

TGIF PROJECT

Getting started with JavaScript

Background

HTML and CSS are the languages for describing the content and visual appearance of web pages. They let you make very beautiful and useful web sites. But if you want to make an interactive web application, like a game or an interactive data display, you need **JavaScript**.

Every web browser knows how to run JavaScript code. With JavaScript, you can

- make parts of web pages appear, disappear, and move around
- respond to mouse clicks, finger taps, drag and drop, and keyboard entry
- calculate complex mathematical formulas
- repeat operations in loops that can run hundreds or thousands of times

Basically, with JavaScript on a web page, you can do anything you can do in any other programming language, with one exception. JavaScript in a web page is not allowed to read or write files on your computer. It is prevented from doing that so that hackers can't write web pages that steal information from you or damage your computer files.

A Quick Demonstration

You can use **JS Bin** to see what JavaScript can do.

1. Click on [this JS Bin link](#) to open a new JS Bin window. This link has been defined so that you should see boxes for entering HTML, CSS, and JavaScript.
2. Copy and paste the HTML below into the HTML box.

```
<p id='test'>
Hello, Moon!
</p>
```

3. Copy and paste the CSS below into the CSS box:

```
#test {
  background:
url(https://c1.staticflickr.com/9/8500/8315697815_69cb2a99ba_b.jpg);
  background-size: cover;
  color: lightgreen;
  font: bold 14px arial;
  width: 140px;
  height: 140px;
  border: 2px solid #999999;
}
```

4. Make sure that the Moon image

URL https://c1.staticflickr.com/9/8500/8315697815_69cb2a99ba_b.jpg still works. Pictures come and go from the Internet all the time. If this one is gone, use your search engine to find another comparable small picture of the Moon and use that URL instead.


JS Bin should automatically display this HTML and CSS for you.

5. So far, this is just HTML and CSS. Now let's add some JavaScript so that something happens when we click on the Moon image. For this code to work though, you have to tell JS Bin to include the jQuery library. Click on **Add Library** in the upper left and select a current version of **jQuery**.

After you do this, you should now see some extra code added to your HTML on the left that loads the jQuery library you selected.

6. Now copy and paste this JavaScript code into the JavaScript box on the right.

```
$('#test').on('click', function() {  
    $(this).text('Goodbye, Moon!').fadeOut(4000);  
});
```

7. 

this:

- When the user clicks on the HTML element with the ID test...
- Change the text of that element to say "Goodbye, Moon!"...
- Then take 4 seconds (4000 milliseconds) to make the element fade away.

This code does quite a bit in a few lines, because it uses functions defined in a very popular library called **jQuery**. It could be written with plain JavaScript, but would take about twice as much code and be much harder to read. You will learn how to use jQuery later on.

8. Now click on the Hello, Moon image in the **Output** box. The text on the image should immediately change to "Goodbye, Moon!" Then the image and text should fade away after several seconds.

Congratulations! You just wrote and ran your first JavaScript!

Video Tutorials

Code along videos are a good way to learn coding. As everyone has his own preferences, it is up to you to find your preferred code along tutor! For instance here is Net Ninja's code along for JavaScript beginners: [Net Ninja YouTube](#)

JavaScript Practice Exercises

While your boss, Paula Davis, is working with the client let's do some practice exercises to understand some of the basic practices of coding in JavaScript.

JavaScript Basics

Part 1: Set up a JavaScript playground

Since you'll be learning to write JavaScript for web pages, the best way to experiment is by making a very simple web page that loads and runs some JavaScript. You may find this page useful when testing out ideas in later tasks.

1. Create a directory for your code, e.g., playground.
2. Inside that directory, create an HTML file, e.g., index.html.
3. In the HTML file, put some simple HTML to show some text that will let you know you're looking at the right file. Be sure to include `<!DOCTYPE HTML>` at the top.
4. Add a script element to this file to load JavaScript from the file main.js. See this example for the syntax. Use the version listed for HTML5.
5. In the same directory, create the file main.js.
6. In the JavaScript file, add a line into your JavaScript file to print the following text in the console: "Starting javascript...".
 - a. In programming, printing to some console window is often called "logging". Hence the function to use in JavaScript is `console.log`. [See this example](#).
7. Open your HTML file in a browser.
8. Open the developer console. See [here](#) for how to do this. Is your message there? If not, try reloading the file.

TIP: If you still don't see the message, check your HTML and JavaScript file for typos. Make sure file names match exactly, including case. Make sure the JavaScript file is in the same directory as the HTML file.

TIP: As you do the exercises below, keep adding new code to do what each exercise asks for. At the end, you'll be asked to submit all the JavaScript for mentor feedback.

Part 2: Writing expressions with variables

To get started, here are some simple exercises to create and use variables and expressions. Variables are how you store data and results of calculations. Expressions are how you calculate values.

Resource: [quick introduction to JavaScript expressions](#).

Exercise 1:

In your JavaScript file create a variable called `myName` with your name as the value. Put your name inside string quotes, e.g., `"my name"`. Then add a line of code to print the variable name to the console after the previous message.

Save your JavaScript file in your editor. Reload the HTML page in your browser. You should see your name printed. If not, investigate and fix.

Exercise 2:

Create a variable called `age` with a number that is your age. Do not use string quotes for numbers.

Add a line to print that variable in the console. Save the file and reload the page. You should see your name, and your age.

Exercise 3:

Create a variable called `ignasiAge` with a value 32. Create another variable called `ageDiff` and set it to an expression that calculates your age minus Ignasi's age. Print the value of `ageDiff`.

Save the file and reload the page. You should see your age, and the difference. If you are younger than Ignasi, you should see a negative number.

Part 3: Writing code with conditionals

Conditionals are forms used programming to tell the computer to do different things, depending on some test, e.g., "if the user is logged in, say "Hi" else say "Please log in."

The most basic conditional form is the `if...then...else....` See the ebook for a discussion.

Exercise 4:

Write a conditional that compares the variable with your age with the number 21. It should print either "You are older than 21" or "You are not older than 21", appropriately, depending on your age.

Save your JavaScript file and reload the page. Make sure you see the correct message. Try changing your age in the JavaScript file to make sure the other message prints when it should.

Exercise 5:

Write a conditional that compares your age with Ignasi's age. This conditional will need to test if you are older, younger, or the same age, and print, appropriately, either "Ignasi is older than you", Ignasi is younger than you", or "You have the same age as Ignasi".

Save your changes and reload the file.

JavaScript Array Functions

Sorting an Array

Exercise 1:

Create an array with all the names of your class (including mentors). Sort the array alphabetically. Print the first element of the array in the console. Print the last element of the array in the console. Print all the elements of the array in the console. Use a "for" loop.

Save the file and reload the page. You should see the first element of the sorted array, the last element of the sorted array, and a list of all the elements in order in the array in the console.

Looping Over an Array

Exercise 2:

Create an array with all the ages of the students in your class. Iterate the array using a while loop, and then print every age in the console. Add a conditional inside the while loop to only print even numbers. Change the loop to use a "for" loop instead of a "while" loop.

Save your changes to your JavaScript file. Reload the HTML page in your browser. You should see every age printed, then only the even numbers printed. If not, investigate and fix.

Functions that Use Arrays

For the following exercises, you cannot sort your array. Be sure your solution works for any array that it is passed!

Exercise 3:

Write a function which receives an array as a parameter and prints the lowest number in the array to the console.

Save the changes to your JavaScript file. Reload the HTML page in your browser. You should see the lowest number in the array printed in the console. If not, investigate and fix.

Exercise 4:

Write a function which receives an array as a parameter and prints the biggest number in the array to the console.

Save the changes to your JavaScript file. Reload the HTML page in your browser. You should see the biggest number in the array printed in the console. If not, investigate and fix.

Exercise 5:

Write a function which receives two parameters, an array and an index. The function will print the value of the element at the given position (one-based) to the console.

For example, given the following array and index, the function will print '6'.

```
var array = [3,6,67,6,23,11,100,8,93,0,17,24,7,1,33,45,28,33,23,12,99,100];  
var index = 1;
```

Save the changes to your JavaScript file and check your browser console. You should see the number at the correct index printed in the console. If not, investigate and fix.

Exercise 6:

Write a function which receives an array and only prints the values that repeat.

- For example, given the following array and index, the function will print '6,23,33,100'.

```
var array = [3,6,67,6,23,11,100,8,93,0,17,24,7,1,33,45,28,33,23,12,99,100];
```

Save the changes to your JavaScript file. Reload the HTML page in your browser. You should see an array of the repeated numbers printed in the console. If not, investigate and fix.

Exercise 7:

Write a simple JavaScript function to join all elements of the following array into a string.

```
myColor = ["Red", "Green", "White", "Black"];
```

Save the changes to your JavaScript file. Reload the HTML page in your browser. You should see the following in your console:

```
"Red", "Green", "White", "Black"
```

If not, investigate and fix.

JavaScript String Functions

Exercise 1:

Write a JavaScript function that reverses a number. For example, if $x = 32443$ then the output should be 34423.

Save your JavaScript file and reload the page. Make sure you see the correct output. If not, investigate and fix.

Exercise 2:

Write a JavaScript function that returns a string in alphabetical order. For example, if $x = \text{'webmaster'}$ then the output should be 'abeemrstw' . Punctuation and numbers aren't passed in the string.

Save your JavaScript file and reload the page. Make sure you see the correct output. If not, investigate and fix.

Exercise 3:

Write a JavaScript function that converts the first letter of every word to uppercase. For example, if $x = \text{"prince of persia"}$ then the output should be $\text{"Prince Of Persia"}$.

Save your JavaScript file and reload the page. Make sure you see the correct output. If not, investigate and fix.

Exercise 4:

Write a JavaScript function that finds the longest word in a phrase. For example, if $x = \text{"Web Development Tutorial"}$, then the output should be "Development" .

Save your JavaScript file and reload the page. Make sure you see the correct output. If not, investigate and fix.

TGIF Sprint 1

Your Task

You have been asked to create a demo web app displaying Congressional data from the ProPublica Congress API on an HTML page. Specifically, you need to create three pages, Home, and one each for the Senate and one for the House of Representatives. Both will look very similar. You should try to avoid duplicating code as much as possible.

There are several steps to this task divided in 4 epics:

- Set up your environment and get API data
- Build Senate and House web pages
- Build Home page and connect them together

Epic 1: Set up your environment and get API data

Task 1: Register your API Key

API Keys

Web services that return data in JSON or XML form have become very popular. They allow a data consumer web site A to get data from a data producer web site B (or several web sites), manipulate that data, and display it to end users. One concern for the producer is that some consumers may overwhelm the producer's servers with frequent requests for large amounts of data.

To control this, many web services require that all requests include an API key. This is a long sequence of characters that you need to include in every call to the data producer. You can often get an API key for free. The producer uses the key to track who is asking for data. The producer can easily limit how often you can ask for data and how much data you can get in one call. The producer may allow a consumer to pay for increased access.

JSON

JSON (JavaScript Object Notation) is a format for representing complex data in plain text form. As the name suggests, the format originated with JavaScript, but Douglas Crockford proposed using it as a general data exchange format instead of the more verbose XML (Extensible Markup Language). While XML was very general, powerful and popular, JSON quickly became more popular, because it was easier to code with and was comparatively more compact.

Get a ProPublica Congress API Key

1. Go to [the ProPublica site](#). There's a link there to a form for requesting an API key. This is free.
2. Wait for the API key to be emailed to you.
 - a. Be patient. This can take a day, perhaps more on weekends.
 - b. If you don't receive a key within a day, check your email spam folder. Automated emails of this sort often trigger spam filters.

Task 2: Create your local project folder structure

Task 3: Setup your tools (VS code or Bracket)

Task 4: Add extensions and plugins to your IDE

Code faster and easier by only adding some extensions directly from your integrated development environment (IDE) such as:

- Emmet for quick code template generation from keywords
- Beautify or Prettier for code auto formatting

Task 5: Create new GitHub repository for your local Git

Task 6: Get the JSON data and convert to JS

For developing and testing your code, it's best to have a local copy of some real data.

- Testing will be faster
- You won't bother the ProPublica server.

To get a local copy, make one AJAX call to get the JSON you need, save that data in a JavaScript file, then use the JavaScript file. In a later task, it will be easy to switch to using live data.

1. Go to [our AjaxTester web page](#)
 - a. AjaxTester is a general form that can be used to test AJAX JSON web APIs.
2. Construct a URL for the ProPublica Congress web service to get the members of the Senate for the 113th Congress.
 - a. Use [this page](#) to see what that looks like.
3. Enter the URL in the Endpoint field of AjaxTester.
4. Click the + Add header button.
5. Enter X-API-Key as the header name.
6. Put your API key as the header value.
7. Click the Ajax request button. If the URL and key are correct, JSON should appear in the textbox on the right.
 - a. You will get an HTTP 0 error if you have an incorrect URL or invalid API key.
8. When you have JSON in the textbox, click on it and hit control-A (Windows and Linux) or command-A (Macintosh), to select the entire JSON text.
9. Open the text editor you use for programming, create a new file, paste the JSON into it, and save under a clear name, e.g., pro-Congress-113-senate.json
10. Repeat the above steps, but for the House of Representatives. Since this has almost 10 times as much data, it will take a little longer to appear. Save in another file, e.g., pro-congress-113-house.json

Convert JSON to JavaScript

- Although you'll use these JSON files in later tasks, to create a standalone demo you need files that are executable JavaScript code. If you try to load the JSON in HTML with `<script src="pro-congress-113-senate.json"></script>`, you will get a syntax error. A JSON object is not executable code. Fortunately, it's easy to fix.
1. Open your JSON Senate file in your editor, and insert
`var data =`
at the very start. This is JavaScript for "save the JSON object in the variable named data." Save this as a new file with the same name but the extension `.js`, for JavaScript.
 2. Repeat with your JSON House file.

Epic 2: Build Senate and House web pages

SPIKE: DOM manipulation

Task 1: Verify that files are connected and log into the console

Confirm that you can load the JSON data from your JavaScript files into a web page.

1. Using the basic html file provided in the email, create an HTML page senate-data.html with the following elements

- a PRE element with the ID "senate-data"
- a SCRIPT element that loads the JavaScript file you previously saved with Senate data
- a SCRIPT element that has source code that puts the value of the variable data (pulled from the JavaScript file you loaded) into the PRE element

TIP: Do not **put** SCRIPT tags in the HEAD, in the middle of the BODY, and at the end of the BODY. Middle is definitely not good. Common practice these days is to **bottom-load scripts**.

TIP: To make the data variable's contents a string (easier to read), use the **stringify** **method**: `JSON.stringify(data,null,2)`

2. In order to write JavaScript that stores text into an HTML element, you can use `getElementById`. You can check Resources for more information.

Example:

```
document.getElementById("senate-data").innerHTML =  
JSON.stringify(data,null,2);
```

3. Open the HTML page in your browser.
4. If you see the JSON data, go on to the next step. If you don't, open your browser's Developer Console, reload the page, and look for error messages.

Task 2: Format the Senate data as a table

Once you are successfully getting the JSON data into your page, you can focus on displaying that data in a user-friendly HTML table.

1. Replace the PRE element on your page with a TABLE element with the same ID.
2. The data variable contains a results field. The results field contains an array of members of the Senate.

3. Change your JavaScript code to be a loop over that array. (This code will be in place of the code you had in the last step that put content into the PRE element.) The loop should construct a string of HTML using an approach such as the following:
 - full name, built from the array element fields for last, first, and middle names
 - party (D, R, or I)
 - state (two letter code)
 - seniority (years in the office they currently hold)
 - percentage of votes with party, with a % added
 - For each element of the array, it should build a TR element. Within that TR element, it should create strings with TD elements, filled with the following data from the array element:
4. After building the string, the JavaScript should store the HTML in the TABLE element.
See Resources for review material on arrays, loops, and using both to construct and display a string combining data and HTML markup.
5. Open the page in your browser. If the page looks correct, go to the next step. If not, open the Developer Console and look for error messages.

TIP: Do not put SCRIPT tags in the HEAD, in the middle of the BODY, and at the end of the BODY. Middle is definitely not good. Common practice these days is to [bottom-load scripts](#).

TIP: Use (... || "") to avoid null's in the table, like the middle name.

Task 3: Style Page with Bootstrap

Style/build out page per wireframes included in the email.

1. Add code to load the Bootstrap 3 CSS file. See Resources for help on doing this.
2. Add the appropriate elements and attributes to your HTML to build out the navigation bar with dropdown menu. See Resources for help on setting up your page to use Bootstrap.
3. Add the appropriate elements and attributes to your HTML to build out the page body and footer per the wireframes.
4. Add the appropriate elements and attributes to your HTML to use the default Bootstrap table styling. The table styling will not have borders, which is what our client prefers. See Resources for help on using Bootstrap's provided classes to style tables and their contents.
5. Check your page in the browser

Task 4: Repeat for House Data

Do the same process for the House data starting with creating an HTML page named house-data.html. When you're through, this will be a much longer page than the Senate page, but you should have very little new work to do.

Epic 3: Build Home page and connect them together

Task 1: Create Home page and load CSS

Using the basic html file provided in the email, create an HTML page (home.html).

Task 2: Apply universal page elements (navbar, page body, footer)

Build out the navigation bar, the page body, and the footer like you did for the House and the Senate pages

Task 3: Add logo to your page

Select an image for the TGIF Logo and an image for the Home page.

Call these images in your HTML file and format with Bootstrap per the wireframes.

Task 4: Interlink all your current pages

Now update your displayed content with helpful links (to the Senators' home pages) and descriptions of your table's contents (with headers).

1. Change your code so that when it constructs the HTML, the name is a link to the Senator's official home page, as given by the URL field in the results array.
2. Open the page, and test several of the links. Make sure they go to the right web page.
3. Add headers for your table to describe the contents of each column (e.g. 'Senator', 'Party Affiliation').

Task 5: Build accordion UI element

Add the appropriate elements and attributes to your HTML to use the Bootstrap Accordion feature.

The content on the home page should expand out when a link or button is clicked. For help on using Bootstrap's classes to create an accordion feature consult the [Bootstrap Documentation](#).

Task 6: Post to GitHub

TGIF Sprint 2

Your Task

You have been asked to create two pages with statistics on the Senate and the House. This will involve:

- looping over the data about the members of the two houses, counting and adding as you go
- constructing an HTML table to display the final results with clear labels and properly formatted numbers

Each statistic requires different logic to calculate. Some are simple counts, others are complex. The best way to organize such code is to define a function to calculate each statistic. Each function may in turn call other functions.

Best practice also suggests separating the calculation of the statistics from the construction of the HTML. This is called separating the model from the view. That way your calculation code can be written and tested separately from the HTML code. Also, the calculation code could more easily be reused in future programs.

So the steps in this Plan of Attack first focus on creating a JSON object to hold all the statistics. We'll refer to this JSON object as the statistics object.

You'll create an HTML page for the Senate, but you won't worry about the HTML itself at first. Just write the JavaScript to do the statistics you need, and use the Developer Console to test how your code works.

Once the JSON statistics object is constructed correctly, then you'll add the code to create the HTML to display the JSON, just as you did in a previous task with the JSON returned by the Congress API.

This sprint is divided into 5 EPICS:

- Build the Senate at a glance table
- Build Senator Attendance page
- Build Senator Party Loyalty page
- Build House Attendance and Party Loyalty Pages
- Implement Checkbox filters into your website

Epic 1: Build Senate at a glance table

Task 1: Create your JavaScript object to store the statistics

To hold your calculation results, you need a JavaScript object, which we'll call the *statistics* object.

The statistics object will have a **key** for each statistic, e.g., "Number of Democrats", and the **value** for that statistic. Use strings for the keys that are what you will want printed on the web page. The values you will calculate in later steps.

1. Open the Attendance starter HTML file. Create a Senate Attendance statistics page from this file.
2. Create a new JavaScript file that is going to contain all your statistic functionalities and instantiate an object named statistics.
3. In the statistics object, include fields for all the statistics the client has requested (e.g., "Number of Democrats"). For now, initialize each value to zero.

When you create the statistic object, remember that you will read from that same object to build table. A good structure of the object would facilitation the task!

To get the number of members in each party...

There are two obvious ways to do this.

- Go through the list of all members in one of the chambers and count the Democrats, Republicans, and Independents
OR
- Make three lists, one for each party, and get the length

If all you needed was the counts, the first approach would be more efficient for that task. But since some of the other statistics will need lists of party members, the second approach will be more efficient overall. So:

1. Open your Senate Attendance HTML file.
2. Write code to create and fill three variables, one for a list of the Democrat objects, one for the Republican objects, and one for the Independents.
3. Then update your statistics object with the number of members in each party, e.g. for the key "Number of Democrats" replace the default value of zero with the length of the list of Democrat objects.
4. Use `console.log()` to print your updated statistics object to the Developer Console so that you can inspect it to verify that it contains the correct answers. See *Resources* for help with `console.log()`

5. Open your Senate Party Loyalty HTML file. In order to be DRY (Don't Repeat Yourself), move your statistics object code to a separate javascript file and load it in both the Senate Attendance page and the Part Loyalty Page.

Task 2: Calculate the average "votes with party" for each party

1. Open your statistics javascript file.
2. To get an average of an array of data, you first sum up the relevant numbers from that array, then divide by the size of the array. Do that with the list of Democrats, using the percent party votes for each member.
3. Store the result in your statistics object.
4. Repeat for the list of Republicans.
5. Call your results in your Senate Party Loyalty HTML file.
6. Use console.log to print this object to the Developer Console so that you can inspect and verify it contains the correct answers.

Task 3: Build the Senate at a Glance table for Senators

Since you've already built a table in the previous sprint, you should have a pretty good idea how to represent this data in a table based on the wireframes from the client.

Epic 2: Build Senator Attendance Page

Task 1: Calculate least engaged based on missed votes data for bottom 10%

1. Open the Party Loyalty starter HTML file.
2. Create the Senate Party Loyalty statistics page from this file.
3. Repeat steps 2-3 that you did to calculate the average votes with party, only do the calculation for missed votes

Task 2: Display top 10% least engaged in the table, sort, and handle duplicate data points

Since you've done a similar data sorting task in the previous epics, the real challenge now is to generalize your code into a few functions that can get these answers, with as little repeated code as possible. Can you come up with one function that, given the appropriate parameters, can do all these statistics?

When you are happy with your results, collect and store them in your statistics object with the appropriate keys. Call the results in Senate Party Attendance HTML file and in the Senate Attendance HTML file.

Task 3: Calculate most engaged based on missed votes data for top 10%

Now, execute the same process only for the opposite problem...which Senators missed the FEWEST number of votes?

Task 4: Display top 10% most engaged in table and sort

TIP: Don't forget to consider missed votes! As Paula explains in the email, you should include all the Senators (and Representatives in the later epic) with the same attendance. Just taking 10% might remove some members who actually attended the same percentage of sessions as ones that you included.

Epic 3: Build Senator Party Loyalty page

Task 1: Build Least Loyal table by calculating and displaying top 10% of Senators who didn't vote with party

This is a non-trivial problem, and there are multiple ways to solve it, some simpler than others.

1. Think about the problem in general, which is
 - Example: 100 values, with duplicates, and you want to find the 10% of those values that are the smallest values in the set.
 - Example: The smallest value in the set is 84.2%, and that value occurs 3 times.
 - Example: 3/100 is not greater than or equal to 10% so repeat for the next smallest value:
 - The next smallest value is 84.5%, and that value occurs 2 times.
 - Now you have the 5 smallest values.
 - 5/100 is still not greater than or equal to 10% so choose the next smallest value and repeat again.
 - Stop when your set of smallest values represents 10% or more of the total.
 - Given a list of N numbers, with duplicates and a percentage K
 - Find the *smallest* value X such that there are M numbers less than or equal to X
 - M / N is greater than or equal to K
2. Develop and test code to solve this problem, then apply it to the data on voting with one's party. Your goal is to generate a list of the names of the 10% of Senators who vote least often with their party.
3. Call your code in the Senate Party Loyalty HTML file and use console.log to check your answers on different test data.

Task 2: Build Most Loyal table by calculating and displaying to 10% Senators who didn't vote with party.

Again, follow the steps only for Senators who voted most often with their party.

Epic 4: Build House Attendance and Party Loyalty Pages

Task 1: Build House At A Glance table

Follow the steps you executed for the Senate data only apply it to the larger data set of the House members.

TIP: there are 100 members of the Senate and 435 members of the House of Representatives.

Task 2: Build House Attendance page

Task 3: Build House Party Loyalty page

Epic 5: Implement Checkbox filters into your website

Task 1: Build checkbox that triggers a function to filter for Senate Democrats only

TIP: Do not use "onClick" property on the HTML tags, use `.addEventListener()` on the JS file instead.

Task 2: Build check-boxes that trigger functions to filter for Senate Republicans and Independants

Add Filter by Party Checkboxes

Add three checkboxes to filter the data table by party (Democrat, Republican, and Independent). Look at checkbox examples in *Resources*.

1. Open the senate HTML file.
2. Create checkbox input section with three checkboxes and the appropriate labels.
3. Write the javascript code that will filter the data by party:
 - a. Get checked box values and put them into an array.
 - b. Use that array to filter the list of members to pass to your function to create the table.
 - c. Call this code whenever a checkbox is changed, i.e., use an *onchanged* event listener.
4. Open the page in your browser. If the checkboxes filter the table correctly, go to the next step. If not, open the Developer Console and look for error messages.

Task 3: Build functionality to allow users to filter by multiple checkboxes

Task 4: Repeat for House data

TGIF Sprint 3

Epic 1: Implement dropdown filters into your website and integrate

Task 1: Create dropdown selector to filter Senate members by State

Add a dropdown menu to filter the data table by state. The filter default state should show all the data. Look at dropdown filter examples in *Resources*.

1. Open the senate HTML file.
2. Create dropdown input section with the appropriate labels and options.
3. Write the javascript code that will filter the data by state:
 - a. Get the selected value.
 - b. Use the same approach to filtering that you did with the party checkboxes.
4. Open the page in your browser. If the dropdown filters the table correctly, go to the next step. If not, open the Developer Console and look for error messages.

Some random HTML and JavaScript coding tips:

- **TIP:** You will have two distinct filters on this page. You definitely don't want the checkboxes and drop down to have the same name. Use content-based names like party-filter and state-filter.
- **TIP:** Use separate functions to handle changes on party and state. All each function should do is update the relevant variable, e.g., what parties and state(s) are selected. Then each can call the same createTable() function to draw the updated table.
- **TIP:** When setting up the state dropdown menu, make sure that one of the options is "all states".

Task 2: Define every different scenario of filtering possible

Think about every different ways that the filters can be used in combination to render different results. If needed, your mentors will discuss this task with you and then lead a spike to share the different scenarios.

Task 3: Code integrated filtering by party and state

Now that you've defined your scenarios for using multiple filters, code it to achieve your expected results.

Task 4: Repeat for House members

Do the same process to create filters (checkboxes to filter by party and a dropdown to filter by state) for the House data by repeating the steps from task 1. This will be a much longer page than the Senate page, but you should have very little new work to do.

Epic 2: Use AJAX to call ProPublica data

Task 1: Fetch JSON data from ProPublica server

You have been asked to upgrade the code you have to use live data with AJAX.

The first important change you'll make will be to replace the code that gets the test data from local files with AJAX calls using **fetch()** to get real data from the ProPublica site instead. The second change is to follow best practices and clean up the JavaScript and HTML code.

TIP: it's a good practice to move most of the JavaScript from the HTML file over into its own .js file.

AJAX and Browser Security

AJAX gives code on a web page the ability to send information to another web server. This raises security concerns. A malicious or hacked web page might contain code that sends your personal information wherever it pleases. To be safe, therefore, by default browsers only allow a web page to send AJAX if:

- It is running on a web server, under some domain, e.g., superior.widgets.com
- The AJAX is sending information to a URL in the same domain, or
- The web service on the other domain implements cross-origin resource sharing (CORS)

DRY

You always want to avoid duplicating code. Duplicated code means that when you find a bug or see a way to improve something, you have to make the same change in multiple places. In software development, this is called being DRY (Don't Repeat Yourself).

In JavaScript, one technique to avoid duplication is to refactor the bits that make code different into a data object. Then you write one body of code, and pass it different data objects for different situations.

For example, suppose you have several pages where some part of each page is to be filled in with data from an AJAX call, and then stored/rendered with Vue. A DRY solution would define a function, call it **renderRemoteData()**, that is called on each page with code like this:

```
<script type="text/javascript">
renderRemoteData({ url: "https:remote-temperature.com",
  target: "display-table"
});
</script>
```

Here, the target is the ID for elements on the page where the Vue data appears, and the URL gives the function the information it needs to make the AJAX call. Just one code file to define **renderRemoteData()** and its subfunctions is needed.

Task 2: Refactor code so that it can handle asynchronous behavior

Now, refactor your code so the functions are able to use the data from the fetch call. As you refactor, think about why it matters to use asynchronous fetching instead of local static data.

Task 3: Add loaders

Add a loader to your website while the data is being fetched, so users know they have to wait for the page to load.

Epic 3: Refactor to merge JS files

[View Tasks & Resources](#)

Task 1: Refactor your code down to 1-2 JS files

Now you should have about 4 JS files. These files contain functions that are quite similar. An excellent practice is to refactor your code so that these functions, that operate in the exact same way but with different data, all exist within the same JS file(s). Review your files and create a plan for how to merge your code so that there is only ONE or TWO JS files. When you have come up with a strategy for your refactoring, check your plan with a mentor and then execute.

Task 2: Submit your code to GitHub

You should already have a dedicated repository for the TGIF project, but if not please create one now.

(Optional but recommended) Epic 4: Incorporate VUE into your project

[View Tasks & Resources](#)

HTML Templating

Constructing HTML by concatenating strings is simple at first, but rapidly gets very hard to maintain as the HTML and CSS gets more complex. Therefore, virtually all web programming frameworks include templating libraries that let you define HTML patterns, in HTML files, and then combine those patterns with data to create the actual HTML on the final pages.

Vue is a JavaScript framework for building user interfaces. It lets you write the HTML you want to use, and add attributes and `{{ value }}` forms to insert data and repeat forms.

For example, suppose we want to display a simple list of links to employees. Here's how you could write that in your HTML file using Vue:

```
<ul>
  <li v-for="employee in employees">
    <a v-bind:href="employee.url">{{ employee.name }}</a>
  </li>
</ul>
```

This is nice and readable, and much easier to edit than JavaScript concatenating strings. Vue adds a few special notations for your HTML. The ones shown here are:

- `{{ expression }}` to insert the value of a JavaScript expression into HTML text
- `v-bind:attribute="expression"` to insert the value of a JavaScript expression into a tag attribute
- `<tag v-for="var in expression">...</tag>` to repeat some HTML element for every item in the list returned by a JavaScript expression. `var` will be set to each item in turn, just like a JavaScript for loop.

For Vue annotations to work requires a few additional steps:

- Put the HTML that you want Vue to scan and change inside an HTML element with some distinct ID. The ID `app` is often used but not required.
- Load the Vue JavaScript library.
- Add JavaScript code to create a Vue object that connects the HTML with the data.

- Set the data.

Here's an outline of how that might work here:

```
<!DOCTYPE HTML>
<html>
<head>
...
</head>
<body id="app">
...
  <ul>
    <li v-for="employee in employees">
      <a v-bind:href="employee.url">{{ employee.name }}</a>
    </li>
  </ul>
...
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script>
var app = new Vue({
  el: '#app',
  data: {
    employees: [
    ]
  }
});

app.employees = [
  { name: "John Smith", url: "http://company/jsmith.html" },
  { name: "Mary Jones", url: "http://company/mjones.html" }
];

</script>
</body>
</html>
```

The Vue object is a place to store data to be displayed. Only when you want to update the display should you update the Vue data. Internally Vue is smart about checking to see what needs to be changed on the page. More complex calculations should be done in JavaScript when preparing the data to pass to the HTML.

CDN (Content Distribution Networks)

Strictly speaking, a Content Distribution Network (CDN) is a network of web servers that contain duplicated content, placed at various locations around the world to reduce congestion and long-distance network traffic. The term has informally been extended to refer to web sites that host commonly needed web resources, such as JavaScript libraries.

TGIF External Resources

ProPublica API

[The ProPublica Congress API home page](#)

Walks you through getting your API key and provides Terms of Use for accessing the data, along with FAQs.

JSON

[How to represent data in JavaScript](#)

- A brief introduction to numbers, strings, and JavaScript objects

[How to convert JSON data to a string](#)

- Useful for showing JSON data during development and debugging

[JSON Basics: What You Need to Know](#)

- An introduction to the JSON data format, including how to create and read JSON strings in JavaScript. (Don't worry about the sections on XML and PHP.)

JavaScript

[The Modern JavaScript Tutorial](#)

- For JavaScript fundamentals: scripts, objects, events, methods
- JavaScript in an HTML page: how to load JavaScript from an external .js file, how to write JavaScript directly into an HTML file
- JavaScript fundamentals: statements, comments, variables, data types, arrays, operators
- JavaScript to get an HTML element with a specific ID and replace its contents
- For JavaScript fundamentals: functions, scope, methods, objects, built-in objects (including DOM)
- Storing Data, Arrays are Objects for help with accessing specific information within an array
- JavaScript fundamentals: decision-making (e.g. comparison, if-else), loops
- NOTE: a better way to write loops is with `forEach()` and `map()`. See [here](#).
- **TIP:** Also see [JavaScript For Loops](#) for a very basic, but interactive introduction to for loops in JavaScript
- JavaScript to build a string of HTML by looping through an array, and then displaying the final result

CSS Styling with Bootstrap

Bootstrap 3 Tutorial

- Comprehensive tutorial on the Bootstrap 3 front-end framework. Allows you to play with code samples directly in the tutorial website. Of especial note:
- [Bootstrap Get Started](#): Goes over how to set up an HTML page to use Bootstrap
- [Bootstrap Tables](#): An introduction to the basic classes Bootstrap provides for styling tables
- [Bootstrap CSS Tables Reference](#): A more comprehensive listing of Bootstrap-provided classes for styling tables
- [Bootstrap Accordion/Collapse](#): Goes over how to hide/show content with Bootstrap

JSFiddle

[JSFiddle](#) provides a custom environment to test your JavaScript, HTML, and CSS code right inside your browser. The many features provided by the JSFiddle interface makes it relevant to almost all Web application developers. The main section of the site is divided into four areas:

1. CSS: Enter CSS to be applied to the HTML used in your tests.
2. HTML: This area allows you to enter HTML to be used in your tests.
3. JavaScript: JavaScript source is entered in this area.
4. Output: The results of executing the CSS, HTML, and JavaScript.

To learn more go [here](#).

Debugging

The Modern JavaScript

- Search 'Debugging'

Displaying JavaScript data as HTML

Once you have data in JSON form, you can use JavaScript to construct HTML that presents that data in a readable form to users. Typically, only a dozen lines of code, more or less, will be needed to do this, but those lines will involve many of the basic programming concepts in JavaScript. Of especial relevance are

- [arrays and JSON objects](#)
- [anonymous functions](#)
- [looping over array data](#) using `map()` with anonymous functions
- [using loops to construct HTML](#)

JavaScript Display Possibilities - http://www.w3schools.com/js/js_output.asp