Welcome!

Lets have some fun making an interactive map!

**Part I**

First open up your Module\_I folder and locate the “index.html” file; open this in your preferred editor of choice. You should see this:

<!DOCTYPE html>

<html>

<head>

<title>Zombie Locations</title>

</head>

<body>

</body>

</html>

Now before we can create a map we need to create a place in our web page for it to live! To do this just add a new “div” element in the body of your html code, and give it an ID. In this example we give it an ID called “mapid”:

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

Now we have a place to put a map…….but we have not provided any dimensions yet. To do this, open up your blank stylesheet.css and add in some css:

#mapid { width: 600px; height: 600px; }

Of course we know that if we have some css then we need to link it to our html code. To do this open your html code and add the following code to the header:

<link rel="stylesheet" href="stylesheet.css"/>

Great! But where is our map actually coming from? For this example we are going to be using Leaflet, so we will need to tell our web page where to find the Leaflet Libraries (one for the Leaflet CSS Stylesheet and one for the Leaflet Javascript). To do this we need to add the following lines to the head of our html code:

<link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.1/dist/leaflet.css" />

<script src="<https://unpkg.com/leaflet@1.0.1/dist/leaflet.js>"></script>

And with those links now added our web page knows where to find Leaflet and all of its goodies! But how do we actually generate a map? Thats easy, we just ask for one! For this example we would like to have a map of New York City which has a latitude and longitude of roughly 40.719190, -73.996589, so to “ask” for this map we just need to write some javascript to create a new variable for the map, and provide our latitude and longitude. Open your java.js file and add the following code:

var map = L.map('mapid').setView([40.719190, -73.996589], 13);

However in order to see anything other than blank space at that location we are going to need to get some tiles! Lucky for us Carto has some great tiles that are freely available! Just add these lines to you java.js file:

var CartoDBTiles = L.tileLayer('http://{s}.basemaps.cartocdn.com/light\_all/{z}/{x}/{y}.png',{

attribution: 'Map Data &copy; <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a> Contributors, Map Tiles &copy; <a href="http://cartodb.com/attributions">CartoDB</a>'

});

map.addLayer(CartoDBTiles);

The first part of the above code pulls the Carto tiles from their source and the second part adds the tiles to our map! Awseome, but dont we need to link this to our html? Yes! Which can be done by opening your html code and adding the following line directly under your “mapid” div element:

<script src="java.js"></script>

That code links our javascript to the html, lets check it out! View your map by opening your html code in your browser (double click on the index.html file).

Great! Now lets have some fun by adding something to our map! How about the location of one of our Zombies? We can do this by placing a “marker” on our map representing the zombies location. To do this we will need to add the following lines to our java.js file:

var marker = L.marker([40.719189, -73.996589]).addTo(map)

The latitude and longitude we used above are actually the location of a fantastic BahnMi spot where I assume this zombie likes to hang out! To view your marker simply reload your webpage! Now we can see where the zombie is, but how will others that view the map know its a zombie? Simple, we can add a pop-up! Directly under the las line of code you wrote in your java.js file, add the following lines:

marker.bindPopup("<b>G.Rider</b><br><b>Favorite Food:</b> Fresh Souls</br><b>Favorite Song:</b> Free Bird</br><b>Favorite Movie:</b> The Jerk</br><b>Favorite Hobby:</b> Poppin-Wheelies!").openPopup();

This code binds a pop-up to the marker with all of the info we have on hand for our zombie friend “G.Rider”. Refresh your webpage to check it out! Note that we use html style elements change the look of some of our words and content.

So what if you want to add multiple points? Great question, we can use something called a geoJson file! These files contain both information regarding your data (such as favorite film, etc) as well as the geographic data needed to plot it on a map (in this case latitude and longitude). They also allow us to add as many points or “features” as we would like. Lets make a small geojson file that contains the info needed for the zombie we already have, and a new zombie we will call “Bubbles”! First thing to do is open up your “myfirstgeojson.geojson” file. Then we want to add the following code:

var geojsonFeature = {

"type": "FeatureCollection",

"features": [

{"type": "Feature",

"properties": {

"Myname": "G.Rider",

"FavFood": "Fresh Souls",

"FavSong":"Free Bird",

"FavMovie":"The Jerk",

"FavHobby":"Poppin-Wheelies!"},

"geometry": {

"type": "Point",

"coordinates": [-73.996590, 40.719191]}

},

{"type": "Feature",

"properties": {

"Myname": "Bubbles",

"FavFood": "Ribs!",

"FavSong":"Welcome To The Jungle",

"FavMovie":"The Princess Bride",

"FavHobby":"Human Hunting"},

"geometry": {

"type": "Point",

"coordinates": [-73.996990, 40.719891]}

}

]

};

Notice the format of the geojson data, we give the type as a “feature” and then in that feature we can have as many “properties” as we like. In this case we just added all of the info from our Zombie profiles. After the properties we then provide the “geometry” which literally is the data needed to plot your data on the map!

Great! Now with your geojson file set up we are going to have to link it up to you html, Crack open your html code and add the following line directly under where you placed your “mapid” div element and above your link to your javascript:

<script type="text/javascript" src="myfirstgeojson.geojson"></script>

This line of code is telling our html where to find this data! Please note that order does matter here! If you put this line of code under your javascript link, then your map wont execute because it will try to run the code that generates the map before it reads in the data!

With our new geojson in place we now need to write some javascript to help “digest” the data for our map. Lets open up our javascript file and add the following code:

var featureClick = function (feature, layer) {

layer.bindPopup(

"<strong>Name:</strong> " + feature.properties.Myname + "<br /><strong>Favorite Food:</strong> " + feature.properties.FavFood + "<br /><strong>Favorite Song:</strong> " + feature.properties.FavSong + "<br /><strong>Favorite Movie:</strong> " + feature.properties.FavMovie + "<br /><strong>Favorite Hobby:</strong> " + feature.properties.FavHobby)

}

This code is letting us create a new variable which is a function that binds all of our zombie info (favorite food, etc) to a pop-up. If this is a bit hard to understand just take a quick look back at your geojson data file to see where everything is pulled from. OK, with that out of the way lets ad another line of code directly under that:

var myPoints = L.geoJson(geojsonFeature, {

onEachFeature: featureClick

}).addTo(map);

This code is the business end of things, we are creating a new variable (called myPoints) and telling it to pull in our geojson file (called geojsonFeature here). We also instruct that for every feature within our data we would like to apply the “featureClick” funciton we created in the last step. So boiling this all down, we are saying pull in our data set, and for every feature within it (two zombies in our case) apply the function that generates a pop-up containing all of our zombie info for that specific feature. Got it? Great! But before we go any further lets comment out our original markers so there is no interference with our new code. To do this simply add two back-slashes in front of each line:

//var marker2 = L.marker([40.719189, -73.996589]).addTo(map)

//marker.bindPopup("<b>G.Rider</b><br><b>Favorite Food:</b> Fresh Souls</br><b>Favorite Song:</b> Free Bird</br><b>Favorite Movie:</b> The Jerk</br><b>Favorite Hobby:</b> Poppin-Wheelies!").openPopup();

Lets check it out! Refresh your browser to see your handy work!

**Part II**

Now that we have mastered creating our own geojson data, lets move on to using some larger data sets that already exist! Perhaps we are interested in knowing if our Zombie friends are living in high poverty areas of NYC. Lucky for us this data already exists inside a larger data set of “Neighborhood” information for NYC. This file is also conveniently located in your folder under the name “NYC\_neighborhood\_data.geojson”. If you open up this file you can see that each feature is a a polygon representing a neighborhood, with many feature properties including things like unemployment (called UnempRate) and percentage of people in poverty (PovertyPer). With this data we can build a choropleth map where each neighborhood is color coded by the percentage of people in poverty. Lets begin!

First we need to crack open our javascript file again. We will begin by building a function that takes the percentage of people in poverty for each neighborhood and converts it into a corresponding color. Add the following lines to your javascript file:

var povertyStyle = function (feature){

var value = feature.properties.PovertyPer;

var fillColor = null;

if(value >= 0 && value <=0.1){

fillColor = "#fee5d9";

}

if(value >0.1 && value <=0.15){

fillColor = "#fcbba1";

}

if(value >0.15 && value<=0.2){

fillColor = "#fc9272";

}

if(value > 0.2 && value <=0.3){

fillColor = "#fb6a4a";

}

if(value > 0.3 && value <=0.4) {

fillColor = "#de2d26";

}

if(value > 0.4) {

fillColor = "#a50f15";

}

var style = {

weight: 1,

opacity: .1,

color: 'white',

fillOpacity: 0.75,

fillColor: fillColor

};

return style;

}

There is a lot going on in that code, but lets break it down. First we set a new variable (povertyStyle) and then point it to where our data on percentage poverty is located (feature.properties.PovertyPer). We then set a default color of “null”, followed by ranges of percentage poverty and the asscoaited colors we would like them to have. So for example if the percentage poverty of a neighborhood is less than 0.1, we would like the fill color to be #fee5d9. At the end we set the style for each polygon with the color range it is associated with, and we also set some other parameters like the fill opacity at 0.75, so they will be semi-transparent.

Great! We have our colors taken care of, what about the pop-up and other info? Great question! To do that we need to add the following lines below the code from above:

var povertyClick = function (feature, layer) {

var percent = feature.properties.PovertyPer \* 100;

percent = percent.toFixed(0);

// let's bind some feature properties to a pop up

layer.bindPopup("<strong>Neighborhood:</strong> " + feature.properties.NYC\_NEIG + "<br /><strong>Percent in Poverty: </strong>" + percent + "%");

}

That code creates a new variable (povertyClick) and using the feature and the layer as the input takes in the raw percentage of poverty and multiplies it by 100 (to scale from 0-100%) for readability (folks like to see percentages from 0-100 not .00-1). Then we simply add the info we want in the pop-up including our new “percent” within our “layer.bindPopup”.

With all that out of the way we now need some code to execute it! Add these lines below what you already have in you javascript file:

var neighborhoodsGeoJSON = L.geoJson(neighborhoods, {

style: povertyStyle,

onEachFeature: povertyClick

}).addTo(map);

That code creates a new variable (neighborhoodsGeoJSON) which executes the color coding function (povertyStyle) and the pop-up function (povertyClick) that we made in the previous steps. Awesome! Now all we need to do is link it up! Crack open your html code and add the following line under your “mapid” div element:

<script type="text/javascript" src="NYC\_neighborhood\_data.geojson"></script>

Like our previous link this code is letting our code know where to find our sweet geojson data! Remember that order matters so make sure this code was not put below your javascript call!!

Great work, refresh your browser to view your new map!!

So whats next? How about layer control! As you make more complex maps with more layers, you may want the ability to turn the layers on and off. To do this crack open your javascript file and add the following lines:

var baseMaps = {

"CartoDB": CartoDBTiles,

};

var overlayMaps = {

"Zombie Map": myPoints,

"Poverty Map": neighborhoodsGeoJSON

};

Here we have made two new variables, one of them is our basemap, which is just the Carto tiles, the second is our overlays, myPoints and neighborhoodsGeoJSON. We gave new names to the over lays which will be more appealing to users (“Zombie Map” and ”Poverty Map”). Finally we need to add one last line of code here:

L.control.layers(baseMaps, overlayMaps).addTo(map);

Which gives us layer control over our two overlays! Refresh your browser to see the results!!