Lab 6: Leaflet.js, APIs, and Plug-Ins.

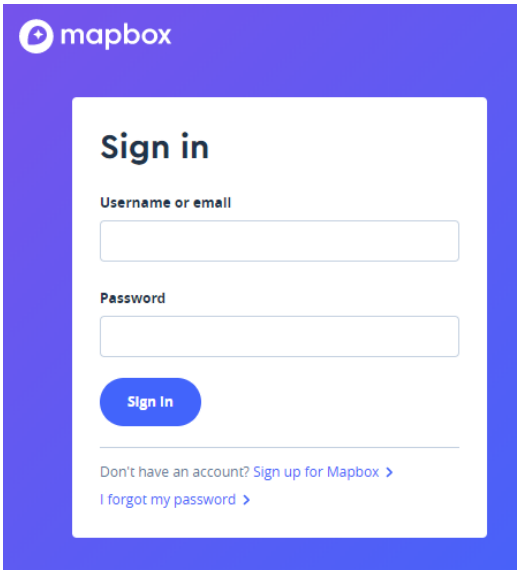
OBJECTIVES

This lab introduces a web mapping/GIS library – Leaflet. Compared to other mapping libraries, leaflet provides a much more intuitive way to create a mapping application. Unlike ArcGIS Javascript API, you may more easily understand the functionality of different classes or methods because of simplified naming schemes. This lab covers the following basic tasks:

1. Load a basic map.
2. Markers, Circles, Polygons, and PopUps
3. Map mashups – The Zoom Viewer
4. Changing the Tile Server and Tile Layer options
5. Draw polylines
6. Using Plug-ins - Geocoding

Task 1. Load a basic map

- Set up an account with mapbox.com, here, to create your accessToken before you start. You get 75,000 free tile requests with this token.



Everything you can do with mapbox is in the API Documentation
https://docs.mapbox.com/api/

- Copy and paste your accessToken to a word document or a text file for future use. Your account on Mapbox will do the same.
- Open the local version of your GitHub Page repository in Atom.
- Create a file in the "javscript" folder called, "secret_keys.js" and in the *username.github.io* directory create a file called, ".gitignore".
- In the .ignore file, type "javascript/secret_keys.js", and save the file.
- In the "secret_keys.js" file, type the following and save the file:
  mapbox_access_token = '*YOUR ACCESS TOKEN HERE'*
- At the top of the <head> element in 'index.html' type:
  <script src="javascript/secret_keys.js"></script>

- Add the following code to the **TOP** of the <head> section. This is to define the leaflet CSS and the Leaflet JavaScript library. The reason why we want it to be at the **TOP** of the head section is because we will want any CSS we put into our CSS file to modify the style of the map to supersede default styling provided by Leaflet. Generally, the last CSS file loaded into the browser window (by line #) is the setting that the browser will use for displaying.:

```
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="https://unpkg.com/leaflet@1.5.1/dist/leaflet.css"
integrity="INSERT FIRST INTEGRITY CODE"
crossorigin=""/>
<script src="https://unpkg.com/leaflet@1.5.1/dist/leaflet.js"
integrity="INSERT SECOND INTEGRITY CODE"
crossorigin="">
</script>
```

Everything you can do with Leaflet is in the Documentation
https://leafletjs.com/reference-1.5.0.html

- **FIRST INTEGRITY CODE:**

  sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZChSR7A==

- **SECOND INTEGRITY CODE:**

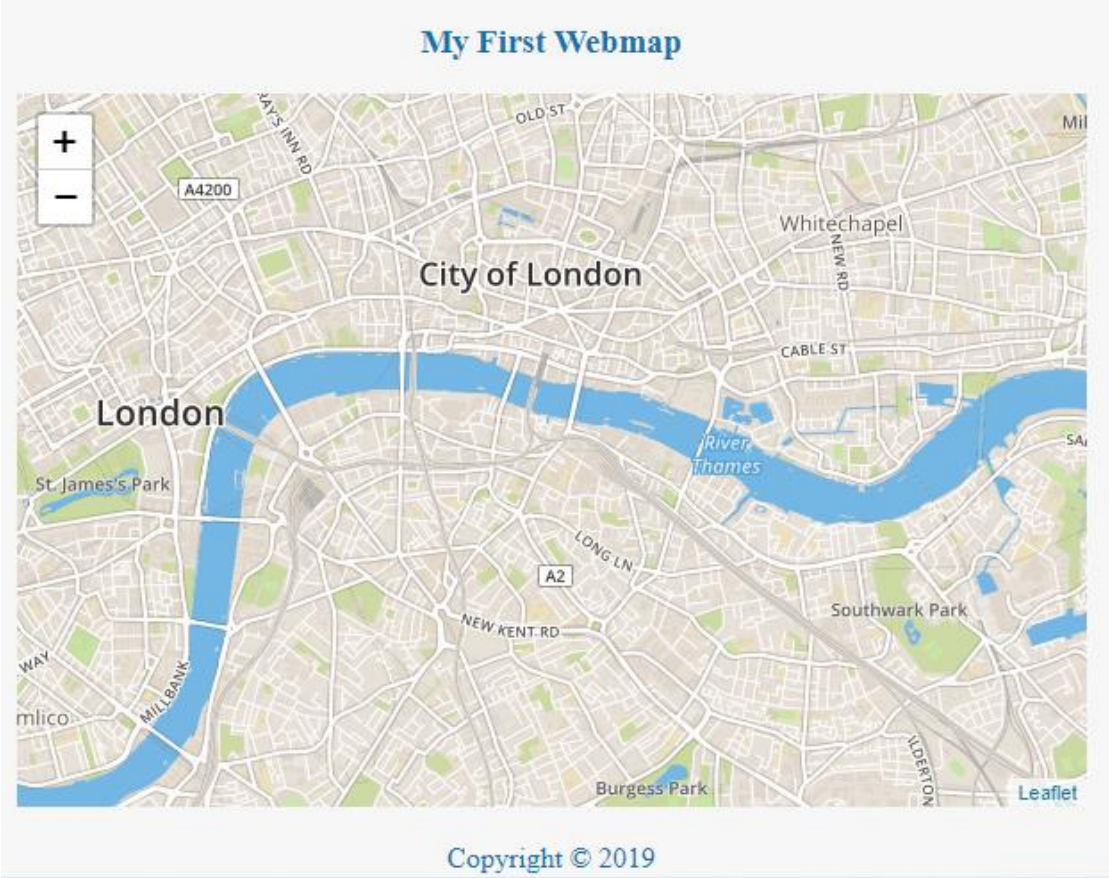
  sha512-XQoYMqMTK8LvdxXYG3nZ448hOEQiglfqkJs1NOQV44cWnUrBc8PkAOcXy20w0vlaXaVUearlOBhiXZ5V3ynxwA==

- Add the following code to the <body> section, make it the next to last div element within the body container div element. It should be below the webmap-reviews div element and above the copyright div element.

```
<h3>My First Webmap</h3>

<div id="mapid" style="width: 600px; height: 400px; margin: auto;">

    <script>

     var mymap = L.map('mapid').setView([51.505, -0.09], 13);


          L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}', {
              maxZoom: 18,
              id: 'mapbox/streets-v11',
              accessToken: mapbox_access_token,
          }).addTo(mymap);

      </script>


  </div>
```

- Save all files, preview in your browser locally to make sure that the code is working, stage the changes, commit with the message “Added a Leaflet webmap with a Mapbox tile layer and .ignore file.” and push your changes.

The bottom of your page should look like this:



Task 2. Markers, Circles, Polygons, and PopUps

- **BEGIN TYPING INSIDE OF** the end of the <script> that you just inserted in the <body> section to add all of the following items. Each time you add a new item (ie. A marker, a circle, etc.), save the file and preview it locally to see what happened. If nothing happens, fix the script before adding the next item.:

- Add a marker:

```
var marker = L.marker([51.5, -0.09]).addTo(mymap);
```

- Add a circle:

```
var circle = L.circle([51.508, -0.11], {
  color: 'red',
  fillColor: '#f03',
  fillOpacity: 0.5,
  radius: 500
}).addTo(mymap);
```

- Add a polygon:

```
var polygon = L.polygon([
  [51.509, -0.08],
  [51.503, -0.06],
  [51.51, -0.047]
]).addTo(mymap);
```

- Add Popup windows for each object:

```
marker.bindPopup("<b>Hello world!</b><br>I am a popup.").openPopup();
circle.bindPopup("I am a circle.");
polygon.bindPopup("I am a polygon.");
```

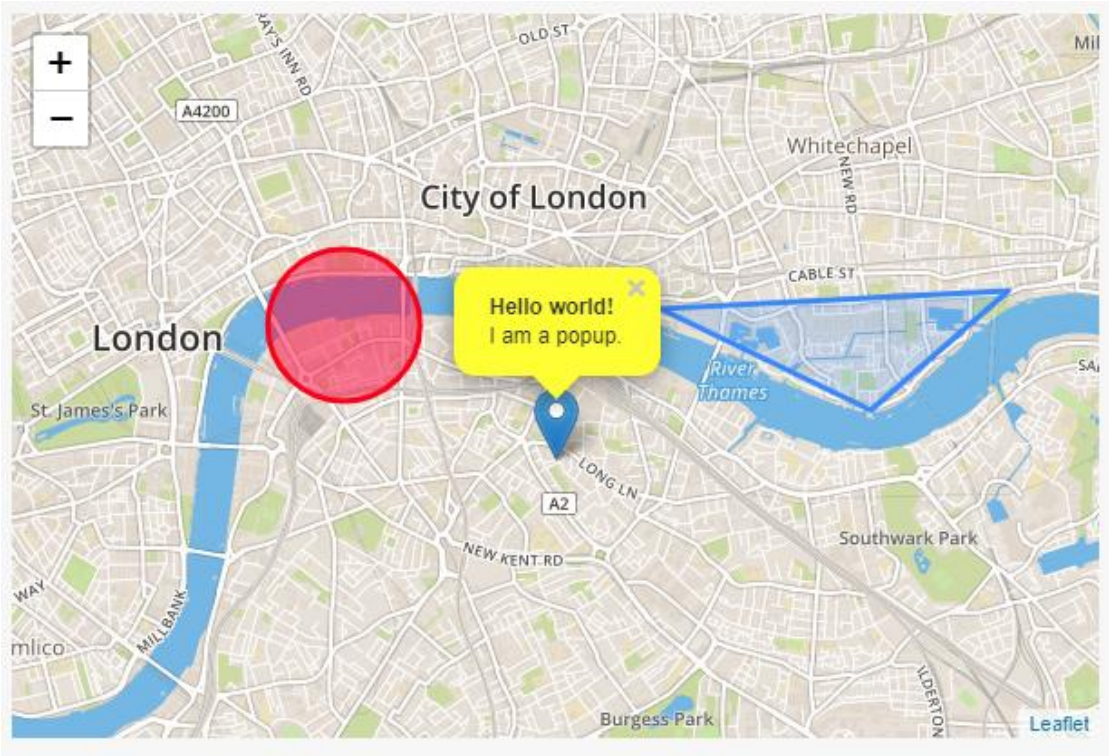
- Now, let's add our own styling. If you were to go to the web address of the Leaflet CSS file provided in our head section, you would find that the popup containers have CSS classes that are used for styling. Create a new CSS file called "map_style.css" and add the following code to it:

```
.leaflet-popup-content-wrapper,
.leaflet-popup-tip {
  background: yellow;
  }
```

- Add the stylesheet to your <head> section, place it appropriately so that it supersedes the Leaflet CSS file:

```
<link rel="stylesheet" type="text/css" href="css/map_style.css">
```

- If you have done all steps appropriately to this point, your map should look like this. If it does not, fix the code before moving on:



- Change the zoom on click for each object:

```
marker.on('click', function(e){
  mymap.setView(e.latlng, 14);
});

circle.on('click', function(e){
  mymap.setView(e.latlng, 13);
});

polygon.on('click', function(e){
  mymap.setView(e.latlng, 13);
});
```

Save all files, preview in your browser locally to make sure that the code is working, stage the changes, commit with the message "Added content to the webmap with custom popups and dynamic zooming." and push your changes.

Task 3. Map mashups – The ZoomViewer

- In this task, we will be adding a listener that dynamically displays content in a ZoomViewer as the user zooms.

- We are going to set the view at the end of the script, and there is no reason to redundantly set the view twice. Modify the variable, mymap, in your <script> section as follows:

```
var mymap = L.map('mapid');
```

- Now add the following lines to the end of your <script> section:

```
var ZoomViewer = L.Control.extend({
  onAdd: function(){
    var gauge = L.DomUtil.create('div');
    gauge.style.width = '200px';
    gauge.style.background = 'rgba(255,255,255,0.5)';
    gauge.style.textAlign = 'left';
    mymap.on('zoomstart zoom zoomend', function(ev){
      gauge.innerHTML = 'Zoom level: ' + mymap.getZoom();
    })
    return gauge;
  }
});

(new ZoomViewer).addTo(mymap);

mymap.setView([0, 0], 1);
```

- When you zoom the map in and out, the number in the ZoomViewer at the top-right of your map should change to indicate your zoom level. Save your script locally and preview it.

Task 4. Changing the Tile Server and Tile Layer options.

Step 1: Change the default mapbox tilesets.

- Modify the "id" in the leaflet tileLayer to 'mapbox/outdoors-v11', it should look like this:

```
L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}', {
    maxZoom: 18,
    id: 'mapbox/outdoors-v11',
```

- Save and view the webpage locally.
- Now, try each of the following 'mapbox/satellite-v9', mapbox/dark-v10', 'mapbox/navigation-day-v1', and 'mapbox/navigation-night-v1'.

- **Start a new word document and answer the following questions: Question 1: What does your map look like after applying 'mapbox.mapbox-traffic-v1'? What is the minimum zoom level for this tileset to be visible?**

You can find and create styles, tilesets, and datasets using mapbox's online studio

https://studio.mapbox.com/

Step 2: Use tile servers other than mapbox.

- Comment out the Leaflet Tile Layer by typing '/*' before it, and '*/' after it. Example:

```
/* MAPBOX TILE LAYER
L.tileLayer('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access_token={accessToken}', {
  maxZoom: 18,
  id: 'mapbox.mapbox-traffic-v1',
  accessToken: mapbox_access_token
}).addTo(mymap);
*/
```

- Create a new Leaflet Tile Layer using Open Street Map:
-

```
L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
}).addTo(mymap);
```

- Save and open your webpage locally.

- Now, replace the tileLayer address above as follows:

```
L.tileLayer('https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png',{
}).addTo(mymap);
```

- ***Now, go to:***

https://wiki.openstreetmap.org/wiki/Tile_servers
Or
http://leaflet-extras.github.io/leaflet-providers/preview/index.html

- Try using several of the different Tile servers by copying and pasting the tiles URL. After viewing several different layers, keep whichever is your favorite.

- **Answer the following question in your word document.:**
  Question 2: Why are {s}, {z}, {x}, and {y} in the address? Provide an example of an address where you manually input {s}, {z}, {x}, and {y}, describe what your map link will show when I open it.

Save all files, preview in your browser locally to make sure that the code is working, stage the changes, commit with the message "Switched to OpenStreetMaps tiles." and push your changes.

Task 5. Draw a polyline

- Add the following variable, 'polyline', below your variable, 'polygon':

```
var polyline = L.polyline([
  [51.506, -0.08],
  [51.502, -0.06],
  [51.507, -0.047]
]).addTo(mymap);
```

- Save your file, and view it locally.
- **Question 3: What river does your polyline follow? Change your tile layer if you need to, but change it back to your favorite OpenStreetMaps tile layer after you get the river name.**
- **Question 4: Take a screenshot of the polyline you drew on your map.**

Task 6. Using Plug-ins - Geocoding

- Go sign-up for a free-trial plan to use a Geocoder API at:

https://opencagedata.com/users/sign_up

- Add the following code to the <head> section.

<link rel="stylesheet" href="https://cdn.jsdelivr.net/gh/opencagedata/leaflet-opencage-search@d4cbd36122efc8d17152b4177ed0e12165305441/dist/css/L.Control.OpenCageData.Search.min.css" />

<script src="https://cdn.jsdelivr.net/gh/opencagedata/leaflet-opencage-search@d4cbd36122efc8d17152b4177ed0e12165305441/dist/js/L.Control.OpenCageSearch.min.js"></script>

- In your <script> section, comment everything that is after the OpenStreetMaps tile layer and before the line that you set the map view on. Then, directly after where you defined the variable 'mymap', add the following code:

```
var options = {
  key: geocoder_api_key,
  limit: 10
};
var control = L.Control.openCageSearch(options).addTo(mymap);
```

- Add your 'geocoder_api_key' variable to the 'secret_keys.js' file.
- Change the text "My First Webmap" to "Geocoding Map"
- Save your page and view it locally.
- Use the Geocode button to search "San Marcos, Texas".
- Search the city name of your hometown and display it on the map.

Question 5: Save a screenshot of the search result of your hometown.

Leaflet has A TON of plugins. You can view many more plugins, including alternatives to this Geocoding plugin here:

https://leafletjs.com/plugins.html

Save all files, preview in your browser locally to make sure that the code is working, stage the changes, commit with the message "Changed map to a Geocoding Map." and push your changes.

HOMEWORK ASSIGNMENT

1. **(20 points)** Using the skills you learned in the previous labs, create an external javascript file called, "map.js" in the same directory as "external.js". Move the script you created in this lab to the "map.js" file. Inside the "index.html" file, call

the external script “map.js” from within the “mapid” div element.

Save all files, preview in your browser locally to make sure that the code is working, stage the changes, commit with the message “Moved map script to an external javascript file, map.js.” and push your changes.

2. **(30 points)** Modify your web map to display the location of the seven wonders ("India's Taj Mahal", "Great Wall of China", "Petra in Jordan", "Brazil's statue of Christ the Redeemer", "Peru's Machu Picchu", "Mexico's Chichen Itza pyramid", "The Colosseum in Rome").

- Use a tile layer base map of your choosing.
- The coordinates of the seven wonders are:
  - 27.174961, 78.042385
  - 40.334245, 116.477652
  - 30.328611, 35.441944
  - -22.951389, -43.2108334
  - -13.163056, -72.545556
  - 20.682778, -88.569167
  - 41.890169, 12.492269
- Use a unique marker symbol for each wonder landmark.
- Customize the popup windows as you see fit (you should not use the default pop up window style). Your customized popups should still be aesthetically pleasing.
- For each landmark, add a pop-up showing the name of the landmark in bold, a photo of the landmark, and a short paragraph introducing the landmark (note: you can search for photos and introductory texts on Google/Wikipedia).

Save all files, preview in your browser locally to make sure that the code is working, stage the changes, commit with the message “Changed webmap to display the Seven Wonders of the World.” and push your changes.

3. **(40 points)** Use plug-ins to improve the look and/or functionality of the Seven Wonders webpage that you just created. The webpage must use at least **three** unique plug-ins for undergraduates and **four** unique plug-ins for graduate students. The plug-ins must be from at least two different categories on the plug-ins page:

https://leafletjs.com/plugins.html

There may be no more than two plug-ins from the same category.

Each time that you finish successfully adding a plugin, save all files, preview in your browser locally to make sure that the code is working, stage the changes, commit with the message “Added ***(INSERT PLUGIN NAME HERE)*** to the Seven Wonders of the World.” and push your changes.

4. **(10 points)** Add a link to your GitHub page (https://*username*.github.io) to the top

of the word document with your answers to questions 1-5 (coming from tasks). At the bottom of this document, list and describe each of the plug-ins you used for homework assignment. You must include a link to the documentation page (or github page) of the plugins you used above each description.

Deliverable: Submit your completed word document in a zip file on Moodle.

Due Date: October, 31, 2021