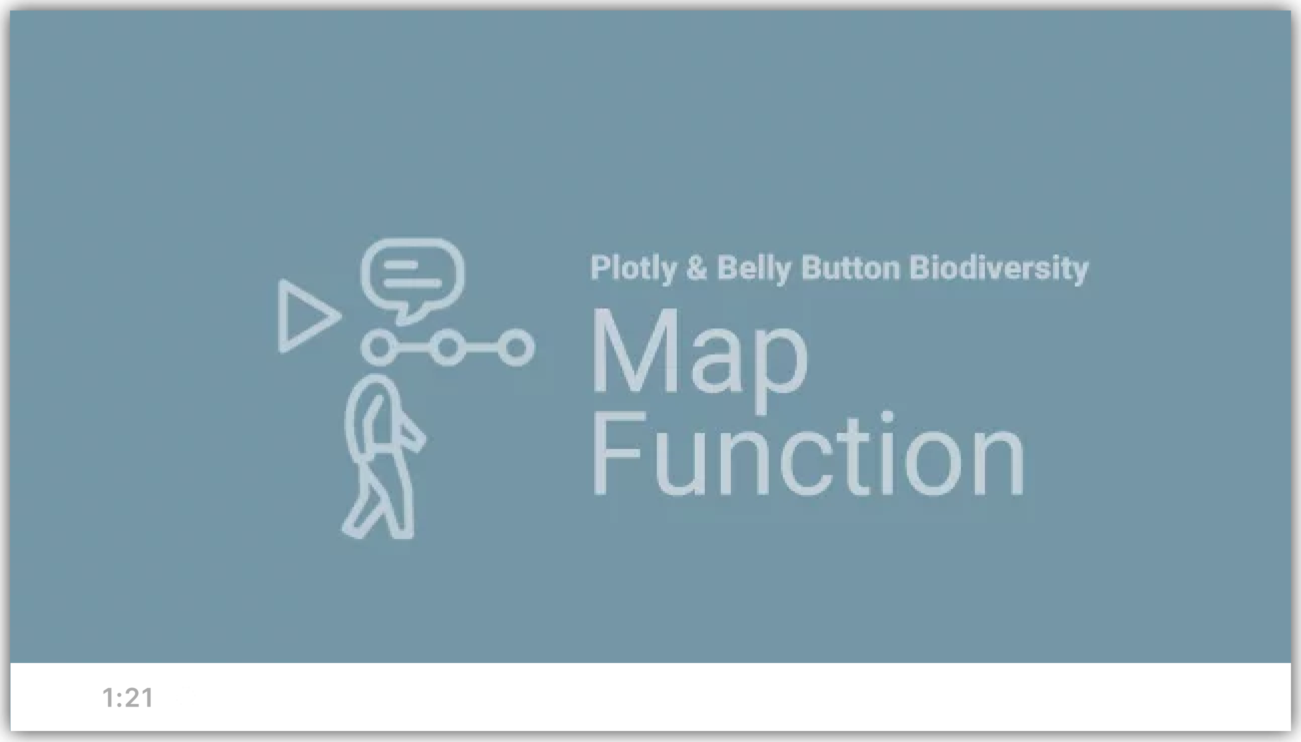
12.2.1

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);
```

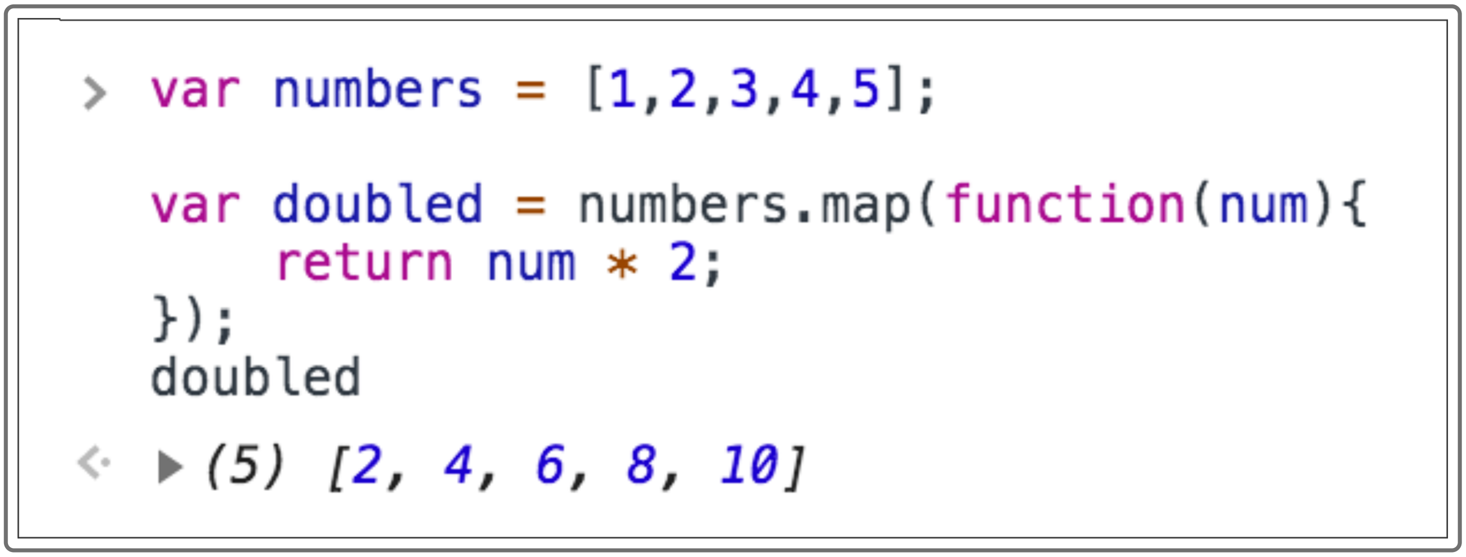
In this code, an array named `numbers` contains five integers: `var numbers = [1,2,3,4,5];`. Let's break down the rest of the code in more detail:

- 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;
});
```

Retake

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.

```
var 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"
    }
];

var 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

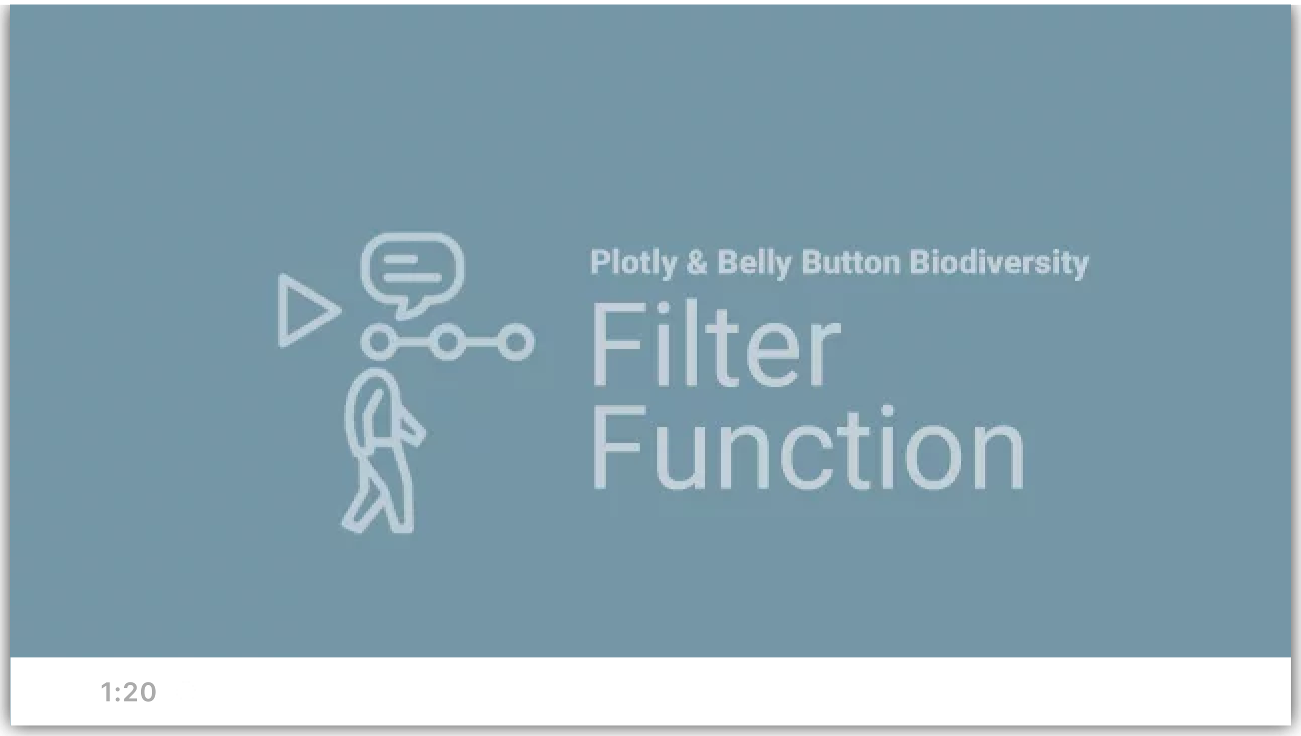
Open your text editor and browser. Modify the code in the previous example to extract the population of each city, instead of the city name.

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.

Retake

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

Let's break down this code.

- The variables `a` and `b` are replaced by `anElement` and `anotherElement`.
- The first two elements that are compared might be 3 and 2. The variable `anElement` is assigned to 3, and `anotherElement` to 2.
- The arrow function performs the subtraction `anElement - anotherElement`, or 3 - 2.
- Since the result is positive (3 - 2 = 1), the order of the two numbers is reversed.
- The `sort()` method compares another pair of elements in the array, for example 37 and 39.
- Since 37 - 39 is a negative number, their ordering is kept.
- The process is repeated until the array is sorted.

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

Retake

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];
var 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.