

Geog 461W - Dr. Alan M. MacEachren
Pre-class Bootcamp

Tutorials originally by Yanan Xin
Modified by Morteza Karimzadeh

Hello GEOG461 Students!

In order to make dynamic web maps, you need to know some basics of HTML, CSS and JavaScript. No programming experience before? No worries. All the things you need to get through this class are included in this document. Read this document carefully until you reach “Putting it all together”. Then start doing things step-by-step as instructed.

I. Intro to HTML

Please read the following HTML tutorial sessions from [W3schools.com](https://www.w3schools.com): Don't miss the fun parts of **Try it yourself >>**

- HTML Introduction **TO** HTML Attributes (5 sessions here)
- HTML Comments
- HTML CSS
- HTML Blocks
- HTML Layout
- HTML JavaScript
- HTML Charset
- [HTML Event Attributes](#)

Other sessions and quizzes are optional. You are welcome to learn as much as you want!

II. Intro to JavaScript

Take a brief look at [this page](#) for a flyby introduction to JavaScript. A lot of the following notes are summarized from the book *Coding with JavaScript for Dummies*, which I think are most related to the contents covered in this class. Copyright belongs to original authors. if you have extra time and energy, and you're new to JavaScript (JS), you're highly encouraged to read the original book, especially Part I. Also please read JS Debugging here on [W3Schools.com](https://www.w3schools.com). You can check out other sections during your spare time. Before we proceed, read the following sections on [W3Schools JavaScript course](#) (From JS Home to JS Functions):

JS Tutorial
JS HOME
JS Introduction
JS Where To
JS Output
JS Syntax
JS Statements
JS Comments
JS Variables
JS Operators
JS Arithmetic
JS Assignment
JS Data Types
JS Functions

Tools needed

Download and install accordingly

- **Web browser**

Google chrome is recommended since all the instructions below are based on Chrome. Also it has an excellent Developer Tool to work with Javascript. However, you can also use Mozilla Firefox and get the [Firebug](#) add on for development and debugging purposes.

To open the Developer Tool, find the Chrome menu at the right-upper corner of your browser, ☰, More Tools -> Developer Tools. The interface looks like [Figure 2.1](#):

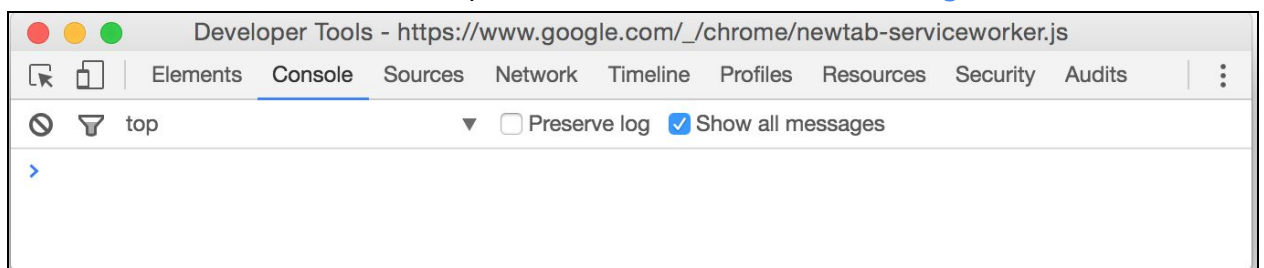


Figure 2.1. Developer Tools of Chrome with the display of console panel

The developer tool can appear in three ways (see [Figure 2.2](#)), on a separate window, bottom of your browser window or right side of your browser window. Click on the three dots symbol ⋮ at the right-upper corner to choose your preferred appearance.

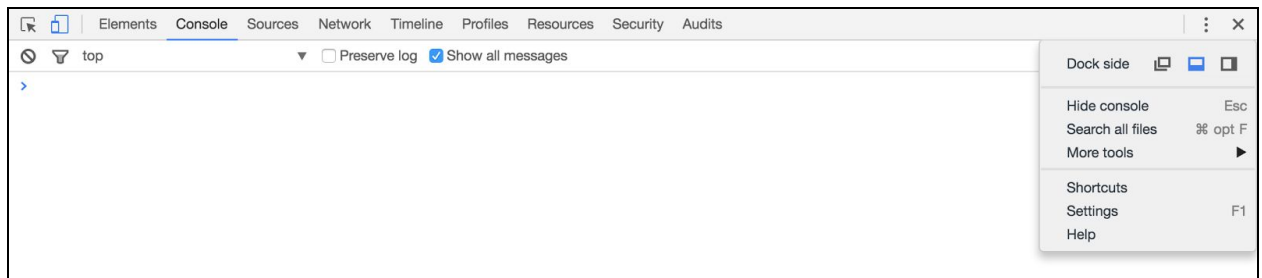


Figure 2.2 Placements of Chrome Developer Tools

If your web page doesn't show up as you expected, you can find error messages in the **Console** tab of this developer tool. Also, JavaScript programmers like to use `console.log("messages you want to show");` to debug their code, which we will cover in our lab tutorials.

- **Code Editor**

[Sublime Text](#) is recommended. Not only because it's very popular among JavaScript programmers, but also it provides a lot of powerful plugins with a simple user interface. You may also use Notepad++ or an IDE of your choice, such as NetBeans or Eclipse.

*It doesn't matter if you use Sublime Text 2 or Sublime Text 3, they both can be downloaded for evaluation purpose for free. The developers expect users who continue to use the software to buy the license (\$70). Specifically, the web site explains future purchase expectations after an evaluation as follows: "Sublime Text may be downloaded and evaluated for free, however a license must be purchased for continued use. There is no enforced time limit for the evaluation."

Run JS in Browser Window

Here are three ways of putting JS in an HTML document:

- ✓ Put it directly in an HTML event attribute (example 01)
- ✓ Put it between an opening and closing script tag (example 02)
- ✓ Put it in a separate document and reference it in your HTML document (example 03)

Example 01 As An HTML Event Attribute

```
<button id="bigButton" onclick="alert('Hello World!');">Click  
Here</button>
```

In this case, when a user clicks on the button created by this HTML element,

a popup will appear with the words "Hello World!". The string (marked by double quotes), `alert('Hello World!');` is actually a javascript function. The JavaScript will be executed after the button is clicked.

Example 02 Within `<script>` and `</script>` Tags

Insert the code within script tags, like line 10 to line 15 below in [Figure 2.3](#):

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Hello, HTML!</title>
5  </head>
6  <body>
7      <h1>Let's Count to 10 with JavaScript!</h1>
8      <p id="theCount"></p>
9      <script>
10     var count = 0;
11     while (count < 10) {
12         count++;
13         document.getElementById("theCount").innerHTML +=
14             count + "<br>";
15     }
16     </script>
17 </body>
18 </html>

```

Figure 2.3 Put JS codes inside script tags

Example 03 As External JS File

```
<script src="myScript.js"></script>
```

In this example, rather than having the Javascript within the HTML, it can be in a separate file. Specifically, you can extract line 10 to line 15 in example 02 and make it a separate file (named `myScript.js`), and then put this js file in the same folder as your html document. That file can be called and executed using just one line of code, as shown above. This one works exactly as example 02, but by separating the js file with html, it gives more flexibility of reusing and modifying the js file (This is how we use leaflet js library throughout this class)

The Placement of JavaScript in a HTML Document

Sometimes you'll see the script tags are placed within the head and sometimes in the body of the HTML document. It actually is not the exact place that matters but rather the

order. The browser will read and execute the webpage from the top down. So if you want to wait to execute your code until all contents are loaded, you can do something like example 01, `onclick = <your javascript>` or like example 02, simply put your code at the very bottom of your page.

Key Rules and Concepts

Now let's take in a few key rules of writing javascripts which can be easily missed by beginners and cause most of errors. The key concepts will help you see the forest for the trees.

A few rules of JavaScript (JS):

- JS is **case-sensitive**. So Ben and ben would indicate a different "person" (we call it an object) in the JS world.
- Watch out for **reserved words**. Some words are reserved in JS world and you cannot use them to name your variables, like *function*, *var*, *while* and so on. These reserved words usually are explicitly colored in sublime text to make their reserved status easy to recognize.
- A statement is like sentence of a paragraph. It's the fundamental building block of JS programs. Statements are ended with **semicolons** in JS.

Key concepts:

1) JS Connector to HTML-- DOM.

DOM (Document Object Model): The Document Object Model is **the interface for JavaScript to talk to and work with HTML documents** inside of browser windows. Think of JavaScript as a set of commands to control all the components in the webpage, thus commands for how to get, change, add, or delete any HTML elements. In order to send out a command to the right object, JavaScript has to know how to identify and call the target component. This is where the DOM (a World Wide Web Consortium standard for accessing documents) comes in.

The DOM can be visualized as an inverted tree, see example below in [Figure 2.4](#) (sample document code) and [Figure 2.5](#) (a diagram illustrating how the HTML document's components are organized):

```
1  <html>
2  <head>
3      <title>Bob's Appliances</title>
4  </head>
5  <body>
6      <header>
7          
9      </header>
10     <div>
11         <h1>Welcome to Bob's</h1>
12         <p>The home of quality appliances</p>
13     </div>
14 </body>
15 </html>
```

Figure 2.4 A simple html document

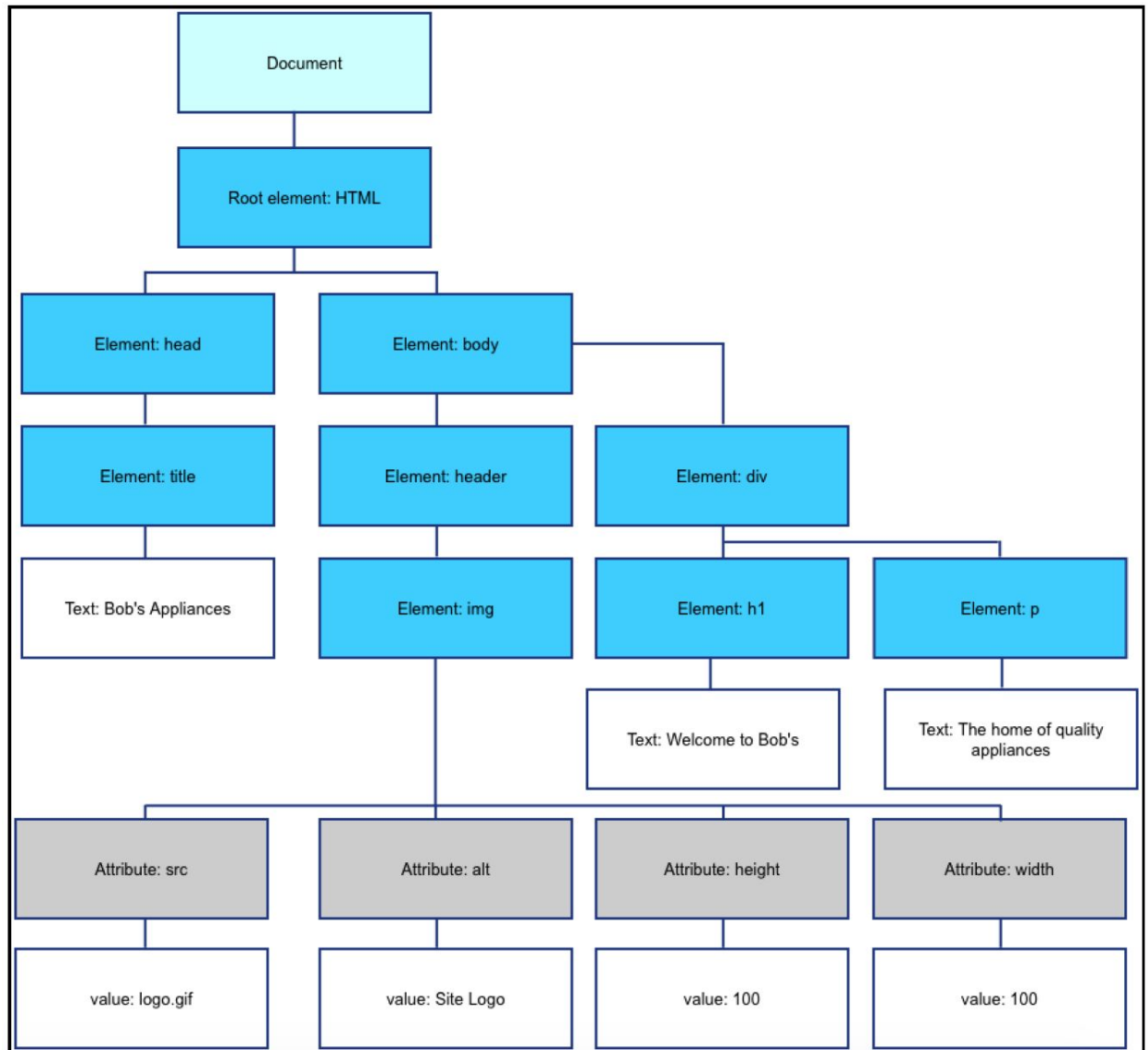


Figure 2.5 Corresponding DOM tree

2) Connector to JS Plugins -- API.

API: Think of the standard for plugs and sockets. See [Figure 2.6](#) below. You have to know how many holes exist in each socket and how they are organized in order to design or choose plugs. Just like the standard for plugs and sockets, APIs, or Application Programming Interfaces, are sets of software routines and standards that guide programmers to interact with each other's application. When a web browser allows a JavaScript program to interact with it, it does so by using APIs.

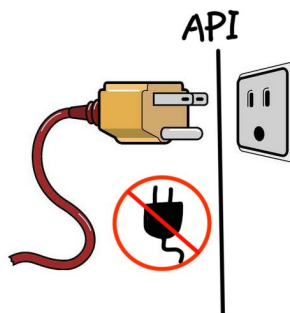


Figure 2.6 A Metaphor of API

Typically, an API is written up as a specification that tells what properties and methods are available to programmers, what arguments can be passed to the methods, and what sorts of values are returned by the properties. Let's take Leaflet's API for example. See [Figure 2.7](#).

Leaflet API:

[Map creation](#)

Factory	Description
<code>L.map(<HTMLElement String> id, <Map options> options?)</code>	Instantiates a map object given a div element (or its id) and optionally an object literal with map options described below.

Figure 2.7 Leaflet API for L.map function

When using Leaflet, you create a map by using the [L.map\(\)](#) function in JavaScript. What about the stuff within the parenthesis? These are parameters of this function. They are separated by commas. `<HTMLElement | String>` indicates the type of the first parameter. And `id` is the descriptive name of the parameter. The question mark `?` after `options` means this `options` parameter is optional. See [Figure 2.8](#) for an annotated version:

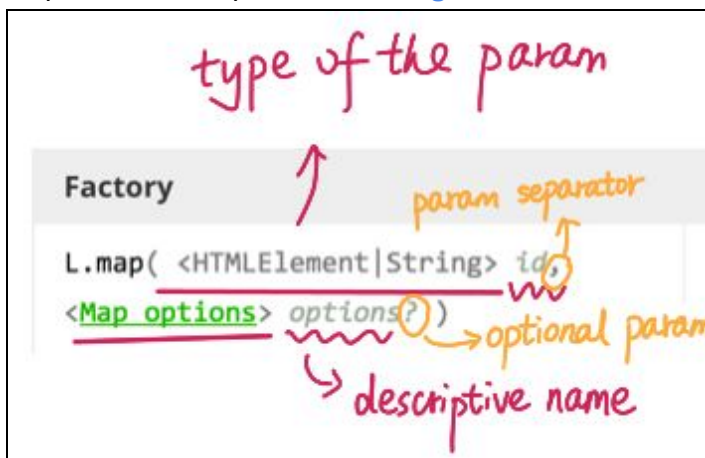


Figure 2.8 Annotated version of L.map function API.

It seems a bit complex. While let's have a look on a sample code below:

Sample code 01:

```
// initialize the map on the "map" div with a given center and
zoom
var map = L.map('map', {
  center: [51.505, -0.09],
  zoom: 13
});
```

In sample code 01, 'map' is the id of the HTML DIV element that we want to hold the map. Center and zoom are all map options. However, when you call the `L.map()` function and pass arguments to it, you don't have to specify the options parameter, like the sample code 02 showing below. It only gives a 'mapid' to the `L.map` function.

Sample code 02:

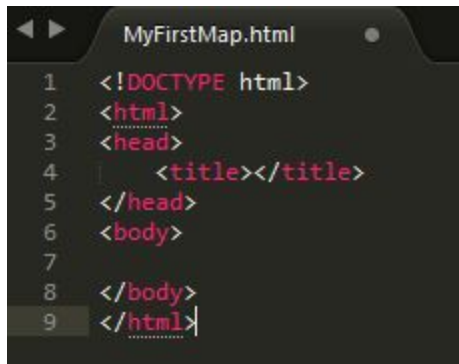
```
<div id="mapid"></div>
<script>
  var mymap = L.map('mapid').setView([51.505, -0.09], 13);
</script>
```

Putting it All Together

Okay! Let's create a simple web map, step by step. I am going to loosely base this section on [Leaflet tutorial](#). So that would be another source to get help if you get stuck. The goal is to progress as much as you can, and if you fail, explain why you think you couldn't complete the assignment. That should earn you full grade for this bootcamp.

First, create a plain HTML file. You know that an HTML file is nothing but a pure textual file, but with a file extension like .html and formatted according to HTML syntax. To create an HTML file, Open Sublime Text and click on New File under the File menu. You will see a blank file appearing. From File menu, click on Save As, and name the file as YourNameFirstMap.html (mine is KarimzadehFirstMap.html) and save it to a location on your own drive where you have administrative rights. If you open the HTML file by double clicking on it, you will see a blank page opening in your browser. By the time we are done with this bootcamp, we want to see a simple web map on this page instead.

Now, let's add the HTML elements. While you can type them in each, let's take advantage of Sublime to take care of it for us. From the Tools menu, click on Snippets, and click on (or type in) HTML and press enter. Your document should look like this by now:

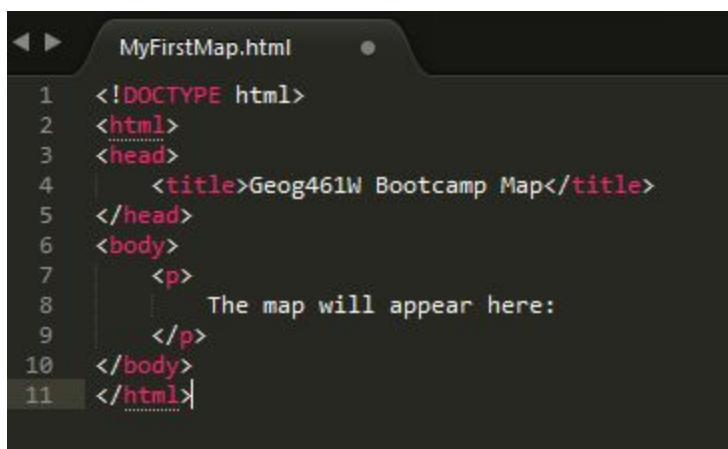


```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title></title>
5 </head>
6 <body>
7
8 </body>
9 </html>
```

The first line is a “declaration”. This tells the browser that it is dealing with an HTML file, that the document type is HTML. The rest are standard HTML elements that you got to know earlier on W3Schools. You can always look up the [W3School HTML Reference](http://www.w3schools.com/html/html_reference.asp) for an exhaustive list of elements if you need a refresher on what a particular element is. You will find yourself looking up resources frequently when developing web applications (like other software applications).

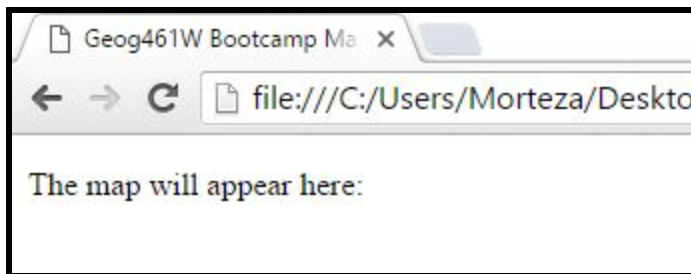
Save your file and refresh the page in the browser. Although you have added several elements, the document is still blank. Why?

Did you notice the title of your page in the browser is the physical address of the HTML file on disk? Let's fix that by putting in a title, and also adding a paragraph element (lines 4, and 7 through 9):



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Geog461W Bootcamp Map</title>
5 </head>
6 <body>
7   <p>
8     The map will appear here:
9   </p>
10 </body>
11 </html>
```

Now, if you refresh the browser, the title should change and you should see your paragraph:



Now, let's get to the actual business: adding a map. First, let's add a DIV element that would hold the map.

```
MyFirstMap.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Geog461W Bootcamp Map</title>
5 </head>
6 <body>
7   <p>
8     The map will appear here:
9   </p>
10  <div id="mapId" style="height: 400px; width:600px"></div>
11 </body>
12 </html>
```

Note that we have also included a “style” attribute, and within it, we have specified a height of 400 pixels and width of 600 pixels for our map container. Later on, you will learn to define style in separate CSS files, but for now, this attribute does what we want to do. It's just not according to best practices, since we are always seeking “[separation of concerns](#)”: Code that deals with content (HTML), should be separate from styling (CSS) and separate from code that manipulates the page or data (JavaScript). For now, we are doing everything in a single file, the HTML file.

- Optional: Refresh your web page. You cannot see the DIV, can you? Because it is empty. You can make sure it exists if you enable Chrome's developer tools (shortcut ctrl+shift+I), click on the “Elements” tab, and mouse over the elements that make your page. Once you hover your mouse over the DIV element, you will see a 600X400px area highlighted on your page. This is the DIV where the map will be put.

Now, we need to include a code like sample 01 or 02 to include the map on the page. As discussed, that's a Leaflet function call. However, we need to first include the Leaflet "library" in our HTML page by "referencing" it. Then, JavaScript will execute the function that is stored in the Leaflet Library when we "call" or "invoke" that function. In order to reference the Leaflet library, add the following line to the "head" section of the HTML page:

```
<script src="https://npmcdn.com/leaflet@1.0.0-rc.3/dist/leaflet.js"></script>
```

Got it? The library is nothing but a (textual) JavaScript file named leaflet.js that is stored on <https://npmcdn.com/leaflet@1.0.0-rc.3/dist/> (you can see a version that is NOT minified -the blank spaces not removed- [here](#)). That is where the Leaflet functions are located, including the L.map() function. You might as well download the library to your own computer and reference a local copy instead of the one that is hosted on a remote server. Each way has its own advantage. Can you think of any?

Similarly, leaflet stores the styles it uses in a CSS file. Reference it in the head section of your file like this:

```
<link rel="stylesheet" href="https://npmcdn.com/leaflet@1.0.0-rc.3/dist/leaflet.css" />
```

You can also click [here](#) to see the actual CSS file. You will learn more about these files throughout the semester! W3Schools has an entire [course](#) dedicated to it. If you know HTML, JavaScript and CSS well, you can call yourself a Web Developer. Confidently.

Okay. Let's add our Script to the file. Which Script you say? The JavaScript function that populates the map within our mapId DIV element! Again, if we stick with best practices and separation of concerns, we should create a separate JavaScript (.js) file, write our JavaScript code in there, and reference it in our HTML file just the way we referenced Leaflet's JavaScript. However, in this bootcamp, we are including everything in a single HTML file. So, let's use inline JavaScript. Add an opening and closing <script> tag in the <body> of your HTML file, like figure 2.3. Your File should look like this now:

```

MyFirstMap.html
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <script src="https://npmcdn.com/leaflet@1.0.0-rc.3/dist/leaflet.js"></script>
5      <link rel="stylesheet" href="https://npmcdn.com/leaflet@1.0.0-rc.3/dist/leaflet.css" />
6      <title>Geog461W Bootcamp Map</title>
7  </head>
8  <body>
9      <p>
10         The map will appear here:
11     </p>
12     <div id="mapId" style="height: 400px; width:600px"></div>
13     <script>
14
15
16     </script>
17 </body>
18 </html>

```

Now, add these three JavaScript lines between the <script> opening and closing tags:

```

//create a map variable that points out to the map hosted in the mapId DIV
var map = L.map('mapId')
//create a variable named osm that holds OpenStreetsMap tile layer
var osm = new L.TileLayer('http://tile.osm.org/{z}/{x}/{y}.png');
//Add the OpenStreetsMap tile layer (osm) to the map and zoom the view on Walker Building with a 17 zoom level
map.addLayer(osm).setView([ 40.793271, -77.867023], 17);

```

Note how I have used two forward slashes to specify a “comment” line. Lines that are commented are purely informational and the compiler will not interpret them as JavaScript code.

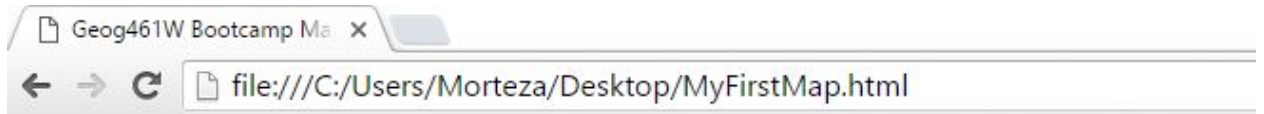
The first line creates a JavaScript variable named “map” that points out to the map that is hosted in the mapId DIV (L.Map() creates that map in the DIV that gets passed to it as parameter). The second line creates a variable named “osm” that holds OpenStreetsMap tile layer. Finally, the third line takes the OpenStreetsMap tile layer (by its variable name, osm) and adds it to the map (by its variable name, map) and zoom the view on Walker Building with a 17 zoom level. Your file should look like this:


```

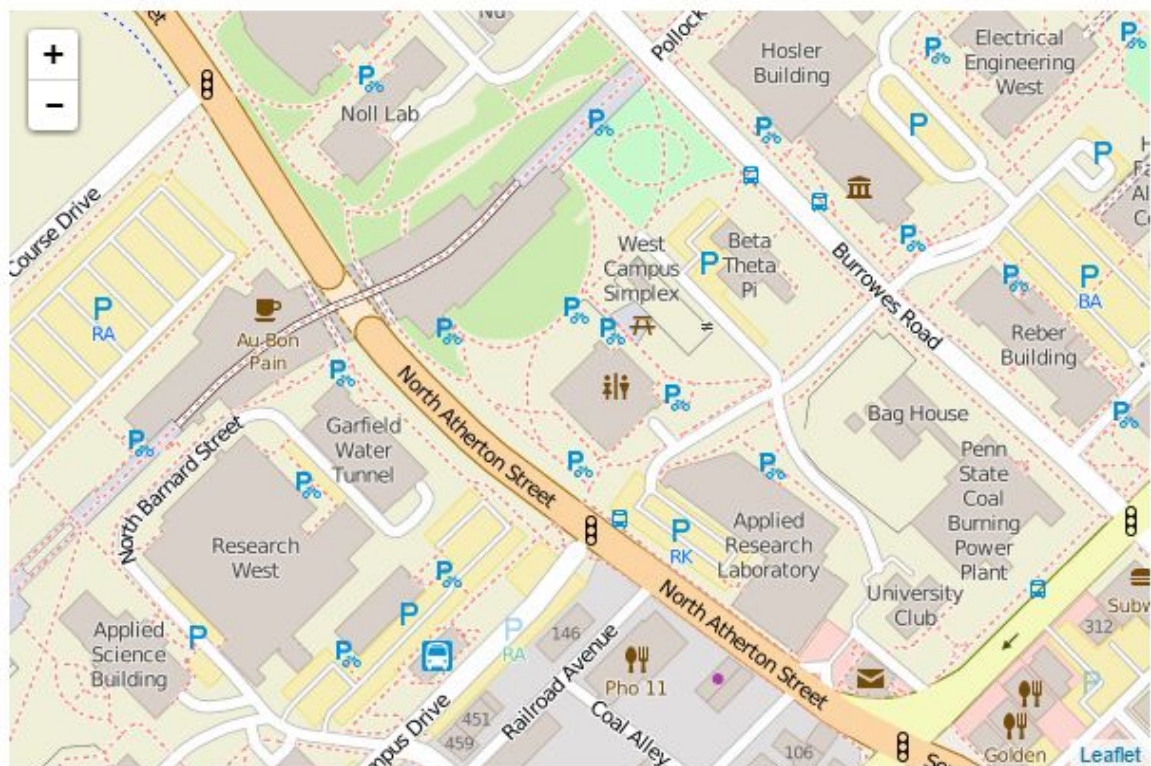
MyFirstMap.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="https://npmcdn.com/leaflet@1.0.0-rc.3/dist/leaflet.js"></script>
5   <link rel="stylesheet" href="https://npmcdn.com/leaflet@1.0.0-rc.3/dist/leaflet.css" />
6   <title>Geog461W Bootcamp Map</title>
7 </head>
8 <body>
9   <p>
10     The map will appear here:
11   </p>
12   <div id="mapId" style="height: 400px; width:600px"></div>
13   <script>
14
15     //create a map variable that points out to the map hosted in the mapId DIV
16     var map = L.map('mapId')
17     //create a variable named osm that holds OpenStreetsMap tile layer
18     var osm = new L.TileLayer('http://tile.osm.org/{z}/{x}/{y}.png');
19     //Add the OpenStreetsMap tile layer (osm) to the map and zoom the view on Walker Building with a 17 zoom level
20     map.addLayer(osm).setView([ 40.793271, -77.867023], 17);
21
22   </script>
23 </body>
24 </html>

```

Save your file, and refresh the browser. This is what you should see:



The map will appear here:



Congratulations! You have created your first web map. We will need to add features to this basemap (tile layer), but let's do that in next labs. You are welcome to do so if you are inclined. Remember, Leaflet tutorials are out there!

Now, save your file and submit as part of your assignment. If you were not able to follow all the way through because you need more help, don't worry. Just submit the HTML file that is completed as much as you were able to, and explain why you were not able to finish the assignment according to instructions. Specifically, tell us where you got stuck, or where the instructions are not clear. An HTML file (complete or not) along with this explanation should earn you full grade for this lab!