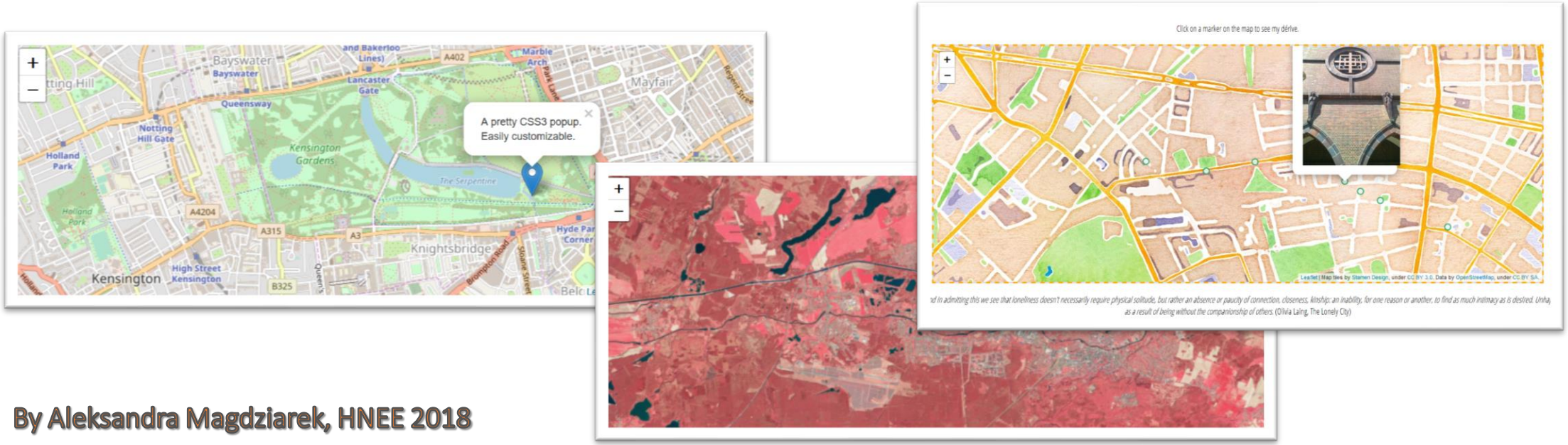
Creating online maps using

Leaflet

An open source JavaScript library

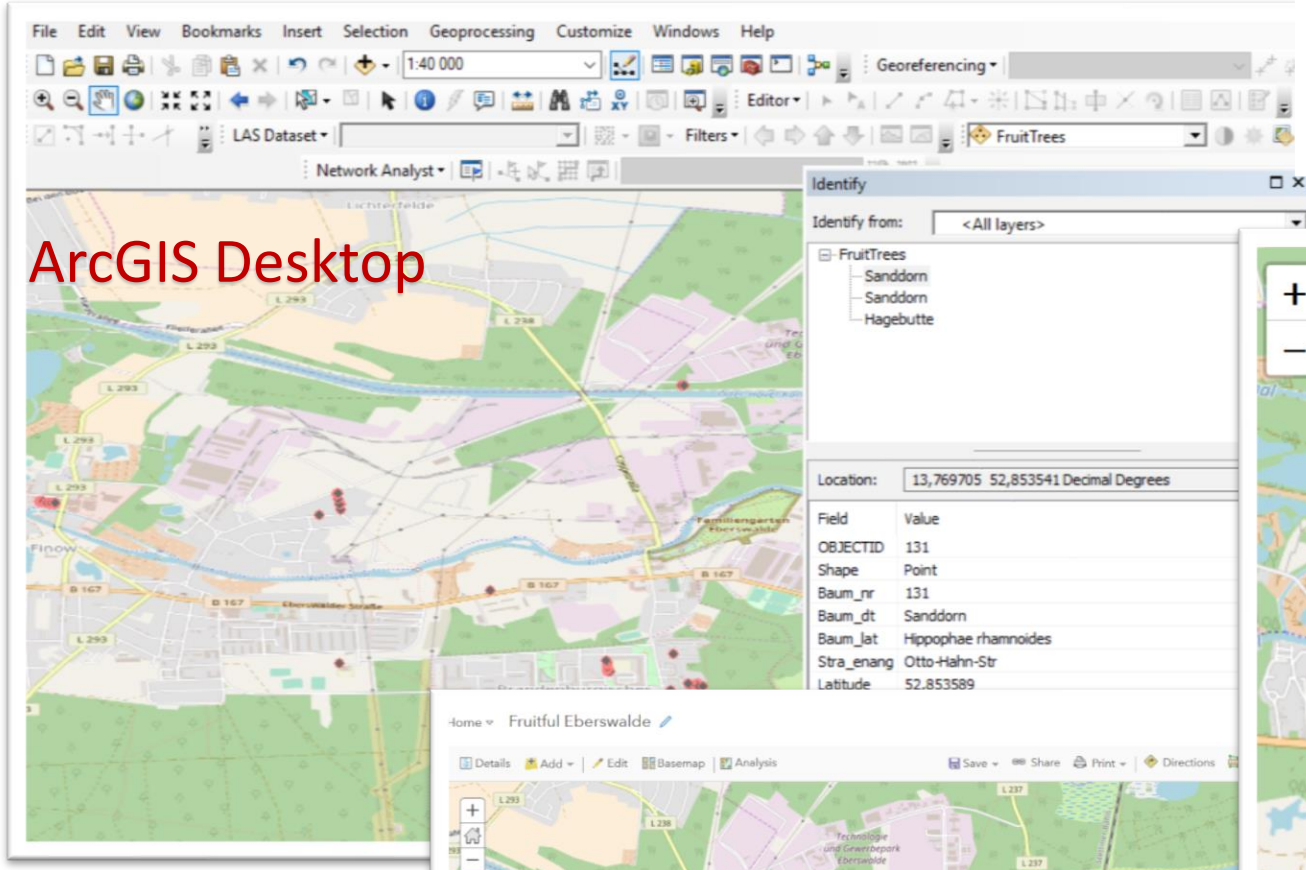


By Aleksandra Magdziarek, HNEE 2018

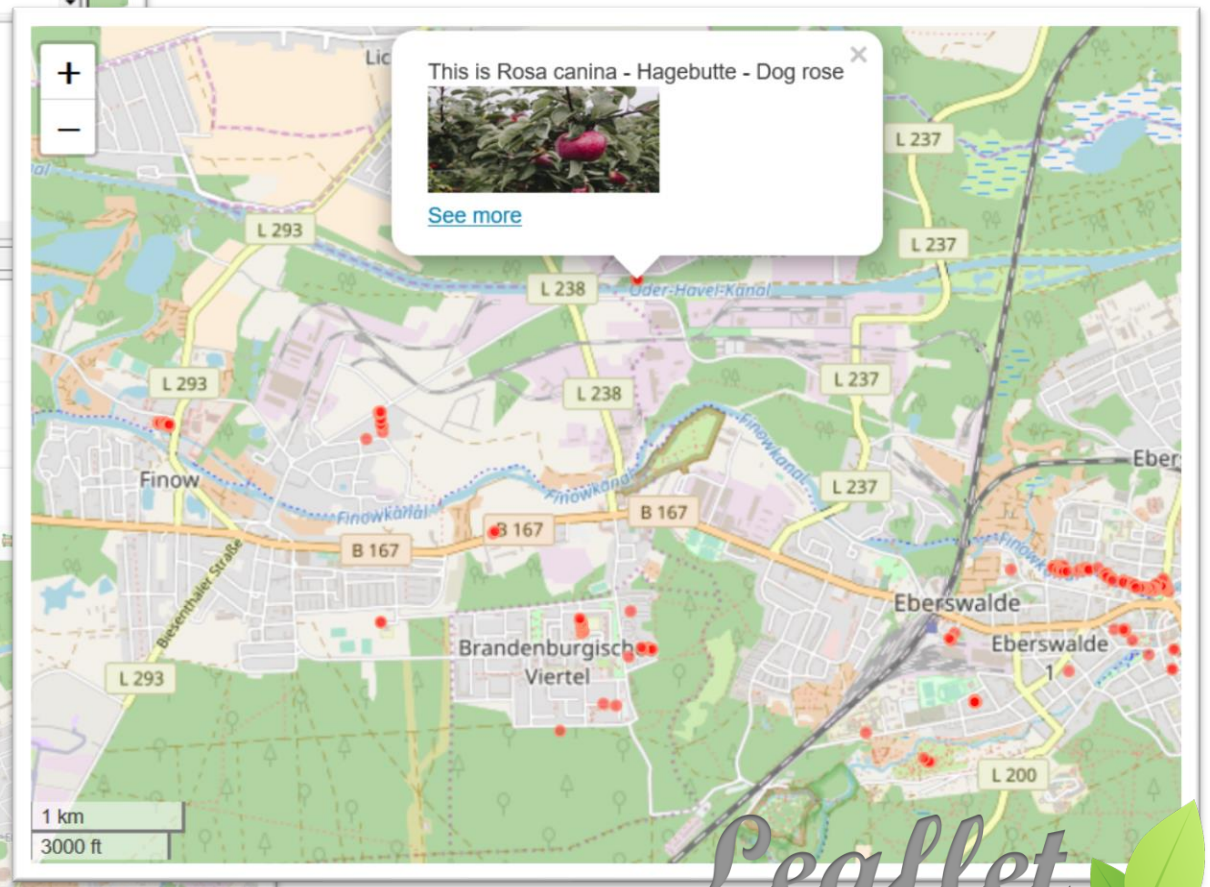
Introduction

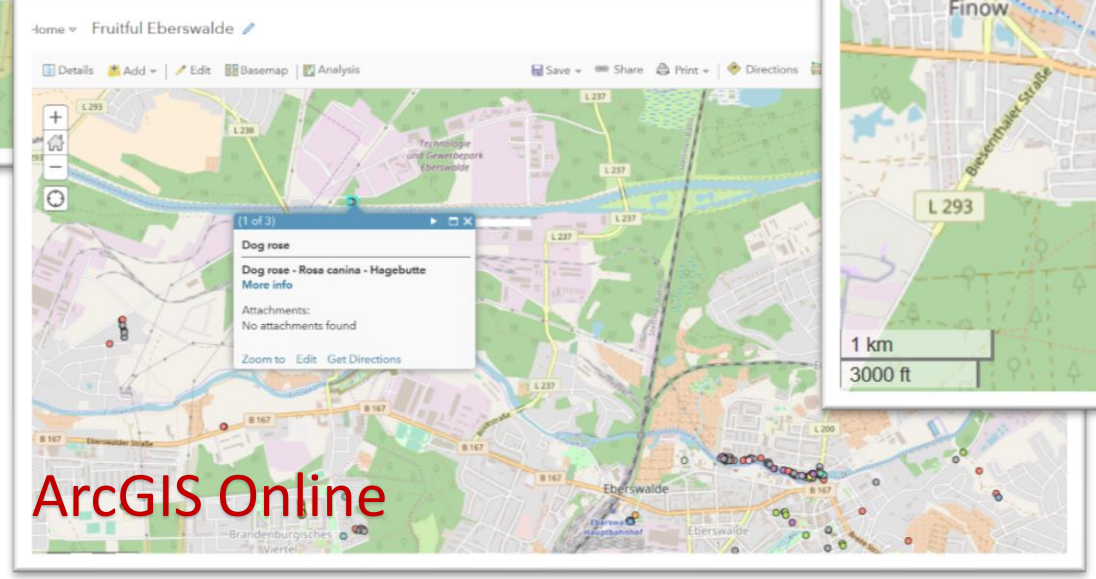
- The first part of this tutorial will take about 30 minutes and will give you an idea of what Leaflet is and how to make a simple map. This is rather light and easy.
- The second part will allow you to replicate a map you've made in desktop GIS but in a programmatical way. Basic knowledge of Java Script and HTML would be helpful but is not necessary.
- You can work with your own geospatial data and copy/ type in code or download the zip folder from here: https://github.com/OlaMag/Leaflet_exercise_Eberswalde

This tutorial will show you how to built a map like this one:



ArcGIS Desktop





ArcGIS Online

ArcGIS ESRI

Leaflet

Why use Leaflet?

- You can create maps for your own use on your desktop computer using GIS software, you can share them online using, eg. the Esri environment or you can do it **programmatically** and **share the content on a website** without any payment or copyright issues.
- There are a few other open source map libraries, eg. Open Layers, but Leaflet seems to have gained traction in recent years as it's very well documented, easy to use and lightweight.

Part 1

How to build a basic map with scale, zoom in function and a point marker

Overview of structure

HTML – a skeleton of your application which you will put things on, put it in a generalized way: it decides about the order elements are displayed. It will be at the top of the script and it uses tags like <head> and <body>

CSS – the style of your application, eg. color, size, font, position of elements. Look for it between <style> tags. Here, for simplicity, we will use inline styling in HTML instead.

JavaScript – gives the application functionality. It will appear between the <script> tags.

Let’s get to work!

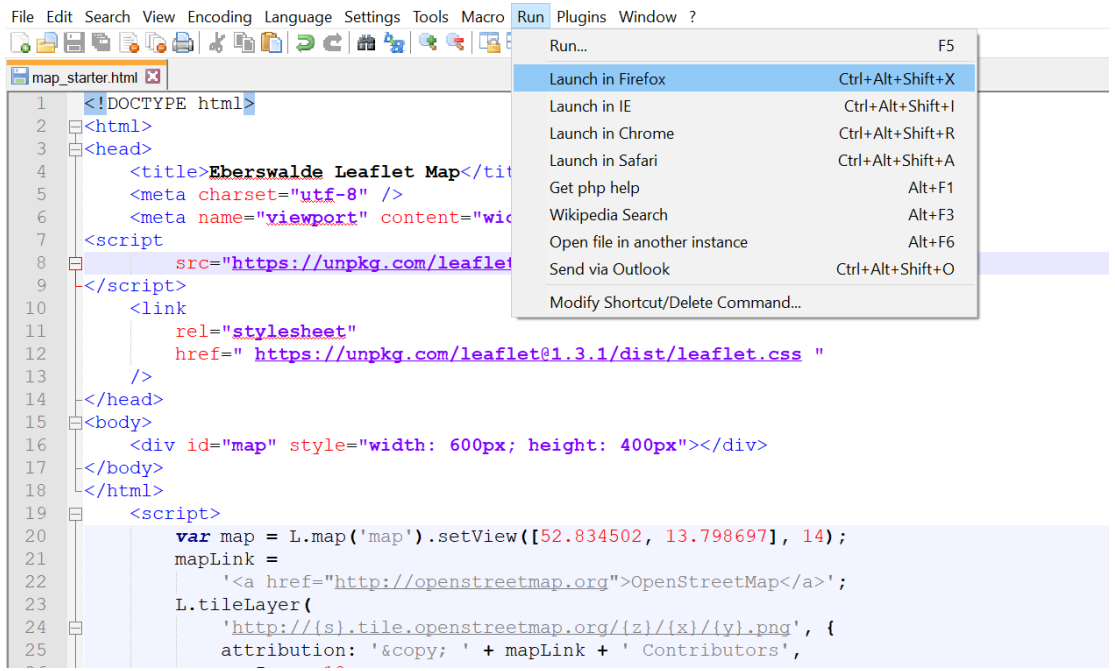
Type in the below code into an editor, eg. Notepad++ or open provided file (map_starter.html) in an editor (copying and pasting the below is risky)

```
<!DOCTYPE html>
<html>
<head>
   <title>Eberswalde Leaflet Map</title>
   <meta charset="utf-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
<script
      src="https://unpkg.com/leaflet@1.3.1/dist/leaflet.js">
</script>
   <link
      rel="stylesheet"
      href=" https://unpkg.com/leaflet@1.3.1/dist/leaflet.css "
   />
</head>
<body>
   <div id="map" style="width: 80vw; height: 80vh"></div>
</body>
</html>
   <script>
      var map = L.map('map').setView([52.834502, 13.798697], 14);
      mapLink =
         '<a href="http://openstreetmap.org">OpenStreetMap</a>';
      L.tileLayer(
         'http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
         attribution: '&copy; ' + mapLink + ' Contributors',
         maxZoom: 18,
         }).addTo(map);
                                    L.control.scale().addTo(map);
   </script>
```

Don’t change anything just now, the order and every character matters here!

Test!

- Double click on the map_starter.html file or run from your code editor (eg. Notepad++)
- A map should open in a browser.



- It’s responsive which means if you change the size of the browser window or display it on your smartphone the map size will change too.

What does it all mean?

Values in green are arbitrary (defined by you)

INTRODUCING HTML:

<!DOCTYPE html>

<html>

<head> <- what is in the head won't be directly seen by the user.

<title>Eberswalde Leaflet Map</title> <- this will be displayed on the tab

<meta charset="utf-8" /> <- a widely used character encoding standard

<meta name="viewport" content="width=device-width, initial-scale=1.0"> <-helps you to make your application responsive to different device sizes

<script src="https://unpkg.com/leaflet@1.3.1/dist/leaflet.js"></script><-this is where the JS code responsible for Leaflet running lives. It has other addresses too, it doesn't matter which one you use but rather use https!

<link rel="stylesheet"

href=" https://unpkg.com/leaflet@1.3.1/dist/leaflet.css "

/> <- this is where the code that determines leaflet style is (see? CSS)

</head> <-this is where head ends, its closing tag, note the forward slash

<body> <- the body determines the order and existence of application's elements

 <- this is where in the skeleton the map goes. Leaflet requires defining the width and height. This here is inline styling – ie. defining style directly in the HTML tag rather than in CSS. The 'vw' and 'vh' are size units that size elements to be relative to the size of the window

</body> <-the end of the body

</html> <-the end of HTML

Now on to the JavaScript part!

INTRODUCING JAVASCRIPT:

<script> <-this is where the instructions start

var map1 = L.map('map2') <- declare a variable with any name, here the name is map1. 'L' is for Leaflet, the library we are using. In this library there is an object called map (L.map) so we are telling the engine to put the map object where the HTML element that has the id='map2' is. People usually call all three simply 'map' but I added the numbers so you can see what it refers to. The only one that cannot be changed is the 'map' in L.map. It's a bit like dataframe in ArcMap, you just add layers and features to it.

.setView([52.834502, 13.798697], 14); <-when the map loads make the central point these coordinates (it's important to use this format [{lat}, {long}]) and the zoom should be 14.

var mapLink =

'OpenStreetMap'; <-declare a variable that hold the link to the map we want to use, eg. OSM

L.tileLayer(<- in the Leaflet library find the Tile Layer

'http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', { <-get the tiles from this address

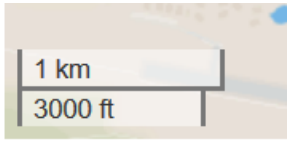
attribution: '© ' + mapLink + ' Contributors', <-give credit to the people who created the tiles. The word 'attribution' is part of Leaflet library, it will not appear on the map but it tells the engine where to place the credits. It makes this strip on the bottom right

maxZoom: 18, <-zoom only to this level, no further

}).addTo(map); <-add the tile to the map. Note each separate line of code ends with a semicolon

L.control.scale().addTo(map); <- amongst controls in the Leaflet library find Scale and add it to the map

</script> <-end of script

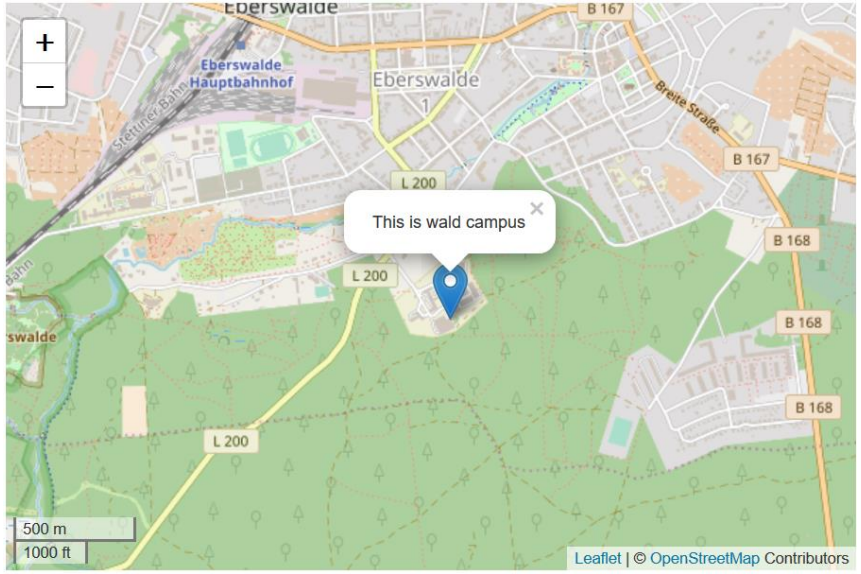


Put a marker on the map

Between the last two lines of code, paste code for a marker:

```
L.control.scale().addTo(map);
L.marker([52.823608, 13.809929])
        .addTo(map)
        .bindPopup(`This is wald campus`)
        .openPopup();
    </script>
```

Can you guess what each line does?



This is not the end of this tutorial but now you can play with the code and see what it does, eg change the green values. After each change save the html file and reopen in the browser to see the result. If nothing displays undo the change and try to fix it. Happy coding!

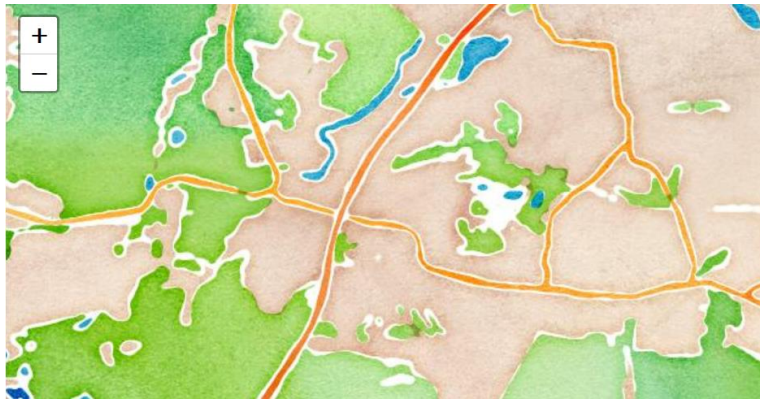
Feeling daunted?

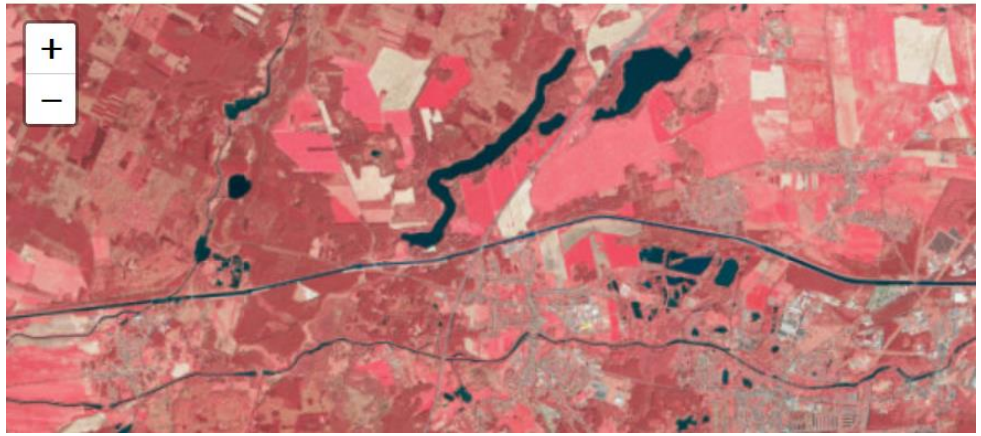
Look for ideas on the next page or check

https://leafletjs.com/

Ideas

- Find your hometown coordinates (go to: https://mynasadata.larc.nasa.gov/latitudelongitude-finder) and paste them into the code to change the focus of the map
- Get creative! Use a beautiful watercolour map from Stamen Design as a tile instead of OSM: http://tile.stamen.com/watercolor/{z}/{x}/{y}.jpg
  Remember to change the attribution.
- Serious science: use a web map service and get an infrared image or digital model to put on your leaflet map





hint: instead of the L.tileLayer(…) type:

```
L.tileLayer.wms("http://isk.geobasis-bb.de/ows/dop20cir_wms", {
layers: "bb_dop20cir_3m",
attribution: "WebAtlasDE BE/BB fix"}).addTo(map);
```

If you get stuck scroll to the end for answers

You might be wondering…

- **How do these few words in our script produce a web mapping application?**

We are standing on the shoulders of giants – the scripts we referenced in HMTL are long lines of code that the creator of Leaflet, Vladimir Agafonkin, and other contributors have written. So we are only using a subset of it and telling the engine where to get the rest from. You can view the full code for leaflet on the project's github page

Part 2

How to add a feature layer: convert a dataset into geoJSON.

This will show you how to work with your own dataset but alternatively you can use files from: https://github.com/OlaMag/Leaflet_exercise_Eberswalde.
If you are using provided data, use map_starter.html for a basemap or if you are using your own data, remember to change the initial location accordingly.

Create a map with a feature layer

In this section you will need a point feature layer. Leaflet doesn't display just any exchange formats that desktop GIS uses. If your point data is in a shapefile or a csv file, you will have to convert it to a format called **geoJSON**. JSON stands for JavaScript Object Notation and has been used on the web for storing data separately from code. Add geocoordinates to it and you get a geoJSON. Here's an example:

```
{
   "type": "Feature",
   "properties": {
      "name": „HNEE Forest Campus",
      "popupContent": "This is where we study!"
   },
   "geometry": {
      "type": "Point",
      "coordinates": [13.807482, 52.825023]
   }
}
```

Here is the tricky part: notice the order!
Leaflet uses [latitude, longitude] format, however geoJSON reverses it to [longitude, latitude]. When you use a convertion tool, it does the reversing for you.

Now you have to make a decision

Option 1: Use your own data locally

- if you want to work with your own data and you have nowhere to store your file online.

- Go to the next page

Option 2: Use GeoJSON online

- if you are working with provided data online or are hosting your own dataset somewhere online
- To host data online you can set up a github account

- Go to page 24

Option 1: Using your own data

Wherever you are storing your data, convert it to csv format. If it's a .csv file with geocoordinates, you're ready. For a shapefile:

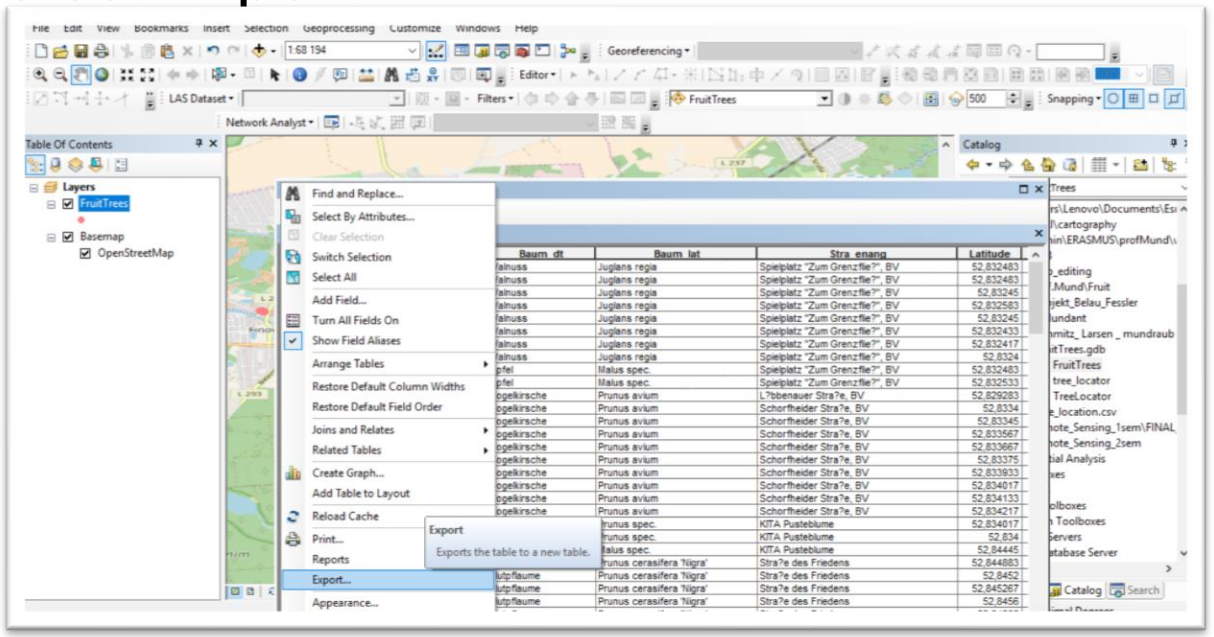
1. Open the file in eg. ArcMap, open its attribute table. Does it have coordinates?

If not, use the tool 'Add XY coordinates' and save the table. If yes, then go to Table Options and from the menu select Export.

2. Save the table as

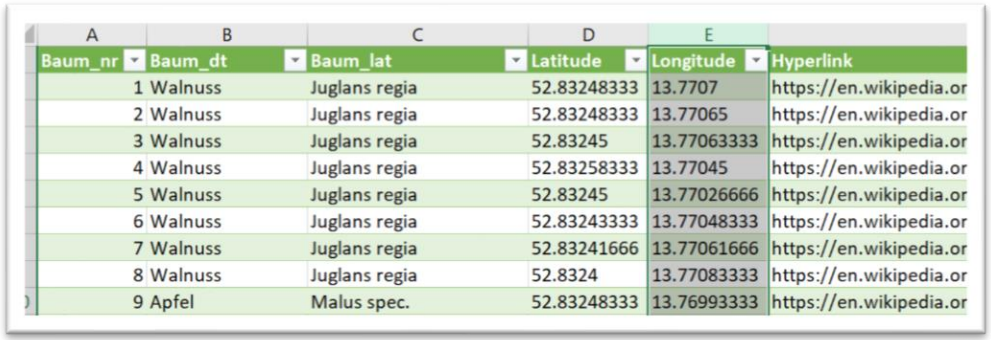
a **text** file in your

directory



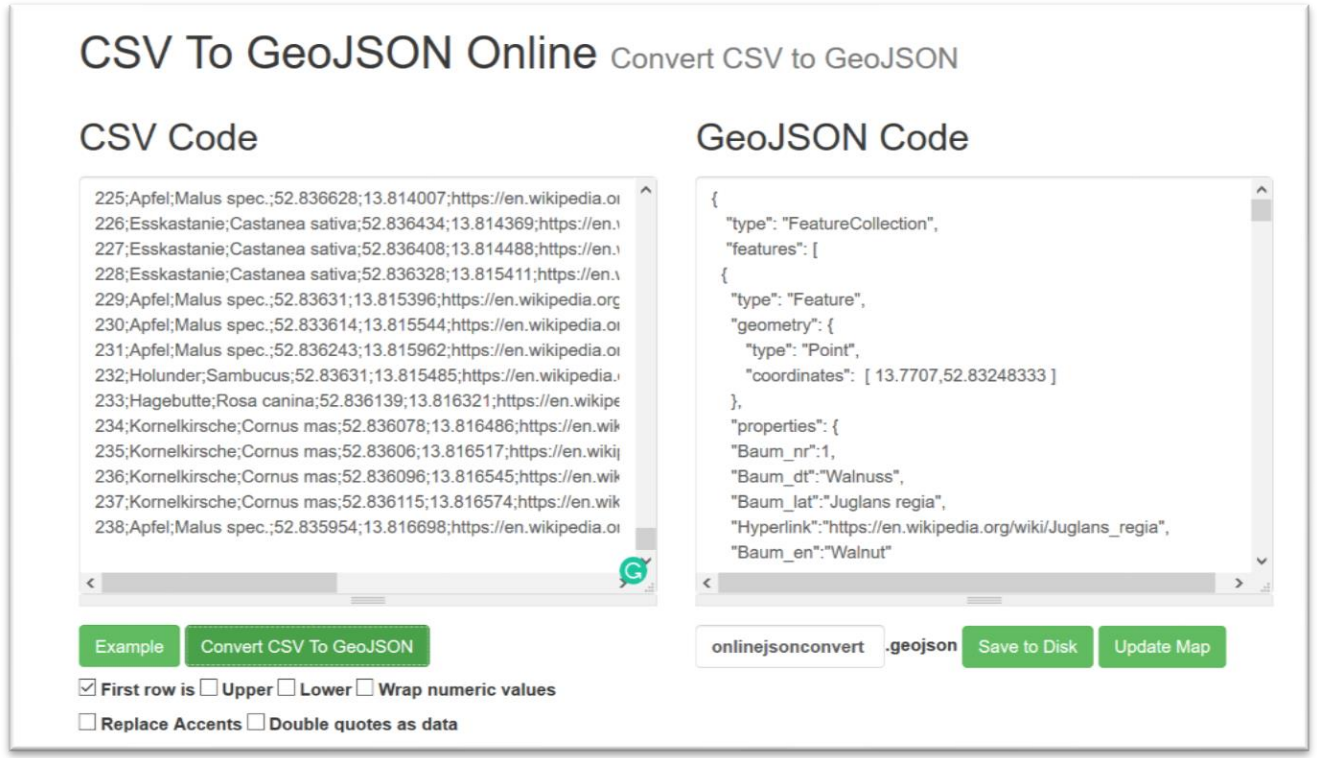
3. If your table is well managed and doesn’t have any redundant data skip this step. If not:

- Open an empty Excel sheet, go to DATA tab and click on ‘From text file/ csv’. Import the text file you created in the previous step. Remove unnecessary columns. Give the columns that contain latitude and longitude explicit names. Make sure dots are used for decimal notation, not commas.
- It should look like this
- Save the file as **.csv**



	A	B	C	D	E	
	Baum_nr	Baum_dt	Baum_lat	Latitude	Longitude	Hyperlink
	1	Walnuss	Juglans regia	52.83248333	13.7707	https://en.wikipedia.or
	2	Walnuss	Juglans regia	52.83248333	13.77065	https://en.wikipedia.or
	3	Walnuss	Juglans regia	52.83245	13.77063333	https://en.wikipedia.or
	4	Walnuss	Juglans regia	52.83258333	13.77045	https://en.wikipedia.or
	5	Walnuss	Juglans regia	52.83245	13.77026666	https://en.wikipedia.or
	6	Walnuss	Juglans regia	52.83243333	13.77048333	https://en.wikipedia.or
	7	Walnuss	Juglans regia	52.83241666	13.77061666	https://en.wikipedia.or
	8	Walnuss	Juglans regia	52.8324	13.77083333	https://en.wikipedia.or
	9	Apfel	Malus spec.	52.83248333	13.76993333	https://en.wikipedia.or

- Go to: https://www.onlinejsonconvert.com/csv-geojson.php or http://convertcsv.com/csv-to-geojson.htm
- Copy the data from your csv file and paste it into the text viewer on one of the convertion websites
- Click convert button
- Copy geoJSON code or save to disc as myGeoJSON.geojson.



GeoJSON stored locally

if you have nowhere to store your file online.

1. Open your myGeoJSON.geojson in a code editor, eg. Notepad++.
2. In the first line of code declare a name for the geoJSON. Eg.

var fruitTrees = //var stands for ‘variable’, the name fruitTrees is a name I have given to my geoJSON. Do the same.

```
 {
  "type": "FeatureCollection",
  "features": [
 {
   "type": "Feature",
…
```

3. After the very last line and the last curly bracket enter a semicolon so it should looks something like this:

```
var nameOfVariable = {
    "type": "Feature",
    "properties": {
        "name": „HNEE Forest Campus",
        "popupContent": "This is where we study!"
    },
    "geometry": {
        "type": "Point",
        "coordinates": [13.807482, 52.825023]
    }
// you probably have more data here but scroll down to the very end
};
```

Add the green part

Add the green part, a semicolon

4. Go to File > Save As. This time we need to save it as a .js file (not like the original geoJSON). Call it myGeoJson.js and **save it in the same folder** as your .html file with your map is.

5. Open your map_starter.html file with your map in a code editor. Remember all the scripts for leafletJS that we referenced in the head? Now we also want to reference our geoJSON. In the head enter the line:

```
</script>
<script src='./myGeoJson.js'></script>
 <link rel="stylesheet" ...
```

This is a relative pathway, the './' means the js file is in the same folder as the html file in which we are referencing it. Make sure it really is! Otherwise the script won't be found.

6. Now you can use your dataset in the .html file by just referencing its name! (eg. fruitTrees or nameOfVariable)

You can skip option 2 now.

Option 2: GeoJSON online

if you are working with my data online or are hosting your data somewhere online

- If you want to use a geoJSON for an online application, you need to publish it online. You can do this for example on github.
- If your geoJSON is online, type in or paste this code snippet between the last two lines of the html file. Change the green parts, leave the rest the same

```
L.control.scale().addTo(map);
var url = "https://raw.githubusercontent.com/OlaMag/Geodata/master/fruitTreesEberswalde.geojson"; //paste the URL to your data source
var xmlhttp = new XMLHttpRequest(); // keep the following lines the same
xmlhttp.open('GET', url, false);
xmlhttp.send();
var data = xmlhttp.responseText;
var fruitTrees = JSON.parse(data); //instead of fruitTrees type the name you want to give to your data. Keep the rest the same

</script>
```

Pause for a moment…

(or skip this and next page if you're in a hurry)

Look at both the csv file and the geoJSON.

- How do you explain to eg. Microsoft Excel where the value you are looking for is?
- How do you think you can explain to JavaScript where the value is in a geoJSON?

Accessing values in objects & arrays

- With Excel it's like playing battleships: H2, A5, etc are cells that have 'an address'. In each one there's a value of one data type, eg. text (string) or number (integer), etc.
- GeoJSON is a JavaScript object and it uses a particular indexing system. The column names are repeated in every record. Eg. my csv file had 239 lines, the same dataset as geoJSON has 3337 lines!
- To understand how this really works you need to read up on objects and arrays and indexing them. In short: we will be saying to the engine: fetch the geoJSON, from it get record number X and inside it get coordinates.

Render points on the map

- Now in both cases (option 1 and 2) you have a reference to your geoJSON and now it is stored in a variable, eg. called by me fruitTrees. Use the following code before the final closing tag for the script and after the variable declaration:

```
for (let i=0; i < fruitTrees.features.length; i++){  //enter the name of the variable you declared above
        tree = L.marker([fruitTrees.features[i].geometry.coordinates[1], fruitTrees.features[i].geometry.coordinates[0]]) //the same name as above, type anything instead of tree
        .addTo(map);
        };
```

Save and run the html file in your browser! Are the points displayed? If yes, carry on, if not, go back and try doing it again.

What does it mean?

for (let i=0; i < fruitTrees.features.length; i++){ <- this is a ‘for loop’. It means we will be iterating through every element in our fruitTrees.features array by giving each point a number from 0 to the index of the last point - 1

tree = L.marker([fruitTrees.features[i].geometry.coordinates[1], fruitTrees.features[i].geometry.coordinates[0]]) <- for each index number we will print a marker on the map using the coordinates contained in our variable fruitTrees. This is this complex indexing mentioned earlier!

.addTo(map); <- add the markers to the map

}; <- close the loop and mark the end of the line with a semicolon

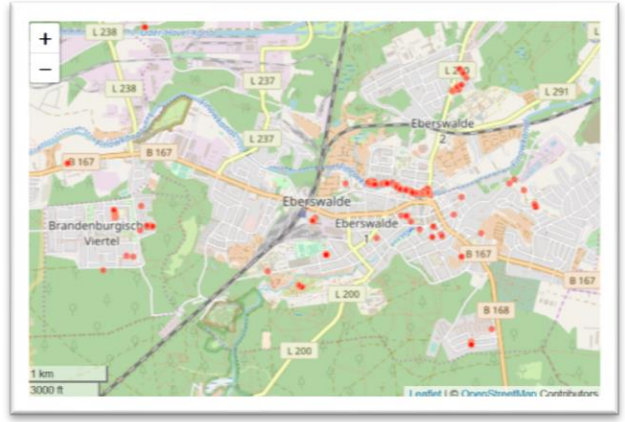
At this point the map should look like this:

Want to style your points?

Before the for loop enter this snippet that defines the style of the points:

```
var style = {
    radius: 3,
    fillColor: “red",
    color: "#fc9272",
    weight: 1,
    opacity: 1,
    fillOpacity: 0.8,
};
```

Once you see this works, try to change the values
In green, on the right side of the colon. Save and run again to see the changes.



Edit the for loop, just add one word ‘style’ representing the above variable:

```
for (let i=0; i < fruitTrees.features.length; i++){
tree = L.circleMarker([fruitTrees.features[i].geometry.coordinates[1], fruitTrees.features[i].geometry.coordinates[0]], style)
.addTo(map)
};
```

Want to add pop-ups?

Adding a simple pop-up is easy. Just add:

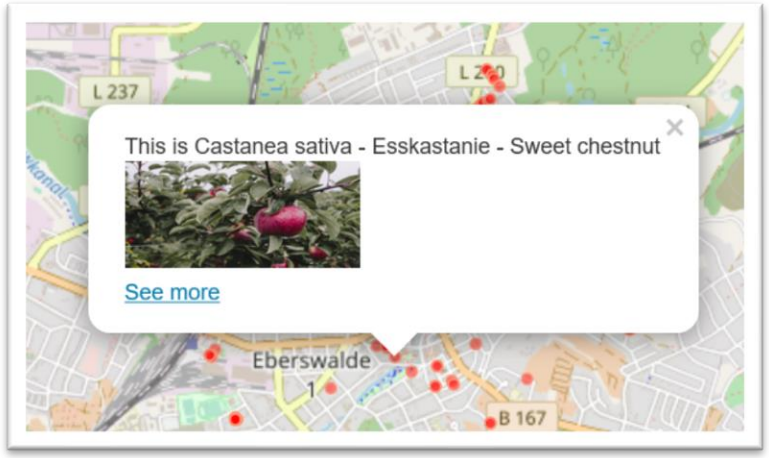
.bindPopup('text here') after .addTo(map), eg.:

```
for (let i=0; i < fruitTrees.features.length; i++){
                    tree = L.circleMarker([fruitTrees.features[i].geometry.coordinates[1], fruitTrees.features[i].geometry.coordinates[0]], style)
                    .addTo(map)
                    .bindPopup('text here');
};
```


What's the difference between this pop-up and the one from part 1? How do they open?

If you want a unique pop-up for each point you should know how to index a JS object and know a bit about HTML and CSS.

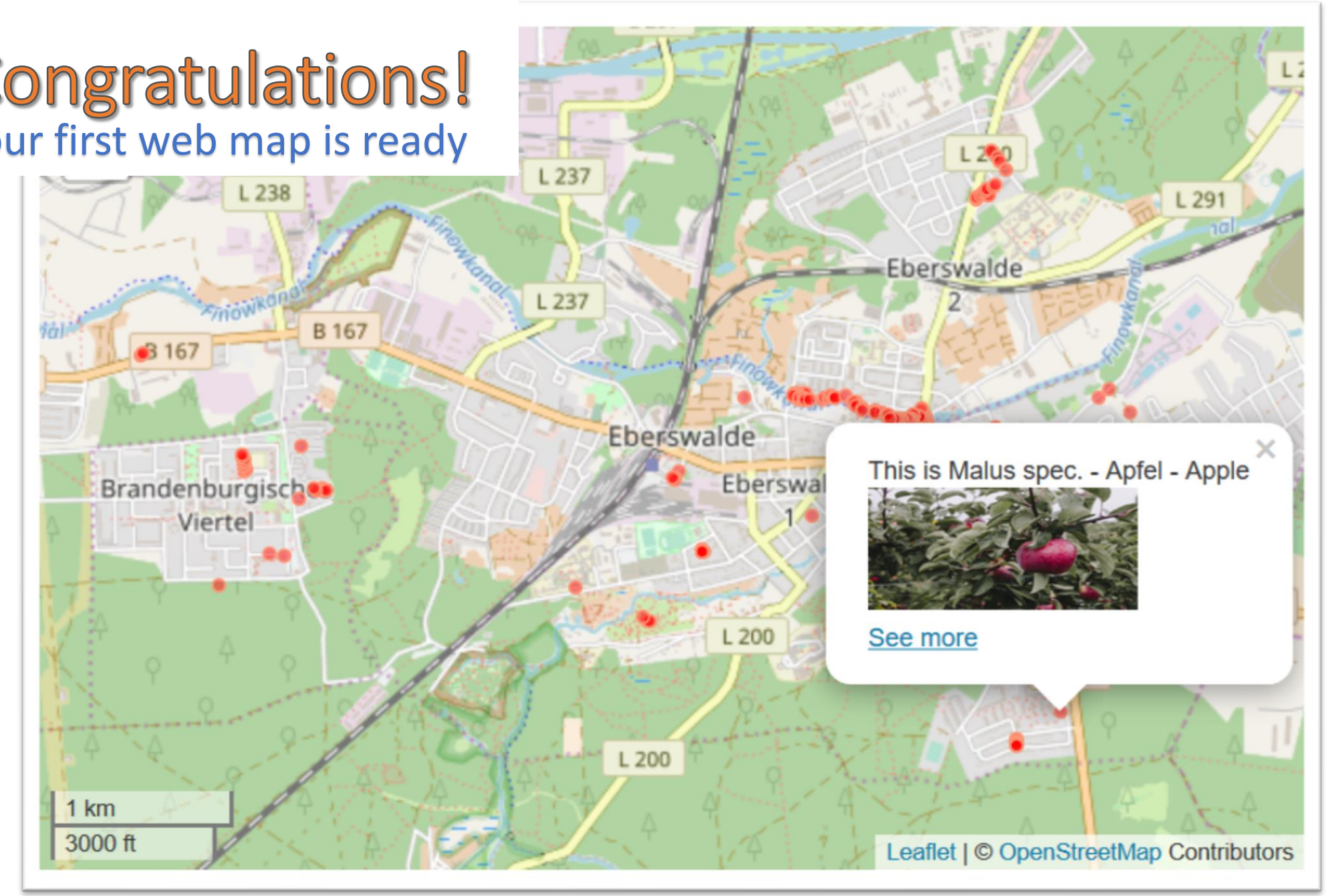
Here is an example where the pop-up will display a set of names in different languages for each point, and a clickable hyperlink to a wikipedia entry and a photo. Look at my data for fruitTrees and try to figure out how the indexing is done.




```
.bindPopup(`This is ${fruitTrees.features[i].properties.Baum_lat} - ${fruitTrees.features[i].properties.Baum_dt} - ${fruitTrees.features[i].properties.Baum_en}
                    <br><img src="https://raw.githubusercontent.com/OlaMag/Leaflet_exercise_Eberswalde/master/p2_pop-up_image.jpg"
                    alt="image unavailable" style="width: 10vw; height: 10vh"></img>
                    <br><a href=${fruitTrees.features[i].properties.Hyperlink}>See more</a>`);
```

Congratulations!

Your first web map is ready



Leaflet Plugins

- Leaflet is a more powerful tool than it looks on the surface and it is all made possible by open source contributors. You can draw spatial features, calculate walking and driving routes, make heatmaps, visualize statistical data and many more.
- Check https://leafletjs.com/plugins.html for a full list.

Accessing values in arrays & objects

- https://www.w3schools.com/js/js_arrays.asp
- https://www.w3schools.com/js/js_objects.asp

Other resources:

- The most helpful and comprehensive publication is ‘Leaflet Tips and Tricks’ available for free on Leanpub: https://leanpub.com/leaflet-tips-and-tricks
- To learn more about HTML, CSS and JavaScript join freeCodeCamp, an excellent community for learning to code for all levels: https://freecodecamp.org
- More on JavaScript: theory explained in ‘You Don’t Know JavaScript’ by Kyle Simpson, available on github for free: https://github.com/getify/You-Dont-Know-JS
- Eloquent JS including example code and projects: http://eloquentjavascript.net/

All the above are freely available but please remember a lot of work has gone and still goes into them so consider donating!

Source of images

- <head></body> tattoo from: http://www.tattoostime.com/head-body-geek-tattoos-on-nape/
- Lightbulb icon: Freepik from https://www.flaticon.com, licence: CC 3.0 BY
- Apple tree in the pop-up in the final map from unsplash.com by Kelly Sikkema

Examples part 1

```
<!DOCTYPE html>
<html>
<head>
   <title>Eberswalde Leaflet Map</title>
   <meta charset="utf-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
<script

src="https://unpkg.com/leaflet@1.3.1/dist/leaflet.js">
</script>
   <link
      rel="stylesheet"
      href="
https://unpkg.com/leaflet@1.3.1/dist/leaflet.css "
   />
</head>
<body>
   <div id="map" style="width: 600px; height: 400px"></div>
</body>
</html>
   <script>
      var map = L.map('map').setView([52.834502, 13.798697], 14);
  L.tileLayer.wms("http://isk.geobasis-bb.de/ows/dop20cir_wms", {
   layers: "bb_dop20cir_3m",
   attribution: "WebAtlasDE BE/BB fix"}).addTo(map);
   L.control.scale().addTo(map);
   </script>
```

This is where you change the coordinates to
Change the focus of the map

Note this is a local service,
if you change the coordinates to a
place outside of Brandenburg
this particular service won't work

Examples part 1 cont.

```
<!DOCTYPE html>
<html>
<head>
   <title>Eberswalde Leaflet Map</title>
   <meta charset="utf-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
<script
      src="https://unpkg.com/leaflet@1.3.1/dist/leaflet.js">
</script>
   <link
      rel="stylesheet"
      href=" https://unpkg.com/leaflet@1.3.1/dist/leaflet.css "
   />
</head>
<body>
   <div id="map" style="width: 600px; height: 400px"></div>
</body>
</html>
   <script>
      var map = L.map('map').setView([52.834502, 13.798697], 14);
      mapLink =
         '<a href="https://stamen.com/">Stamen Design</a>';
      L.tileLayer('http://tile.stamen.com/watercolor/{z}/{x}/{y}.jpg',
{
         attribution: '&copy; ' + mapLink
         }).addTo(map);
 L.control.scale().addTo(map);
   </script>
```