Graphical user interface, text, website

Description automatically generated

Module 11: UFO Sightings with JavaScript

[Introduction to Module 11 1](#_Toc94634639)

[11.0.1: Creating Dynamic Content 1](#_Toc94634640)

[11.0.2: Module 11 Roadmap 1](#_Toc94634641)

[11.0.3: Getting Ready for Virtual Class 2](#_Toc94634642)

[11.0.4: JavaScript, Bootstrap, and UFOs 2](#_Toc94634643)

[JavaScript Basics 2](#_Toc94634644)

[11.1.1: Overview of ES6+ 2](#_Toc94634645)

[11.1.2: JavaScript in the Real World 2](#_Toc94634646)

[11.1.3: Writing JavaScript 2](#_Toc94634647)

[Building Webpages with JavaScript 3](#_Toc94634648)

[11.2.1: JavaScript Components 3](#_Toc94634649)

[11.2.2: Organize Your Repository 4](#_Toc94634650)

[11.2.3: JavaScript Objects 4](#_Toc94634651)

[11.2.4: Storyboarding 5](#_Toc94634652)

[Functional JavaScript 6](#_Toc94634653)

[11.3.1: Getting Started with JavaScript Functions 6](#_Toc94634654)

[11.3.2: Simple JavaScript Functions 6](#_Toc94634655)

[11.3.3: From Simple Functions to Arrow Functions 7](#_Toc94634656)

[JavaScript for Loops 7](#_Toc94634657)

[11.4.1: Use a JavaScript for Loop 7](#_Toc94634658)

[11.4.2: Practice Using for Loops in JavaScript 8](#_Toc94634659)

[Building Dynamic Tables 8](#_Toc94634660)

[11.5.1: Introduction to Dynamic Tables 8](#_Toc94634661)

[11.5.2: Add forEach to Your Table 9](#_Toc94634662)

[11.5.3: Add Filters 10](#_Toc94634663)

[11.5.4: Use the “If” Statement 11](#_Toc94634664)

[Build the HTML 11](#_Toc94634665)

[11.6.1: Bootstrap Components 11](#_Toc94634666)

[11.6.2: Add the Data 11](#_Toc94634667)

[11.6.3: Customize the Page with CSS 11](#_Toc94634668)

[Application 11](#_Toc94634669)

[Module 11 Challenge 11](#_Toc94634670)

[Module 11 Career Connection 11](#_Toc94634671)

[Class Notes Day 1 12](#_Toc94634672)

[Class 1 Activity 1 12](#_Toc94634673)

[Class 1 Activity 2 12](#_Toc94634674)

[Class 1 Activity 3 12](#_Toc94634675)

[Class 1 Activity 4 12](#_Toc94634676)

[Class 1 Activity 5 12](#_Toc94634677)

[Class 1 Activity 6 12](#_Toc94634678)

[Class 1 Activity 8 12](#_Toc94634679)

[Class Notes Day 2 12](#_Toc94634680)

[Class 2 Activity 1 12](#_Toc94634681)

[**Class 2 Activity 2** 12](#_Toc94634682)

[**Class 2 Activity 3** 12](#_Toc94634683)

[**Class 2 Activity 4** 12](#_Toc94634684)

[**Class 2 Activity 5** 12](#_Toc94634685)

[**Class 2 Activity 6** 12](#_Toc94634686)

[**Class 2 Activity 7** 12](#_Toc94634687)

# Introduction to Module 11

## [11.0.1: Creating Dynamic Content](https://courses.bootcampspot.com/courses/978/modules/items/358752)

**Overview**

Javascript powers the user experience on a webpage – creating a DYNAMIC webpage – by accepting user inputs and adjusting accordingly (like a [Next] button, or [Search \_\_\_\_] feature. We will insert Javascript into an HTML page, use DevTools, CSS, & Bootstrap to build & style our webpage, and test it as we go.

## [11.0.2: Module 11 Roadmap](https://courses.bootcampspot.com/courses/978/modules/items/358754)

**What You Will Learn:**

* 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:**

0 Introduction to Module 11(15 mins)

1 JavaScript Basics (1 hour)

2 Building Webpages with JavaScript (1 hour)

3 Functional JavaScript (1 hour)

4 JavaScript for Loops (1 hour)

5 Building Dynamic Tables (1 hour)

6 Build the HTML (2 hours)

5 Application (5 Hours)

## [11.0.3: Getting Ready for Virtual Class](https://courses.bootcampspot.com/courses/978/modules/items/358757)

(Git pull)

## [11.0.4: JavaScript, Bootstrap, and UFOs](https://courses.bootcampspot.com/courses/978/modules/items/358761)

Take a Javascript file on UFO sighting information and display it as a table. Sifting thru it would be challenging, so we will add filters (thanks to Javascript) to make this table fully dynamic, meaning that it will react to user input. We’ll put it on an HTML page for easy viewing. You'll customize your webpage using Bootstrap, and equip your table with several fully functional filters that will allow users to interact with our visualizations.

Create the UFOs repository

# JavaScript Basics

## [11.1.1: Overview of ES6+](https://courses.bootcampspot.com/courses/978/modules/items/358768)

JavaScript was developed back in 1995.

There was a major update to JavaScript not too long ago, called ECMAScript (aka “ES#”) to standardize Javascript with “proper syntax.” The biggest update was ES6. Referring to it as ES6 / JavaScript / JS in this module. It made JS:

* Faster b/c
* streamlined code = easier to both read and write
* quality of life improvements –adding Python-like generators, for...of loops, functions

The latest update is ES11.

## [11.1.2: JavaScript in the Real World](https://courses.bootcampspot.com/courses/978/modules/items/358773)

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 The [National Ecological Observatory Network (NEON)](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)](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](https://courses.bootcampspot.com/courses/978/modules/items/358776)

JavaScript has 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.

JavaScript syntax tips:

* **Case sensitive** - considers upper- and lower-case words to be different
* **Case style** – camel case (python uses snake case)
* **Semicolons** – technically optional, but recommended. Use when ending a statement;
* **Printing** – use console.log() because print() will actually try to print to a printer instead of our console

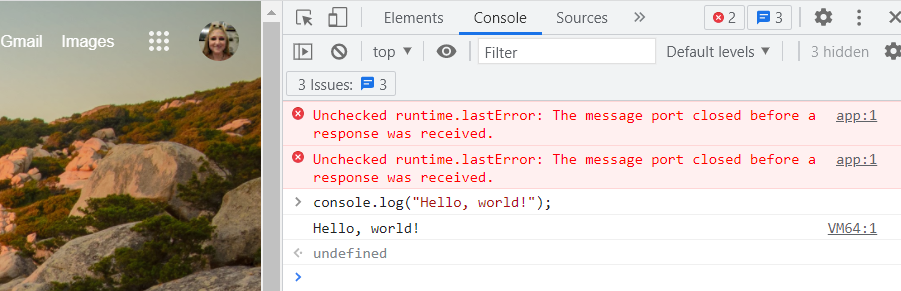
Print statement in JS: console.log("Hello, world!");

Print statement in Python: print("Hello, world!")

* **Statements** perform actions - Assigning a variable
* **Expressions** create values - Using arithmetic to create a new value
* **{Code blocks}** – { usually indented 2 or}

{ four spaces to make reading the code easier}

Testing Simple Statements



* 1. Go to a website > activate DevTools
  2. Click on **Console** - command line interface tool to test JS, (like terminal is used to test Python).
  3. Test: type “console.log("Hello, world!"); > [enter]

# Building Webpages with JavaScript

## [11.2.1: JavaScript Components](https://courses.bootcampspot.com/courses/978/modules/items/358784)

Dive in and start working with some of the basic components:

**Variables Test in Console:**

Graphical user interface, text, application

Description automatically generatedVariable statement in JS: var y = 2;

Variable statement in Python: y = 2

In JavaScript, a variable isn't always just a variable.

Different variables help developers define what the specific use is.

|  |  |  |  |
| --- | --- | --- | --- |
| Var | declaration is global | applies to the program instead of being contained in a block of code | Least specific |
| Let | Confined to code block | might want to use this variable again later to hold different data, but in this code block I'll only use it once | Semi-specific |
| const | Never reassigned | won't be reassigned or redeclared, either in a block of code or within the program as a whole | Most specific |

Text

Description automatically generated

**Arrays**

* Grouping data together in a **list.**
* Open data.js with VS code
* The structure of the array begins much like a Python list: with a square bracket. But the data inside is arranged a bit differently. Array looks more like a Python dictionary than a simple Python list.
* Semi-colon ends the code block:



**Convert the Array to a Table**

1. create the appropriate variables using var, let, or const.
2. 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](https://courses.bootcampspot.com/courses/978/modules/items/358788)

Establish a solid folder structure now to keep track of all the things using at once to create a webpage: an HTML file, JavaScript files, images (for customizing the webpage) and a CSS style sheet. 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.

|  |  |
| --- | --- |
| A picture containing calendar  Description automatically generated |  |
|  |
|  |
|  |
| 🗀 not being moved or altered externally |
| 🗀 hold css file |
| 🗎 customize our webpage |
| 🗀 images we want to add to our website when it's time to customize it |
| A picture containing calendar  Description automatically generated🗀 hold our JavaScript |
| 🗎 code that builds the HTML table and fills it with data from data.js |
| 🗎 raw data |
|  |
| 🗎 window to our work: table, article summary, titles, filters all be displayed here |
|  |
|  |
|  |

## [11.2.3: JavaScript Objects](https://courses.bootcampspot.com/courses/978/modules/items/358791)

Take a closer look at JavaScript objects and how to interact with them… not as simple as it looks

var data = [

object /array = *data* variable

{properties of the JavaScript object}

Several ways to access the array’s

properties / key-value pairs

 many things can be—or are—objects:

array , dates, functions, booleans *can* be

  {

    datetime: "1/1/2010",

    city: "benton",

    state: "ar",

    country: "us",

    shape: "circle",

    durationMinutes: "5 mins.",

    comments: "4 bright green circles…."

  },

## [11.2.4: Storyboarding](https://courses.bootcampspot.com/courses/978/modules/items/358794)

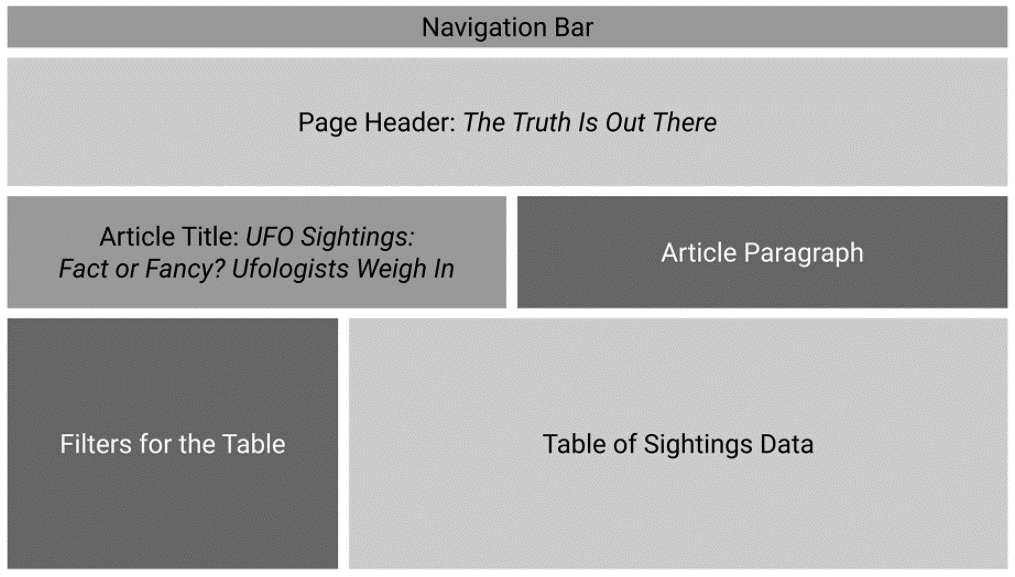
Essentially needs to build two things: the webpage that will allow users to view the data (HTML) and a dynamic table that will present it (JavaScript). Determine the layout of a webpage, so it's important to complete this step early in order to save time later. It's like building a house. You need to know how it's all going to fit together before you start building!

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. We will switch between building the JavaScript table (app.js) and the HTML page (index.html).

Graphical user interface, text, application, email, website

Description automatically generated

**Create a Storyboard**



**Align the Code**

putting our plans into action / transitioning our storyboard into a webpage

building our components

Generate the table with JavaScript

import the data . go to app.js in VS code // COMMENT CODE IN JS

A screenshot of a computer

Description automatically generated with medium confidence

# Functional JavaScript

## [11.3.1: Getting Started with JavaScript Functions](https://courses.bootcampspot.com/courses/978/modules/items/358800)

Graphical user interface, text, application

Description automatically generated

**function** - define function

**hello** – name it

**(name)** – hold parameters, if any

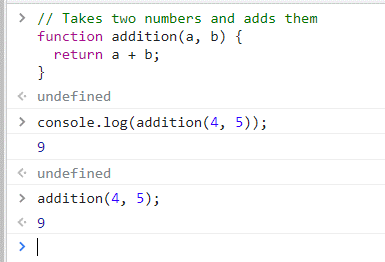
**{ }** – function goes inside

* No rules for indentation
* Instead, put all inside the {}

**console.log** – calling/printing a function

* Keep inside {}

## [11.3.2: Simple JavaScript Functions](https://courses.bootcampspot.com/courses/978/modules/items/358803)



Function:

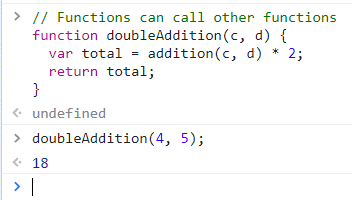
Call the function:

Shorted the call syntax:

( **parameters** )

Placeholders for the values. Example: function addition(a, b) { return a + b; }

It's not a good idea to have more than two parameters per function. Too many arguments can significantly slow down and even crash your code.



**Functions can also call other functions**.

Function “doubleAddidtion” (parameters c & d)

{

Variable “total” = the previous function (with 2 new parameters) \* 2



call “total’

}

⦸ or clear()+[enter] clears the working area of your console.

🞁 (keyboard arrow) access what you've written by cycling through executed code.

## [11.3.3: From Simple Functions to Arrow Functions](https://courses.bootcampspot.com/courses/978/modules/items/358808)

complete the same functions as regular functions, but they use a more compact and concise syntax that makes a code script shorter and easier to read. Any standard function in JavaScript can be refactored into an arrow function.

Long version w/out arrow: function square(number) { return number \*\*2; } Both produce same result:







Shorter version using arrow: let square = (number) => number\*\*2;



Explanation:

* The arrow function collapses the function from 3 lines to 1 line, which is a significant reduction in characters.
* The function keyword is not part of the arrow function. This is because the arrow symbol (=>) indicates that this block (or line) of code is a function.
* The return keyword and console.log() are removed because with this new syntax, JavaScript inherently knows what will be returned.

long version: function doubleAddition(a, b) { var total = addition(a, b) \* 2; return total; }



arrow version: let doubleAddition = (a, b) => addition(a, b) \* 2;



NOTE - While arrow functions are clear and readable, there are still cases in which traditional functions are necessary. For example, when we want to place a function within another function, we would need to use a traditional function.

# JavaScript for Loops

## [11.4.1: Use a JavaScript for Loop](https://courses.bootcampspot.com/courses/978/modules/items/358815)

for loop iterate through items, such as names in a list

"undefined"  - how you know that you've successfully executed the line of code and the array has been saved locally

* If you close your console and reopen it, the code will have been erased and you'll need to start over.

Python for loop: for i in user\_list: print(i)

JS for loop: function listLoop(userList) { for (var i = 0; i < userList.length; i++) { console.log(userList[i]); }

* for - trigger that indicates we'll be iterating through a list
* var i = o - assign an iterable variable and set its value to zero
  + In this loop, think of the letter 'i' to mean 'iterate.' When we assign a zero value, we're starting a counter from the beginning. You can also think of it in terms of list comprehension–the first name of the list has an index value of zero, for example.
* i < userList.length – if i is still smaller than the total number of i’s in userList, move on to the next step.
* i++ - increases the iterable by 1.
  + We're using list comprehension here; the for loop knows to iterate to the next name because the index number has increased by 1.
* When the length of i is equal to the total number of items in the list, the for loop will complete its iterations and the next line of code will be executed.
  + For example, Jeff's index position is 3; when i is equal to 3, the loop is complete. This is because there are no names after Jeff's, nothing with an index value of 4.

Text

Description automatically generated with medium confidenceText

Description automatically generated

[shift]+[enter] to go to the next line in Console

## [11.4.2: Practice Using for Loops in JavaScript](https://courses.bootcampspot.com/courses/978/modules/items/358818)



Here's our array:

Want each item in the array to be printed to the console.

Add a message to go with each item reading "I love [vegetable]" with each iteration

Text

Description automatically generated

Details:

* We're using the keyword for to initiate the loop.
* We also start the loop at the beginning by assigning an iterable as zero with var i = 0;.
* **.length** attribute… Next, we tell the loop to continue working through the array until… as long as the iterable ("i") is less than the number of vegetables in our array: i < vegetables.length;.
  + When I = vegetables.length, the statement will become false, and thus will stop looping.
  + We need < always b/c the numbering strats at 0, so if we went to <+ length, it would go 1 past the length. (if there are 12 items, we would want #s 0-11. So to list thru all 12 items, we would do x < 12 🡪 #’s 0-11
* Finally, we increase our iterable by 1 by adding i++; which tells JavaScript to move to the next item in the array until there are no more items.
* Let's say we also want each item in the array to be printed to the console. To do this, we'll add a console.log statement inside the curly brackets. Let's add a message to go with each item, too, so it will read "I love [vegetable]" with each iteration.

This time we'll loop through numbers without using an array & count up to a fifth value.

Graphical user interface, text, application

Description automatically generated

# Building Dynamic Tables

## [11.5.1: Introduction to Dynamic Tables](https://courses.bootcampspot.com/courses/978/modules/items/358824)

Build the table to display all of the UFO sightings. Iterate through the array of objects in our data file and then append them to a table row. All of this will happen within a function, which makes the code self-contained.

Function names should be called what they do: verbNoun. Ex: *buildTable* or *handleClick*

Text

Description automatically generated

## [11.5.2: Add forEach to Your Table](https://courses.bootcampspot.com/courses/978/modules/items/358829)

Create another function specifically for building the table. Data from the data.js file will be inserted into the table, row by row. This sounds like iterating through an array using a for loopforEach, doesn't it?  
  
This time, we'll use a forEach function, which loops through the array in the same way as a for loop. The difference is that forEach works *only* with arrays. Another benefit is that forEach can be combined with an arrow function, once again making the code more concise and easy to read.

Text

Description automatically generated

## [11.5.3: Add Filters](https://courses.bootcampspot.com/courses/978/modules/items/358832)

There will be hundreds of rows of sightings in the table, which is entirely too much for one person to reasonably look through and study. Therefore, the next step is to add the ability to filter the data. We'll be using D3.js to help with this part.

**Data-Driven Documents / D3** - a JavaScript library that adds interactive functionality, such as when users click a button to filter a table. It works by "listening" for events, such as a button click, then reacts according to the code we've created.  
  
Crate a “filter by date”

Text

Description automatically generated

## [11.5.4: Add the “If” Statement](https://courses.bootcampspot.com/courses/978/modules/items/358835)

"If there is a date already set, then use that date as a filter. If not, then return the default data."

if ( condition ) { code to execute};

if (a date is entered/ the search) {Filter the default data to show only the date entered};

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

=== 🡪 tests for STRICT equality. (date in the table has to match our filter exactly)

== 🡪 tests for LOOSE equality.

A screenshot of a computer

Description automatically generated with medium confidence

**Listen for the Event**

We have a function that *handles* a click, but how does the code know when a click happens?

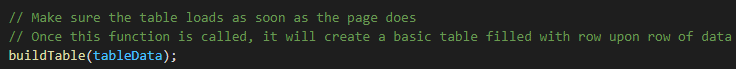
Another aspect of D3.js is that it can listen for events that occur on a webpage, such as a button click. The next code we add will be tied to the filter button we'll build on our webpage.

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.



**Build the Final Table**

Make sure the table loads as soon as the page does. 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.



# Build the HTML

## [11.6.1: Bootstrap Components](https://courses.bootcampspot.com/courses/978/modules/items/358843)

Test the webpage by building the HTML page I’ll be showcasing my work on.

Keep the different ids she created in app.js in mind while she pieces this webpage together. There will be a button (#filter-btn) somewhere as well as a #datetime id nestled into some tags. She'll need to build the base of the table so that the code we helped construct knows where to put the table's data. This means that the columns and rows will all need to be defined manually.

go back to the index.html file that we created earlier. If you haven't already, open that file in VS Code. Next, use a shortcut to autofill the basic HTML layout by typing an exclamation mark on the first line, then press enter on your keyboard.

Text

Description automatically generated



Title = "UFO Finder"



Link to Bootstrap's content delivery network (CDN)

Text

Description automatically generated



Link to the stylesheet. Since we created a stylesheet and index.html file at the same time, we'll just link to the style.css file that's in our css folder.

Text

Description automatically generated



Graphical user interface, application

Description automatically generated with medium confidenceBegin setting up the body of the HTML, where we'll store our components.

**Wrapper class** adds a bit of extra functionality to Bootstrap. It helps group the elements (e.g., title, paragraph, table, and filters) and specifies the styling in our stylesheet.



**Review the Storyboard again**

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Build the Navbar**

Within the wrapper …



navbar =  indicates to Bootstrap that we want a component that fits across the top of the page. navbar-expand-lg = provides extra responsiveness to the default navbar behavior. When the viewing area is reduced from a large to a smaller screen, the navbar will collapse or resize itself smoothly.

**Add functionality to our navbar**

In this case, we don't need to redirect readers to another section of the webpage. Instead, we want to reset the webpage after a filter has been applied to the table. This is accomplished by linking to the homepage, index.html





navbar-brand = is a type of default styling that helps with the site’s aesthetics

href = points to the index.html file we're working on. When a user clicks on that link, the page will reload and the default unfiltered table will appear, ready for new input.

Need to display text in the navbar and complete the link. For now, use "UFO Sightings”

**Test It**

Make sure our link to Bootstrap and the navbar is implemented correctly.

Navigate to File Explorer > go to index.html file > open with… Chrome

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence**Add the Jumbotron**

Bootstrap looks for certain classes within HTML tags to indicate where to apply styling, such as by adding a "jumbotron" class to a div tag. Text nested within these tags will have visual enhancements automatically added. For example, a header tag nested within a jumbotron will be larger and bolder than a header tag on its own. In our code editor, let's add a jumbotron with a header that reads "The Truth Is Out There."



Graphical user interface, text, application, Word

Description automatically generated

**Add the Article Title and Paragraph**

Utilize the grid system, which will let us assign screen space to each element

Will need to be set up in a certain order:

1. A screenshot of a computer

   Description automatically generated with medium confidencethe container, then a row

container-fluid =  ensure that both elements we're adding will span the width of the viewport

row = nested class makes sure that the title and paragraph will align neatly along the page

1. how many columns each element will require?

title requires less width than the paragraph. assign four columns to the article title and the remaining eight to the paragraph. Remember that each element will be within its own div.

**Scaffolding is in place to hold the title and article**

Insert the title and paragraph Dana has chosen. Using an <h3 /> tag, nest the article title ("UFO Sightings—Are They for Real? Ufologists Weigh In") in the first column.

<small> = tag nested to add a little bit of extra styling to help de-emphasize the second portion of the title.



In the second div, the one that uses eight columns, add the article paragraph:

*Are we alone in the universe? For millennia, humans have turned to the sky to answer this question. Now, thanks to research generously funded by W. Avy, a UFO-enthusiast and amateur ufologist, we can supplement our sky-searching with data analysis. <br> <br>"The release of this analysis is well-timed: It coincides with the celebration of World UFO Day, which is a moment for ufologists around the world to connect, relax, and sample a range of UFO-themed snacks," said Dr. Ursula F. Olivier, the world's preeminent expert on circular sightings. "Citizen-scientists can be especially helpful in both cataloguing sightings—which is hugely helpful for us in our search for aliens—and in helping us celebrate the work that has already been done, such as this data visualization project, which will help us raise awareness of the ubiquity of sightings!" <br><br>Not everyone is ready to celebrate, however. Local CEO and vocal anti-alien activist V. Isualize reached out to our reporters to go on record as firmly opposed to any attempts to provide access to this data. "If there are aliens, they certainly would like to be left alone," she stated, before directing us to the Leave Aliens Alone (LAA) community engagement initiative she founded and funds. <br><br>So what do YOU think? Are we alone in the universe? Are aliens trying to contact us, or do they want to be left alone? Dig through the data yourself, and let us know what you see.*

Text

Description automatically generated

**Create the Table Filter**

Build the section for the filter. The ids created in our JavaScript code (#filter-btn and #datetime) will come into play here.

1. **Set up the next row** that will hold the filter section and the table.

Determine how many columns to designate for the filter section and how many for the table.

Look at our storyboard again.

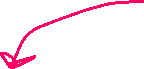
Text

Description automatically generated

Text

Description automatically generatedCreate a div with space for 3-column (filter) & a 9-column (table.s)

1. **Build the filter**



1. Text

   Description automatically generatedAdd a form tag, then build the form by nesting additional elements within it. Let's give the form a name so that users will know what it's for.

Add what readers will be searching for. Our JavaScript code has a filter setup to search by date. Add an input box for a date. We'll also need the button that we referenced (#filter-btn) so that searches can be executed. To add these items as cleanly as possible, we're going to create a set of list tags, nested within an unordered list tag. We'll also include some Bootstrap classes to keep things extra neat.

Text

Description automatically generated

1. starting the unordered list. We're going to give this a class of "list-group." Using this specific class lets Bootstrap apply predetermined styling to the list. We can spruce it up further if we want to after we're done.



1. Next we need to add the list items: one for the input, one for the button. Each <li /> tag will have a class of "list-group-item."

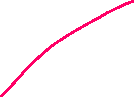
Text

Description automatically generated

1. Now let's add the date input field in the first <li /> tag. We'll add two new HTML components here: label and input. The label will be used as a prompt to encourage users to input a date. The input field is where users will complete the input.







1. This label represents a caption for the date item. The text Enter Date serves as the actual label.





* type="text" means that the code will look for text to be input.



* The placeholder is an example of a date to search, so users know both the location and the format to use when inputting a date.
* id="datetime" is what our JavaScript code will look for when the button is clicked and the function is executed.



add our button tag with a few additional attributes: the id we defined in our JavaScript code (#filter-btn), a type, and a class. When the button is clicked, the input from earlier is picked up by our JavaScript code and then applied to the filter.

* type="button" tells the browser that, by default, the button does nothing. However, our custom JavaScript script will overwrite the default behavior—as if the button is waiting for instruction.
* class="btn btn-default" attribute adds some Bootstrap styling to the mix to help keep the element neat and tidy.
* Graphical user interface, text, application

  Description automatically generatednamed our button "Filter Table," by nesting it between the opening and closing button tags.

1. Prep the Table for Data

build the table. In our app.js script, we use D3 to select the "tbody" HTML tag. We're going to build that component and link the JavaScript and HTML files.

The table and filter components are both inside the same container, so we only need to construct the table HTML.

An HTML table has several nested layers:

Bootstrap styling: presents a table that is slightly striped to give variation between the rows of data.

Text

Description automatically generated

column headers *element*

everything inside will be

a row of data

column names/ headers

body container (stored data)

table container

Take another quick look at one of our data.js objects.

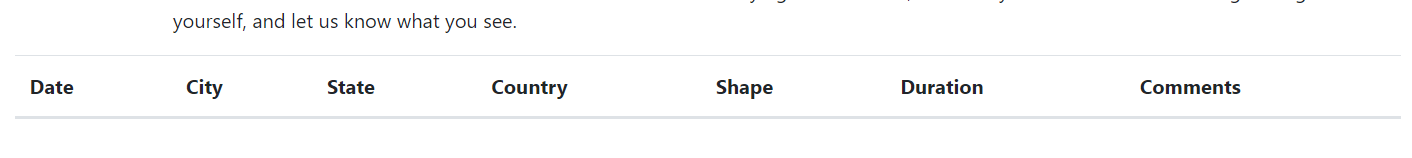
Text

Description automatically generated with medium confidence

Text

Description automatically generatedThe information in our object is present as key-value pairs (KVPs). Each object will have the same key, but different values—these keys (such as datetime, city, and state) will be our table headers.

Add a <th /> tag for each table header. Let's clean up the text a bit by using proper capitalization and spacing.



## [11.6.2: Add the Data](https://courses.bootcampspot.com/courses/978/modules/items/358846)

the only way to truly test it is to tie it together with the JavaScript code we created earlier. We'll tie them together by adding <script /> tags then linking to the file's location. This is very similar to when we added a link to our stylesheet. Only this time, the links to our scripts will be at the bottom of the page.

When adding multiple <script /> links to a webpage, the order matters. The order we link our files will be the order they are executed. If we link app.js before data.js, then the app will try to build the table before the data has loaded. This will generate an error and break the table.

At the bottom of the page, under the last <div /> tag, we will need to add our scripts. There are three we need to include, in the following order via a script tag:

1. A link to a D3.js CDN
   * allows the library to "listen" in on our code, or react to user input. For example, if we did not add this link, our d3.select section of code in app.js wouldn't know where to insert data.
2. The file path to data.js – link because the UFO sightings data needs to be loaded before it can be accessed
3. The file path to app.js – “

Text

Description automatically generated

## [11.6.3: Customize the Page with CSS](https://courses.bootcampspot.com/courses/978/modules/items/358851)

spruce the page up a bit

CSS syntax, as with all coding languages, is very exact. Each customization entry follows a pattern:

*Area of customization is set, usually ID’d by its element, class attribute, or id attribute: value of the customization;*

Text

Description automatically generated

style.css

index.html We chose to apply the styling to the body of the page because it is one of the outermost elements—every other visual element lies within the <body> of the page. If we were to replace "body" with "h1," then only the h1 elements would be affected. Similarly, because we want to apply the same styling to the entire page, we can add a "bg-dark" class to our body element. In our index.html file, update the opening <body> tag



Let's also make use of Bootstrap's built-in styling, starting with the navbar. In the index.html file, let's add navbar-dark bg-dark to the <navbar> classes



Update the jumbotron to something more appealing by **changing the font and background**. This time, we'll add a Bootstrap class called "display-4" to the <h1> element inside the jumbotron, which will change the font style and size for us.



style.css Make the background more interesting. Dana has chosen a space image to use as a background instead of a solid color; **download it and save it to the "images" folder** you created in your repo.

[Download nasa.jpg](https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_11/nasa.jpg) To **assign this as our new background**, we'll need to use our style.css sheet. Open that file in your code editor. What we'll do next is create an entry that will style only the jumbotron element.

* .jumbotron - tells the stylesheet to apply the changes only to divs with a class of "jumbotron."
* background image - then tell the code where the image is stored.
* background size - is set to span the entire width and height of the element, no matter the size of the viewport.
* text align - be in the center of the element.

A screenshot of a computer

Description automatically generated with medium confidence

The form element in the HTML will also need a small update: The text is now white, but the form's background is also white by default. Identifying what the form is telling us is difficult in its current state, so let's update the font color.

First, let's look at the form element in our HTML to identify which one we'll apply the styling to.

Text

Description automatically generated

# Application

## [Module 11 Challenge](https://courses.bootcampspot.com/courses/978/modules/items/358857)

modify the code in your index.html file to create more table filters. In addition to the date filter you created in this module, you’ll add filters for the city, state, country, and shape, as shown in the following image:

A screenshot of a computer

Description automatically generated with medium confidence

## [Module 11 Career Connection](https://courses.bootcampspot.com/courses/978/modules/items/358859)

* Replace the handleClick() function in your app.js file with a new function that saves the element, value, and id of the filter that was changed.
* Create a new function to loop through the dataset and keep only the results that match the search criteria. The webpage will be updated with the search criteria after pressing "Enter".
* Rename the app.js you created in this module as app\_1.js or something similar to avoid using the wrong file for the Challenge.

1. Download the ufo\_starterCode.js, rename it app.js, and place it in the js folder of your UFOs GitHub repository. The starter code includes the code used to populate the table from this module.
2. In the index.html file, remove the list (<li></li>) element that creates the button.
3. Create four more list elements: city, state, country, and shape. These will be similar to the "Enter Date" list element. Each element should have the same "id" as the object properties in the data.js file.
4. In Step 1 of the app.js file, create an empty filters variable to keep track of all the elements that change when a search is entered. This variable will be used in Step 5 to store the property “id” and the value that was entered from user input.
5. Next, you will need to write code for two functions whose names we’ve provided in the starter code, updateFilters() and filterTable().
   * The updateFilters() function will replace your handleClick() function and update the filters variable you created in Step 1.
   * The filterTable() function will filter the table data by the value that is entered for the "id" that has changed.
6. For Step 2, located before the buildTable(tableData) function at the end of the starter code, modify the event listener from this module so that it detects a "change" on each input element and calls the updateFilters() function.
7. In Step 3, we’ve provided the function, updateFilters(). Inside this function, you’ll write code in Steps 4-5 to update the filters based on user input.
8. In Step 4a, create a variable that saves the element that was changed using d3.select(this).
9. In Step 4b, create a variable that saves the value of the changed element’s property.
10. In Step 4c, create a variable that saves the attribute of the changed element’s id.
11. In Step 5, write an if-else statement that checks if a value was changed by the user (variable from Step 4b). If a value was changed, add the element’s id (variable from Step 4c) as the property and the value that was changed to the filters variable you created in Step 1. If a value was not entered, then clear the element id from the filters variable.

**HINT how to update the filters based on user input – MOVIE**

* Using d3.select(this), you can select the changed element and retrieve the changed property and HTML id.
* Watch the video for a detailed overview of how to update filters based on user input.

1. In Step 6, inside the updateFilters() function, call the filterTable() function that will be used in Step 7.
2. In the filterTable() function in Step 7, write code to filter the table based on the user input that is stored in the filters variable.
3. In Step 8, create a variable for the filtered data that is equal to the data that builds the table. This variable will hold the updated table data based on the user input.
4. In Step 9, loop through the filters object and store the data that matches the filter values in the variable created in Step 8.
5. In Step 10, rebuild the table with the filtered data by passing the variable created in Step 8.
6. Deploy the web app on your GitHub pages.

# Class Notes Day 1

## Text Description automatically generatedClass 1 Activity 1

Python v. JS

Text

Description automatically generatedText

Description automatically generated

## Text Description automatically generatedClass 1 Activity 2

Python v. JS

Text

Description automatically generatedText

Description automatically generated

## Graphical user interface, text, application, email Description automatically generatedClass 1 Activity 3

Text

Description automatically generated

## Class 1 Activity 4

## Text Description automatically generatedClass 1 Activity 5

Text

Description automatically generated

## Text Description automatically generatedClass 1 Activity 6

Graphical user interface, text, application

Description automatically generated

Text, letter

Description automatically generated

## Class 1 Activity 8

Text

Description automatically generated

A picture containing text

Description automatically generated

A picture containing text

Description automatically generated

A picture containing text

Description automatically generated

# Class Notes Day 2

## Class 2 Activity 1 .forEach

Text

Description automatically generated

**Class 2 Activity 2**

1. This link: <https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.3.7/css/bootstrap.min.css>
2. when you take out the “min” from the URL it puts the “whitespace” back in
3. And this code brings in the Bootstrap library

Text

Description automatically generated



Text

Description automatically generated



Text

Description automatically generated



Text

Description automatically generated

Text

Description automatically generated



Graphical user interface, text, application, email

Description automatically generated

Confirm if something exists

Text

Description automatically generated

Object.entries

Graphical user interface, text, application, email

Description automatically generated

**Class 2 Activity 3**

**Class 2 Activity 4**

Text

Description automatically generated

Text

Description automatically generated

Add Bootstrap Style… ↓

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

**Class 2 Activity 5 D3 tables & graphs**

D3 auto-populates data into table, you just have to name the column headers.

Text

Description automatically generated

Text

Description automatically generated



**Class 2 Activity 6**

**Class 2 Activity 7**