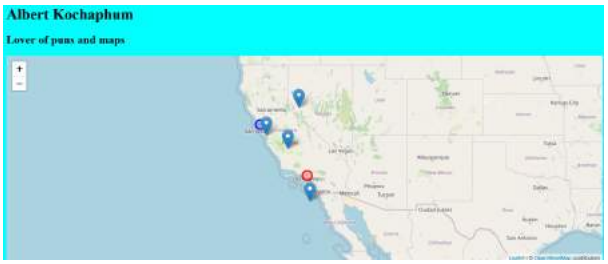
JavaSCrypt of the Necrodancer and FUNctions

Creating our first map with Leaflet.js and learning to use JavaScript!



Goals

- Add a Leaflet map
- Understand how JavaScript works with HTML and CSS
- Understand how JavaScript variables, functions, methods work together

Starting Template Code for lab #2

Use the following template code or your lab assignment #1:

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">
    </head>

    <body>
        <header>
            <!-- hint: you can make a menu with other links here if you'd
like -->
        </header>

```

```
        <div class="main">
            <!-- hint: the majority of lab 1 assignment can go here -->
        </div>
        <div id="footer">
            Copyright(2022)
        </div>
    </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";
}

#contents{
    grid-area: content;
}
```

Last update: 2023-04-09

Extra: Basics of JavaScript

> **Note**
>
> This lab will be covered on Thursday 4/13

JavaScript makes sure our page knows how to function and react. There are different frameworks for JavaScript, like React.js and vue.js, but this class will be focusing on vanilla JavaScript with ES6+ standards.Read more about the standards here.

In `HTML`, `JavaScript` must be contained within a script tag. In our `<head>` tag, let's add a `<script></script>` tag.

Any Javascript that is in the `<head>` tag will load first.

Any JavaScript that is in the `<body>` tag will load later.

JavaScript functions to run after the `HTML body` loads, so putting the `<script>` after the `</body>` becomes necessary.

This will be relevant when we bring in `Leaflet.js` because the `Leaflet` library needs to be loaded first! That means it should go in the header, while our own custom `JavaScript` comes after, preferrably later in the `<body>` tag, you can even kick our JavaScript file out of the body and put it into a `<footer>` tag!

Let's a-(variable)-go!

Variables are like **boxes** that hold information. They can be **numbers**, **text**, or even collections of other variables! In programming languages we call those variable **types**. With JavaScript, variables are automatically assigned types based on their declaration. We'll discuss more next week, but what you need to know for now is how to **declare** variables.

In JavaScript all declarations and lines should end with a semicolon `;`, which is like a `.` in English that says, my statement is done.

This is an example of a declaration:

```
var day = 8;
var name = "Albert";
```

In front you see the `var` **keyword** that tells the web browser, “Hey this is a variable!”. In this example, `day` is a **numeric** type with a value of `8` and `name` is a **string** type. Each type has certain properties and uses, for example you can add **numbers** together using something like `day + day`, but you adding strings will simply concatenate and not total them.

> **What is a keyword?**
>
> In most coding languages, a **keyword** is a word that tells a program to treat the following text, numbers, or characters in a specific way. For example, `var myName` says treat `myName` as a variable. This means you **CANNOT** name a variable `var`, Jar Jar Binks cousin Var Var Binks is **VARy** bad for JavaScript to see! i.e. `var var` Also note, you cannot use `spaces` in variable names!

With JavaScript ES6, `let` and `const` keywords were introduced to declare variables. This change means that the recommend practice is to no longer use the `var` keyword. `let` and `const` variables get declared in the same way:

```
let day = 8; 1
const name = "Albert"; 2
```

1. The `let` keyword **LETS** a variable CHANGE! 😄
2. The `const` keyword declaration keeps a variable **CONST**ant!

Let vs Const vs Var

> **What is the difference?**
>
> 1. The `let` keyword declaration LETS a variable change
> 2. The `const` keyword delcaration a variable **CONSTant** and will never change.
> 3. The `var` allows varaibles to change or never change depending on **where** it was declared! VERY PROBLEMATIC!

Because `var` can be changing (mutable) and unchanging at the same time, so `var` was changed into off into two different variable types, `let` and `const`.

Scopes: Local vs. Global

Where you `declare` a variable sets the scope to either a local one (limited to a function or area in the code) or global (can be accessed by anything/anywhere else in the code).

So, bye bye `var` and `LET` us welcome our new `CONST` variables to the JavaScript programming world.

TLDR

DO NOT USE `var` unless you need to code for Internet Explorer.

Console.log()

By itself, our script tag does nothing. So, one VERY helpful JavaScript tool (method) that we should familarize ourself with is `console.log()`, because it allows us to test our code.

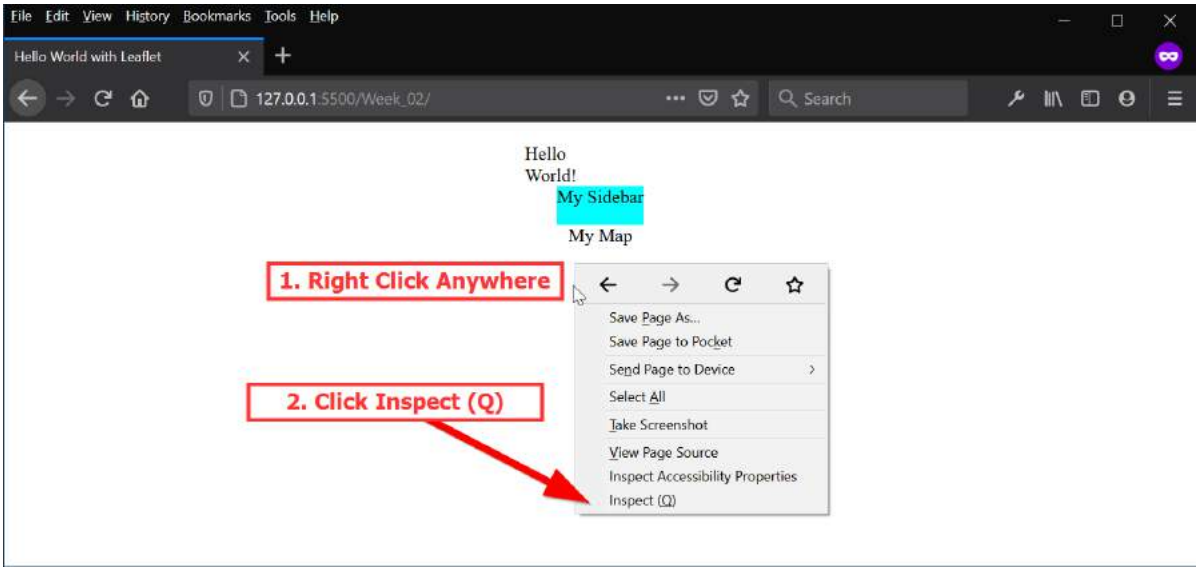
Add the following script:

```
<script>
    console.log('Hello Asian Am 191! :)');
</script>
```

Nothing happened?! What!?

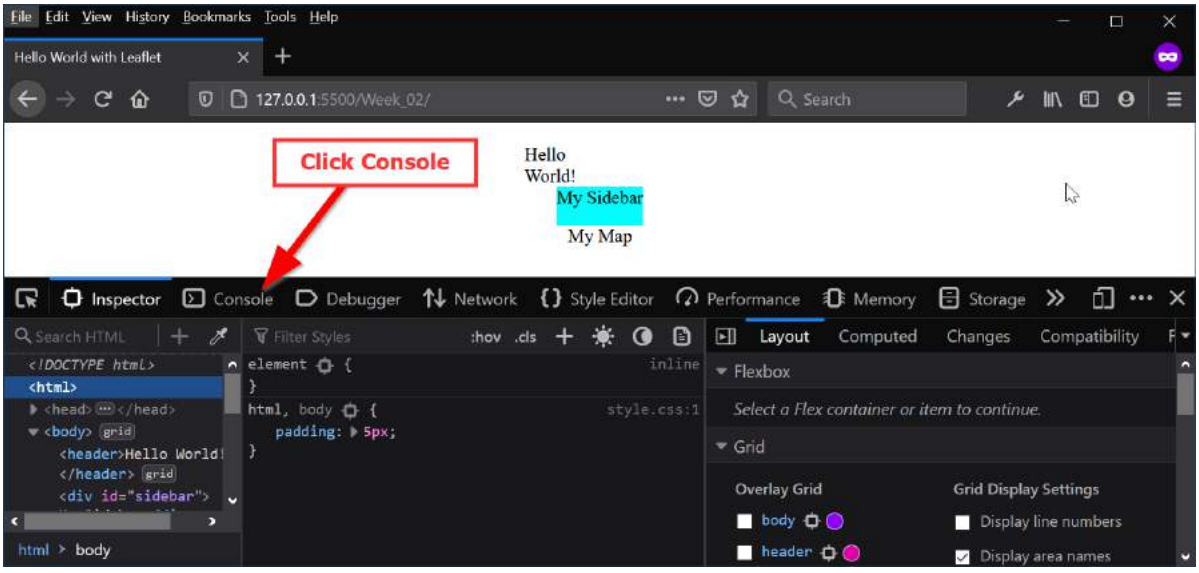
Actually, you are about to unlock your full web developer potential!

In Firefox, right click anywhere on the page and the click `Inspect Element`:

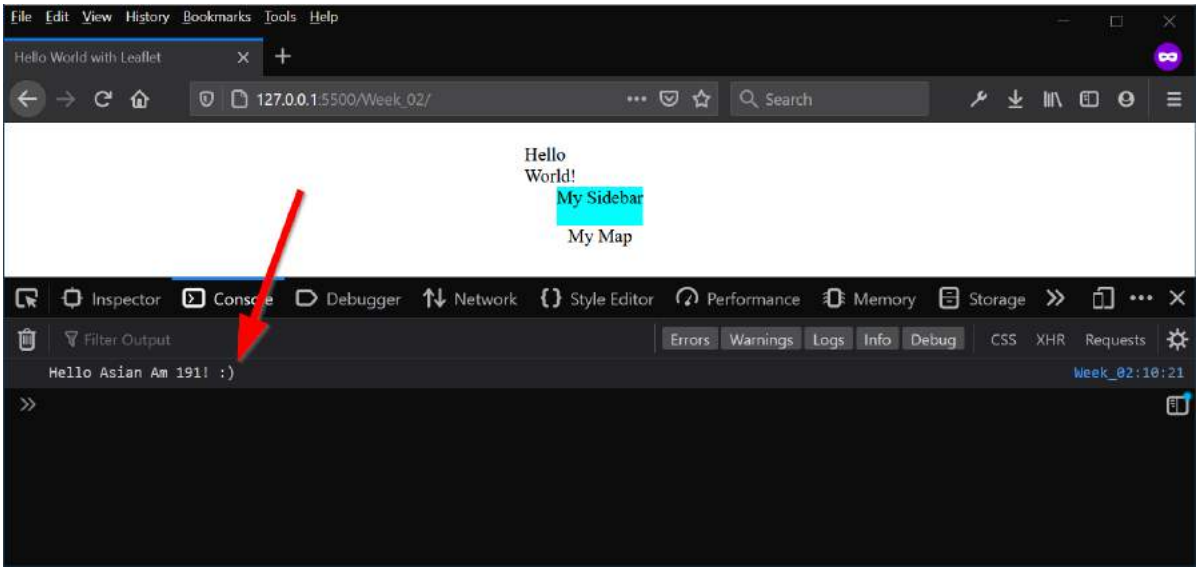


This opens the Developer Toolbar!! 🎉🎉 You can also find it by going to the **Menu** and going to **Web Developer** and then **Web Developer Tools**.

Click on the ⌨ Console button:



Yay! Our message is there!



Linking to another JavaScript file

Similar to the CSS files, we should move the JavaScript file into its own file (and folder) to avoid cluttering the HTML file with JavaScript.

Importing different libraries, whether it it `CSS` or `JavaScript` is the main way unlock skills and level up our webpage.

BUT!!! Instead of the `<link>` that we use with `CSS` we use the `<script>` tag:

Linking JavaScript

```
<script src="YOUR_SCRIPT_NAME.js"></script>
```

The `src` attribute is location of your file.

Linking CSS

```
<link src="SOME_CSS_FILE.css">
```

Notice that when you use `<link>` there is no `</link>`, but with `<script>` you must close the tag with `</script>`.

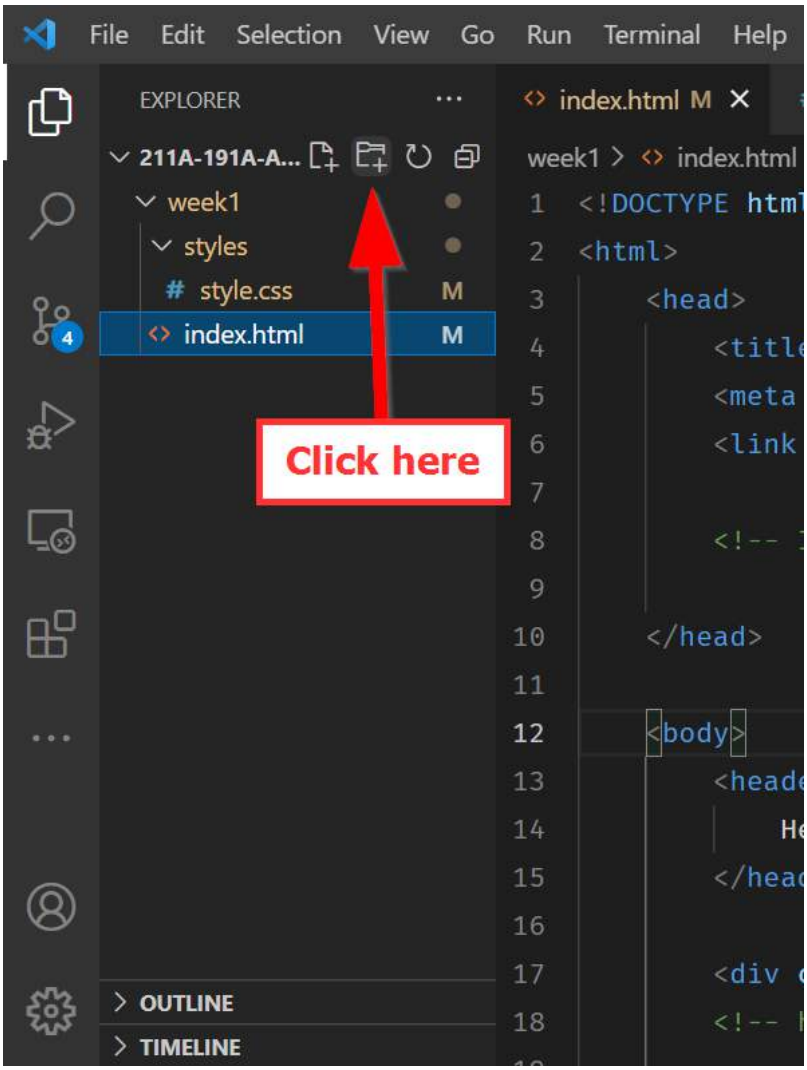
⚽ In-class Exercise #3 - JuSt link your JS file

Tasks

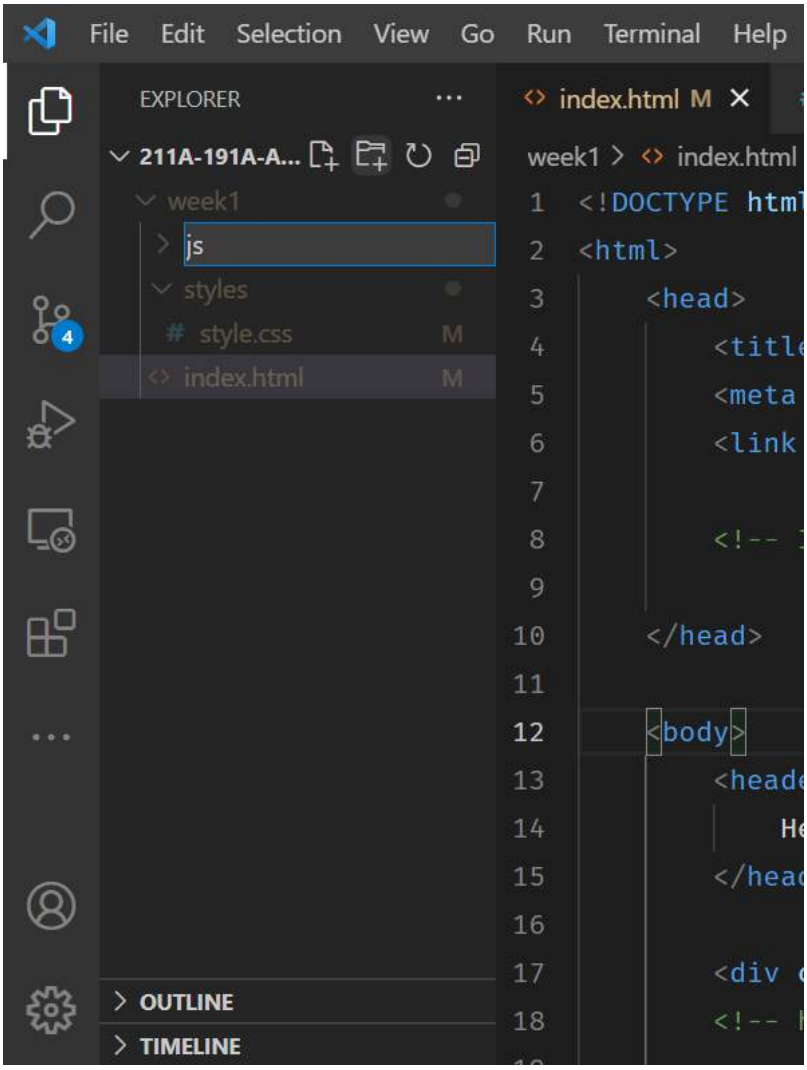
1. Create a new folder called `js`
2. Add a `JavaScript` file in there called `init.js`
3. Add JavaScript method: `console.log()` with a message of your choosing
4. Get your message to show up in the console

Answer

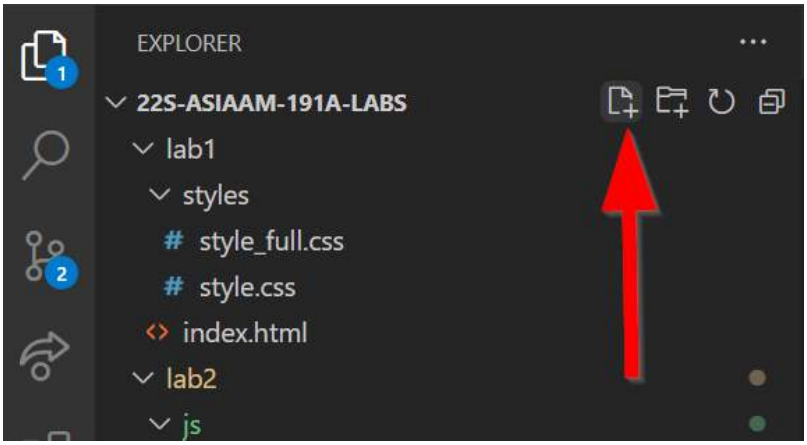
1. Click on the `New Folder` button:



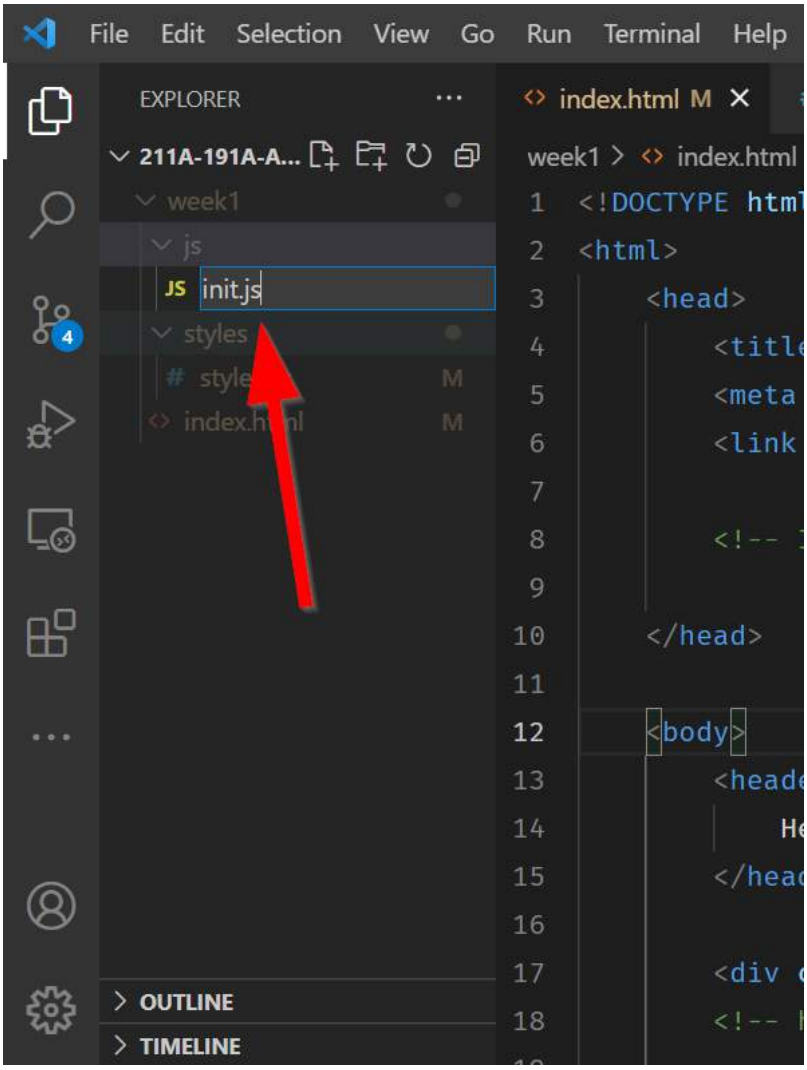
2. Type in `js`:



3. Click on the New File button:



4. Give it a name, like `init.js`, which in this case stands for the `initial JavaScript file of our page`



5. In the `index.html` file and before the end of the `<body>` element include the following:

index.html

```
        //
        // ... HTML Truncated for brevity ...
        //
        <script src="./js/init.js"></script>
    </body>
<html>
```

/js/init.js

```
console.log("Hello Asia-Am 191A! :)")
```

> ⚠ **Important!**
>
> Never include `<script></script>` tags inside of a Javascript file, those are `HTML tags` !!! Do so will break your page, because you are mixing two different languages: `HTML` with `JavaScript`. 😢

Hello Leaflet… Finally..

OK, why did we do ALL of that? Well, when we use Leaflet, we actually need to bring in Leaflet's external CSS and JavaScript files!

So, in our header, let's add the following:

```
<!-- 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>
```

Now, let's go ahead and add a container for our map.

After `<div id="main"></div>` add a new `<div></div>` tag, and give it an ID attribute of "map":

index.html

```
<div id="main">
    <div id="map"></div>
</div>
```

With our container ready to go, open up the JavaScript file again and add the following Leaflet code template:

js/init.js

```
1  // JavaScript const variable declaration
2  const map = L.map('the_map').setView([34.0709, -118.444], 15); (1)
3
4  // Leaflet tile layer, i.e. the base map
5  L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
6      attribution: '&copy; <a
7  href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
8  contributors'
9  }).addTo(map); (2)
10
11 //JavaScript let variable declaration to create a marker
12
```

```
    let marker = L.marker([34.0709, -118.444]).addTo(map) 3
            .bindPopup('Math Sciences 4328 aka the Technology Sandbox<br> is the
    lab where I work in ')
            .openPopup();
```

1. `L.` is the `Leaflet` class that allows us to use built-in Leaflet tools. The `.` is like a chain of commands, but similar to `css`, Leaflet wants us to know `where our map is!` In the documentation, the map constructor expects an ID, which we called `the_map`. We then use `setView` to set the `Latitude (y)`, `Longitude (x)`, and `Zoom` of the initial map.
2. We use a `L.tilelayer` class to add a basemap to our map.
3. We use the `L.marker` to add a point to our map. Notice there is no `;` here because the code continues.

Can you see the usage of latitude and longitude in the code?

What is latitude and Longitude

- Latitude ranges from -90.0 to 90.0 and measures the distance north or south from the equator (**y-axis**).
- Longitude ranges from -180 to 180 measures distance east or west of the prime meridian (**x-axis**).

What is `L.map` and `L.tile`?

`L.map` is Leaflet's lingo for its own mapping Application Programming Interface (API). Every API has its own unique language to utilize it. To learn more about Leaflet's API visit here: https://leafletjs.com/reference-1.7.1.html

⚽ Class Exercise #4 - Adding more markers

Tasks

1. Add some new markers!
2. Customize the initial map
3. Add a new element `<div id="contents">` inside the `main` element.
4. Optional: change the base map or add some `html` into the marker popups.

Looking at the code above a little bit, we can see some latitude/longitude pairs. Your task is to copy the marker code add more markers of your choosing.

> ⚠️ **Unique variable names**
>
> When you create new marker variables, you **must** give the marker variable a new name, like `marker2` or you will simply override the previous marker! 😭

To find latitude/longitude of coordinates, you can use this website or another tool or your choosing:

- https://www.latlong.net/

Optional: Not happy with the basemap?

See if you can switch the basemap out by visiting the following link and changing `L.tileLayer` on your map:

- https://leaflet-extras.github.io/leaflet-providers/preview/

🏁Checkpoint

Check to see if your code looks likes the following before moving on:

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>
            <!-- hint: you can make a menu with other links here if you'd
like -->
        </header>

        <div class="main">
            <div id="contents">
                <!-- hint: the majority of your assignment 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

```
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 const variable declaration
const map = L.map('the_map').setView([34.0709, -118.444], 15); // (1)!

// Leaflet tile layer, i.e. the base 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); // (2)!

//JavaScript let variable declaration to create a marker
let marker = L.marker([34.0709, -118.444]).addTo(map) // (3)!
        .bindPopup('Math Sciences 4328 aka the Technology Sandbox<br> is the
lab where I used to work!')
        .openPopup();
```

Last update: 2023-04-09

Hello Leaflet… For realz..

Returning home to the HTML/CSS/JS analogy

Recall from last week's lab (before things went downhill) and the pre-lab reading that a webpage is like a house:

- HTML is the scaffolding/foundation of the house
- CSS is the paint, carpets, etc. that makes the house look nice
- JavaScript is the **appliances** that adds function to the house

Today we will be focusing on the **appliances** in more detail.

Why start with HTML and CSS first?

In order to do any JavaScript coding, you need to make sure your content has a place to show up! Additionally, with Leaflet, we actually need to bring in Leaflet's CSS and JavaScript files into our HTML house. Recall that last week, the map did not show up because I forgot to include Leaflet!

Also, recall that we can do that by **externally linking** Leaflet just like we did with our CSS (`styles/style.css`)!

Prepping our HTML

Let's keep all our content in the `.main` class, but move all our portfolio information into a new `div` with an `ID` of `contents` :

```
        <div class="main">
            <div id="contents">
                <!-- hint: the majority of lab 1 assignment can go here --> 1
            </div>
        </div>
```

1. You should be able to paste the `main` contents of your first lab assignment here.

Compatibility with Lab assignment #1

If you are using your `week1` lab assignment, you can copy and paste anything with the `div` elements inside the hint! Of course, this will not work if you have different `div` or changed the `css` too much! Adapt as needed!

In our `index.html` header tag right before `</header>`, we will need to add the Leaflet.js library. We add it right before the end to make sure they are the last items to load before the body starts loading.

Linking Leaflet.js files

index.html

```
<!-- 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>
```

Adding `the_map` container

In our body, our map needs a **PLACE** to go, so let's add a `<div>` for our map.

`divs` vs. `spans`, what's the `DIV`fference?

`divs` are generic `HTML elements` that stand for `division` of content, you can think of them as boxes of content. `Spans` are like divs, but for text content. Check out Mozilla Developer Network for more information:

- `elements`
- `divs`
- `spans`

After the `<div id="contents"></div>` element, add a new `<div></div>` element, and give it an **ID** attribute of `the_map` like the follwing:

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

Our current `index.html` should look like the following:

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>
            <!-- hint: you can make a menu with other links here if you'd
like -->
        </header>

        <div class="main">
            <div id="contents">
                <!-- hint: the majority of lab 1 assignment can go here -->
            </div>
            <div id="the_map"></div>
        </div>
        <div id="footer">
            Copyright(2022)
        </div>
    </body>
</html>
```

Prepping the CSS

Let's incorporate our new `div` that has the `map` `id` into our CSS now.

Adding `the_map` container

```
#the_map{ (1)
    height:80vh; (2)
```

```
    grid-area: main_map; 3
}
```

1. `#the_map` tells this CSS selector to look for an `ID` called `the_map`
2. `height` defines how tall this `div` should be, `80vh` means 80% of the vertical height.
3. We are going to name this `grid-area` the following: `main_map` it will be referenced in the `.main` class, but you can name it whatever you want; just be consistent.

Why `height: 80vh;`?

For Leaflet's map to show, it must have a height defined in order to show up. So you can play around with the height in your assignments, but be sure to include some height!!

Putting `the_map` `id` into our main `class`

Find our previous CSS style for the `main-content` it should be a class called `main`. Remember, because it is a `class` it starts with a `.` NOT a `#` like IDs do!

IDs vs. Classes

- IDs: There can only be ONE unique ID on a HTML page and in **CSS** you refer to it with a `#`, like `#the_map`.
- Classes: There can be multiple classes on an HTML page and in **CSS** you refer to it with a `.`

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

Our CSS should look like the following:

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;
}
```

With our container ready to go, open up the JavaScript file again and add the following Leaflet code template:

js/init.js

```
// JavaScript const variable declaration
const map = L.map('the_map').setView([34.0709, -118.444], 15); (1)

// Leaflet tile layer, i.e. the base 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); (2)

//JavaScript let variable declaration to create a marker
let marker = L.marker([34.0709, -118.444]).addTo(map) (3)
        .bindPopup('Math Sciences 4328 aka the Technology Sandbox<br> is the
lab where I work in ')
        .openPopup();
```

1. `L.` is the `Leaflet` class that allows us to use built-in Leaflet tools. The `.` is like a chain of commands, but similar to `css`, Leaflet wants us to know `where our map is!` In the documentation, the map constructor expects an ID, which we called `the_map`. We then use `setView` to set the `Latitude (y)`, `Longitude (x)`, and `Zoom` of the initial map.

2. We use a `L.tilelayer` class to add a basemap to our map.
3. We use the `L.marker` to add a point to our map. Notice there is no `;` here because the code continues.

Can you see the usage of latitude and longitude in the code?

> **What is latitude and Longitude**
>
> - Latitude ranges from -90.0 to 90.0 and measures the distance north or south from the equator (**y-axis**).
> - Longitude ranges from -180 to 180 measures distance east or west of the prime meridian (**x-axis**).

What is `L.map` and `L.tile`?

`L.map` is Leaflet's lingo for its own mapping Application Programming Interface (API). Every API has its own unique language to utilize it. To learn more about Leaflet's API visit here: https://leafletjs.com/reference-1.7.1.html

⚽ Class Exercise #2 - Adding more markers

> **Tasks**
>
> 1. Add some new markers!
> 2. Customize the initial map
> 3. Optional: change the base map or add some `html` into the marker popups.

Looking at the code above a little bit, we can see some latitude/longitude pairs. Your task is to copy the marker code add more markers of your choosing.

> **Unique variable names**
>
> When you create new marker variables, you **must** give the marker variable a new name, like `marker2` or you will simply override the previous marker! 😭

To find latitude/longitude of coordinates, you can use this website or another tool or your choosing:

- https://www.latlong.net/

Optional: Not happy with the basemap?

See if you can switch the basemap out by visiting the following link and changing `L.tileLayer` on your map:

- https://leaflet-extras.github.io/leaflet-providers/preview/

🏁Checkpoint

Check to see if your code looks likes the following before moving on:

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>
            <!-- hint: you can make a menu with other links here if you'd
like -->
        </header>

        <div class="main">
            <div id="contents">
                <!-- hint: the majority of lab 1 assignment 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

```
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 const variable declaration
const map = L.map('the_map').setView([34.0709, -118.444], 15); // (1)!

// Leaflet tile layer, i.e. the base 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); // (2)!

```

```
//JavaScript let variable declaration to create a marker
let marker = L.marker([34.0709, -118.444]).addTo(map) // (3)!
        .bindPopup('Math Sciences 4328 aka the Technology Sandbox<br> is the lab where I used to work!')
        .openPopup();
```

Last update: 2023-04-09

JavaScript FUNctions

Before we dive into functions, we need to talk a little more about JavaScript!

Some 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
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

Let's warm up by using some variables in our `init.js` file.

js/init.js

```
// original code
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);

// adding markers
let work = L.marker([34.0709, -118.444]).addTo(map)
        .bindPopup('Where I used to work on campus')

let home = L.marker([37.7409, -122.484]).addTo(map)
        .bindPopup('Family in SF')

let random = L.marker([39.7409, -122.484]).addTo(map)
        .bindPopup('Third Point')
```

Time for FUNctions

Programmers are often programming because they have to get something done, but a true programmer likes to automate (as well as copy and paste).

Let's edit our `init.js` and replace the `marker` variable with the following:

js/init.js

```
// adding markers
let work = L.marker([34.0709, -118.444]).addTo(map)
        .bindPopup('Where I work on campus')

let home = L.marker([37.7409, -122.484]).addTo(map)
        .bindPopup('Family home in San Francisco')
```

```
let random = L.marker([39.7409, -122.484]).addTo(map)
        .bindPopup('Third Point')
```

Your `init.js` should look like this:

js/init.js

```
// original code
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);

// adding markers
let work = L.marker([34.0709, -118.444]).addTo(map)
        .bindPopup('Where I work on campus')

let home = L.marker([37.7409, -122.484]).addTo(map)
        .bindPopup('Family home in San Francisco')

let random = L.marker([39.7409, -122.484]).addTo(map)
        .bindPopup('Third Point')
```

Would it be cumbersome to add 10 points like this? What about 100? 1,000?

That's where functions come in handy!

Our first function!

Functions are declared like variables by a keyword, however instead of `let`, `const`, or `var` we use the keyword… `function`, so original!

A basic function looks likes this:

```
function our_first_function(){
    console.log('hello from our first function')
}
```

Let's try to apply what this looks like with marker creation!

We know from the documentation and our previous usage that `L.marker`, needs `latitude` and `longitude`. So, we can automate the marker creation by creating a function like this:

js/init.js

```
function addMarker(lat,lng,message){ 1
    console.log(message) 2
    L.marker([lat,lng]).addTo(map).bindPopup(message) 3
    return message 4
}
```

1. `function` is the declaration of our function, `addMarker` is the name, and `lat,lng,message` is the parameter, which are **passed in** to a function to be utilized. `Parameters` are optional, but parentheses `()` are not!! The `{` is the begining of the function.
2. The `console.log` in the body will tell us if the function is working.
3. Here we use the `L.marker()` to add a marker
4. The return is used to exit a function and return a value.

Why did we include a third parameter called `message`?

It allows us to customize our popups!

Notice how the how function accesses our parameters: - `L.marker` uses `lat`, `lng` - `bindPopUp` uses `message`

Function parameters

You can pass in variables into functions and multiple parameters are seperated by a comma. In this function, there are 3 parameters: `(lat,lng,message)`. Remember that if you even if you have NO parameters, you must include the parenthesis `()` like follows: - `js#! function our_first_function(){return "hello world"}`

Go ahead and check the console!

WHAT?! Nothing has changed! 🤔

Using Functions

In order for a function to run, it needs to be “plugged-in”. This is called “invoking” or “calling” the function. When a function has no parameters, you can call it like so:

```
    function_name()
```

But since our function does have parameters (namely the `lat`, `lng`,and `message`), you must specify what those are when you call the function.

Add this to the end of our `init.js` file:

> js/init.js

```
addMaker(37,-122,'you are awesome! you automated a marker function')
```

> ⚠ **Warning about the order of parameters!**
>
> The order of the parameters (`lat`, `lng`, `message`) is must be **SAME** order that the function reads them!! Try switching `37` and `-122` to see what I mean.

Now your console should return the “message” AND you should see a new marker on the map!

Inside `function` blocks you can create variables, change HTML, and do all sorts of things like play videos and even create games.

⚽Class Exercise #3 - Using the marker function

Create your own marker function that does the following:

- Utilizes at least `four parameters`
- Declare a `new variable` inside the function
- `Returns` a value

Use your function to create 3 markers with it.

Done

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

// use the function
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')
```

If you finished early, try these extra challenges: - Try to style your pop-up with 2 attributes!

Bonus Exercise - Create your own function

Create your own function that does the following:

- Utilizes at least `two parameters`
- Declare a `new variable` inside the function
- `Returns` a value

Bonus Answer

```
// create function
function addNumbers(value1,value2){
    let result = value1 + value2
    return result
}

// use the function
addNumbers(1,10)   // result: 11
```

Shortcut: String Literals

```
let popup = `${zoomLevel} + ${zoomLevel}`
```

String literals or template strings allow you to subsitute variables into strings with the `${VARIABLE_NAME}` syntax inside the place holders.

Declaring a string with ` instead of `' '` or `" "`, allows you to convert `variables` to strings. For example, the zoom level normally would be treated as a number, but when we brought it in with the `${}` combination it became a string so it could not be summed.

This technique will be helpful for our pop-ups as follows:

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

1. Notice how `title` is added to the `<h2>` tag element and `message` is added to an `<h3>` element?

Alternative Method

You can also bundle the pop-up into a variable, and then use that to populate the `bindPopup()` call:

```
let popup_message = `<h2>${title}</h2> <h3>${message}</h3>`
L.marker([lat,lng]).addTo(map).bindPopup(popup_message)
```

🏁Checkpoint

js/init.js

```
// declare variables
let zoomLevel = 5;
const mapCenter = [34.0709,-118.444];

// use the variables
const map = L.map('the_map').setView(mapCenter, zoomLevel);

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
}

// use our marker functions
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')
```

index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Basic Leaflet Map</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>
        <div class="main">
            <div id="contents">
                <!-- hint: the majority of lab 1 assignment can go here -->
            </div>
            <div id="the_map"></div>
        </div>
    </body>
    <script src="js/init.js"></script>
</html>
```

Last update: 2023-04-09

✅ Final Template Code

index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Basic Leaflet Map</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>
        <div class="main">
            <div id="contents">
                <!-- hint: the majority of lab 1 assignment can go here -->
            </div>
            <div id="the_map"></div>
        </div>
    </body>
    <script src="js/init.js"></script>
</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 variables
let zoomLevel = 5;
const mapCenter = [34.0709,-118.444];

// use the variables
const map = L.map('the_map').setView(mapCenter, zoomLevel);

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
}

// use our marker functions
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')
```

Now you should be ready to take on the lab assignment!