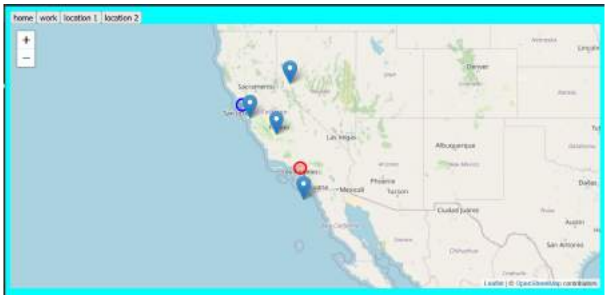
# Here’s GeoJSONny and UX

Adding functionality to our Leaflet.js map!



**Goals**

- Create a GeoJSON online and it to the map with JavaScript
- Understand how JavaScript works with HTML and CSS
- Understand how JavaScript variables, functions, methods work together

## Starting Template Code for lab #3

Use the following template code or your lab assignment #2:

**2 Labs, 2 day!**

Because we will be doing 2 labs today, we will making two copies of this code. You can save yourself time by doing the following:

- Make a copy of `index.html` as `part1.html`
- Make a copy of `js/init.js` to `js/part1.js`

**index.html**

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <title>Hello World</title>
        <!-- hint: remember to change your page title! -->
        <meta charset="utf-8" />
        <link rel="shortcut icon" href="#">
        <link rel="stylesheet" href="styles/style.css">

        <!-- Leaflet's css-->
        <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />

        <!-- Leaflet's JavaScript-->
        <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js">
</script>
    </head>

    <body>
        <header>
            <!-- space for a menu -->
        </header>

        <div class="main">
            <div id="contents">
                <!-- page contents can go here -->
            </div>
            <div id="the_map"></div>
        </div>
        <div id="footer">
            Copyright(2023)
        </div>
        <script src="js/init.js"></script>
    </body>
</html>
```

**styles/style.css**

```
body{
    display: grid;
    /* grid-template-columns: 1fr;  */
    grid-auto-rows: auto 1fr;
    grid-template-areas: "header" "main_content" "footer";
    background-color: aqua;
    /* height: 100vh; */
}

header{
    grid-area: header;
}

#footer{
    grid-area: footer;
}
```

```

.main{
    grid-area: main_content;
    grid-template-areas: "content" "main_map";
    display: grid;
}

#contents{
    grid-area: content;
}

#the_map{
    height:80vh;
    grid-area: main_map;
}
```

**js/init.js**

```
// declare the map
const map = L.map('the_map').setView([34.0709,-118.444], 5);

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);

addMarker(37,-122,'home','home land!')
addMarker(32,-118,'work','where i work land!')
addMarker(39,-119,'location 1','random location')
addMarker(36,-120,'location 2','another random location')

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}
```

Last update: 2023-04-17

# Part 1: Functions and the DOM

## The HTM-Elements: Avatag the last Airbender

Remember, when you see tags in HTML, like `<body></body>`, they are referred to as elements, so for example:

```
<water>Katara</water>
<air>Aang</air>
<earth>Toph</earth>
<fire>Zuko</fire>
```

Above we have four elements. Each element has a `content`, for example, the `earth` element's content is `Toph`. Unfortunately, despite how exciting those elements are, the most common HTML element is the `<div></div>` element, which is a generic container.

The `DOM` (Document Object Model) is basically where HTML elements exists and it has an API (Application Programming Interface) that JavaScript can interact by using functions.

## Making JavaScript interact with HTML-ements!

**Objective**

Make a button that we can click on to fly to a location for each of the markers you made.

1. Add a new function to our `addMarker` function
2. Create the function to add buttons
3. Add a function to move the map

**Creating elements?!**

To create HTML elements with JavaScript you need to use the createElement method.

## Create the function to add buttons

Next we will add our new function. Notice how we are using the `lat`, `lng`,and `title` from the `addMarker` function? That’s why it was helpful to do step one first.

**js/init.js**

```
function createButtons(lat,lng,title){
    const newButton = document.createElement("button"); ①
    newButton.id = "button"+title; ②
    newButton.innerHTML = title; ③
    newButton.setAttribute("lat",lat); ④
    newButton.setAttribute("lng",lng); ⑤
    newButton.addEventListener('click', function(){
        map.flyTo([lat,lng]); ⑥
    })
    document.getElementById("contents").appendChild(newButton); ⑦
}
```

1. Creates a new button `element`
2. Gives the button a unique `id`
3. Gives the button a `title`
4. Sets the `latitude`
5. Sets the `longitude`
6. Tells Leaflet where to `flyTo()`, which is the latitude/longitude
7. This targets the `id` where the buttons should be added to! In this case it is the div with the id `contents`!

## Call the `createButtons()` in our `addMarker` function

Remember, the only way functions work is if they are called, so the last step is to call the `createButtons()` in our `addMarker()` function.

**js/init.js**

```
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2> <h3>${message}</h3>`)
    createButtons(lat,lng,title); ①
    return message
}
```

1. This is the line that calls our `createButtons()` function!

Try clicking the button on the webpage and it should fly to the location of that marker!

## 🛑End of Lab Part 1🛑

This is the end of part 1 for today's lab!

Let's change our `html` and `js` file names, to `part1.js` and `part1.html` to get ready for the second part of the lab.

Your final code should look like the following:

**js/part1.js**

```
// declare the map
const map = L.map('the_map').setView([34.0709,-118.444], 5);

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);

addMarker(37,-122,'home','home land!')
addMarker(32,-118,'work','where i work land!')
addMarker(39,-119,'location 1','random location')
addMarker(36,-120,'location 2','another random location')

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    createButtons(lat,lng,title);
    return message
}

function createButtons(lat,lng,title){
    const newButton = document.createElement("button"); // adds a new button
    newButton.id = "button"+title; // gives the button a unique id
    newButton.innerHTML = title; // gives the button a title
    newButton.setAttribute("lat",lat); // sets the latitude
    newButton.setAttribute("lng",lng); // sets the longitude
    newButton.addEventListener('click', function(){
        map.flyTo([lat,lng]); //this is the flyTo from Leaflet
    })
    document.getElementById("contents").appendChild(newButton); //this adds the button to our page.
}
```

**part1.html**

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World</title>
        <!-- hint: remember to change your page title! -->
        <meta charset="utf-8" />
        <link rel="shortcut icon" href="#">
        <link rel="stylesheet" href="styles/style.css">

        <!-- Leaflet's css-->
        <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />

        <!-- Leaflet's JavaScript-->
        <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js">
</script>
    </head>

    <body>
        <header>
            <!-- space for a menu -->
        </header>

        <div class="main">
            <div id="contents">
                <!-- page contents can go here -->
            </div>
            <div id="the_map"></div>
        </div>
        <div id="footer">
            Copyright(2022)
        </div>
         <script src="js/part1.js"></script>
    </body>
</html>
```

Last update: 2023-04-17

# Part 2: Creating a GeoJSON file

Learning how to connect information from our surveys to our map will be the key for our class projects, so first we will practice by creating a GeoJSON file of our own!

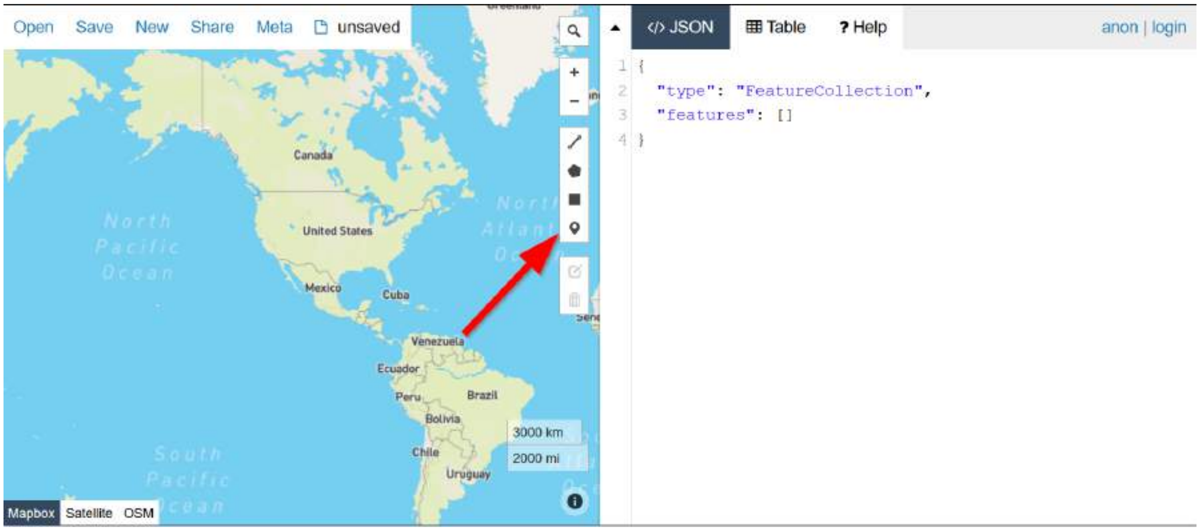
## The power of people-based web mapping

Let’s put to practice what web development and GIS can do for empowering our own stories.

Head over to GeoJSON.io:

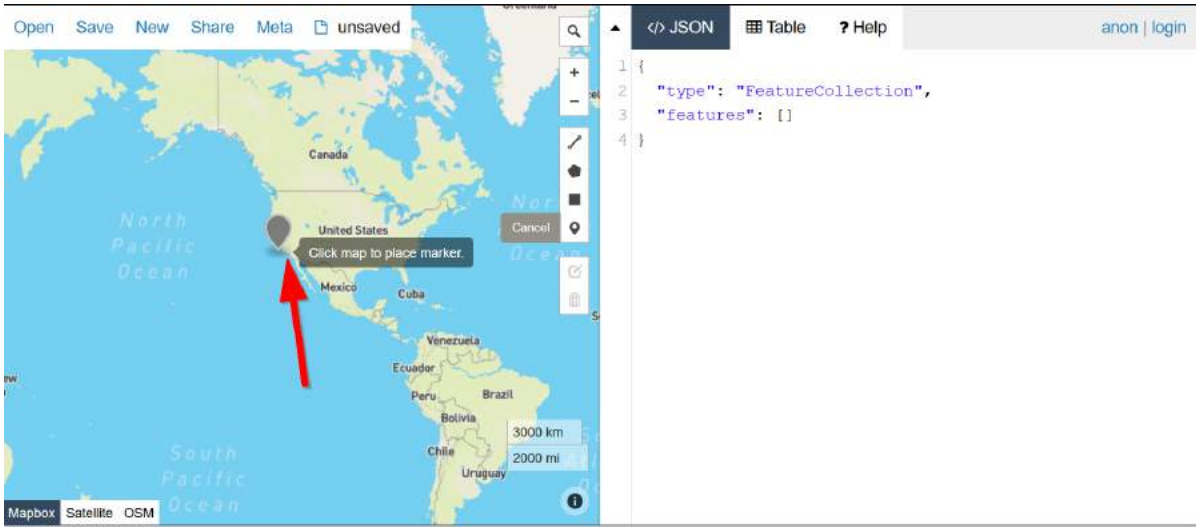
- http://www.geojson.io/

Click on the marker 📍 tool:



```
{
  "type": "FeatureCollection",
  "features": []
}
```

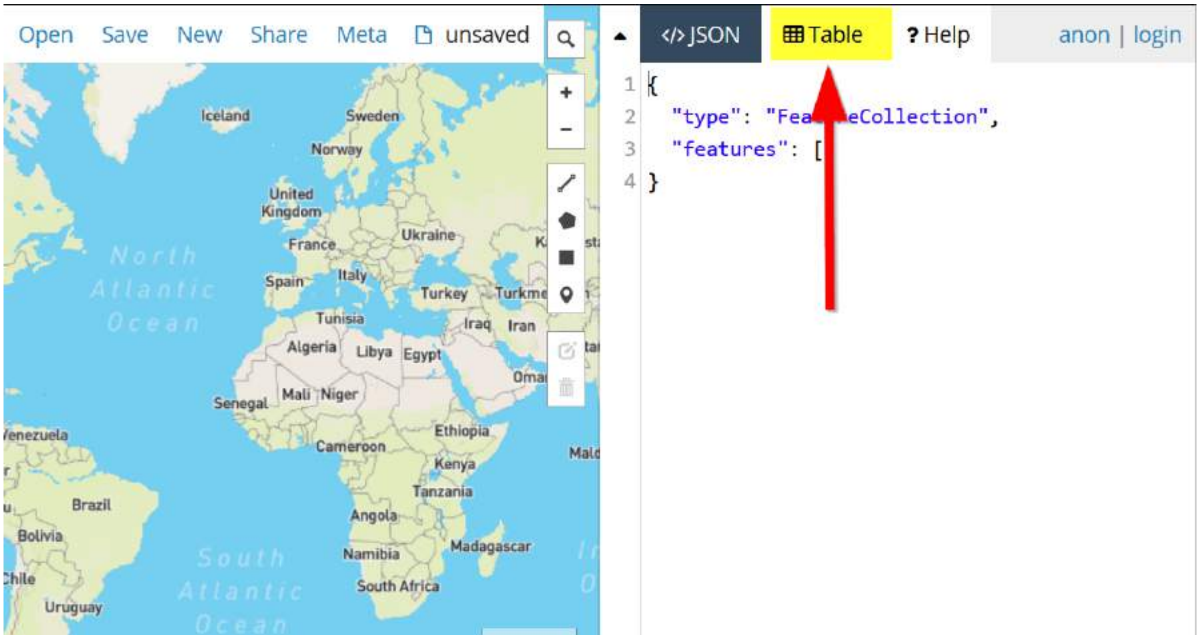
Click on a location of interest to you:



Click map to place marker.

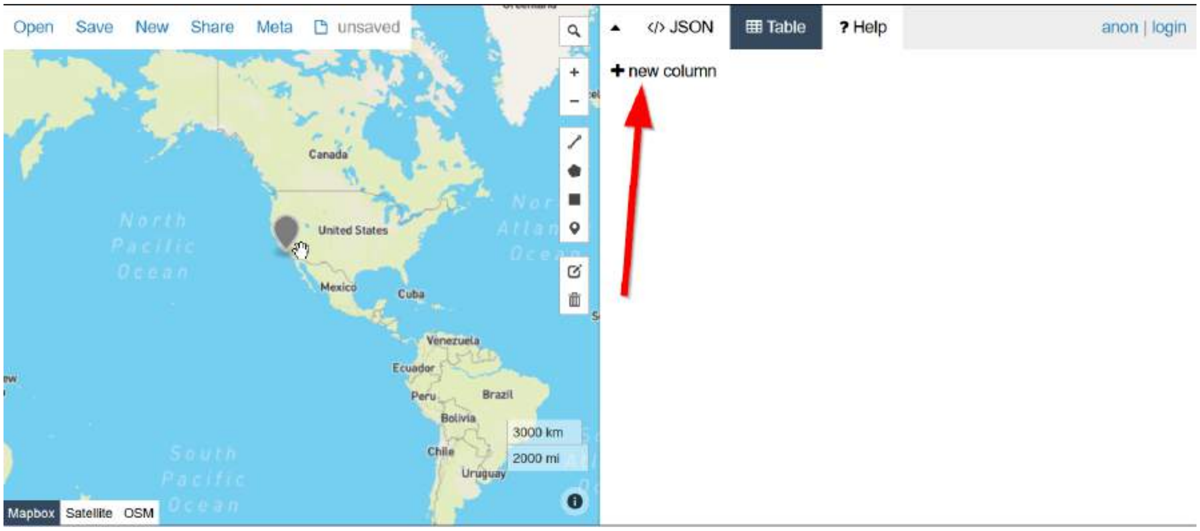
```
{
  "type": "FeatureCollection",
  "features": []
}
```

Switch to the Table view by clicking on ⊞ Table:

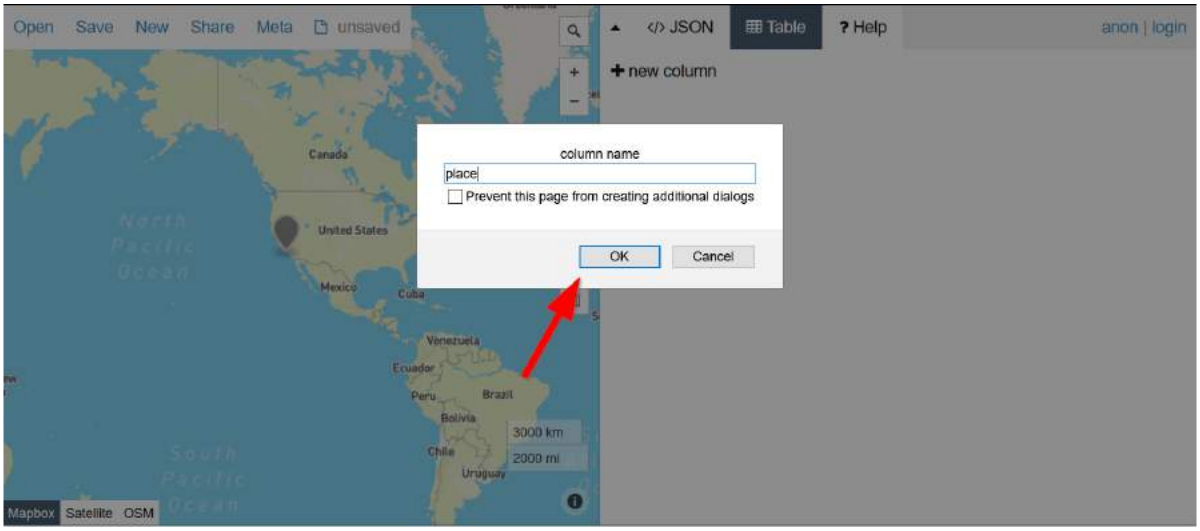


```
{
  "type": "FeatureCollection",
  "features": [
}
```

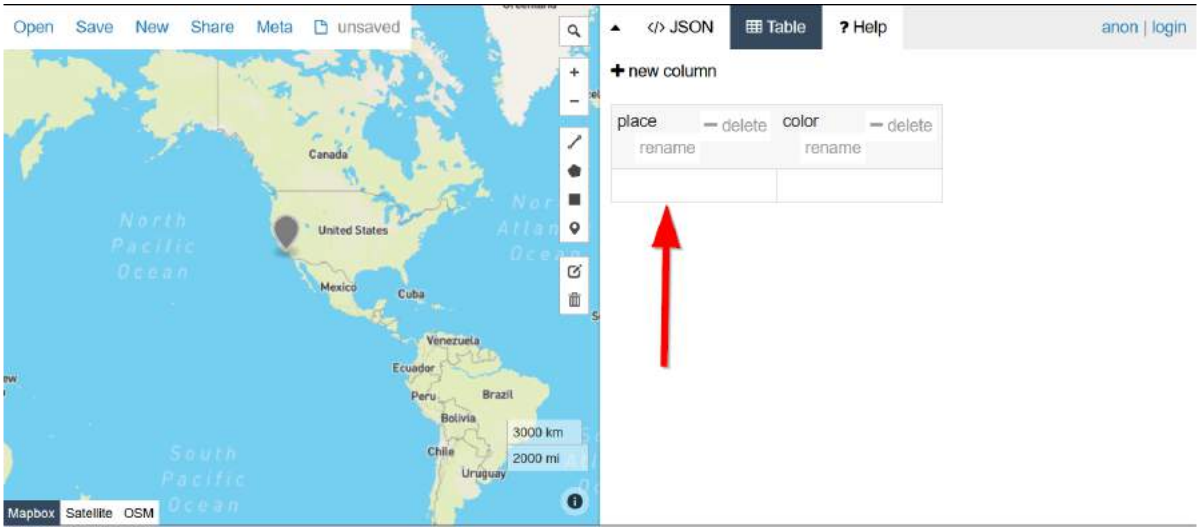
Add a data column by clicking on ➕ new column :



Call it place and click `OK`:

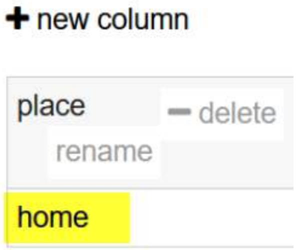


Click inside the `place` column



| place — delete rename | color — delete rename |
|---|---|
| | |

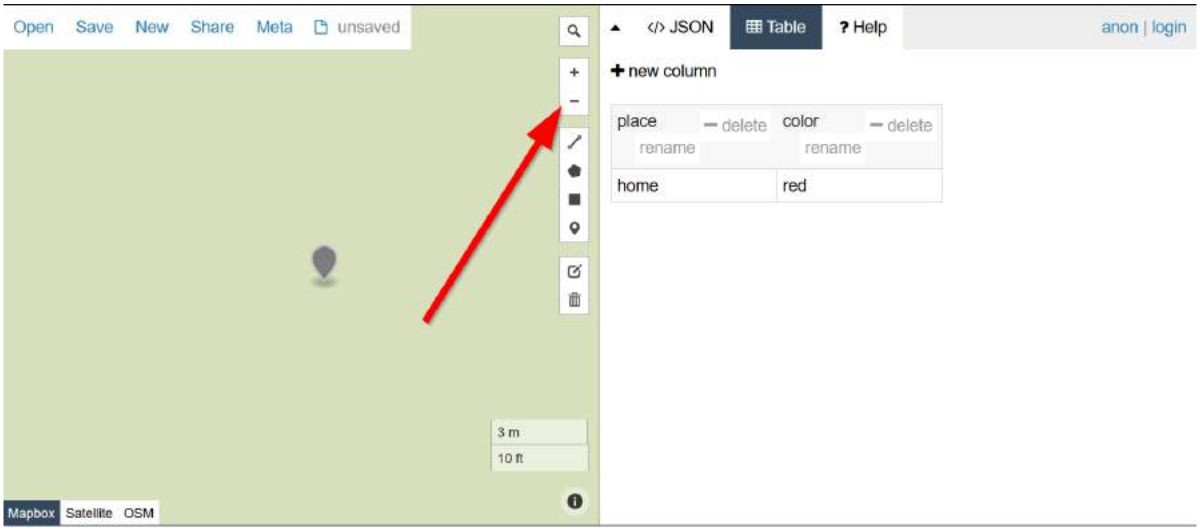
Type in a description for the `place`, in this case I called it home.



+ new column

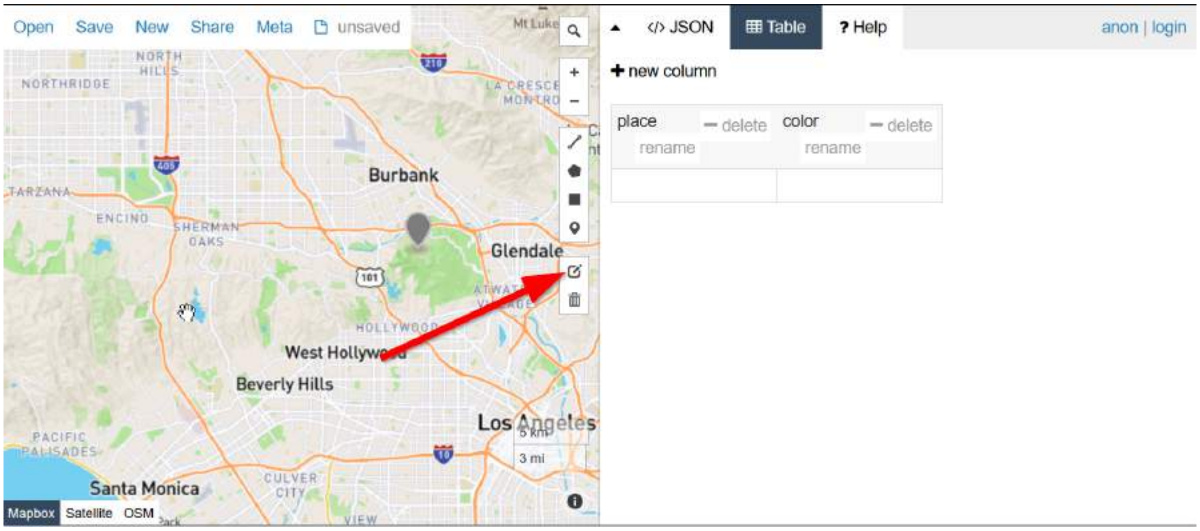
| place — delete rename |
|---|
| home |

Zoom out by pressing the — button or ++ minus ++ key:



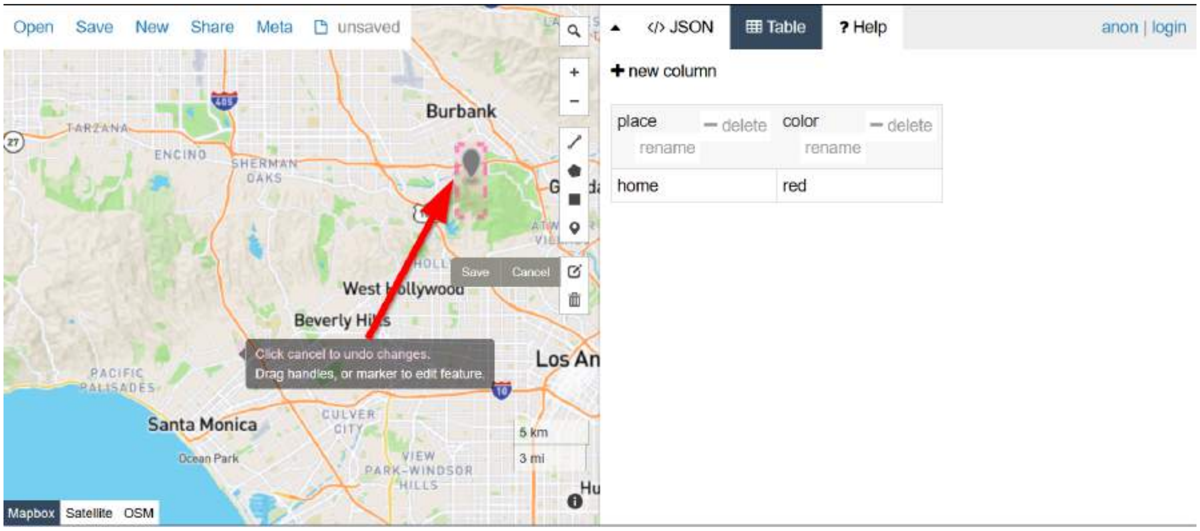
| place — delete rename | color — delete rename |
|---|---|
| home | red |

Click the edit ✎ button:



| place — delete<br>rename | color — delete<br>rename |
|---|---|
| | |

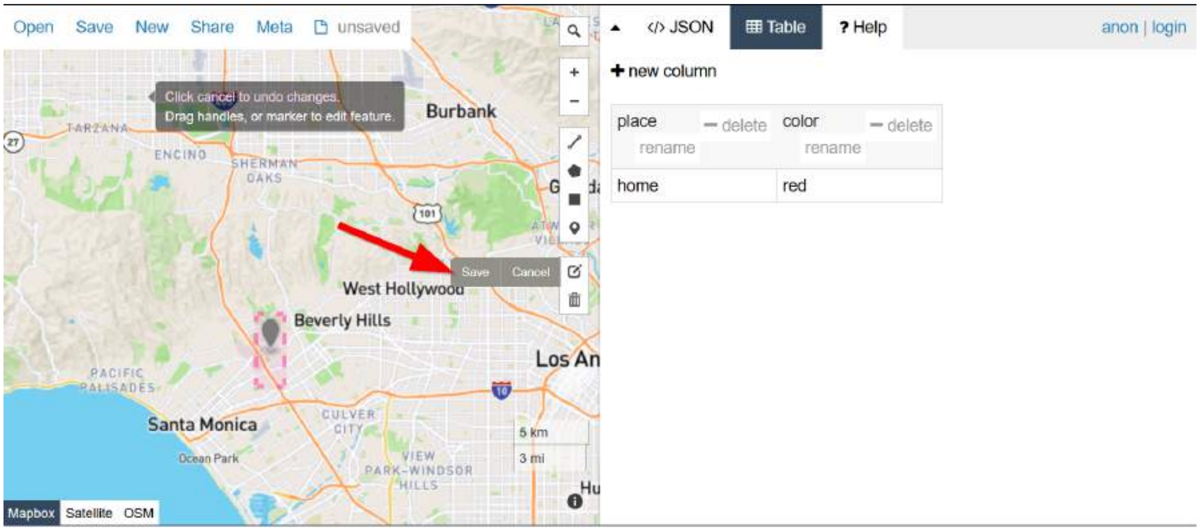
Click on the marker and move it the adjust the location:



Click cancel to undo changes.
Drag handles, or marker to edit feature.

| place — delete<br>rename | color — delete<br>rename |
|---|---|
| home | red |

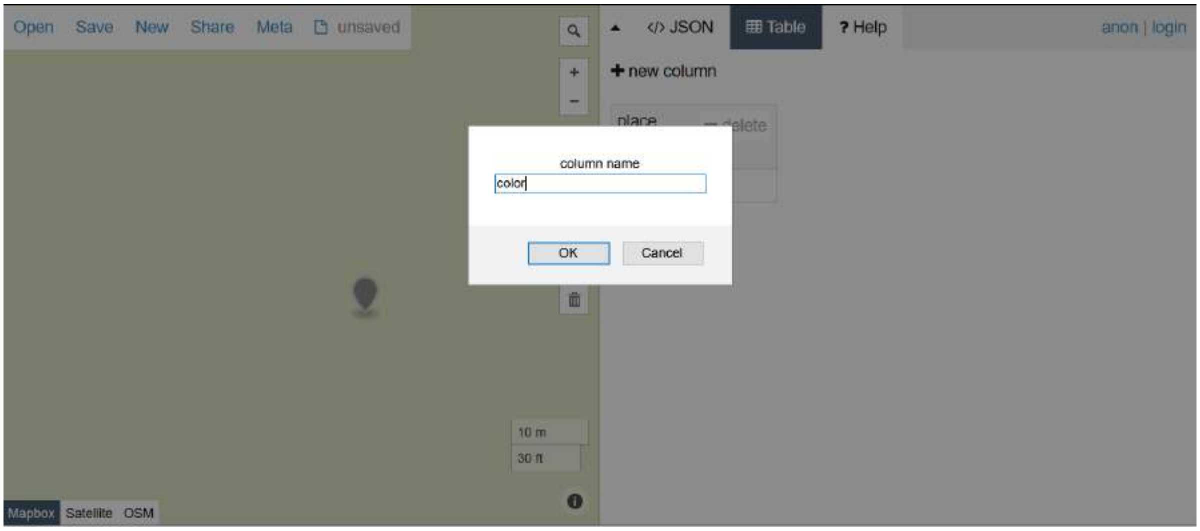
Click the edit ✎ button and then Save to save your edits:



Click cancel to undo changes.
Drag handles, or marker to edit feature.

| place — delete rename | color — delete rename |
|---|---|
| home | red |

Add a new column called color, to put some color to your map later.

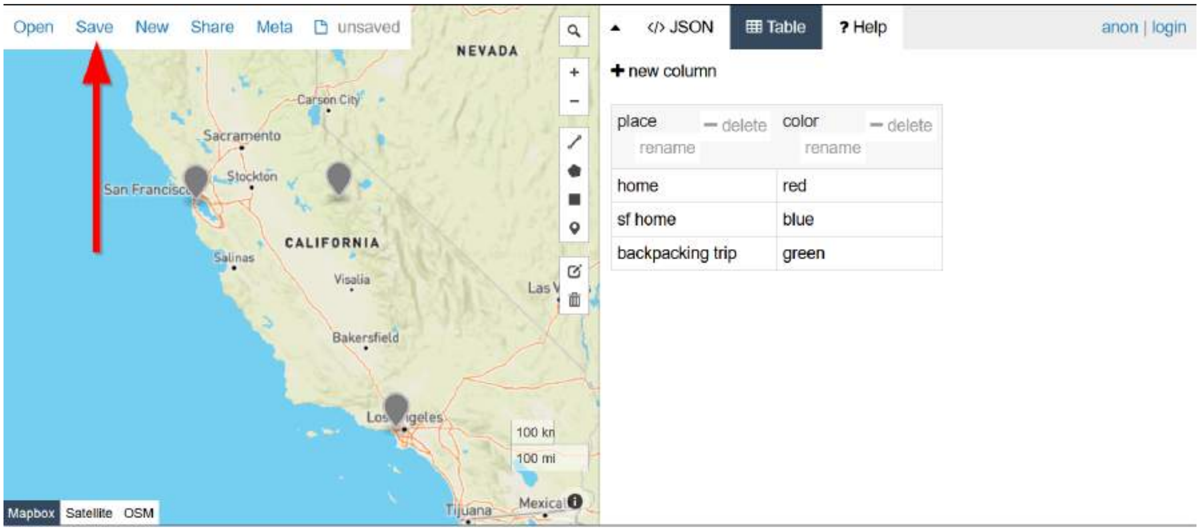


column name

color

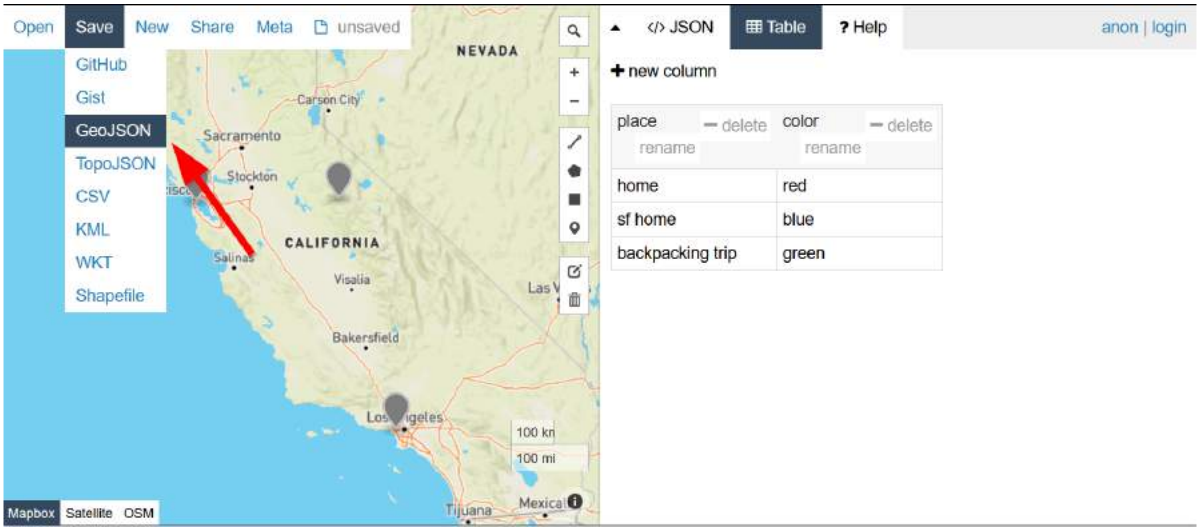
OK Cancel

When you are done, save your file by going to the top menu's Save option:



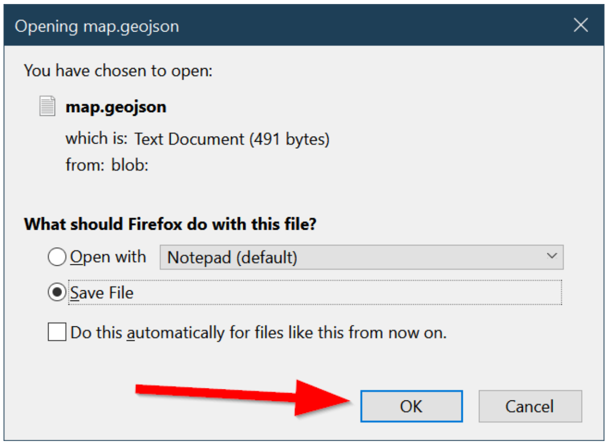
| place | color |
|---|---|
| home | red |
| sf home | blue |
| backpacking trip | green |

Click GeoJSON:



| place | color |
|---|---|
| home | red |
| sf home | blue |
| backpacking trip | green |

Download the file to your computer:


Opening map.geojson

You have chosen to open:

**map.geojson**

which is: Text Document (491 bytes)
from: blob:

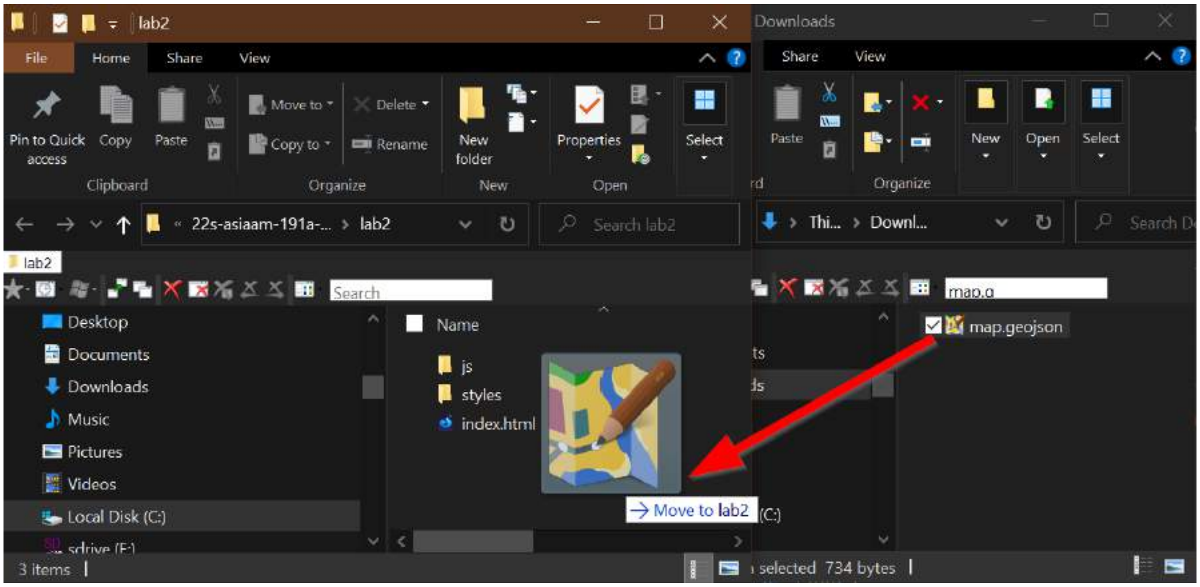
**What should Firefox do with this file?**

○ Open with Notepad (default)

● Save File

□ Do this automatically for files like this from now on.

Copy the file into your project folder:



## ⚽In-class Exercise #1 - Leaving your mark(er) on the map!

Go back and add more points to your GeoJSON file.

**Tasks**

1. Add some points into your GeoJSON file
2. Save the file and add it to your lab3 folder

After finishing the exercise, think about how empowering it was for you to be able to add data to the map yourselves. Whether you were clicking random spots or trying to find your old favorite places to visit, the ability to mark things is a reclaiming of mapping for yourself. This sense of staking a claim is what is meant when we refer to “empowering community voices”.

## 🏁Checkpoint - Check your GeoJSON

1. Make sure your GeoJSON file is in your `week3` folder!
2. Take note of the filename!

Last update: 2023-04-17

# FETCH and THEN

Time to dive back into scary JavaScript waters! Before doing so, `let`‘s just make sure we are warmed-up for our swim!

## Back to JavaScript variables again!

### Need a variable refresher?

#### Revisiting more variable definitions

What we really need to understand about variables is that they act like boxes where you can **store** or **take** information out of. - `const` acts like a locked safe that will not let you put anything into it after you define it - `let` is like a regular box. - `var` is `VARy` problematic because it can be both locked and unlocked

Here are some of the types in JavaScript:

```
//number
let box1 = 5;
let box2 = 5.0;

//string
let box3 = 'five';
let box4 = "five";

// string literal, uses backticks and ${variable} to bring in another variable
let box5 = `this is from box #4: ${box4}`;

// array, which is a list of things
let box6 = [1,2,3,4,5];

// Object, stores variables together, can be of different types!
let box7 = {"number": 'five', "value":5};

// boolean (true or false)
let box8 = true;

// null value
let emptyBox;
```

Remember, to declare a variable or give it a value you must use the `=` symbol, like so:

```
let my_variable = "exist!";
```

#### Anatomy of a variable declaration

- `let` is the keyword declaration of a variable
- `my_variable` is the variable’s name
- `"exist!"` is the value for this variable
- `;` defines the end of a line in JavaScript

\`\`\`

## Anatomy of a JavaScript Object

An `object` is a unique variable that can store many other variables! Think of it as a big box where many other boxes can be put inside.

> ```
> let myJavaScriptObject = {"key_name": "value", "key_2_name":"value"}
> ```

Your object can look like this too:

> ```
>     let myJavaScriptObject = {
>         "key_name": "value",
>         "key_2_name":"value"
>         };
> ```

**Wait! Didn't we see this somewhere?**

Yep! It was in the GeoJSON we created!

**The meaning behind GeoJSON**

GeoJSON actually stands for **"geographic" JavaScript Object Notation**! 🤯

In a **JavaScript object**, each value has a `key` and a `value`.

The `:` symbol seperates the `key` from the `value`, like this:

```
let myObject = {"key":"value"};
```

- Everything in an object is contained within the curly braces `{}`
- Anything to **left** of the `:` is the key
- Anything to **right** of the `:` is the value
- New **key-value pairs** are separated by a comma, `,`
- ⚠️ **Warning** ⚠️! Never end an object with a `,` !!!!

## Accessing an object's property

To access an `object`'s properties we use the `.` notation.

For example, `myObject.key` will return the value, which in this case is.. `value`!

**⚠️ No 🚀 spaces 🪐 in variable names!**

You **cannot** use spaces in variable definitions like `let my map;`, so stick with `camelCase` or `snake_case` when naming varibles with multiple words. When defining `key`s in `objects`, you can use spaces, but it is not recommended.

If you do encounter a `key` with a space in it, like, `let anObject = "my annoying key": "is this"`, you cannot use the `.` syntax to access it you must use this alternative method: `anObject["my annoying key"]`

## ⚽In-class Exercise #2 - Variables and console.log

**Tasks**

1. Re-copy this week's lab template with `index.html` and `init.js`
2. Replace the hard coded values of `const map = L.map('the_map').setView([34.0709, -118.444], 5);` with an `object`.
3. Get the `object` to show up in the console.

**Reminder: Working with our Dev Console**

In VS Code, start Live Server by clicking `Go Live`.

After Firefox runs, open the **Console**:

- You can either right click anywhere on a page with the mouse and clicking on `Inspect` or press `F12` on the keyboard.

Remember to think of the **Console** as the Command Line/Terminal for your browser.

**Answer**

Your code should look like the following:

```
    let mapOptions = {'center': [34.0709,-118.444],'zoom':5}
    const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);
    console.log(mapOptions)
```

1. In the console, type in `mapOptions` (or whatever you chose to name your object) then press `Enter ⏎`.
2. You should see your JavaScript object, `mapOptions`!

**Reflection**

Think about the benefits of having variables in an object, is it easier to read for you? Harder?

Knowing how to check the console will help us test our JavaScript code and even run functions and methods without leaving the browser!

## A **method** to my madness?!

With that refresher about `variables` and practice with `objects` in mind, do you remember how our **`functions`** in last week's lab took a `variable` and did something to it?

Variables have built-in **functions** called `method`s!!!

For example, `string`-type variables have `methods` for changing the string, like making all the letters `UPPERCASE` or splitting a character based on a ` `. To access a method, you use the `.` after the variable has been declared as that type.

**Calling methods for what they are `()`**

Since **`methods`** *are* **functions**, you must call them in the same way with the `()` at the end. This is because some `methods` have parameters you can fill in.

**A helping `console` hand!**

You can check what methods are available right in Firefox’s web developer console! Most modern web browsers have this feature as well.

To give this a try, copy and paste this right into your web browser and see what happens!

```
let myString = "hi, this is a test string"
let divideBySpace = myString.split(" ")
console.log(divideBySpace)
```

**Did you get this response?**

`result: Array(6) [ "hi,", "this", "is", "a", "test", "string" ]`

If you did, yay! This is an array (list), of 6 strings!

## ⚽In-class Exercise #3 - What other methods are available?

As with all languages, learning to look-up things is important to expand what you can say and do! The following is a table of where you can find some methods:

| Location | Type |
|---|---|
| MDN) | Strings |
| W3 | Strings |
| W3 | Numbers |
| W3 | Arrays |
| W3 | Objects |

**Tasks**

1. Visit one of the links above or search online to find other methods.
2. Get the result to display in your console.
3. Bonus: read the next section try to `chain` multiple `methods` together.

**Answer**

Here is an example of an uppercase method:

```
let myString = "hi, this is a test string"
let divideBySpace = myString.toUpperCase()
console.log(divideBySpace)
```

Result:

`"HI, THIS IS A TEST STRING"`

## Method chaining

In JavaScript whenever you see a `.` after a parenthesis `()` ,it means you are chaining a function to follow it.

For example:

```
let myString = "hi this is a test string"
let divideBySpace = myString.toUpperCase().split(" ")
console.log(divideBySpace)
```

The output should look a little bit different than last time thanks to the `toUpperCase()` method!

**Output**

`Array(6) [ "HI", "THIS", "IS", "A", "TEST", "STRING" ]`

# Time to `fetch` and `then` do something

To access data, we will use the JavaScript Fetch API to **`fetch`** our GeoJSON file and **`then`** add it to our map.

When we access the GeoJSON file with the `Fetch` API we then get many `methods` to use with it.

A `fetch` looks like this:

`fetch("map.geojson")`

**Wait! No `variable` declaration?!** 😢

Why do you think so?

**Answer**

The `fetch` API is actually a built-in function for `JavaScript`, much like `console.log()`!

## Good? Let's carry on `then`!

`fetch` actually does nothing by itself! It needs to do something with the data. Thus, `fetch` is almost always used together with the `then` method as follows:

```
fetch("map.geojson") ①
    .then(function aGenericDataFunctionName(data){ ②
        return data.json() ③
    })
    .then(function anotherGenericDataFunctionName(data){ ④
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data).addTo(map) ⑤
    });
```

1. `map.geojson` is location of the GeoJSON file relative to our file. If you moved the file to a subdirectory called `data`, then you would have to make this `data/map.geojson`.
2. Here is our first chain, we are trying to `fetch` our **geojson** file. We will call a `generic` function in here.
3. For the next step we need a `json`, so we `return` the value as a `json` with the `.json()` method!
4. This is the next `then` i.e. our second chain!
5. This calls `L.geoJSON()` and adds our `data` to the map.

## Anoynmous `functions`

Since our `.then` is a one-time call, it does not need named functions as a part of it!

So we can make our `function` **anoymous** by removing the name part of it.

Here’s how the simpler `fetch-then` should look:

```
fetch("map.geojson")
    .then(function (data){
        return data.json()
    })
    .then(function (data){
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data).addTo(map)
    });
```

Looks much better, right? Well… We can shorten it even more!!!

## WHAT IS THIS `=>` 😡?!!!

The `=>` is a shortcut to define an `anoynmous` function and is called an `arrow-function`!

Here is how it looks in practice:

```
fetch("map.geojson") ①
    .then(response => { ②
        return response.json();
    })
    .then(data =>{ ③
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data).addTo(map) ④
    });
```

1. `map.geojson` is location of the GeoJSON file relative to our file. If you moved the file to a subdirectory called `data`, then you would have to make this `data/map.geojson`.
2. Here is our first chain, we are trying to `fetch` our **geojson** file.
3. This is our next chain, we are trying to add it to our map!
4. The `addTo(map)` is similar to our `marker.addTo(map)` function call!

The map should now have a blue tint over it and you cannot interact with it. Not really useful.

Going forward we will use the `arrow-function` because it is shorter, but if you want to use the other methods, feel free to.

## 🏁Checkpoint

Before moving on, check to see if JavaScript code looks like the following:

**js/init.js**

```javascript
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

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);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

fetch("map.geojson")
    .then(response => {
        return response.json();
    })
    .then(data =>{
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data).addTo(map)
    });
```

Last update: 2023-04-17

# Adding more to our `L.GeoJSON`

Remember that putting a variable into a `type` gives you access to different methods?

Rather than just stopping at `L.geoJSON(data).addTo(map)` we are going to expand that part of the code to style the GeoJSON when we add it!

## Clickable GeoJSON recipe

This is the basic Leaflet recipe for a clickable geojson:

```
// the leaflet method for adding a geojson
L.geoJSON(data)
    .bindPopup(layer => {
        return "you clicked a geojson!";
    }).addTo(map);
```

## Adding GeoJSON functionality

Now that we have that recipe, we need to put it somewhere… Where is the best place for it?

**Answer**

```
fetch("map.geojson")
    .then(response => {
        return response.json();
    })
    .then(data =>{
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data) ①
        .bindPopup(layer => {
            return "you clicked a geojson!"; ②
        }).addTo(map); ③
    });
```

1. This is where we added the clickable geoJSON recipe!!
2. Notice we are going to a generic `you clicked a geojson` message here!
3. This is where we add the `GeoJSON` to the map.

Rather than just simply returning the `popup` as a generic `you clicked a geojson`, let's use our GeoJSON's `place` property that we created in the first part of the lab!

## Checking our logs!

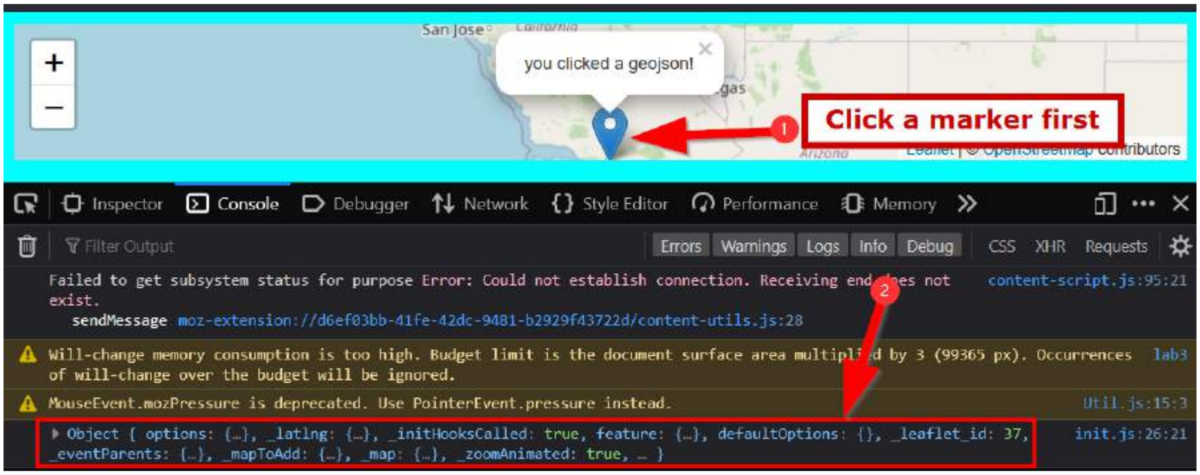
Let's `console.log()` our layer to see how it looks:

**Where should the `console.log()` go?**

Correct, line 3!

```
L.geoJSON(data,
    }).bindPopup(layer => {
            console.log(layer)
            return "you clicked a geojson!"
    }).addTo(map);
```

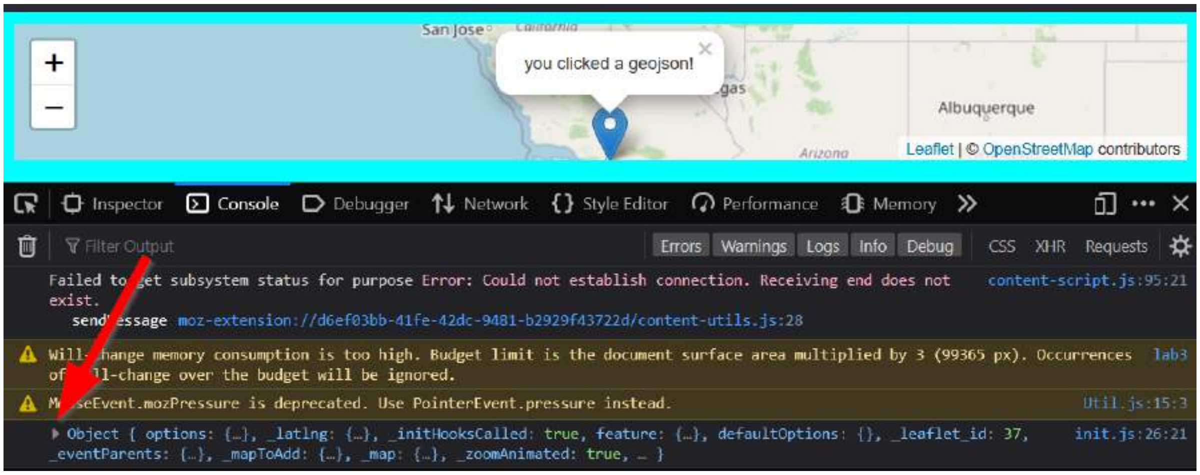
Now when you click the marker, this should pop-up in the console:



```
Failed to get subsystem status for purpose Error: Could not establish connection. Receiving end does not    content-script.js:95:21
exist.
  sendMessage moz-extension://d6ef03bb-41fe-42dc-9481-b2929f43722d/content-utils.js:28
⚠ Will-change memory consumption is too high. Budget limit is the document surface area multiplied by 3 (99365 px). Occurrences   lab3
of will-change over the budget will be ignored.
⚠ MouseEvent.mozPressure is deprecated. Use PointerEvent.pressure instead.    Util.js:15:3
▸ Object { options: {…}, _latlng: {…}, _initHooksCalled: true, feature: {…}, defaultOptions: {}, _leaflet_id: 37,    init.js:26:21
_eventParents: {…}, _mapToAdd: {…}, _map: {…}, _zoomAnimated: true, … }
```

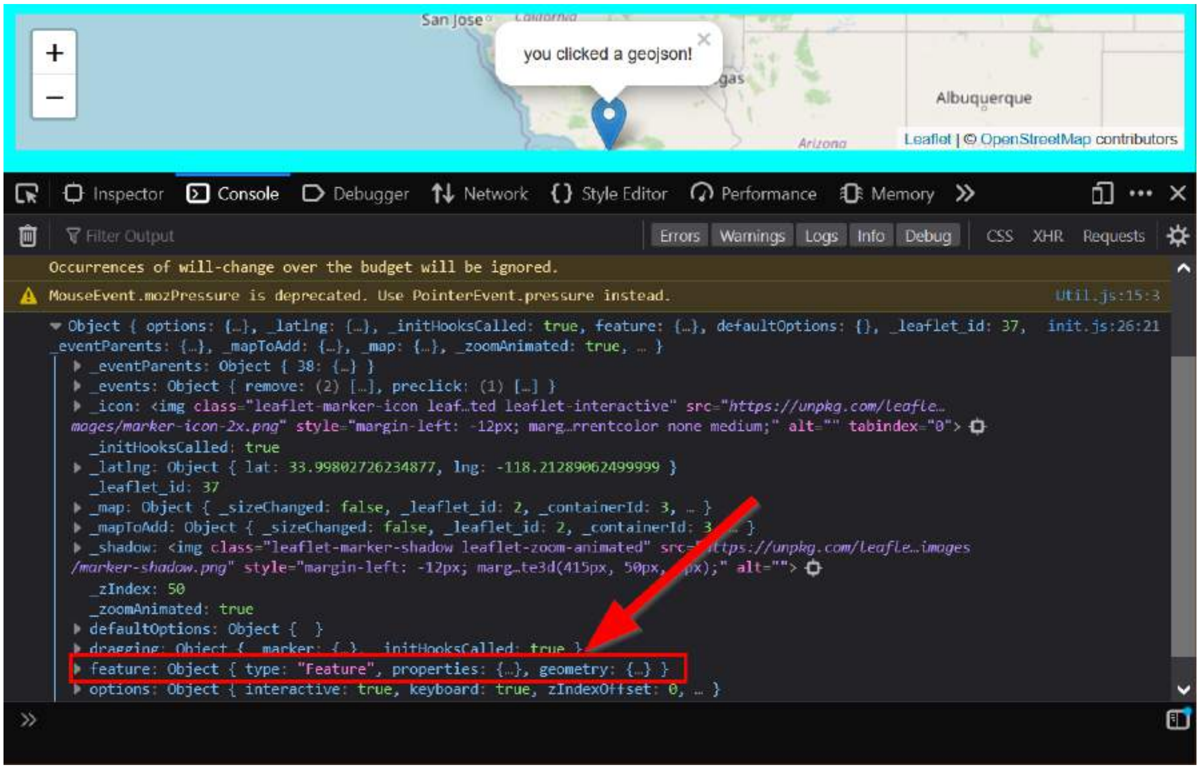
```
Object { options: {…}, _latlng: {…}, _initHooksCalled: true, feature: {…},
defaultOptions: {}, _leaflet_id: 37, _eventParents: {…}, _mapToAdd: {…}, _map:
{…}, _zoomAnimated: true, … }
```

We can drill down into our GeoJSON by clicking on the arrows:



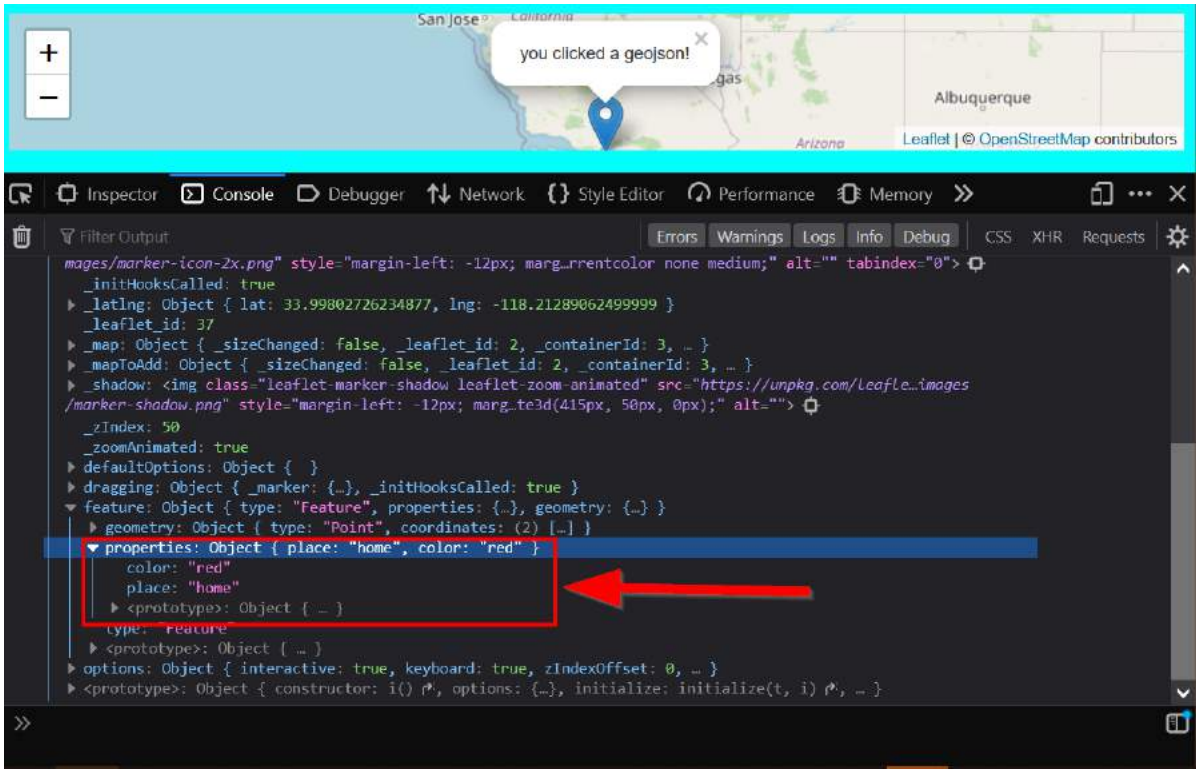
```
Failed to get subsystem status for purpose Error: Could not establish connection. Receiving end does not    content-script.js:95:21
exist.
  sendMessage moz-extension://d6ef03bb-41fe-42dc-9481-b2929f43722d/content-utils.js:28
⚠ Will-change memory consumption is too high. Budget limit is the document surface area multiplied by 3 (99365 px). Occurrences   lab3
  of will-change over the budget will be ignored.
⚠ MouseEvent.mozPressure is deprecated. Use PointerEvent.pressure instead.                              Util.js:15:3
▶ Object { options: {…}, _latlng: {…}, _initHooksCalled: true, feature: {…}, defaultOptions: {}, _leaflet_id: 37,   init.js:26:21
_eventParents: {…}, _mapToAdd: {…}, _map: {…}, _zoomAnimated: true, … }
```

Find the `feature` property and click the arrow to expand it:



```
Occurrences of will-change over the budget will be ignored.
⚠ MouseEvent.mozPressure is deprecated. Use PointerEvent.pressure instead.                               Util.js:15:3
▼ Object { options: {…}, _latlng: {…}, _initHooksCalled: true, feature: {…}, defaultOptions: {}, _leaflet_id: 37,   init.js:26:21
_eventParents: {…}, _mapToAdd: {…}, _map: {…}, _zoomAnimated: true, … }
  ▶ _eventParents: Object { 38: {…} }
  ▶ _events: Object { remove: (2) […], preclick: (1) […] }
  ▶ _icon: <img class="leaflet-marker-icon leaf…ted leaflet-interactive" src="https://unpkg.com/leafle…
  mages/marker-icon-2x.png" style="margin-left: -12px; marg…rrentcolor none medium;" alt="" tabindex="0">
    _initHooksCalled: true
  ▶ _latlng: Object { lat: 33.99802726234877, lng: -118.21289062499999 }
    _leaflet_id: 37
  ▶ _map: Object { _sizeChanged: false, _leaflet_id: 2, _containerId: 3, … }
  ▶ _mapToAdd: Object { _sizeChanged: false, _leaflet_id: 2, _containerId: 3, … }
  ▶ _shadow: <img class="leaflet-marker-shadow leaflet-zoom-animated" src="https://unpkg.com/leafle…images
  /marker-shadow.png" style="margin-left: -12px; marg…te3d(415px, 50px, 0px);" alt="">
    _zIndex: 50
    _zoomAnimated: true
  ▶ defaultOptions: Object {  }
  ▶ dragging: Object { _marker: {…}, _initHooksCalled: true }
  ▶ feature: Object { type: "Feature", properties: {…}, geometry: {…} }
  ▶ options: Object { interactive: true, keyboard: true, zIndexOffset: 0, … }
```

Look at the `properties` and notice what is in there!



```
mages/marker-icon-2x.png" style="margin-left: -12px; marg…rrentcolor none medium;" alt="" tabindex="0">
  _initHooksCalled: true
▶ _latlng: Object { lat: 33.99802726234877, lng: -118.21289062499999 }
  _leaflet_id: 37
▶ _map: Object { _sizeChanged: false, _leaflet_id: 2, _containerId: 3, … }
▶ _mapToAdd: Object { _sizeChanged: false, _leaflet_id: 2, _containerId: 3, … }
▶ _shadow: <img class="leaflet-marker-shadow leaflet-zoom-animated" src="https://unpkg.com/leafle…images
/marker-shadow.png" style="margin-left: -12px; marg…te3d(415px, 50px, 0px);" alt="">
  _zIndex: 50
  _zoomAnimated: true
▶ defaultOptions: Object {  }
▶ dragging: Object { _marker: {…}, _initHooksCalled: true }
▼ feature: Object { type: "Feature", properties: {…}, geometry: {…} }
  ▶ geometry: Object { type: "Point", coordinates: (2) […] }
  ▼ properties: Object { place: "home", color: "red" }
      color: "red"
      place: "home"
    ▶ <prototype>: Object { … }
    type: "Feature"
  ▶ <prototype>: Object { … }
▶ options: Object { interactive: true, keyboard: true, zIndexOffset: 0, … }
▶ <prototype>: Object { constructor: i(), options: {…}, initialize: initialize(t, i), … }
```

**Right! Those are the columns and values we created from the first part of the lab!**

This is called `traversing` the `object` path, and it works the same way when we linked our `photos` or `.css`. The key difference is that it is within one file!

Recapping how we got here, we 1) went into the object (`layer`), then 2) clicked on `feature`, then clicked on 3) `properties`.

To access the place name, we will need to specify that with `place`.

As a result, our path should look like this:

```
layer.feature.properties.place;
```

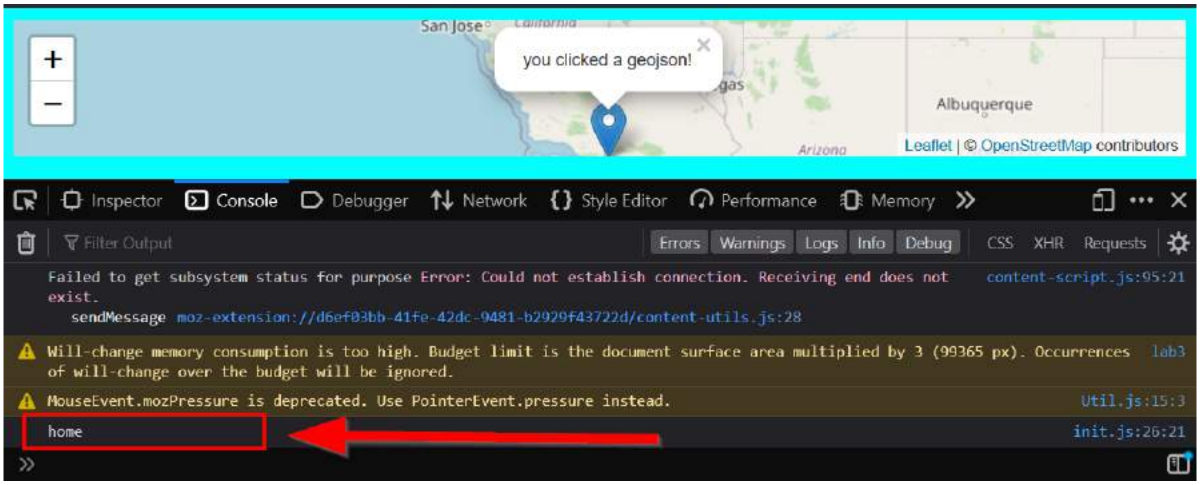
**OMG!! The `.` returns?!**

Aha, very observant! Similar to `chaining` methods, we use the `.` to chain going down an `object` path. Why is that?! Well.. It has something to do with `classes`, but thats out of the `scope` of this class. (multiple coding puns intended.) If you really want to learn more, click here to read about Object-Oriented Programming and JavaScript: click if you dare!.

Let's `console.log()` the result to make sure we have the right path down:

```
    L.geoJSON(data,
        }).bindPopup(layer => {
                console.log(layer.feature.properties.place)
                return "you clicked a geojson!"
        }).addTo(map);
```

When you click a point, the correct `value` should show up:



```
Failed to get subsystem status for purpose Error: Could not establish connection. Receiving end does not    content-script.js:95:21
exist.
  sendMessage moz-extension://d6ef03bb-41fe-42dc-9481-b2929f43722d/content-utils.js:28
⚠ Will-change memory consumption is too high. Budget limit is the document surface area multiplied by 3 (99365 px). Occurrences    lab3
of will-change over the budget will be ignored.
⚠ MouseEvent.mozPressure is deprecated. Use PointerEvent.pressure instead.    Util.js:15:3
home    init.js:26:21
```

Woo!! Now let's `return` this value instead of the generic message:

```
    L.geoJSON(data,
        }).bindPopup(layer => {
                console.log(layer.feature.properties.place)
                return layer.feature.properties.place
        }).addTo(map);
```

Now when you click on the map, the `place` values shows up!

## Utilizing our GeoJSON's `color` property

Before we finish this module, let's take what we learned one step further and use our `color` property too.

While `bindPopUp()` was nice and an outside method, changing the color needs to be inside of the `L.geoJSON()` call. So we have to attach it to an object inside as follows:

```
        L.geoJSON(data, { ①
            style: layer => { ②
                return {color: layer.feature.properties.color}; ③
            }
        }).bindPopup(layer => {
            return layer.feature.properties.place;
        }).addTo(map);
```

1. Here we add a `{` to start our new object
2. `style` is what Leaflet's `L.GeoJSON()` wants, so we have to use that
3. We are assigning `layer.feature.properties.color` here!

> **YOU LIE!!! THIS DOES NOT WORK**
>
> Correct! This code will not work because… A GeoJSON's color property can only be set for `L.CircleMarkers`, `lines`, or `polygons` but **not** regular `L.markers`.

## Converting our GeoJSON to `CircleMarkers`

Since `{style: "red"}` or any color won't work for our marker, we need to convert it into a circle marker using the `pointToLayer()` method! Again, this has to be inside the `L.geoJSON()` because that is where **Leaflet** must know what color to make the features.

We will use the `arrow-function` so we can type fewer characters:

```
        L.geoJSON(data, {
            pointToLayer: (feature, latlng) => { ①
                return L.circleMarker(latlng, {color: feature.properties.color});
②
            }
```

1. Here we pass in our `feature` and `latlng` into the simplified `=> function`
2. Now we convert it to a `L.circleMarker()`, with `latlng` being the first parameter, then setting `color` to the `feature.properties.color`.

The `fetch`'s final `.then` should now look like the following:

```
fetch("map.geojson")
    .then(response => {
        return response.json()
    })
    .then(data =>{
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data, {
                pointToLayer: (feature, latlng) => {
                    return L.circleMarker(latlng, {color:
feature.properties.color})
```

```
            }
        }).bindPopup(layer => {
            return layer.feature.properties.place;
        }).addTo(map);
})
```

## 🏁Last Checkpoint

Our final `init.js` file should look like this:

**js/init.js**

```
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

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);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

fetch("map.geojson")
    .then(response => {
        return response.json()
    })
    .then(data =>{
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data, {
                pointToLayer: (feature, latlng) => {
                    return L.circleMarker(latlng, {color:
feature.properties.color})
                }
            }).bindPopup(layer => {
                return layer.feature.properties.place;
            }).addTo(map);
    })
```

If everything works up until now, then you are ready to take on the lab assignment!

# ✅ Final Template Code

**index.html**

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Hello World</title>
        <!-- hint: remember to change your page title! -->
        <meta charset="utf-8" />
        <link rel="shortcut icon" href="#">
        <link rel="stylesheet" href="styles/style.css">

        <!-- Leaflet's css-->
        <link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />

        <!-- Leaflet's JavaScript-->
        <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js">
</script>
    </head>

    <body>
        <header>
            <!-- space for a menu -->
        </header>

        <div class="main">
            <div id="contents">
                <!-- page contents can go here -->
            </div>
            <div id="the_map"></div>
        </div>
        <div id="footer">
            Copyright(2022)
        </div>
        <script src="js/init.js"></script>
    </body>
</html>
```

**styles/style.css**

```css
body{
    display: grid;
    /* grid-template-columns: 1fr;  */
    grid-auto-rows: auto 1fr;
    grid-template-areas: "header" "main_content" "footer";
    background-color: aqua;
    /* height: 100vh; */
}

header{
    grid-area: header;
}

#footer{
    grid-area: footer;
}

.main{
    grid-area: main_content;
    grid-template-areas: "content" "main_map";
    display: grid;
}

#contents{
    grid-area: content;
}

#the_map{
    height:80vh;
    grid-area: main_map;
}
```

**js/init.js**

```javascript
// declare variables
let mapOptions = {'center': [34.0709,-118.444],'zoom':5}

// use the variables
const map = L.map('the_map').setView(mapOptions.center, mapOptions.zoom);

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);

// create a function to add markers
function addMarker(lat,lng,title,message){
    console.log(message)
    L.marker([lat,lng]).addTo(map).bindPopup(`<h2>${title}</h2>
<h3>${message}</h3>`)
    return message
}

fetch("map.geojson")
    .then(response => {
        return response.json()
    })
    .then(data =>{
        // Basic Leaflet method to add GeoJSON data
        L.geoJSON(data, {
                pointToLayer: (feature, latlng) => {
                    return L.circleMarker(latlng, {color:
feature.properties.color})
                }
            }).bindPopup(layer => {
                return layer.feature.properties.place;
            }).addTo(map);
    })
```

You can use this template to finish this week’s lab assignment!

Last update: 2023-04-17