11.5.4 Use the “If” Statement

Navigating through functions and `for` loops in JavaScript has helped Dana feel more comfortable with coding in this new language. The next part she'll start working on is adding an `if` statement.

The `if` statement in JavaScript is similar to the Pythonic `if` statement, though the syntax is a little alien in comparison. Dana will need to explore how to create `if` statements in JavaScript and then incorporate it into her code as part of the filtering component.

Much like in Python, an `if` statement in JavaScript will check for conditions before executing the code. Our code will check for a date filter, so our `if` statement should read as follows; "If there is a date already set, then use that date as a filter. If not, then return the default data."

Look at a basic JavaScript `if` statement, but just for practice as this won't be part of our `app.js` code.

```
// if-statement syntax
if ( condition ) { code to execute }
```

In its most basic form, the `if` statement looks similar to a function. Write pseudocode about what we want our code to do.

```
// pseudocode practice
if (a date is entered) {
```

```
    Filter the default data to show only the date entered
};
```

That makes a little more sense. We want JavaScript to check for a date. If one is present, we want it to return only the data with that date. Now return to our app.js file to add our `if` statement.

```
if (date) {
    filteredData = filteredData.filter(row => row.datetime === date);
};
```

Take a closer look at the line that's inside our `if-statement`.

Take a look at the syntax for the `.filter()` method: `row => row.datetime === date);`. This line is what applies the filter to the table data. It's basically saying, "Show only the rows where the date is equal to the date filter we created above." The triple equal signs test for equality, meaning that the date in the table has to match our filter exactly.

IMPORTANT

> There are two ways to test for equality in JavaScript: `==` and `===`. While they look similar, there are differences. A triple equal sign (`===`) is checking for **strict equality**. This means that the type and value have to match perfectly.
>
> A double equals sign (`==`) is checking for **loose equality**. This means that the type and value are loosely matched. For more information about equality in JavaScript, read **JavaScript — Double Equals vs. Triple Equals (https://codeburst.io/javascript-double-equals-vs-triple-equals-61d4ce5a121a)**.

When we look at our complete `if` statement, it should appear as follows:

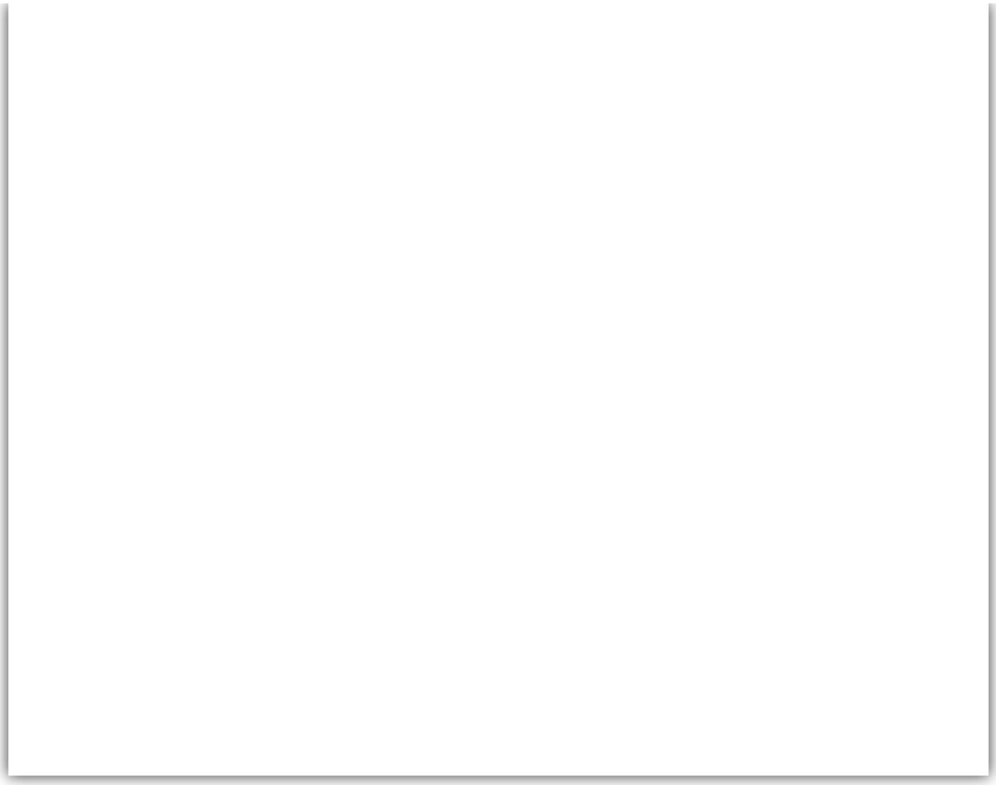
```
if (date) {
    filteredData = filteredData.filter(row => row.datetime === date);
  };
```

This is great! Our `handleClick()` function tells the code what to do when an event occurs (such as someone clicking a filter button), and it can apply that filtered data using an `if` statement. Being able to do all of this is great, especially since it involves creating functions written in a syntax that isn't the easiest to learn. There is one more step to complete with this function, though: building the table using the filtered data.

Build the Filtered Table

Thankfully, we've already set up a function to build a table: `buildTable();`. Now we just need to call it. Remember, we're building the function with the filtered data, so we'll use that variable as our argument.

Under our `if-statement`, let's call the `buildTable` function.



After we pass `filteredData` in as our new argument, our full `handleClick()` function should look like the one below:

```
function handleClick() {
  // Grab the datetime value from the filter
  let date = d3.select("#datetime").property("value");
  let filteredData = tableData;

   // Check to see if a date was entered and filter the
  // data using that date.
  if (date) {
    // Apply `filter` to the table data to only keep the
    // rows where the `datetime` value matches the filter value
    filteredData = filteredData.filter(row => row.datetime === date);
  };

   // Rebuild the table using the filtered data
  // @NOTE: If no date was entered, then filteredData will
  // just be the original tableData.
  buildTable(filteredData);
};
```

Listen for the Event

Our code is almost ready to be attached to the HTML component of our webpage. There are just a couple of loose ends to tie up. One is the clicking that will happen when someone filters the table. We have a function that *handles* a click, but how does the code know when a click happens?

Another aspect of D3.js is that it can listen for events that occur on a webpage, such as a button click. The next code we add will be tied to the filter button we'll build on our webpage.

Under our `handleClick()` function, add the following line of code:

```
d3.selectAll("#filter-btn").on("click", handleClick);
```

Our selector string contains the id for another HTML tag. (We'll assign a unique id to a button element in the HTML called "filter-btn".) This time it'll be included in the button tags we create for our filter button. By adding this, we're linking our code directly to the filter button. Also, by adding `.on("click", handleClick);`, we're telling D3 to execute our `handleClick()` function when the button with an id of `filter-btn` is clicked.

NOTE

> A "click" isn't the only event that D3.js can listen for—there are a variety of actions that can be listened for and handled. For example, a tooltip displays when you place your mouse over a specific element on a webpage.
>
> Events aren't limited to mouse events, either. They can include keyboard, text composition, forms—the list is lengthy and some of the events are quite advanced.

Build the Final Table

There is only a single step left before we can build the HTML component of the webpage: making sure the table loads as soon as the page does. Dana's readers will need to see the original table to even begin to use the filter we've set up. At the very end of the code, we'll call our `buildTable` function once more—this time using the original data we've imported. Type the following code:

```
buildTable(tableData);
```

Once this function is called, it will create a basic table filled with row upon row of unfiltered data pulled straight from our array.

All together, our code in the `app.js` file will look as follows:

```
function handleClick() {
  // Grab the datetime value from the filter
  let date = d3.select("#datetime").property("value");
  let filteredData = tableData;

   // Check to see if a date was entered and filter the
  // data using that date.
  if (date) {
    // Apply `filter` to the table data to only keep the
    // rows where the `datetime` value matches the filter value
    filteredData = filteredData.filter(row => row.datetime === date);
  }

   // Rebuild the table using the filtered data
  // @NOTE: If no date was entered, then filteredData will
  // just be the original tableData.
  buildTable(filteredData);
}
```

```
// Attach an event to listen for the form button
d3.selectAll("#filter-btn").on("click", handleClick);

// Build the table when the page loads
buildTable(tableData);
```

Now you're ready to start building your webpage!

ADD/COMMIT/PUSH

Make sure to save your work and add, commit, and push it to your repo!