Javascript – Module 11

* Is a front end development language
* Front end means you are focused on the appearance, look, feel for user who is interacting with the data
* Javascript allows for customizations that enhance user experience
* Has dashboards for data visualization
* Build a dynamic webpage by insert javascript into html
* Dynamic = accepts user input

**What You Will Learn**

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

* Explain the strengths and weaknesses of JavaScript "standard" and JavaScript version ES6+.
* Describe JavaScript syntax and ideal use cases.
* Build and deploy JavaScript functions, including built-in functions.
* Convert JavaScript functions to arrow functions.
* Build and deploy forEach (JavaScript for loop).
* Create, populate, and dynamically filter a table using JavaScript and HTML

**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 11(15 mins)
* JavaScript Basics (1 hour)
* Building Webpages with JavaScript (1 hour)
* Functional JavaScript (1 hour)
* JavaScript for Loops (1 hour)
* Building Dynamic Tables (1 hour)
* Build the HTML (2 hours)
* Application (5 Hours)

**11.0.4**

# JavaScript, Bootstrap, and UFOs

Dana wants to create a website about UFO sightings from her home town

Use .js data file, create a table, filter, add to html file

Customize webpage using Bootstrap

**11.1.1**

# Overview of ES6+

ECMAScript, also referred to as "ES," is a scripting language designed to help standardize JavaScript, has given JavaScript Proper Synatax

There have been many updates to ES over the years, though the sixth update was a major one. You'll probably see "ES6+" mentioned out in the wild pretty often; this is a reference to the "big" update (ES6) as well as the later ones. It's also commonly known as "ES2015" or "ECMAScript 2015."

## Benefits of the ES6 Update

JavaScript after the ES6 update is like the newer computer. This update included many updates to the syntax, which streamlined the code and made it easier to both read and write. Additional, quality of life improvements were implemented as well, such as adding Python-like generators and for...of loops. Even functions were updated and streamlined!

for...of loops is a new syntax associated with JavaScript, so it's okay to not be familiar with it yet! We'll discuss this syntax in more detail as we learn more about the language.

In this module, our focus will be on basic JavaScript and ES6 capabilities such as arrow functions.

**11.1.2**

# JavaScript in the Real World

JavaScript is one of the powerhouse languages out in the wild today. While its strength is in creating visually appealing and dynamic content, it is starting to grow into other fields as well. Tensorflow, a popular machine learning tool, even has its own JavaScript library now.

It's pretty easy to start feeling daunted by everything JavaScript can do, so Dana is more interested in examples of similar websites—ones that use filters on lots of data.

* **Online shopping websites:** These are a great example of dynamic content. They contain filters for departments, and then filters for items within those departments. Filters on top of filters!
* **Ecological data:** The [National Ecological Observatory Network (NEON) (Links to an external site.)](https://data.neonscience.org/browse-data?showAllDates=true&showAllSites=true&showTheme=org) has very large and diverse datasets; these are also displayed on their website as dynamic tables with multiple filters.
* **Weather data:** [The National Snow & Ice Data Center (NSIDC) (Links to an external site.)](https://nsidc.org/data/search/#keywords=permafrost/sortKeys=score,,desc/facetFilters=%257B%257D/pageNumber=1/itemsPerPage=25) also has very large datasets presented in table format on their website. These tables include filters and parameters that can be applied to their table.

**11.1.3**

# Writing JavaScript

**While** JavaScript is clearly capable of a variety of tasks, Dana plans to start with something a bit more manageable. Instead of building an entire dashboard right away, first she'll create a filterable table to display the data. She decides to dig into the syntax of the language. It's very different from other languages Dana has encountered before, so she wants to be sure she understands the basics before she begins to build anything.

One major component of each coding language is its **syntax**. For example, Python is a pretty clean and easy-to-read language; there aren't many semicolons, and the indentation and spacing makes sense. SQL, on the other hand, includes semicolons, but it also has guidelines and requirements when it comes to indentation and spacing.

JavaScript is no different: there are guidelines and requirements for writing it. But because JavaScript can be added to an HTML page, there are more guidelines and requirements than for languages that can only live in a .js file or Jupyter notebook such as Python. There are a few important things to remember about JavaScript syntax. We'll start with the following:

* Case sensitivity
* Semicolons
* Statements and expressions
* Code blocks

We'll be sure to get in lots of practice so that Dana can feel 100% confident in her skills.

## Case Sensitivity

JavaScript is case sensitive. **Case sensitivity** means that JavaScript considers upper- and lower-case words to be different. For example, if we were to assign the words "data" and "Data" as variables, we would be able to save different information in each word. Of course, actually doing this with the word "data" could lead to confusion pretty quickly. Instead, just remember JavaScript cares about capital letters.

Similarly, JavaScript uses different naming conventions than Python that involve case sensitivity. Different languages utilize different methods to link words without using spaces, which is called a **case style**.

JavaScript's code style, according to coding guidelines and syntax, is camel case. You'll encounter this case often as you begin to practice your coding. It's especially useful when declaring variables.

**NOTE**

Camel case is the preferred naming convention in JavaScript. This is especially helpful in cases where Python data is used. For example, we would know that variables named with snake case originated from the Python side of things.

## Semicolons

Much like SQL, when coding in JavaScript it's good practice to end statements with a semicolon. Technically, they are optional when it comes to executing your code, but they are helpful because they tell JavaScript that a particular line or block of code is complete. It's considered a best practice to include semicolons throughout your code. You'll encounter many semicolons throughout this module.

Let's use a print statement as an example. In JavaScript, a print statement is called a **console log**. To print "Hello, world!" to the console, we would use this line:

// Printing a string with JavaScript

console.log("Hello, world!");

**NOTE**

While the print() function does exist in JavaScript, it will actually try to print to a printer instead of our console.

This statement is almost identical to a basic Python print statement, as shown below.

# Printing a string with Python

print("Hello, world!")

Both methods will print the string (in this case, "Hello, world!"). But in addition to switching "print" with "console.log," in JavaScript, a semicolon has been added at the end of the statement.

#### Testing Simple Statements

Simple JavaScript statements such as console.log() can be tested using DevTools. For example, follow these steps to test console.log("Hello, world!").

1. Go to a site like [Google (Links to an external site.)](http://www.google.com/) and activate your DevTools. This is where we'll access our console; the console is the command line interface tool we'll use to test JavaScript, much like our terminal is used to test Python.

You can use any site to open your DevTools; it isn't a requirement to use the Google search page.

1. Click the "Console" tab at the top of the screen.
2. Type console.log("Hello, world!"); on the first line and then press Enter.

The Console tab in DevTools will become a very important tool when we begin to code later on. The Console tab will allow us to see if an error has occurred and, if so, which line of code is causing the disruption.

## Statements and Expressions

When describing JavaScript code, the terms "statements" and "expressions" are both used, and often. Here's how to distinguish between the two:

* Statements perform actions.
* Expressions create values.

Assigning a variable is an example of a statement. Using arithmetic to create a new value is an expression.

#### Code Blocks

Code blocks, which we will see more often as we start writing functions, are denoted by curly brackets. Code inside the curly brackets are typically indented two to four spaces. This isn't required to run the code, but it does make reading it easier and follows the coding guidelines.

**11.2.1**

# JavaScript Components

**Now** that Dana knows where she'll encounter JavaScript in the wild—as well as some of the key differences between JavaScript and Python—it's time to get down to business. Understanding the basics of a programming language from text is one thing, but putting that understanding into practice is another. Dana is ready to dive in and start working with some of the basic components: variables and lists.

We're going to begin our practice by familiarizing ourselves with some basic JavaScript components: **variables** and **arrays**.

## Variables

Before ES6 came along, there was a single way to declare a variable: **var**. You've already worked with variables in Python, but this concept in JavaScript is a bit different. Let's compare a Python variable to a JavaScript variable:

|  |  |
| --- | --- |
| Python | JavaScript |
| y = 2 | var y = 2; |

Python's way of assigning a variable is quite simple: type the name of the variable followed by its value: y = 2.

JavaScript is similar, but with two additions: add var before the variable, and then add a semicolon after the value, like this: var y = 2;.

Let's test our JavaScript variable assignment using DevTools. If you still have your console open and ready to go, great! If not, go ahead and bring another up to practice with.

**SKILL DRILL**

If needed, visit a webpage such as [Google (Links to an external site.)](http://www.google.com/) and activate your DevTools. Click the Console tab to activate the console. Then do the following:

1. On the first line, type var y = 2;and press Enter.
2. On the next line, type console.log(y); and press Enter.

The value of y should print to your console.

End of text box.

Of the many additions that came along with ES6, two more ways to declare variables were also introduced: let and const.

This can be a bit trickier than it seems, because in JavaScript a variable isn't always just a variable. There are specific uses for different variables, and using let and const instead of var helps developers define what the uses are. Let's check them out in more detail.

*Each variable has its adv and disadv depending on application, for let and const, they are also considered block-scope, it will only work inside the braces it was declared in. Means you can declare variables with the same name within different blocks with different meanings.*

*Const cannot be either redefined or redeclared (hence constant within its block)*

*Let can be redefined but not redeclared*

*Var can be redefined and redeclared*

#### Create Variables with let

The biggest difference between var and let is that the var declaration is global, meaning it applies to the program instead of being contained in a block of code.

When a developer chooses to use let, it basically means "I might want to use this variable again later to hold different data, but in this code block I'll only use it once." In ES6+, let is typically used in place of var. We'll be using let in this module, but both are encountered out in the wild.

#### Create Variables with const

The const declaration is more specific than let. Instead of being contained within a block of code, const tells JavaScript that the variable won't be reassigned or redeclared, either in a block of code or within the program as a whole. The following table highlights the key differences of var, let, and const:

|  |  |  |
| --- | --- | --- |
| **Least specific** | var | Variable used in entire program |
|  | let | Variable used in a code block |
| **Most specific** | const | Variable used once |

Now that we've discovered three different ways to declare variables in JavaScript, let's take a look at arrays.

#### Arrays

When coding in Python, data can be grouped together in a **list**. The same is true of JavaScript. In fact, Dana was inspired to learn JavaScript because the data is already stored in a JavaScript array! Let's take a look at the data to see what we're working with. Start by downloading the JavaScript file below:

[Download data.js (Links to an external site.)](https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_11/data.js)

**GITHUB**

This data.js file is a large part of this project. Save it in the new repository you cloned to your computer.

When you open the file, you'll see that it contains a lot of data. Look at the construction of this array.

First, the name of the array, data, is declared with var:

var data = [

The structure of the array begins much like a Python list: with a square bracket. But the data inside is arranged a bit differently.

Each entry is inside the square brackets, like a Python list. That's where the similarities begin to taper off.

In this particular JavaScript array, we're not recording a single item and moving on to the next, much like a simple list (such as [1, 2, 3]). However, here we're recording an entire event: date, location, type, and even comments are saved inside a single array. Not only that, but multiple events are recorded. Because we have so much information, the array looks more like a Python dictionary than a simple Python list. Take a look at the example below:

{

datetime: "1/1/2010",

city: "benton",

state: "ar",

country: "us",

shape: "circle",

durationMinutes: "5 mins.",

comments: "4 bright green circles high in the sky going in circles then one bright green light at my front door."

},

Within a set of curly brackets, we can see the key-value pairs such as the date, city, and state.

While this looks somewhat similar to a Python dictionary, there is one key difference. Scroll to the last link of the data file and see if you can spot the difference. You can use keyboard shortcuts to reach the bottom quickly, instead of scrolling through the whole file.

There's a lot going on in this file: for Dana's article to be a success, she will need to make these sightings easier for people to visually parse by converting them from their current state, a JavaScript array, into an HTML table.

#### Convert the Array to a Table

To convert the array to a table, we're going to take the following code and turn it into the table shown below:



The first step in transitioning the data from an array to a table is to create the appropriate variables using var, let, or const. Open VS Code and create a file in our repo folder named app.js. This is where we'll keep the code that builds the HTML table and fills it with data from data.js.

**11.2.2**

# Organize Your Repository

**Dana** has started to get into a JavaScript coding rhythm. But before she begins to create code for real and then commit scripts to her repo, she needs to organize it.  
  
Building a page that contains JavaScript will require Dana to link additional JavaScript files to the index.html file that she'll be working on later. This means keeping track of multiple things at once: an HTML file, JavaScript files, images (for customizing the webpage) and a CSS style sheet. Therefore, it's a good idea for Dana to establish a solid folder structure now instead of when she's elbow deep in creating her JavaScript functions.

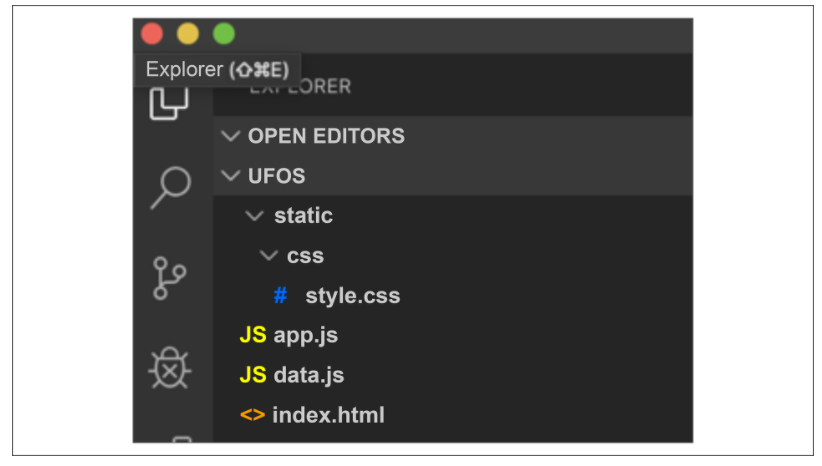
Before we get too far along with our coding, we need to set up a file organization system for our repo. The end result of this project will be an HTML page or application, so we need to establish the proper folder structure accordingly. At a high level, here's what we'll do:

* Create the index.html file.
* Create a subfolder to hold the CSS file (style.css).
* Create a subfolder for images.
* Create a subfolder to hold JavaScript.

First, in the repo folder we established earlier ("UFOs"), create the index.html file. This file is the window to our work: the table and Dana's article summary (along with titles and filters) will all be displayed through this file. We won't be coding it yet—that will come later—but we're creating it now so that it will be ready for us when it's time to build the page.

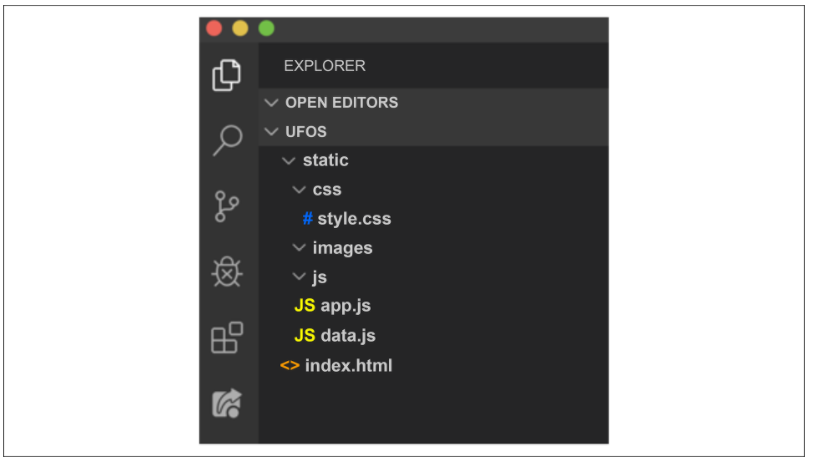
Next, create a subfolder in the repo folder named "static." This static folder will hold our static CSS file; this only means that it isn't being moved or altered externally. In VS code, right-click the menu and select "New Folder," and then name it "static."

Inside the static folder, create another subfolder named "css" to hold the style.css file. You can use the same right-click method to bring up the creation menu, but this time select "New File." We'll customize our webpage using the style.css sheet, but for now we can leave it blank. Here's what the folder structure should look like so far when viewing it in VS Code.



The next subfolder we'll create is our "static" folder to hold whatever images we want to add to our website when it's time to customize it. Create the folder now and name it "images." But for now, move on to the next step—we'll add images later.

The third and final subfolder we'll create is one to hold our JavaScript. Name the folder "js" and move the data.js and app.jsfiles into it so that your folder structure looks like this:



Establishing this folder structure is a best practice when creating webpages with JavaScript. It's important to keep things organized when creating a webpage using JavaScript components, as there are even more moving pieces than a static website. We'll be linking to images and a style sheet as well as JavaScript scripts. The organization presented here provides clearly designated spots to store the code we'll be working on, making it easier to locate them as we go.

**11.2.3**

# JavaScript Objects

**It's** been a good day of research. Dana is far more familiar with some of the basic components of JavaScript: she now knows that variable declaration can actually occur three ways, and the array she is working with is similar to a Python dictionary. It's a great start, but Dana's still a little fuzzy on the array. It looks like a JSON, or a dictionary, so it's more complex than a simple list, right?  
  
Dana's intuition has served her well: the JavaScript array is indeed a bit more than a simple list. Let's take a closer look at JavaScript objects and how to interact with them, which will help us as we begin to create our code.

Coding in JavaScript requires proficiency with JavaScript objects. And, in JavaScript, many different things can be considered an "object." We've actually already encountered one! Let's look at a snippet of code from ourdata.js array:

var data = [

{

datetime: "1/1/2010",

city: "benton",

state: "ar",

country: "us",

shape: "circle",

durationMinutes: "5 mins.",

comments: "4 bright green circles high in the sky going in circles then one bright green light at my front door."

},

As mentioned earlier, this looks very similar to a Python dictionary or something we'd find in a JSON file. In this code snippet, everything within the curly brackets is considered to be properties of a JavaScript object. The object is our variable: data.

There are several ways we can access the properties, also called key-value pairs or objects, in the array.

Also, objects are not limited to being contained within an array. In fact, an array itself is an object. Dates are also objects, as are functions; and Booleans can be objects. Basically, many things can be—or are—objects. We'll get plenty of practice with objects as we start to build our website.

Before building the website, we should plan it out. By using a storyboard and mapping the elements out beforehand, it will be easier to assemble them later.

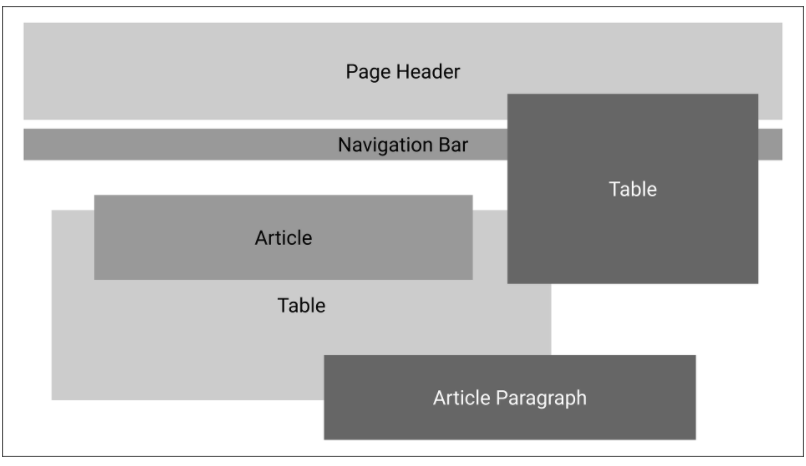
**11.2.4**

# Storyboarding

**Dana** has grown more familiar with JavaScript syntax and her basic code is gaining momentum. She's ready to start putting it to use!  
  
Dana's goal is to create an interactive webpage that allows readers to parse the data around UFO sightings. So, she essentially needs to build two things: the webpage that will allow users to view the data (HTML) and a dynamic table that will present it (JavaScript).  
  
Dana wants to storyboard her website to have an idea of what her readers will see when they view the final product. Storyboarding is incredibly useful in determining the layout of a webpage, so it's important to complete this step early in order to save time later. It's like building a house. You need to know how it's all going to fit together before you start building!  
  
Once the template has been created, Dana can begin to code the JavaScript portion by first importing the data and then referencing it with a variable.

Typically, developers build HTML and JavaScript elements somewhat simultaneously because they complement each other. For example, the JavaScript table will be referenced within the HTML code, and different HTML components will be referenced within the JavaScript code. Because these files are so closely linked, Dana will switch between building the JavaScript table (within the app.js file) and the HTML page (within an index.html file).

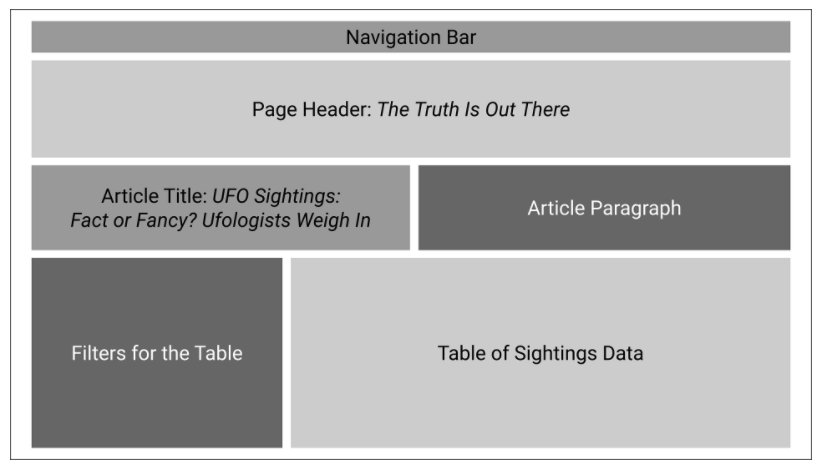
Dana also has a solid idea of how she wants her webpage to look, but it's easy to get lost in the details of building a webpage without a visual reference. A visual reference such as a storyboard will help Dana outline all of the elements she wants included, such as the article title, a summary, and the table itself. Then, when she begins creating JavaScript code to include the table, she'll know exactly which HTML components she'll be connecting to her table. Dana already knows she'll have several individual components on the webpage, shown below:



Now she just needs to figure out how to assemble them. This is where a storyboard comes in.

## Create a Storyboard

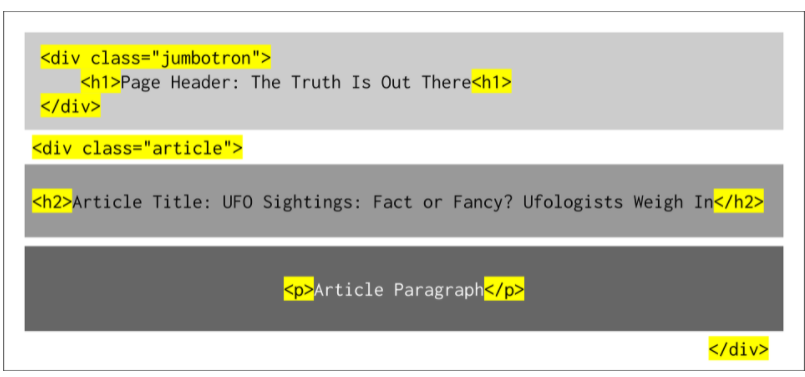
A **storyboard** serves as a kind of blueprint for your site and helps with the transition from idea to finished product. Think of it as a map of the webpage.



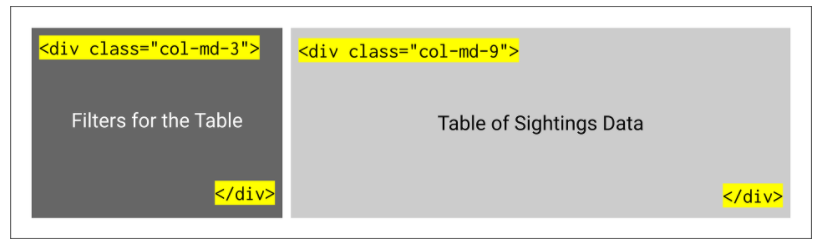
This step is key for a couple of reasons.

First, knowing how we want the webpage to look before building it will save us time later. Second, it helps us make sure we've captured everything we want displayed. Sometimes, seeing the map of the website helps us ensure that all the elements we want displayed are included.

We already know what components we want to use, such as a Jumbotron for the header, and the grid system for the filters and table. See the following image:



We also have an idea of how many columns we want each table component to use.



Now that a storyboard is in place, we can really get going! Let's align our code.

**Align the Code**

When we align our code, we're putting our plans into action, such as when we start transitioning our storyboard into a webpage. We'll start by building our components. The first one will be the table we generate with JavaScript. Open your app.js file with VS Code. The first thing we're going to do is import the data. This won't look like an import from Python. For starters, the double backslash ( // ) is how you comment your code in JavaScript.

In your code editor, type the following to declare a variable, tableData, using const.

// import the data from data.js

const tableData = data;

Next, we need to point our data to our HTML page. Specifically, we need to tell JavaScript what type of element the data will be displayed in. We already know that the data will be displayed in a table, so in our code editor we'll reference the tbody HTML tag using D3.

**IMPORTANT**

D3 is a JavaScript library that produces sophisticated and highly dynamic graphics in an HTML webpage. It is often used by data professionals to create dashboards, or a collection of visual data (such as graphs and maps), for presentation.

Return to your code editor and type the following:

// Reference the HTML table using d3

var tbody = d3.select("tbody");

With this code, we:

1. Declare a variable, tbody
2. Use d3.select to tell JavaScript to look for the <tbody> tags in the HTML

Although we aren't building the HTML right now—we'll do this after we put together the code—we already know that the data will fit into that tag because it's a standard table tag that is used often in HTML, with or without JavaScript enhancements.

**11.3.1**

# Getting Started with JavaScript Functions

**Dana** has started to build her code, which is really exciting! When she imported the data, she took the first step in building her website. The next step is to build the table to sort and store the data.  
  
Dana knows that building this table will introduce a new level of complexity involving for loops and functions. Thankfully, JavaScript and Python have similar logic, so after Dana reviews and practices with code similar to what she'll use in her project, she'll be ready to start integrating it into her code.

Functions in Python and JavaScript have similar logic: we provide the language with a set of instructions to follow, which can then be reused as needed. Watch the following video to learn more about JavaScript functions.

In Python, a simple print statement looks like this:

# Simple Python print statement

def print\_hello():

print("Hello there!")

In this code, the function is declared with the keyword def followed by the name of the function, a set of parentheses, and a colon, with the indented code below.

To write a print statement in JavaScript, we begin the same way: by declaring the function. To do this, we use the keyword function. (**Note:** Remember that the JavaScript syntax uses console.log instead of print.)

// Simple JavaScript console.log statement

function printHello();

At this point, the process diverges from Python. The next step is to add a set of curly brackets, and then add the indented code between them.

// Simple JavaScript console.log statement

function printHello() {

console.log("Hello there!");

}

**SKILL DRILL**

Return to the console tab of your DevTools and run the JavaScript function you just reviewed.

End of text box.

Get more practice with Python and JavaScript functions in the following activity.

Let's take a closer look at basic functions in JavaScript.