Lesson-1

Create a webmap map

In this lesson we will be creating a basic webmap using the leaflet javascript library, basic HTML, and basic CSS. This lesson assumes little to no experience with HTML, Javascript and/or CSS.

The intent of this lesson is to provide you with a basic handle on how to create a webmap and to understand the functions of the different components that make up a webmap. We will learn how to load data from different providers and will create vector geometry for displaying in the map.

You can think of HTML/Javascript/CSS like this; Picture a multi-use tower (apartments, officespace, retail). In the complex we have individual rooms for each use-type and each of those rooms are going to be styled differently and have different functions.

- HTML is the supporting structure of the complex (columns, girders, basement, roof).
- Javascript is the infrastructure (gaslines, power, internet, tv) and also determines what use-type a room will be and how that room will be used.
- CSS is the styling (carpets, paint, window dressings).

This is a pretty general analogy, since all three have the ability to do some part of each other. But for our purposes, this will suffice.

Pre-requisites

The Operating System will not matter as all three (Windows, Mac, Linux/Unix) will display the same.

You will need a text editor for this tutorial. Notepad on Windows will work fine as will TextEdit on Mac. If you are on Unix/Linux, you are ahead of the game. Personally, i like to use SublimeText as it has syntax hilighting, different themes, and works on all three platforms. But choose what suites you best.

Setup

Before we begin creating files lets create a folder to store our webmap.

1. Make a folder somewhere on your computer to use as a workspace. I'm creating mine at: > C:\Documents\James\Leaning_Material\Lesson1

All the code that I create for this lesson will be stored in here.

One of the best tools to see what is going on in your webmap is the inspect tool. This is included in every major browser and can be accessed by right clicking on the map and choosing inspect in the dialog that pops up.

TODO: Show images of the inspect tool

HTML

We will create a basic skeleton for our webmap using HTML. HTML is simply a set of text tags which are interpreted by a web browser and tell it how to display. For example the tag `<b> test </b>` tells the browser to style the text between the tags as bold. **test**. The tag without a backslash starts the tag `<b>`, the tag with a backslash ends the tag `</b>`.

We will start out by creating a text file with an .html extension. There will be no data in this text file at the moment, we will add that later.

1. In sublime (or notepad or textedit) choose create new file and save this file with the name index.html in your Lesson 1 folder.
2. Within the new file, we start out by defining our html and declaring the DOCTYPE. This tells the browser what version of HTML we are using. The following tells the browser we are using html 5.
   `<!DOCTYPE html>`
3. Next we will create a container to store our html tags. All the html we will subsequently write will be contained between these `html` tags `<html>`

   `</html>`
4. Lets now add a container for metadata as well as metadata itself `<head> </head>` Within the head are tags for the
   - page title (displayed in the browser tabs): `<title></title>`,
   - links to CSS codes: `<link></link>` ,
   - information tags: `<meta></meta>` and,
   - simple page styling `<style></style>`. We will use all of these in our document. Indentation and spaces between elements is not neccessary but help with readability. Don't worry about what is in the link tag right now, we will get to that in a bit. The style tag simply tells that page that we do not want any padding or margings and that anything within our body, html and map tags should take up the entire screen or 100%.
5.
```
<!DOCTYPE html>

<html>
<head>

    <meta name="description" content="Web Mapping Example"></meta>
    <meta name="keywords" content="HTML,CSS,JavaScript"></meta>
    <meta name="author" content="James Banting"></meta>
    <meta charset="UTF-8"></meta>

    <title> Leaflet Web map example</title>

    <link rel="stylesheet"
href="http://cdn.leafletjs.com/leaflet/v0.7.7/leaflet.css"></link>
    <style>
        body {
            padding: 0;
            margin: 0;
        }
        html, body, #map {
            height: 100%;
            width: 100%;
```

```
            }
        </style>
    </head>
    </html>
```

We now have a completely valid HTML document that doesnt display anything. If you double click the index.html document in a file explorer, it should open in a web browser and display a black page. Note the text on the browser tab.

6. Now lets add a map. We first need to create a container to display content on the page. Below our ending `</head>` tag and before our ending `</html>` tag, place `<body></body>` tags. These tags contain the text that actually gets displayed on the webpage. I've omitted the beginning part of the html.

```
    </head>
        <body>

        </body>
    </html>
```

Now that we have a container for what gets displayed on the map, we will add a divison or section tag `<div></div>`. This is the element that will hold our map. All html tags can have attribute names within the tags. Attributes such as *id* help to distingusin indiviaual sections. Other attributes such as *class* can be used to assist with styling. We will name our div tag "map"

```
</head>
        <body>
            <div id="map"></div>
        </body>
    </html>
```

We dont need to put anything between the ``` < div>``` tags since jabascript will take care of creating the map.

7. Now comes the JavaScript part. We will use two different methods for linking javascript. One will link to a Content Delivery Network (CDN) and one will link to a local copy of javascript. A CDN is a way to provide fast access to javascript libraries and ensures that any changes to the javascript file are accessibile to multiple users. A local copy on the other hand is served by your server. This is fast for development and completely acceptable if the application you create is the only applicaiton accessing the library. We will use the CDN link for the leaflet JavaScript file and use a local copy for our mapping application (this is linked by folder and filename). To add the JavaScript links we need to set up the tags that define them. Below the `<div>` tags add two (2) `<script></script>` tags. Within the `<script>` opening tag, we will place a *src* attribute that tells the browser where to fetch the JavaScript library from. **Order is important here since we want the leaflet library to load before our local library does**.

```
    </head>
        <body>
            <div id="map"></div>
            <script
src="http://cdn.leafletjs.com/leaflet/v0.7.7/leaflet.js"></script>
            <script src="js/map_app.js"></script>
        </body>
    </html>
```

Don't worry about the file and folder in the second script tag. We will create those in the next section. If you copy and paste the url within the src attribute into your browser, you can see the actual leaflet code. This is what your browser uses to create the webmap.

Your html code should look like this (also availble in the provided index.html file).

```
    <!DOCTYPE html>

    <html>
    <head>

        <meta name="description" content="Web Mapping Example">
        <meta name="keywords" content="HTML,CSS,JavaScript">
        <meta name="author" content="James Banting">
        <meta charset="UTF-8">

        <title> Leaflet Web map example</title>

        <link rel="stylesheet"
href="http://cdn.leafletjs.com/leaflet/v0.7.7/leaflet.css" />

    </head>
        <body>
            <div id="map"></div>
            <script
src="http://cdn.leafletjs.com/leaflet/v0.7.7/leaflet.js"></script>
            <script src="js/map_app.js"></script>
        </body>
    </html>
```

In the next section we will create the javascript that renders our map.

Javascript

it is important to understand what makes up a webmap. The majority of webmaps we come across today allow for interaction with the user. Whether that be through 'slippy' maps (panning, zomming) or popups that show attributes.

Javascript is the language which allows for and drives this interaction. We could write our own javascript code to allow for all the interactions we could want, but that is hard and time consuming. Instead we include libraries in our application which give us the interaction we are looking for. These libraries have been tested quite extensively and are used around the world for webmapping. Two of the more prominent libraries are Leaflet and OpenLayers3. Both are excellent mapping libraries. Why not stand on the shoulders of giants? Leaflet is a smaller library in terms of size and has quite a large number of community generated plugins. OpenLayers has been around for a long time and has a rich history in the Free and Open Source Software (FOSS) Community. It also has a huge Application Programming Interface (API) which means that if you need to build a feature rich appliction and leaflet plugins dont do exactly what you are looking for, OpenLayers is probably your best bet.

For this tutorial we will be using leaflet since it is so easy to get up and going with.

Since we have our HTML built and ready to read from a JavaScript file, lets go ahead and create that file.

1. In sublime (or notepad or textedit) choose create new file and save this file with the name

mapp_app.js in a js folder within your Lesson 1 folder. The structure should look like this:
```
Lesson 1
└─index.html
│
└──js
    └─map_app.js
```

2. Since we already defined our mapping container in the html code, all we have to do is tell the browser what to put in the container. In the map_app.js file add the following JavaScript code:
   ```
   var map = L.map('map').setView([54, -114], 6);
   ```
   This creates an empty webmap for us, centered on Alberta (Latitude: 54, Longitude: -114) and a zoom level of 6 (Higher zoom level means more detail). Leaflet comes with some default settings for us even though we don't have anything to display. If you open index.html in your browser, you should see a plus and a minus in the top right. These are your zoom controls. The bottom left shows the attribution for any layers that get added. This is all customizable should you wish.
3. Now, lets add some data to our map. Directly below the `var map = ....` add the following:
   ```
   L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
           maxZoom: 18,
           attribution: '&copy;  <a href="http://openstreetmap.org">OpenStreetMap</a> Contributors',
           id: 'mapbox.streets'
       }).addTo(map);
   ```
   This creates a leaflet tile layer (Raster) using OpenStreetMap as the data source. Any tile provider can be used here, even your own (ESRI, GeoServer, etc.). The maxZoom part tells the map not to load this layer beyond zoom level 18 (Which is very close to the Earth's surface). The attribution will add text to the bottom left of the map. Reload the index.html page in your browser to see the map.

You should now have a slippery webmap up and running in 6 lines of javascript.

Custom Layers