Tutorial Leaflet WebMap

In this tutorial you will learn how to make some simples leaflet maps by using JavaScript code within an HTML page.

So what is Leaflet?

I consider Leaflet to be the fastest and most fail-safe way of making an interactive map that can be used on a mobile device. This is thanks to the fact that Leaflet is written in JavaScript and only "weighs" 38 KB.

And why is that so useful?

First, because JavaScript is a language used by all modern webpages, and some experts consider JavaScript to be the "most ubiquitous programming language in history" (O'Reilly 2011). Second, the fact that it is only 38 KB in size makes it very fast to load, compared to the size of ESRI, for example (almost 1GB). Clearly you are not going to use Leaflet for making GIS models, but it is good for displaying maps, especially interactive maps. Another beauty of Leaflet is that it is free, and every month new plugins are added giving the library more complex applications.

Will I have to learn HTML and JavaScript?

If you know HTML and JavaScript, this will be easier, but you do not need to know HTML or JavaScript to do this tutorial or to make use of simple Leaflet tools. Also, major GIS software (eg: QGIS and ESRI) and programing languages (eg: Python and R) have plugins or libraries that allow you to export maps, including Leaflet maps, so you can actually create Leaflet maps without knowing how they work. However for the tutorial we will get our hands dirty and write the script in HTML.

You can do this tutorial using a simple text editor (like notepad or textedit), however I suggest you install Visual Studio Code in your computer. It is really a great software for coding and works on Mac, PC and Linux systems. It is especially useful for helping you catch simple typos that cause errors.

Part 1.

Let's get started: create a new file in Visual Studio Code and save the empty file as `myleafletmap1.html`

```
<html>
    <head>
        <title>My First Leaflet Map</title>
    </head>
    <body>
    This is not as hard as I thought.
    </body>
</html>
```

This previous lines are all HTML code. The `<html>` (this is called a "tag") is indicating that this is the beginning of an HTML file, the `<head> tag` is indicating that this is the beginning of the HTML metadata (data about data).
`<title>` indicates that this is the beginning of the title that will appear when you put your mouse over the tab of the webpage.
`<body>` indicates the beginning of the instructions for the body of the HTML page. In this case we just added some text that you should be able to see inside the page.
`</body>` (notice the slash) indicates the **end** of the body.
`</title>` indicates that this is the end of the title.
`</head>` indicates that this is the end of the head, and
`</html>` indicates that this is the end of the html.

If you save and open this file, you should see a page with the text that you wrote in the body, and if you move the mouse to the tab in your browser, you should see the title.

Next now let's work a bit with JavaScript. You **do not** need to install any software to use it. JavaScript is everywhere, in fact, this is what makes it very powerful. However it was developed a while ago, so like HTML, the way to write JavaScript might seem weird.

After `</title>` we will add the following lines:

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css"
   integrity="sha512-
xwE/Az9zrjBIphAcBb3F6JVqxf46+CDLwfLMHloNu6KEQCAWi6HcDUbeOfBIptF7tcCzusKFjFw2yuvEp
DL9wQ=="
   crossorigin=""/>
```

These lines indicate the location of the CSS style file needed for displaying leaflet map on your HTML file. CSS is the language that describes the style that is used in the HTML page. In this class we will **not** go over the CSS language.

After the previous lines you can add the following lines:

```
<script src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js"
   integrity="sha512-
gZwIG9x3wUXg2hdXF6+rVkLF/0Vi9U8D2Ntg4Ga5I5BZpVkVxlJWbSQtXPSiUTtC0TjtGOmxa1AJPuV0C
Pthew=="
   crossorigin=""></script>
```

These lines are indicating where to look for the leaflet library, which you might notice is called `leaflet.js`. The .`js` stands for JavaScript.

After the `<body>` line add the following line

```
<div id="mapid"></div>
```

The `<div>` tag is to create an HTML element that we will style with CSS, and use to perform JavaScript tasks. We are calling this container "mapid", but we could have called it whatever you wanted.

After `</div>` add the CSS that will describe the size of the mapid container by inserting the following lines. The following lines indicate that the element mapid will have a size of 900 pixels in width by 500 pixels of height.

```
<style>
                    #mapid{ width: 900px; height: 500px; }
</style>
```

Now after `</style>` we will add some JavaScript lines. In JavaScript it is easy to insert comments (explanations of your code) by adding a double slash (//) before the comments. Lines starting with double slash do not get interpreted. Hence I will put a comment after each line so you understand what is happening.

First you need to let HTML know that a script is coming, so we do that by typing `<script>`.

```
<script>
// this means that the following lines will be script
// (it could also point to an external script file).

var map = L.map('mapid').setView([35.91, -79.05], 17); // YOU CAN CHANGE THE ZOOM

// this line is creating the variable map by  initializing the L.map class (a
Leaflet class) with the mapid element that we created before. setView is a method
that can be applied to the class L.map : you can give it a latitude, longitude
and a zoom.  The higher the value of the zoom, the finer the initial scale of the
map.

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
}).addTo(map);

// L.tileLayer can display a tile layer. You can check out other tile layer
examples at https://leaflet-extras.github.io/leaflet-providers/preview/. The
location of the tile layer needs to be in single quotes. You can also indicate
the attribution for that layer. In this example we used the Open StreetMap
```

```
attribution. The '&copy; <a> and the other information you see here is HTML code
adding a copyright sign, and making a hyperlink to OpenStreetMap.
// At the end we include .addTo(map), which is indicating to add the tile layer
to the variable map.

L.marker([35.911271, -79.049807]).addTo(map)
    .bindPopup('Hey I am at Carolina Hall 322') // YOU CAN CHANGE THE MESSAGE
    .openPopup();
// end of script
</script>
```

Here we are making a pin located at Latitude 35.911271 and Longitude -79.049807.
Again notice that we wrote the `.addTo(map)` to indicate that this needs to be added to the map. Also we indicated other things that will happen if we drag the mouse over the icon.
.bindPopup indicates that a popup with the message 'Hey I am at Carolina Hall 322' should be attached.
.openPopup is indicating that it should open a popup when clicked and close the rest.

Save the document and open it by double clicking the file wherever you saved it.
A Web browser should open the document and you should see the HTML.
(If you want this document to be online, let me know: I can show you, but I will have to explain a few more techniques which are not part of the objective for today's tutorial. These also require a little more scripting or interactions with other software.)

Now go to your file and make changes in the areas that I indicated that it was safe to make changes, such as the zoom level, the message, and the tile background layer.

If you had some trouble making your document, I have already created an HTML that works named ***part1leaflet.html .***

Part 2.

In this second part of the tutorial we will make a map starting with a point shapefile. Did you know that shapefiles are super annoying? If you need to share them, you have to zip them because the .shp will not work without .prj and .dbf and all the other little files that make the shapefile. Also shapefiles have the limitations of 255 characters that are accepted inside a column. Because of this, the most common geographic data format in the internet is the GeoJSON, and it is very easy to use and to see what it contains. Leaflet understands GeoJSON, because it is in the native format of how we write objects in JavaScript (JSON stands for JavaScript Object Notation). GeoJSON are very easy to make with QGIS or R and ArcGISPro.

To not overwhelm you with data structures, in this second part we will only work with a point GeoJSON that contains only one point, the location of the UNC Well. Open the GeoJSON in a text editor and you should see something like this:

```
{
```

```
"type": "FeatureCollection",
"crs": { "type": "name", "properties": { "name":
"urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
{ "type": "Feature", "properties": { "id": 1, "descrip": "unc well" },
"geometry": { "type": "Point", "coordinates": [ -79.0512352278691,
35.912064676772367 ] } }
]
}
```

As you can see it has a lot of information. But it is easy to decipher. Notice that there is some text in double quotes "" followed by a colon : and then some information that can be text or numbers. The first text is a **keyword**, and the following information is its **value**. For example, the first keyword-value pair we see is `"type":"FeatureCollection",` this is telling us that this is considered a list of features.

Then we see the `"crs":` which means the projection. Notice the word CRS84, which relates to EPSG4326, which you may remember from 370 or 491,is a projection that uses WGS84 as datum and decimal degrees as units.

Then we see the "features" keyword, and inside the features value, there is another list key-value pairs. Notice the fields `"id":1` and `"descrip": "unc well"` which are equivalent to a table in a shapefile that has the a column named "Id" and a column named "description" and each row having information in the feature. Also notice the `"coordinates": [ -79.0512352278691, 35.912064676772367 ]` in Lon, Lat.

The easiest way to bring this information to the map is by doing the following.

1. Go to the file GeoJSON file and open it with a text editor.
2. At the top of the file, before the data starts, add the following text.

   ```
   var uncwell =
   ```

3. What this is doing is making the GeoJSON to be interpret as a JavaScript variable when we call it from our HTML script.
4. Your new file (it is a tiny script) should look like this

   ```
   var uncwell = {
   "type": "FeatureCollection",
   "crs": { "type": "name", "properties": { "name":
   "urn:ogc:def:crs:OGC:1.3:CRS84" } },
   "features": [
   { "type": "Feature", "properties": { "id": 1, "descrip": "unc
   well" }, "geometry": { "type": "Point", "coordinates": [ -
   79.0512352278691, 35.912064676772367 ] } }
   ]
   }
   ```

Now in your HTML file do the following.
Directly under the `<script…></script>` tags where you called in Leaflet.js at the top of the file, call in this new script by typing

```
        <script src = "/folder_location/old_well_4326.geojson"></script>
```

Remember, like R, most programs in the internet understand the forward slash in the directory location, and not the back slash (\) that Windows uses.

In some place after the variable map was created and before the end of the JavaScript you are writing (the HTML `</script>` line) place this line:

```
L.geoJSON(uncwell).addTo(map)
```

`L.geoJSON` will open the variable uncwell and add it to the map. The variable uncwell was created in the tiny script file old_well_4326.geojson and called in by using the line src = "/folder_location/old_well_4326.geojson" that you just added.
Save the HTML file and open it. You should now see a marker over the UNC Well.

If you had some trouble making your document, I have already created an HTML that works named ***part2leaflet.html*** . However, you will need to change the file location and file name of the GeoJSON.

Part 3: Symbology

Now let's make some symbols and some fun popups. All these lines should be in that area of the HTML were we are doing most of our javascripting. The objective will be to put a UNC Well symbol on the map.

Add the following lines:

```
// create custom icon variable
        var uncIcon = L.icon({ // notice the L.icon which is a Leaflet object with properties
        iconUrl: 'https://faopharmacy.unc.edu/files/2018/04/Old-Well-Icon-Example.png',
        // this points to a jpg image obtained from the internet
        iconSize: [50,50], // size of the icon
        popupAnchor: [0,0] // where the icon is located relative to the lat lon of the point.
        });


//YOU COULD CHANGE the iconURL, iconSize and popupAnchor to see how the map changes

// create an HTML variable that will pop up with a UNC Well photo

var customPopup = "UNC Well<br/><img
src='https://global.unc.edu/files/2020/02/spring_4_09__12_054-768x511.jpg'width='100px'/>";

// YOU COULD CHANGE the https link to another picture.
// YOU COULD CHANGE the 100px to other values like 50px or 400px to see what happens
// the following line should go at the end, before script closing tag

        L.geoJSON(uncwell, { // a
        pointToLayer: function (feature, latlng) { // b
        return L.marker(latlng, {icon: uncIcon});}}) // c
        .bindPopup(customPopup) // d
        .addTo(map); //
```

In case you are interested in understanding what is happening behind the scenes: notice that this single line has the format `object(…).changeMe().addTo(map);` but it is broken into separate lines for clarity.

a. We provide uncwell as a property of object `L.geoJSON`
b. Since the object `L.geoJSON` does not accept information about icon style, we need to go a bit deeper inside the object to add style. We use the `pointToLayer` option and create a function that requires a `feature` and a `latlng`.
c. return is a JavaScript statement whose purpose is to return something from the function in which it appears. Here, it returns an object of type `L.marker` located at `latlng`, with the property "icon" set to `uncIcon`.
d. Using the method `bindPopup` (on the object of type `L.geoJSON`) and declaring to use variable `customPopup`
e. Finally using the addTo method (on the object of type `L.geoJSON`) to add all this to the map.

Now try making some changes, I have indicated some of the behavior that could be changed in the comments.

If you had some trouble making your document, I have already created an HTML that works named ***part3leaflet.html*** . However, you will need to change the file location and file name of the GeoJSON.