Make a Web Map: Using LeafletJS

(Based on an assignment by Mike Foster)

So, you know some basics about webpages and their elements. Now you want to make a web map. Great! This session is the first part of a series that introduce Leaflet, a Javascript library used to create interactive, web-based, mobile-friendly maps. Well, we haven't learned Javascript yet… that is okay. We will use Leaflet to introduce the concepts, then cover Javascript in a later session. With Leaflet, you can create a simple map in as little as three lines of code, or you can build complex, dynamic, and complex maps that contain hundreds of lines. It is up to you! This tutorial assumes you have worked through basics of a website, and have a working knowledge of HTML and CSS, and a very basic working knowledge of Javascript. The tutorial will build your knowledge of Javascript, and specifically the LeafletJS library, by building a web map from the ground up.

What is Leaflet?

Leaflet is an open-source JavaScript library for interactive web maps. It's lightweight, simple, and flexible, and is probably the most popular open-source mapping library at the moment. Leaflet is developed by Vladimir Agafonkin (currently of MapBox) and other contributors.

What Leaflet does: "Slippy" maps with tiled base layers, panning and zooming, and feature layers that you supply. It handles various basic tasks like converting data to map layers and mouse interactions, and it's easy to extend with plugins. It will also work well across most types of devices.

How this tutorial works: It's structured around examples that progressively build upon one another, starting from scratch. We will start with an empty webpage, then progressively add components to make a Leaflet map. It assumes a basic knowledge of HTML and JavaScript, or at the very least assumes the will to tinker with the code to better understand what it does. It won't explain every little object or array, but will contain plenty of links. Many code blocks will show only a snippet of code, highlighting the changes over previous examples. Click the "View this example on its own" link underneath a map to see complete code. For thorough documentation, see the Leaflet site.

A couple quick tips.

- Use a text editor for writing your code, such as Sublime Text or Notepad++.
- Keep all of your components in one folder. This will make locating specific files in your code much easier.
- To put your map on the web, upload the folder with your website and map components (that we will create) to your web server. A note, however, most of this tutorial will work locally, but at the end we are going to add an external file. You can use WinSCP to copy your data.

What we are going to create

Our goal will be to start from scratch and build the following map, with a tile base layer, some mapped data, and some basic interactivity. Our end result will be the following map, showing an area around campus.

1. Create a Webpage and Simple Map

a. Work within our working folder

We want to work within a contained folder that serves as a server directory. Use the provided materials folder. This folder will store all of the files associated with our specific web page and web map. Save subsequent files here. **This is the folder in which we are running our Python server.** This is referred to as our web folder.

b. Setup a web page for our map

Open up your text editor. Once in your text editor, we are going to set up an empty **index.html** template for our web page that will contain our web map and web map elements. The components will be the same as always, note the head, title, and body.

Enter the following code into your blank HTML page.

```
<!DOCTYPE html>
<html>
<head>
    <title>Leaflet Map</title>
</head>
<body>
    <!-- Our web map and content will go here -->
</body>
</html>
```

From here, we will do the following four things to add a map to our page:

- Reference the Leaflet CSS and JavaScript files.
- Add a div element to our page that will hold the map.
- Create a map object in Javascript that will interact with the map div element
- Add the tiled OpenStreetMap basemap to our map object using tileLayer

c. Reference the Leaflet CSS and JavaScript files

We need to load Leaflet into our web page before we can start using the library. There are two options for doing this, we can download the library files from the Leaflet download site, or we can use the hosted version. We are not planning on changing the JavaScript or the CSS, so it is easiest to use the hosted libraries. Reference these in your HTML by adding the following lines of code.

Within the head section, after title, copy and paste the following. This adds the Leaflet CSS file to our web page and includes Leaflet styles.

```
<!-- External Stylesheets -->
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.5.1/dist/leaflet.css" />
```

Link to the JavaScript library at the bottom of the body section of our site, putting it at the bottom allows our page to load faster. Copy and paste the following. This adds the Leaflet JS file to our web page and is the Leaflet Javascript library.

```
<!-- Add the Leaflet JavaScript library -->
<script src="https://unpkg.com/leaflet@1.5.1/dist/leaflet.js"></script>
```

We can now begin working with the Leaflet library.

d. Add a map div

Add a div to the body that will hold the map. This is just like any other div element we might use. We will set the style right in the div using the style attribute, and not the CSS file, otherwise all map divs we create will have the same styling.

```
<div id="map" style="width: 705px; height: 375px"></div>
```

e. Use Javascript to create the map object

Now we can start coding the map using JavaScript. The Leaflet library is referenced by using L. followed by the class. The first step is to create the map object using the map class. Set the variable map to be our Leaflet map object. More reading on L.map can be found in the extensive Leaflet documentation. Set the center of the map to be at Auraria (39.743218, -105.004568) and zoom level to 14. Enter the following in our document at the end of the body section.

```
<script>
    // Create variable to hold map element, give initial settings to map
    var map = L.map('map',{ center: [39.743218, -105.004568], zoom: 14});
</script>
```

Note the script tags, this is where we will will put all of our JavaScript for the map.

f. Add a tiled basemap with tileLayer

The last step in getting a basic map running is add a layer. We are going to use what is called a tile layer, which is a fundamental technology behind many web maps.

There are many tile layers you can add to your maps. The one we are going to use today is from OpenStreetMap. To add a tile layer to the map, we use the L.tileLayer class. Place the following code within your script tags.

```
// Add OpenStreetMap tile layer to map element
```

```
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', { attribution: '© OpenStreetMap' }).addTo(map);
```

Note the attribution. Here we can provide reference for the source of the base map, and any other attribution for map elements we want to provide. It will appear in the lower right corner of our map by default, but you can change this. Read more about attribution here.

Our Basic Map

Save your HTML document and copy it over to your web server folder using WinSCP. Then, go to your server's web page. You will see the map we just created!

Additional Tile Layers

There are a number of resources that have tile layers you can use with Leaflet JS. An excellent resource for examining and previewing tile layers is called Leaflet Provider. Scan through available tile layers and preview them, and try replacing the L.tileLayer with one of the other tile layers in your code.

Loading a WMS

You can also add Web Map Services to your Leaflet maps using L.tileLayer.wms. The example provided in the documentation shows adding a weather layer to your map.

2. Add Individual Data to our Map

To introduce adding data, we will learn how to add markers, polylines, and polygons to our map. There are multiple methods for adding data, including methods in which you can large datasets. Before getting ahead of ourselves though, this section will show how you can simple points, polylines, and polygons to your map.

a. Adding Points (aka Markers)

To add a point to your map, we use the L.marker class. To add a point, we specify a latitude and longitude, then add the marker to our map. Enter the following line of code in the script block in the body section of the document, following the tile layer.

```
// Create point feature for Auraria Library
var myDataPoint = L.marker([39.743119, -105.002959]).addTo(map);
```

b. Adding Polygons

Adding polygons is very similar, we use the L.polygon class. Specify a latitude and longitude for each node, then add to our map. Set the style just the same. Enter the following in our script.

```
// Create area feature for campus, style and add to map
var myArea = L.polygon([[39.740652, -105.010233], [39.743218, -105.011864],
```

```
    [39.748473, -105.002487], [39.740283, -104.999037]],
    {color: 'blue', weight: 4}).addTo(map);
```

Save your document and refresh your browser.

Other Simple Vector Data Types

There are a number of other simple data types and groups you can add, read more about them in the Leaflet documentation. These include:

- Path
- MultiPolyline
- MultiPolygon
- Rectangle
- Circle
- CircleMarker

Feature Groups and GeoJSON

Leaflet also supports adding groups of features using class called L.featureGroup. If we wanted, we could have restructured our code to the point, line, and polygon above by placing them all in a feature group.

Additionally, Leaflet is designed work natively with a geographic data format called GeoJSON. GeoJSON are lightweight JavaScript objects that are commonly used to pass and load data to web maps. We will look at them later in this exercise, but first, lets add some interactivity by including some pop ups on the data we have already added. At this point we have some features loaded on to our map, and it should look something like the following.

Where did I get my Lat / Lon values? *A nice trick can be navigating to Google Maps, right clicking on a location on the map, and selecting 'What's here?'. This will provide latitude and longitude values for that location you can then copy.*

3. Add Interactivity with Pop ups

Leaflet makes it easy to add pop ups to your data points. Pop ups are a simple way to add interactivity to your map. When a viewer clicks on the pop up, information will be displayed. Pop ups are included by binding them to the marker or feature that you wish to apply a pop up on. When the visitor clicks on this feature, the bound pop up will appear.

Add a pop up to our marker layer

There is a marker sitting on our map at Auraria Library. Let's add a popup that tells us this is Auraria Library. The simplest way to add a pop up is to use the bindPopup method of L.marker. Enter the following code in your `script` block in your HTML document, and we will step through what is happening.

```
// Bind popup to Data Point object
```

```
myDataPoint.bindPopup("This is Auraria Library.");
```

Save and refresh your map. Click on the marker. You will see a pop up stating "This is Auraria Library." appear. bindPopup is 'method' of our marker object class, just like addTo('map') we used above. Methods are actions that can be performed on objects, and marker is an object. Also note, within the quotations where we wrote "This is Auraria Library.", we can write HTML as content. Change the code to the following, and see what happens.

```
// Bind popup to data point object
myDataPoint.bindPopup("<h3>Auraria Library</h3><p>Denver, CO<br>Information about Auraria Library.</p>");
```

This means, within our popup, we can add links, images, lists, and many other HTML elements. This includes videos! For fun, try adding a YouTube clip!

Add pop ups to our other data features

Just like with the marker object, we can add pop ups to our other features using the bindPopup method. Use the variable that we set the feature to (ie `var = myDataPoint`, `var = myDataLine`, and `var = myArea`) as the object, then use bindPopup. Enter the following block of code in your script tags, after the other code.

```
// Bind popup to area object
myArea.bindPopup("Auraria Campus");
```

Save and refresh. Click on the data features to see the pop ups in action.

4. Introduction to Map Events

What the map does when a user interacts with it and its various components is called an Event. A pop up is a very basic built-in event, but there are many other events that Leaflet can handle. Take a at some of the events the map object can handle here. The main one we will focus on here is what happens when a visitor clicks on the map, or what happens when there is a "Mouse Event". When a visitor clicks on the map, you can return a handful of different properties. One of them is latitude and longitude, making it easy to add simple functionality that will return latitude and longitude of the location where the mouse was clicked.

Find Latitude and Longitude of a Mouse Click

To complete that task, we need to do a couple of things.

1. Add an empty pop up object to our map.
2. Write a function that creates a pop up.
3. Tell the map that when it is clicked, run this function

The following snippet of code does just that. Read through it and see how it addresses each step. Enter this into document in between the script tags.

```
// Create an Empty Popup
var popup = L.popup();

// Write function to set Properties of the Popup
function onMapClick(e) {
    popup
        .setLatLng(e.latlng)
        .setContent("You clicked the map at " + e.latlng.toString())
        .openOn(map);
}

// Listen for a click event on the Map element
map.on('click', onMapClick);
```

Let's break this down a bit. First we created our empty pop up object using L.popup(). The next part is a JavaScript function. A JavaScript function is a block of code designed to perform a particular task and is executed when calls it.

More information on Javascript functions.

In this case, the function changes the properties of the pop up, or in our case, sets them, because the pop up object is empty. It sets the lat and lon of the pop up to the location of the mouse click, sets the content of the popup to say what that location is (has to convert it to a string to do so), and then opens the popup on the map. Finally, the last piece, is an event on the map object that says on click, run the function onMapClick. Complex, yet surprisingly simple all at once.

Save and refresh. Click on the data features to see the pop ups in action.

More on events is found in the Leaflet documentation, under each object, look at the events and properties that are available for you to utilize and manipulate.

5. Use JQuery and Add a GeoJSON

Learning the fundamentals of adding small datasets to our map along with some basic interactivity is important, but often times you will be adding larger datasets to our maps, sometimes containing hundreds of features. Leaflet is designed to work natively with a data format called GeoJSON. GeoJSON is JavaScript object that contains geographic data. This might not make sense right now, and that is fine, we will discuss JavaScript in depth in the next session. But for now, lets look at one way to load a GeoJSON into our map.

The GeoJSON: Denver Libraries

In the downloaded materials, there is a **data** folder that contains a file called **"libraries.geojson"**. This is a dataset of libraries in Denver. Quite handy if you need a caffeine fix and are looking for where to go! Open up the GeoJSON in your text editor to see what a GeoJSON looks like.

Use the JQuery JavaScript Library

The first thing we want to do is add a super common and super useful JavaScript library called jQuery to our page. jQuery makes it easy to manipulate a web page by finding elements on the page, setting their styles and properties, handling interaction events, and more. It has a nice helper method, called getJSON(), we will to use to load our GeoJSON file onto our map.

Enter the following line of code at the bottom of the body section to add the jQuery library to our page after the lines that add the Leaflet JavaScript library.

```
<body>
    ...
        <!-- HTML Page Elements are here -->
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
        <!-- Web Map Scripts are here -->
    ...
</body>
```

Add the GeoJSON

To add the GeoJSON to your map, use $.getJSON(). The $ signifies we are using an object from the jQuery library, and getJSON is the object. Within the brackets, we put the location of the GeoJSON on our server ("libraries.geojson"), then a callback function (function(data)) that executes if the GeoJSON is found. The callback function will contain the code that will add the GeoJSON to our map.

To add the GeoJSON to the map, use L.geoJson() the Leaflet library. Pass the dataset to L.geoJson, then add it to the map element. Simple enough right?

Add the following code to your script, between the script tags.

```
// load GeoJSON from an external file
$.getJSON("libraries.geojson",function(data){
    // add GeoJSON layer to the map once the file is loaded
    L.geoJson(data).addTo(map);
});
```

Simple enough right? Click save and refresh your page to see the GeoJSON added to the map.

Add Popups to Show Library Name

We can see the points, but they might not be very useful without adding popups. We can add popups in a very similar manner as above, except since we are using a full dataset, it doesn't make much sense to add them one by one. The L.geoJson object has a option called onEachFeature that runs a function on each feature when it is added to the map. We can use this to run a function that adds a popup to each feature when it is added to the map. The syntax, which goes in brackets after we specify the data we are adding, looks like the following. Enter this into your `getJSON` and `L.geoJson` functions.

```
$.getJSON("libraries.geojson",function(data){
    // add GeoJSON layer to the map once the file is loaded
    L.geoJson(data,{
        onEachFeature: function (feature, layer) {
            layer.bindPopup(feature.properties.LIBRARY_NA);
        }
    }).addTo(map);
});
```

Once entered, save and refresh your page. Click on one of the popups, you will see the name of the libraries appear. Don't worry if you are a bit confused, we will explain what happened in depth during the next session!

Other Data Formats

There are also a couple of very useful plugins, one called Leaflet Omnivore, that can read in other data formats, and Leaflet Shapefile, that will add a shapefile to your map. We will cover these at a later date, when we look at plugins and the additional capabilities they add to Leaflet.

What to Turn in for Lab 8

- Submit your link to Blackboard before the start of class on November 7.

For the project…

- **First things first – copy your map file and the libraries.geojson file to your webserver, and submit the link on Blackboard.**

For the project, you will create an interactive web map using Leafletjs. You should pick three out of the following skills:

1. You may want to have multiple data layers that you can turn on and off.
2. You may want to have some interactive information in a side panel that is informed by the map.
3. You may want to have the user enter in or annotate information on the map.
4. You may want to give the user the option to change background tiles.
5. You may want to have the map compute area, or length based on the user's import.
6. A skill approved by the instructor.

More importantly, your project should be coherent thematically – it should include information to tell a story. We'll have presentations in class at the end of the semester.