resource restrictions on browsers, and understands that a server app must give the same web address to the data file as well as the associated HTML, CSS, and JavaScript files.

Once Roza completes the belly button data dashboard, she'll have one major task left: sharing it with the rest of the world. Running a local server allows Roza to test her code on the specific computer, but deploying her project onto a public server enables her to share it with her colleagues as well as the study participants. Let's help Roza with the steps necessary to complete this step.

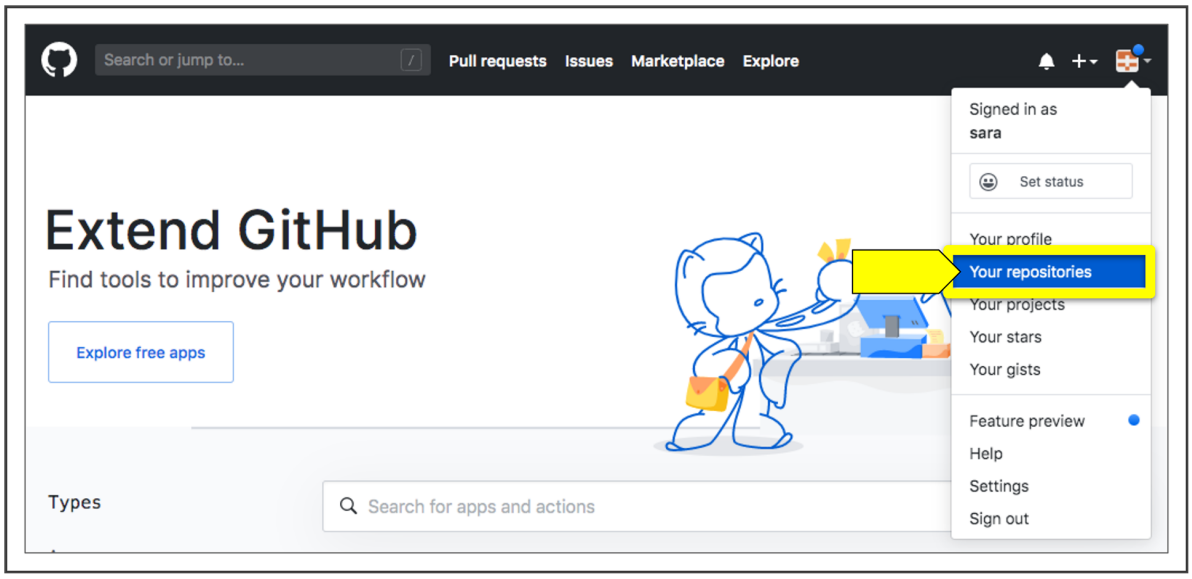
We will deploy a sample project to GitHub Pages. Click below to download the project files.

Download the data files
(https://courses.bootcampspot.com/courses/138/files/13714/download?wrap=1)

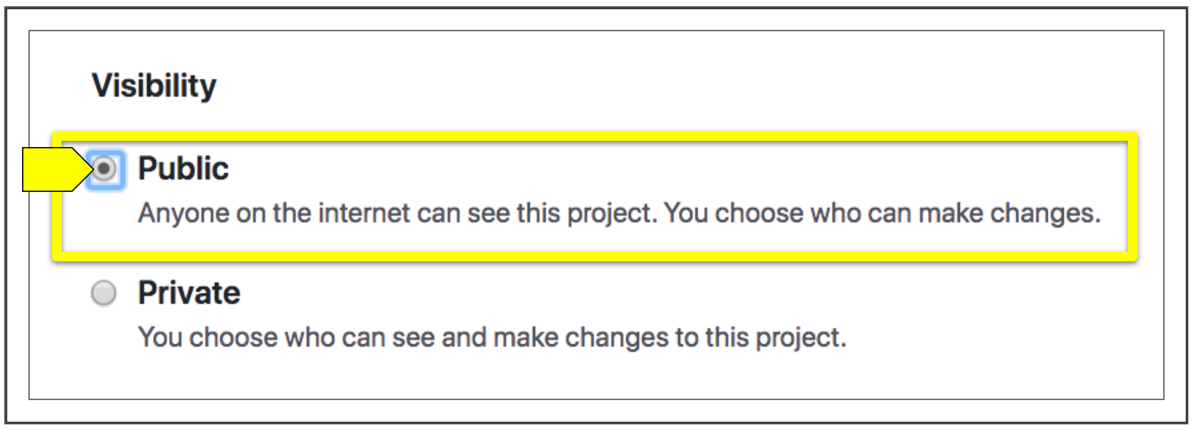
We have previously used GitHub to make our code available to the public and to work on projects collaboratively. GitHub offers a free service called GitHub Pages to deploy projects like Roza's—that is, to run code on a GitHub server to make it viewable to the public. Without a server, anyone who wants to view Roza's dashboard would need to download the project files and run a local server, such as Python's HTTP server.

We'll deploy a sample project (the code you just downloaded) to GitHub Pages. Behind the scenes, GitHub Pages runs a server app for your project, much like Python's HTTP server. However, whereas the local Python server makes the dashboard available only for the user of the specific computer on which it is run, GitHub Pages makes it available to anyone with an internet connection and a web browser. First, we'll use the sample project code to walk through the steps of deploying a project to GitHub Pages. Then we'll repeat these steps to deploy the belly button dashboard.

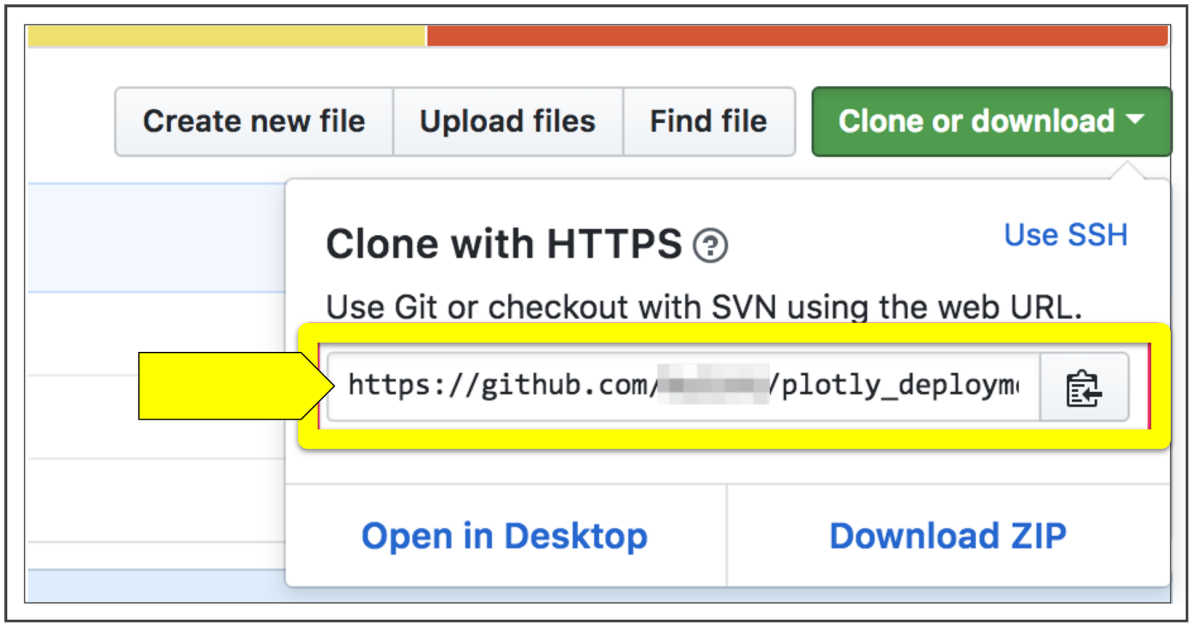
First, to create a new repository on GitHub, navigate to **https://github.com/ (https://github.com/)** . Click the arrow next to your profile icon, and then select "Your repositories."



Select the option to make the repository public:



Next, clone the repository by copying the URL:



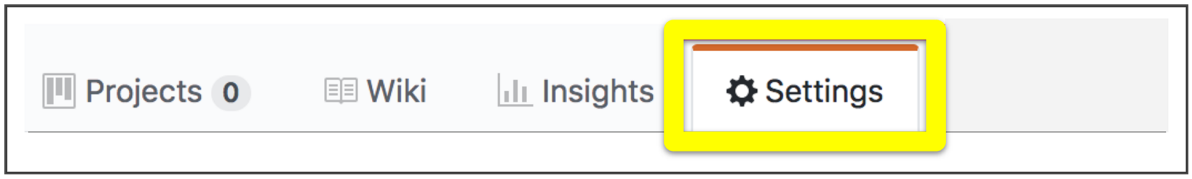
Type `git clone` in the command line and press Enter.

```
$ git clone https://github.com/ .git
```

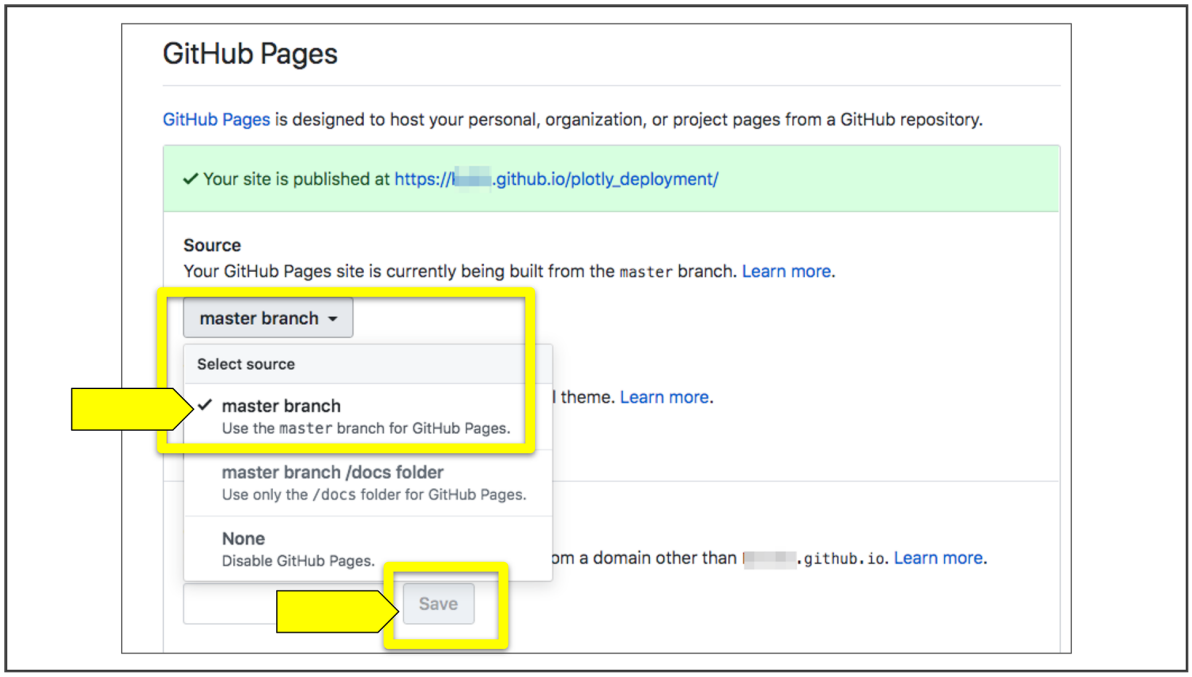
Next, push the project to GitHub by following these steps:

1. In the command line, navigate to the newly created repository directory.
2. Copy and paste the HTML, JavaScript, and JSON files that you downloaded into the repository.
3. In the command line, run `git add .`
4. Run `git commit -m ""`.
5. Run `git push origin master`.

Then navigate to your project page on GitHub and click on Settings to configure for deployment:

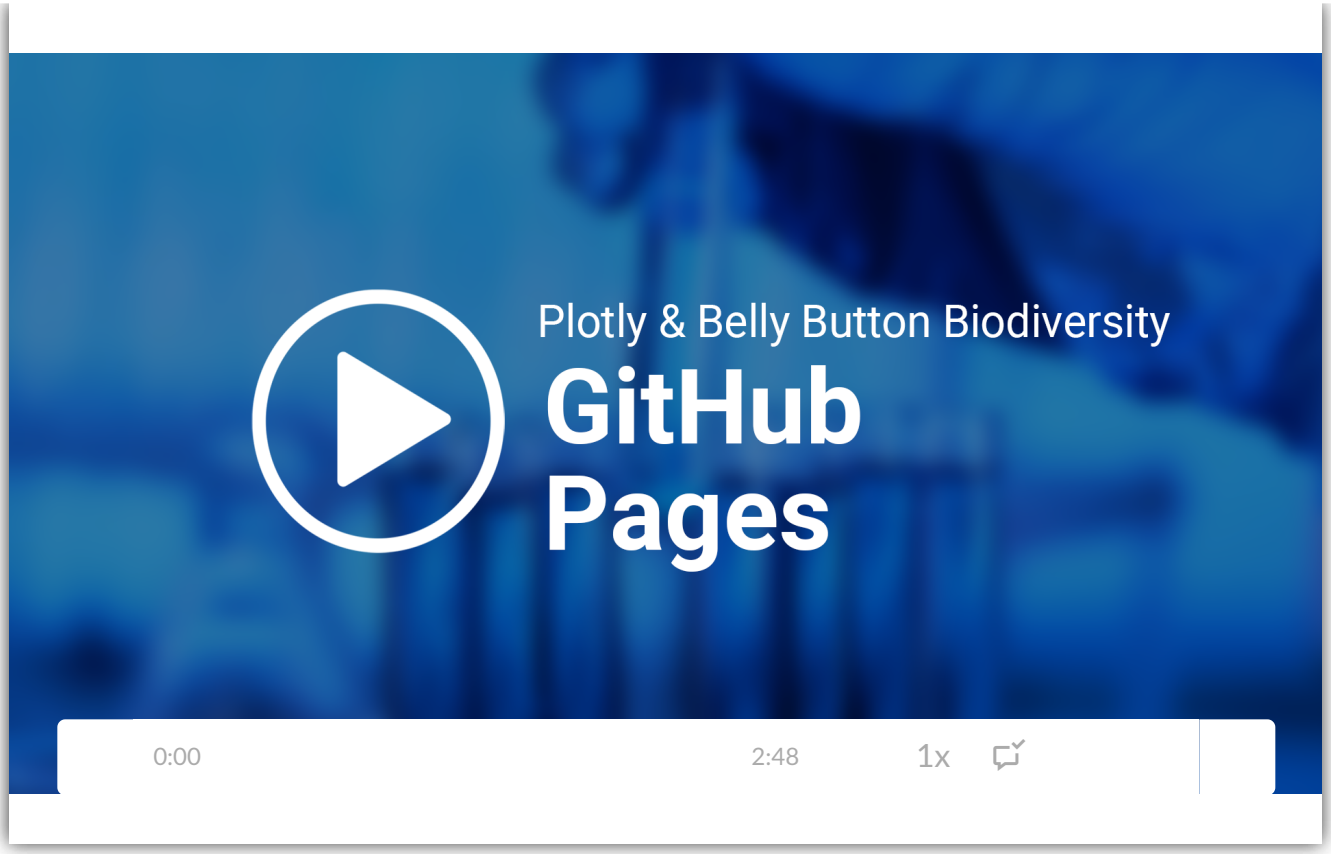


Scroll down the Settings page to the GitHub Pages menu. Select “master branch” from the dropdown menu, and then click Save.

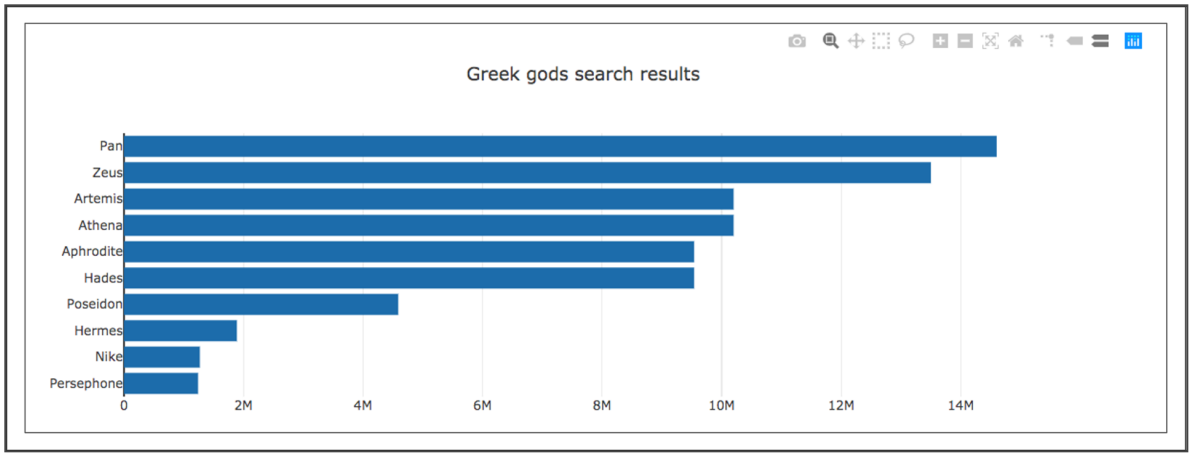


The project should now be deployed to GitHub Pages. You can confirm this by checking that the URL of the deployed page is listed under the GitHub Pages options. You can also navigate to the URL of the deployed page, which will be in this format: `<your account name>.github.io/<your repository name>`. For example, if your GitHub account name is `data_student`, and you named your repository `plotly_chart`, the URL of the deployed page would be `data_student.github.io/plotly_chart`.

Here is a video of the previous steps:



This is what you should see when you navigate to the URL:



Congratulations! You have now successfully deployed a project to GitHub Pages. Now the entire world will be able to access your data visualizations and insights. You've acquired a broad range of skills, from using Plotly to create visualizations to deploying them on the web. You are now ready to take on the challenge.

Module 12 Challenge

Submit Assignment

Due Apr 5 by 11:59pm **Points** 100 **Submitting** a text entry box

> Roza feels comfortable with all the elements she'll use to complete her dashboard, from Plotly to HTML to JavaScript to deployment. You'll now help her finish the rest of the dashboard.

In this challenge, you will complete the belly button dashboard and deploy a working version to GitHub Pages.

Background

Roza has a partially completed dashboard, but she needs to finish it. She has a completed panel for demographic information and now needs to visualize the bacterial data for each volunteer. Specifically, her volunteers should be able to identify the top 10 bacterial species in their belly buttons. That way, if Improbable Beef identifies a species as a candidate to manufacture synthetic beef, Roza's volunteers will be able to identify whether that species is found in quantity in their navel.

Objectives

The goals of this challenge are for you to:

- Create a bar chart of the top ten bacterial species in a volunteer's navel. Use JavaScript to select only the most populous species.

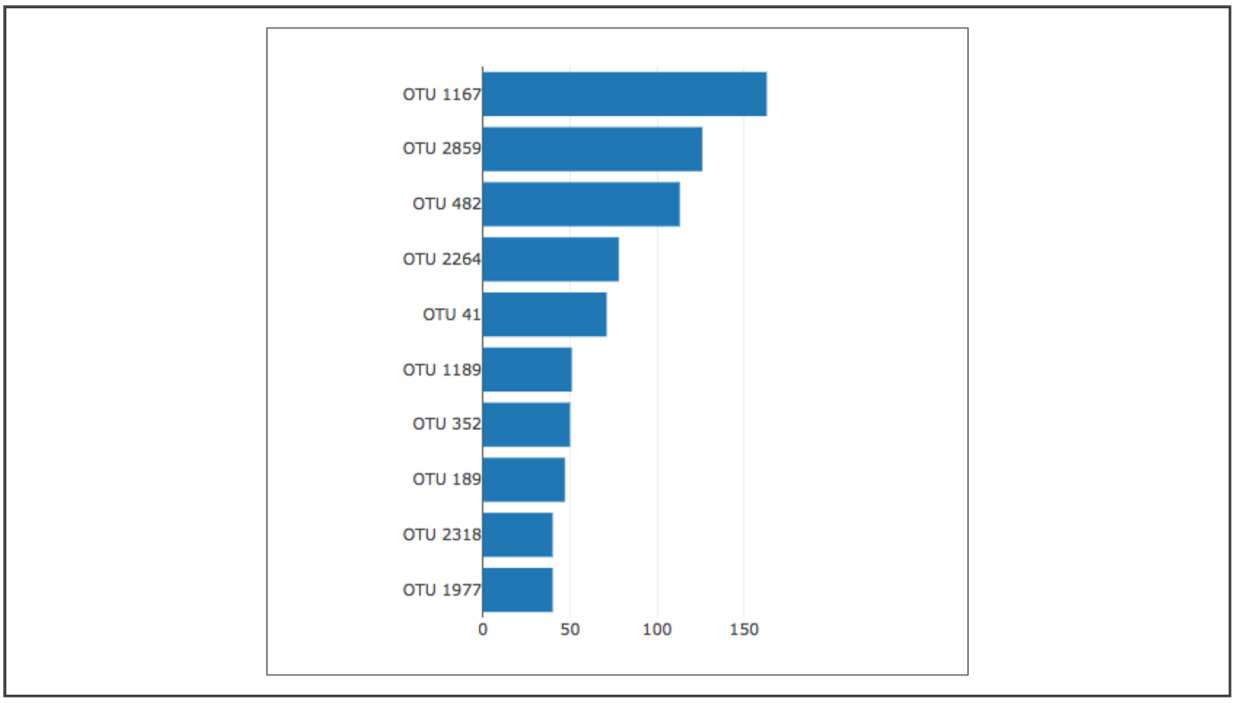
- Create a bubble chart to visualize the relative frequency of all the bacterial species found in a volunteer's navel.
- Complete the demographic information panel, if you have not done so.

Instructions

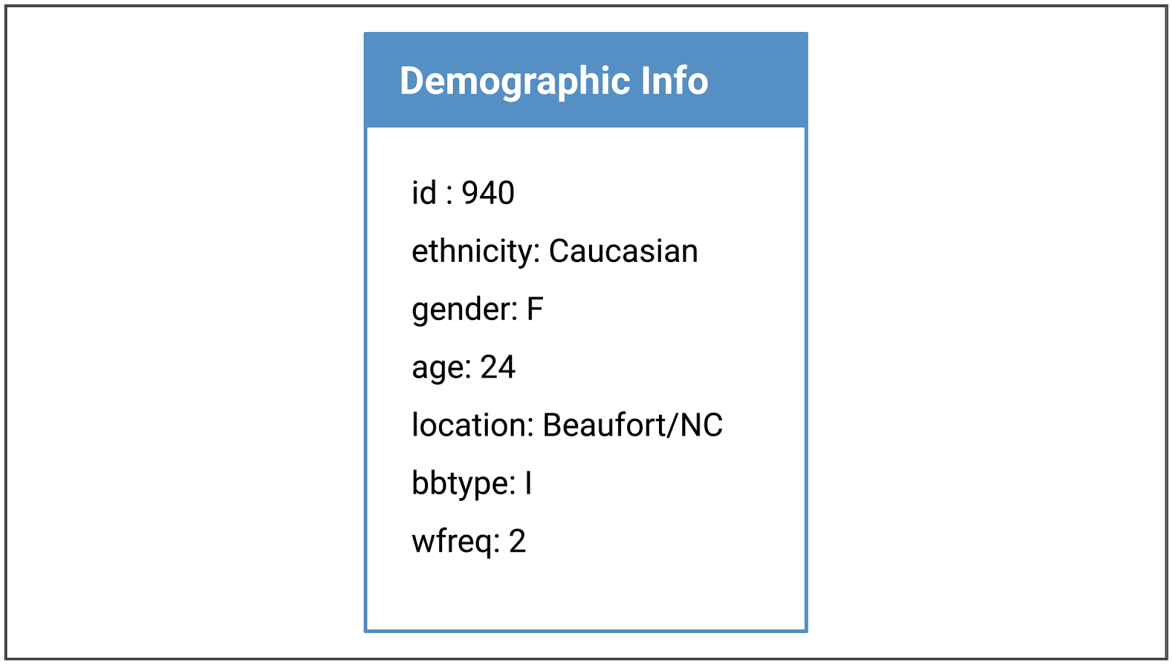
Continue working with the `samples.json` dataset, and complete the following tasks.

- When an individual's ID is selected, the top 10 bacterial species (OTUs) should be visualized with a bar chart. Create a horizontal bar chart to display the top 10 OTUs found in that individual.
  - Use `sample_values` as the values for the bar chart.
  - Use `otu_ids` as the labels for the bar chart.
  - Use `otu_labels` as the hover text for the chart.

Your bar chart should look like the following.



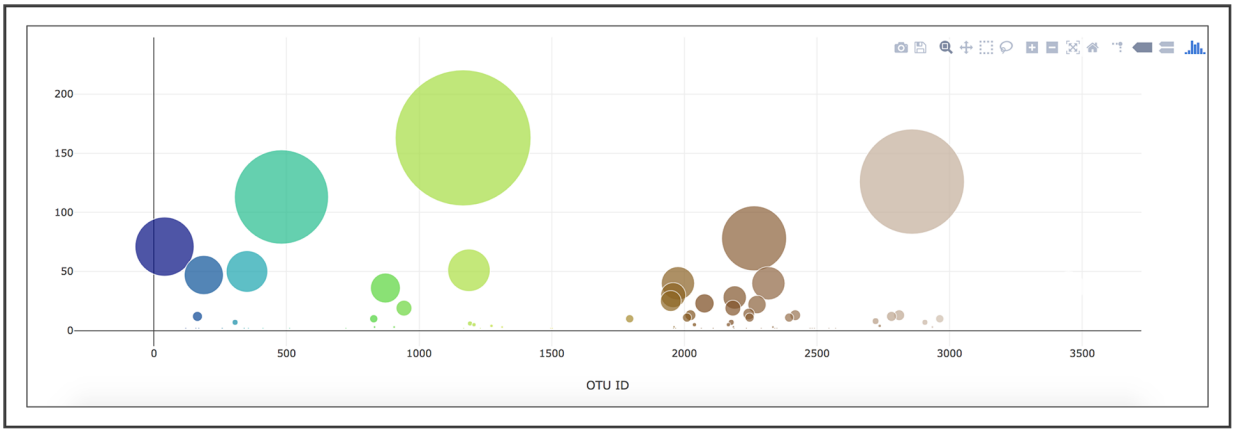
- In the Demographics Info panel, display all the key-value pairs of the selected individual's demographic data. The result should look like the following:



- Create a bubble chart that displays each sample:
  - Use `otu_ids` for the x-axis values.
  - Use `sample_values` for the y-axis values.
  - Use `sample_values` for the marker size.
  - Use `otu_ids` for the marker colors.
  - Use `otu_labels` for the text values.

Hint: For more information about creating a bubble chart, see the **Plotly documentation (https://plot.ly/javascript/bubble-charts/)**.

Your bubble chart should look like this:

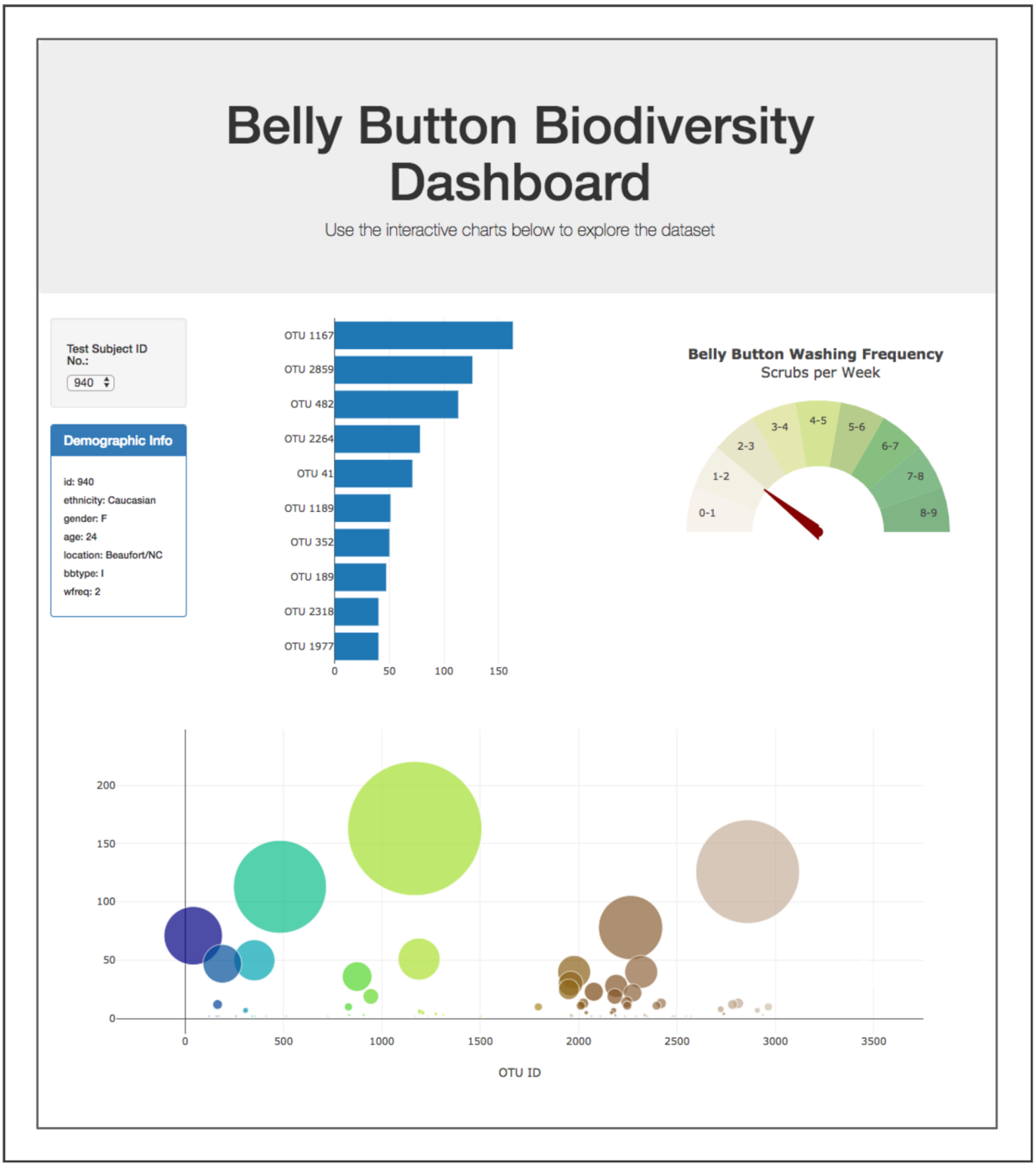


- When the dashboard is first opened in a browser, ID 940's data should be displayed in the dashboard. In other words, the dashboard should not be blank

when a user opens it in a browser.

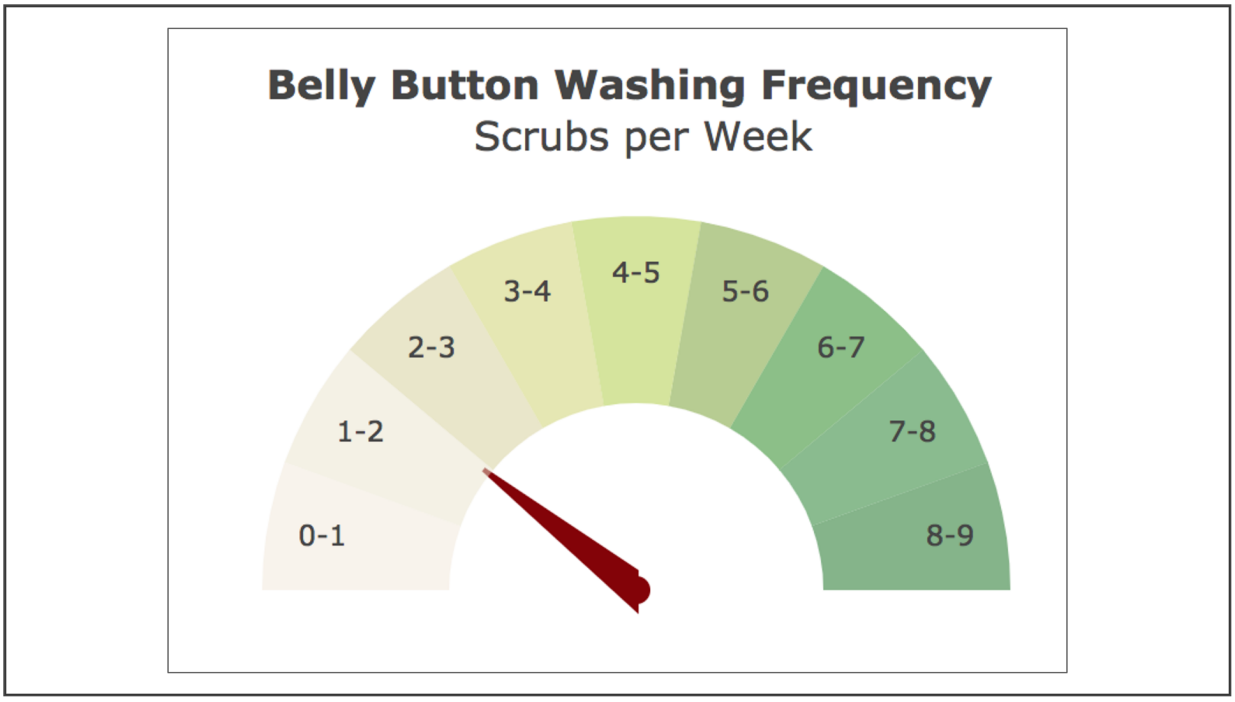
- When a new ID number is selected from the dropdown menu, all the plots and the info panel should be updated.
- Customize the layout to create an attractive dashboard. Use your imagination!

The completed project should resemble the following dashboard, though it should not be identical:



Extension

Adapt the gauge chart from **Plotly documentation (https://plot.ly/javascript/gauge-charts)** to plot the weekly washing frequency of the individual. You will need to modify the example gauge code to account for values ranging from 0 through 9. Update the chart whenever a new sample is selected. Your gauge chart should look like this:



Rubric

Please **download the detailed rubric** to access the assessment criteria.

Submission

Deploy your app to a free static page hosting service. GitHub Pages is recommended. Then, submit the link to the webpage (your deployment) as well as a link to your GitHub repository by pasting both links in the text box. Note that you must include both links in your submission.

Make sure your repo is up to date and contains the following:

- A `README.md` file describing your project
- The coding files created during the module:
  - `index.html`
  - `samples.json`
  - `plots.js`
- A link to the deployed page for your portfolio

Note: You are allowed to miss up to two Challenge assignments and still earn your certificate. If you complete all Challenge assignments, your lowest two grades will be dropped. If you wish to skip this assignment, click Submit then indicate you are skipping by typing “I choose to skip this assignment” in the text box.

Module 12 Rubric						
Criteria	**Ratings**					**Pts**
Bar Chart Please see detailed rubric linked in Challenge description.	**35.0 pts** **Mastery**	**26.0 pts** **Approaching Mastery**	**17.0 pts** **Progressing**	**9.0 pts** **Emerging**	**0.0 pts** **Incomplete**	35.0 pts
Bubble Chart Please see detailed rubric linked in Challenge description.	**35.0 pts** **Mastery**	**26.0 pts** **Approaching Mastery**	**17.0 pts** **Progressing**	**9.0 pts** **Emerging**	**0.0 pts** **Incomplete**	35.0 pts
Metadata and Deployment Please see detailed rubric linked in Challenge description.	**30.0 pts** **Mastery**	**23.0 pts** **Approaching Mastery**	**16.0 pts** **Progressing**	**9.0 pts** **Emerging**	**0.0 pts** **Incomplete**	30.0 pts
						Total Points: 100.0

Module 12 Career Connection

Introduction

Congratulations! This week you learned how to work with Plotly, `D3.json()`, as well as how to parse data in JSON format, using JavaScript to create and deploy interactive charts to GitHub Pages. Feeling a little overwhelmed? That's okay and is to be expected. In this section, we're going to review how all of this new material is applicable to your new career.

You might remember from an earlier Career Connection that most of our students who obtain **Employer Competitive** status are successful in getting positions after their boot camp has ended. How you can use the material you've learned this week to become more employer competitive?

1. **Deploy your own interactive chart to GitHub.** Pick a topic that you are interested in and research an API that will give you the data that you'll need. A great place to get started is **Rapid API (https://rapidapi.com/?utm_source=google&utm_medium=cpc&utm_campaign=1674315309_7653915426 9&utm_term=rapid%20api_e&utm_content=1t1&gclid=CjwKCAjwxt_tBRAXEiwAENY8hYl80EA4_3yhyyHyL6r5SKp6M2vwXPCtGXomEZLF4UERT6aL8LAIOBoCzQQQAvD_BwE)** or **Data.gov (https://www.data.gov/developers/apis)** to find some cool data to play with. Once you have the API you need, use what you've learned in this module to pull the data, format it, and create some interactive charts. Deploy it to GitHub Pages and add in some text to explain what you're doing and why.

2. **Link a PDF of your resume to your portfolio or GitHub page.** Use your portfolio as an opportunity to market yourself. If you haven't done so already, add a button to your portfolio that allows prospective employers to download a copy of your resume. Also, you might consider adding a section on how to contact you and/or get your resume on your newly deployed interactive chart page.
3. **Update your resume with your new skillset.** If you haven't done so already, add Plotly and D3 to your resume in the "Technical Skills" section. Display it loud and proud!

Technical Interview Preparation

Technical interviewing is a skill that requires constant practice and development. Moreover, being tested on your knowledge of a particular topic in an interview can be a nerve-wracking experience. Here are some common interview questions to consider. Before viewing the answer to each question, jot down some things you might say in response.

Common Technical Interview Questions

Q: How would you create a bar chart with Plotly.js to display data?

A: Remember, you aren't expected to remember the exact syntax—employers know that you can look this up when you're working. But you should have an idea of how to conceptually sketch out how this might work. For example, you might touch on the following points:

- How you would fetch data from the API (`d3.json()` to send an HTTP request and load `.json` file)
- Use JavaScript methods and Math object to massage the data into the appropriate JSON format
- Use Plotly.js `newPlot` method to create the bar chart, passing in the JSON for the data

While this isn't a complete answer, it does give the employer a solid idea that you know what you're talking about. Remember, concepts are better than syntax details, but the details are a great bonus if you can remember them.

Q: How would you use the JavaScript method `.filter()` to manipulate data?

A: JavaScript's `.filter()` method is used to essentially filter out the data you don't want. It will return all items from the array that match the given filter statement.

Case Study: NBA Shot Data

If you've ever watched a sports game on TV, you may have noticed the prevalence of data. Before, during, and after the game, viewers are bombarded with enormous amounts of data, such as predictions based on data trends or performance comparisons of individual players and teams.

Imagine that you've just been hired on by your favorite NBA team, your dream job. Your new boss points out that a lot of draft decisions and game strategies have been based on the coaches' gut instincts: who they feel is a good player, who makes the best shots, and who is a better player in comparison to another on the team.

For example, Player X is regularly played on the court as a starter because he usually scores 30–40 points per game. However, a quick look at the data shows he is actually making only about 35% of all the shots he takes—so, in reality, he's missing a lot of opportunities! Corporate management has been pushing for more data-driven decision making in order to maximize the team's performance.

After reading this prompt, the technical interviewers ask the following questions:

Q: What data points would you want to track in order to push data-driven decision making for the coaches?

A: There's no one correct answer for this question, but some data points that would probably be worth tracking are percentage of shots made, the spots where shots are made from, and the percentage of shots made from those various spots.

Q: How would you visually display this data in an attractive and useful format for corporate management that are not data scientists?

A: Again, this isn't the kind of question that has a right or wrong answer. The goal here is to think about the various ways in which you might visualize the data using an interactive chart and your justifications for doing so. In this scenario, you might consider an interactive view of a basketball court with the points mapped out where the shots are taken from. Each of those points might vary in color intensity —the lighter the point, the lower the percentage of shots made.

Continue to Hone Your Skills

If you're interested in hearing more about the technical interviewing process and practicing algorithms in a mock interview setting, check out our **upcoming workshops (https://careerservicesonlineevents.splashthat.com/)** .