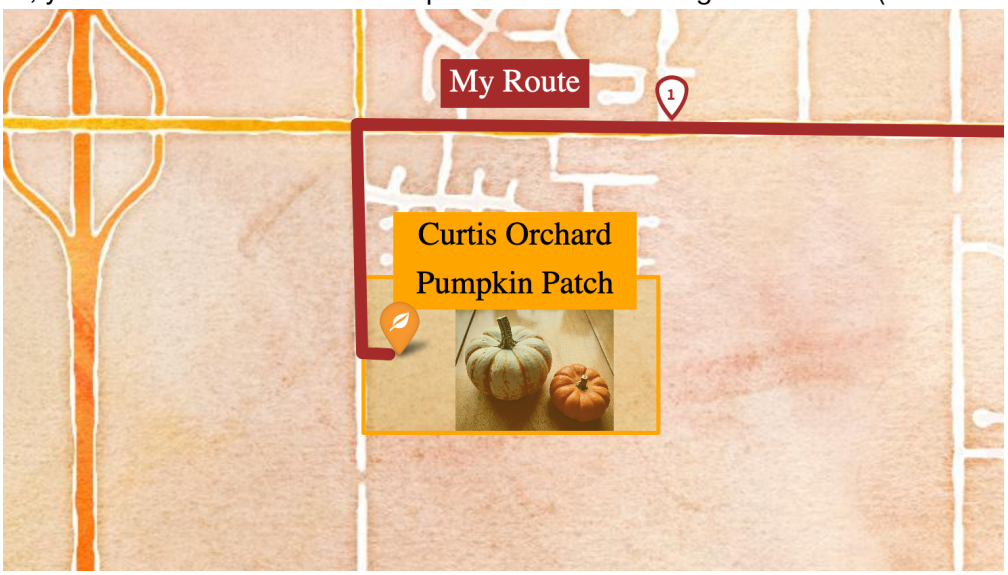
Create a Map Using Python!

If you haven't already, it's first good to check out the first notebook in the series: Finding Coordinates (Finding%20Coordinates.ipynb).

After this tutorial, you will be able to create a map that looks something like this one (code at the end):



Important Note for PDF Versions!

This notebook will look much worse on a PDF!! Maps are shifted to the upper left so that content at least appears, however background boxes and some icons do not display properly when saved in this way.

Part 1: Running Cells

Code is instructions written in a programming language. That means instructions that are written in a way that a computer can understand and follow. This notebook contains code that can be run, so you can look at the code and see what it does in the same place! Code can be run from **code cells**, which look like this:

```
In [ ]:
```

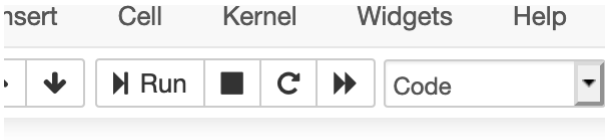
Code cells can contain text or code. If some text starts with this sign: `#` it is a **comment**, or a note for a human. The cell right below this sentence is a code cell that contains a comment.

```
In [1]: # This is a code cell that contains a comment!
```

When you **run** a code cell, a number will appear in between the brackets. Often some **output** will be made by running the cell. This will usually be displayed below the cell.

```
In [7]: 2+2
Out[7]: 4
```

To **run a cell** click within the grey box and press the `Run` button on the toolbar. You can see the `Run` button in the image below.



If you change something in a cell you need to **run it again** (press `Run` another time) to update the output.

When something doesn't run correctly an error will appear. That looks like this:

```
In [15]: 2+2m
           File "<ipython-input-15-25b71f23351f>", line 1
             2+2m
                ^
         SyntaxError: invalid syntax
```

Try running this cell! What do you think the output will look like?

Hint: think of the order of operations!

```
In [2]: 2-(2+1)
Out[2]: -1
```

A comment can also be used to prevent some code from running. This can be useful for testing out things!

```
In [3]: 2-(2+1) # this gives more information but the code still runs
Out[3]: -1

In [4]: # 2-(2+1) # now the whole line is a comment (no output)
```

Part 2: Introducing the Folium Python Library

For this tutorial, we will use the **Python** as the programming language. Python is a common language used for writing code. A **Python library** is a collection of code that someone else has already written that is intended to be used over and over again! We will use the **Folium library** to help map things. It's really useful because you won't have to build a whole system from scratch in order to create a cool map and display it on the screen!

First we need to tell Python that we are using the Folium library. We do that using the `import` command, where we type the name of the library in the form that Python is expecting (Python is expecting it to be lowercase) after the word `import` .

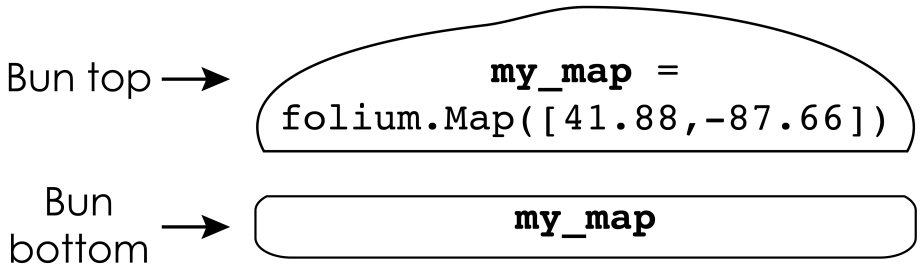
Run this next cell to tell Python to use the Folium library. Here we are just telling Python to use the library we want, so there is no output.

```
In [5]: import folium
```

Writing Python code is like writing out a recipe. There are some parts that need to be written very specifically in a certain form and others parts that are more flexible. Try to follow the examples closely until you learn what parts are flexible. If you have a space or comma in the wrong place or a missing bracket (`[` or `}`) or parenthesis `)` Python will likely get mad and you will get an error!

Creating Your First Map: Build a Map Sandwich!

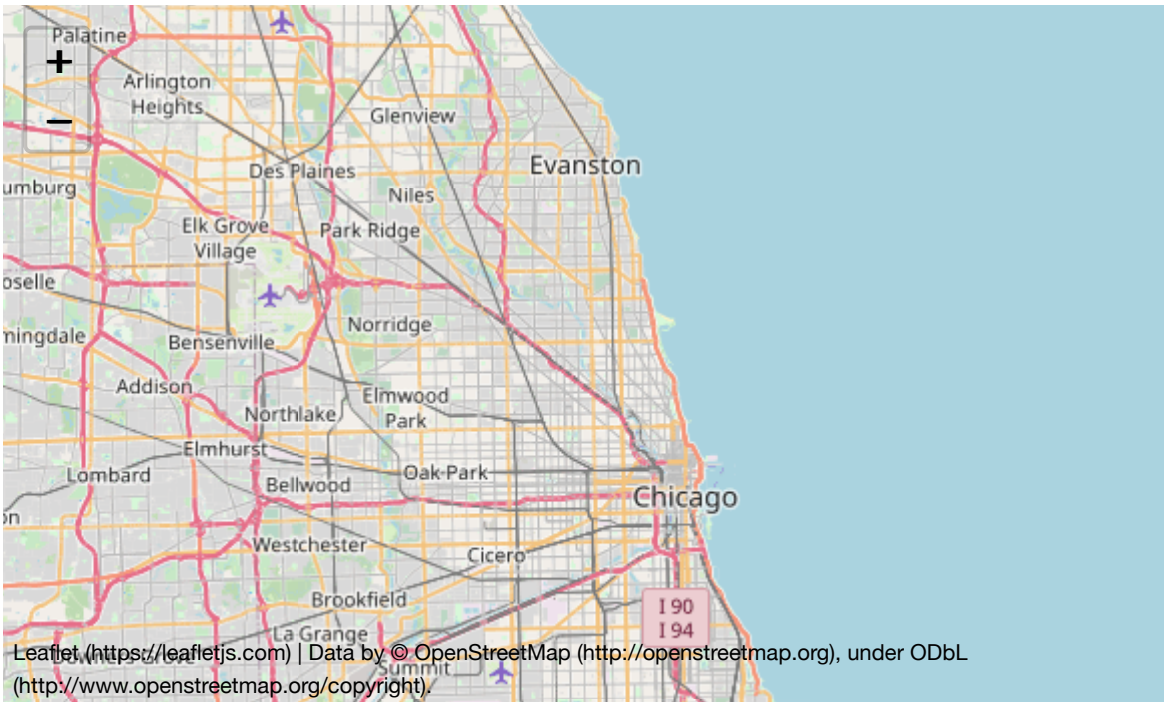
Creating a map with **Folium** is like building a sandwich. To create any map, you must have two pieces: we can call them the top of the bun and the bottom of the bun. Like so:



This diagram shows the simplest example of a map that you can make. You can see map this creates by **running** the next cell.

In [6]:
```
my_map = folium.Map([41.8233, -87.4158])

my_map
```

Out[6]:



Let's talk about this map

What do you see? Where is this? Take a moment to describe different things you can see on this map and what you can do with it.

I see quite a bit already! There is a background, with **blue** water for the lake and rivers, **tan** for the land, roads in **red**, **orange**, and **grey**, and parks in **green**. I see names of cities (like Chicago), neighborhoods, and interstates. I see symbols for airports ✈.

Also you can interact with the map! You can zoom in and out by clicking the + and − buttons and drag the map with your mouse to move from side to side. If you want to go back to the starting position, **run** the cell again!

All of this with 2 lines of code - not bad!

This basic part of the map is called the **base map**. Everything else you add to the map will go on top of this.

What are the pieces here?

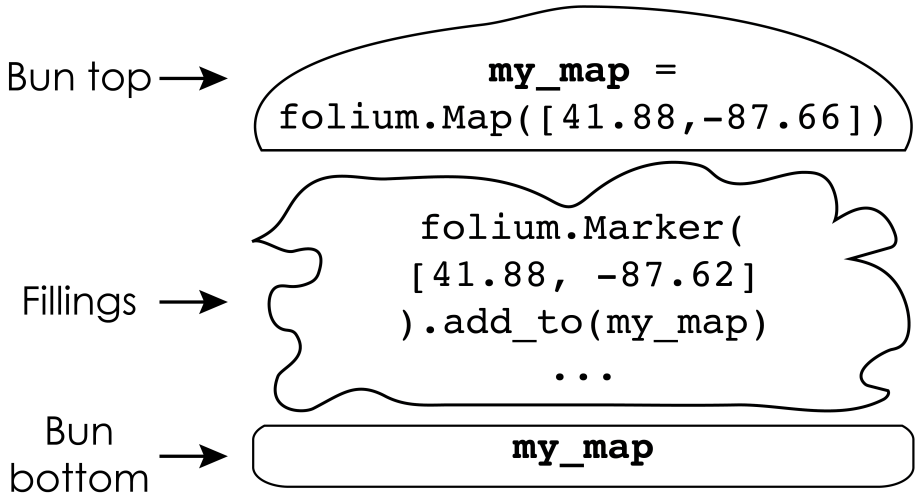
Lets break it down (these section introduces common ways to talk about each of the pieces):

- `my_map` - this is a **Python variable name**, which is a fancy way of saying the name that you call your map. Every time you need to refer to your map you need to use the same name written the same way (i.e. lower case with an underscore between the two words).
- `=` - this is the way in Python you link some data, in this case your actual map, to a variable name. You can say you assigned a unique version of a map object to the variable named `my_map`.
- `folium.Map(` - this is part of the recipe that you can't change and have it still work. This tells the Folium library that you want to make a map.
- `[41.88,-87.66]` - these should look familiar - these are the **coordinate number pairs** you saw in the 1st notebook. This tells the library where you want your map. (Do you remember what each number refers to?) They must be in the square brackets (`[]`) because that's how Folium wants it!
- `)` - this tells Folium that you have given it everything you wanted to make the first map. If you delete this Folium will be angry.
- `my_map` - this tells Python to actually show your map! Note that if you delete this your map will still be created but you won't see anything. That's why you should have this on your last line for every map you map in this tutorial.

And that's it!

However, it isn't a very interesting map yet is it? This sandwich needs some fillings!

Creating Your First Map: Adding Fillings!

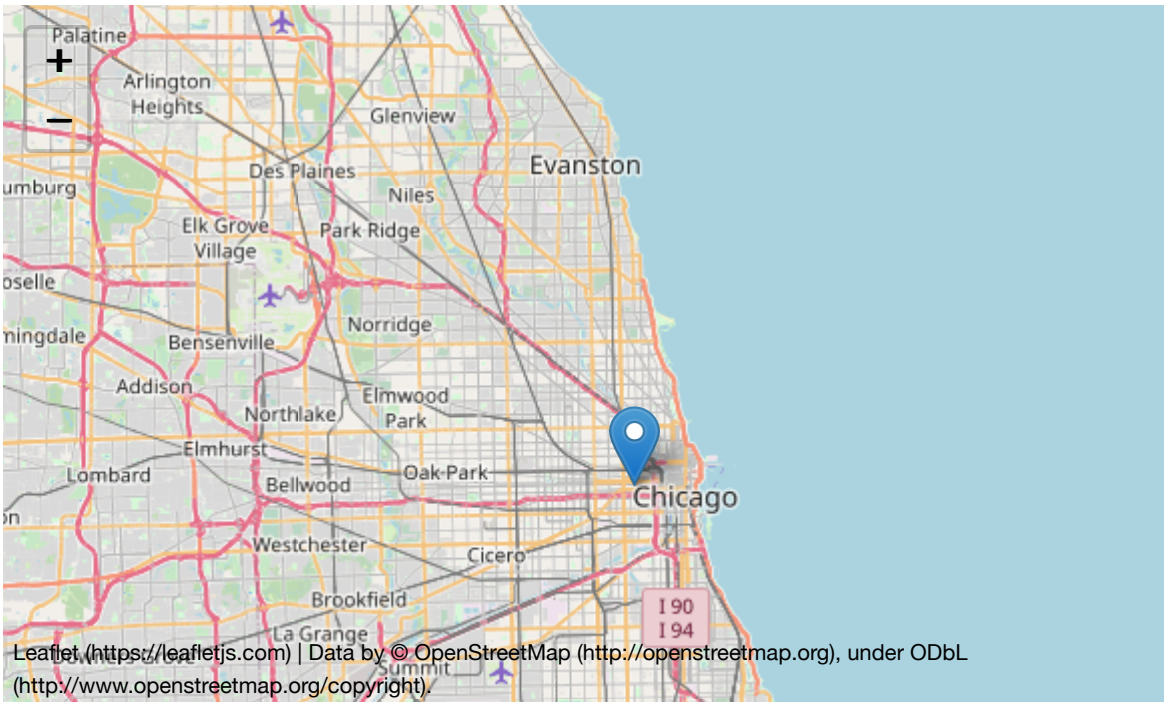


What I'm calling **fillings** here is actual content that you can add to your map. While an actual sandwich can only be so tall before you can't eat it, our maps can have as many fillings as you want and you can mix even filling types (we'll explore that later).

In the next cell we have a basic map with one filling: a **marker**. Think of a marker as a pin that you can stick in a certain point on a map. Before you run the next cell, try to imagine what you think the marker might look like.

In [7]:
```
my_map = folium.Map([41.8233, -87.4158])
folium.Marker([41.88, -87.66]).add_to(my_map)
my_map
```

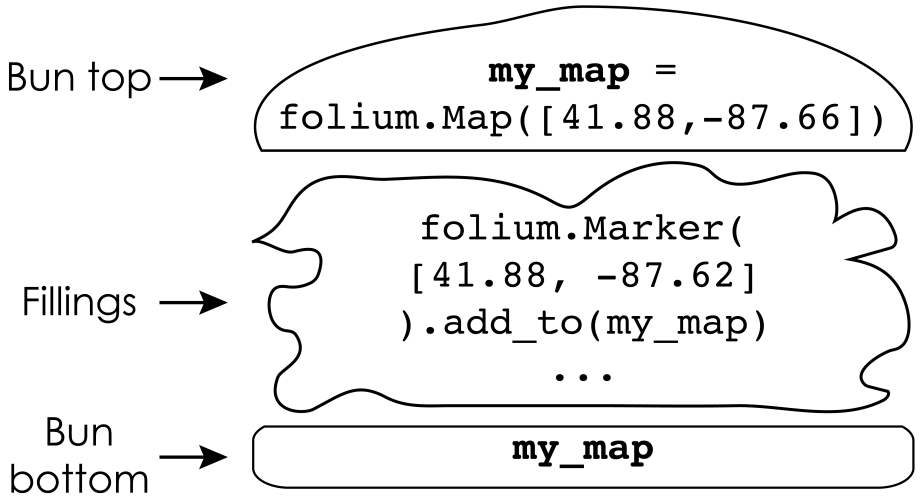
Out[7]:



The new pieces here are `folium.Marker()` and `.add_to(my_map)` . Folium needs the `.add_to()` function exactly as written to link your marker to your map. You link it by putting the variable name for your map, in this case `my_map` in the parenthesis after `.add_to` .

Part 3: Changing the Base Map

Coming back to the map as a sandwich, note that the top of the bun is thicker than the bottom and the filling section is the thickest. Here the top can expand a little bit, the fillings can expand a lot, but for now the bottom will always stay the same size.



Let's start with the top bun. The next cell is similar to our original maps, however more information is specified in the top bun line using the following **keywords**: `location` , `tiles` , and `zoom_start` . Basically everything that you can use to customize the **base map** needs to go between the parentheses (`()`) that appear after `folium.Map` . This tells Folium that these extra pieces of information go with the base map itself and not another part of the map.

Each keyword group inside of the parentheses contains three pieces: a `keyword` , an equals sign (=), and some information. If the keyword group is not at the end of the list, you need a comma (`,`) after the group to tell Python that another keyword group will come next. Note that keywords are case sensitive and need to be spelled exactly as they are shown here (including any underscores `_`) to work!

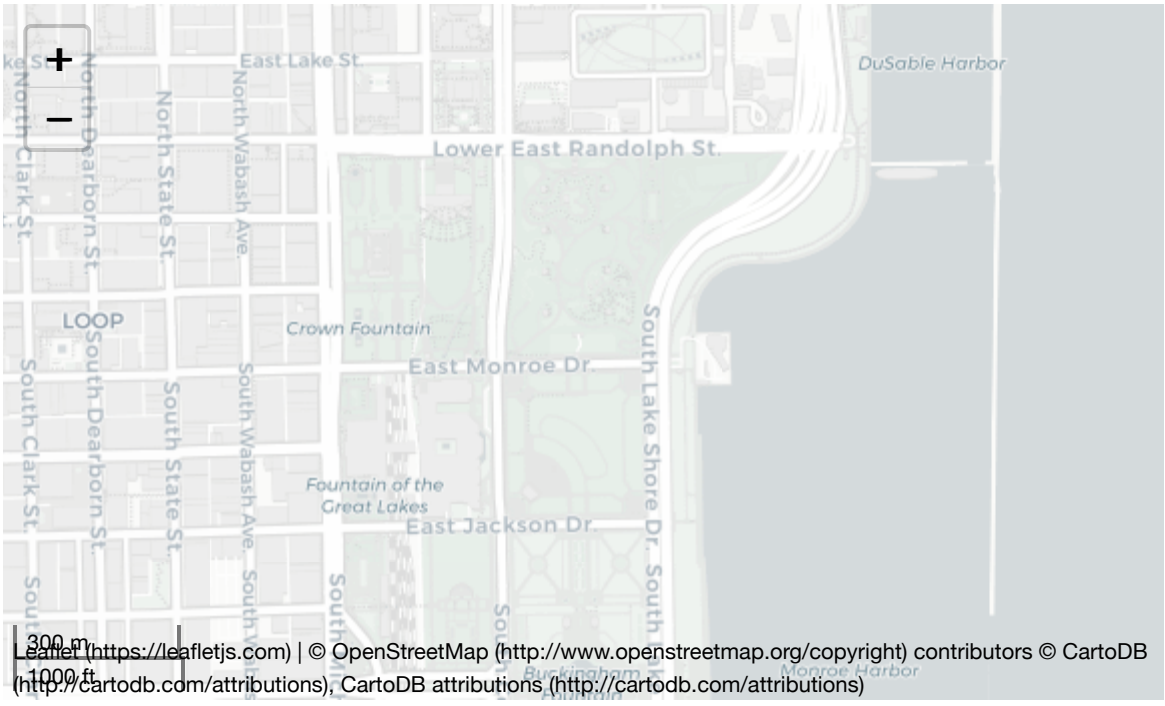
In the next example, we can change the background by changing the `tiles` **keyword** value. Possible background choices you can use with the `tiles` **keyword** include: 'OpenStreetMap' (the default option), 'Stamen Terrain', 'Stamen Toner', 'Stamen Watercolor', 'CartoDB positron', and 'CartoDB dark_matter'. Again, these must be written exactly as they are here. Experiment to find your favorite!

`zoom_start` sets the initial map zoom to a different value. Change the green number after `zoom_start` - when you make the number bigger what happens when you re **run** the cell? What do you think would happen if you made the number smaller?

Can you tell what the `control_scale` keyword does? Try changing **`True`** to **`False`** for this keyword (make sure only the first letter is uppercase). If it is written correctly it will show up as **bold and green**.

In [8]:
```
my_map = folium.Map(location=[41.8772, -87.6102], tiles='CartoDB positron',
                    zoom_start=15, control_scale=True)
my_map
```

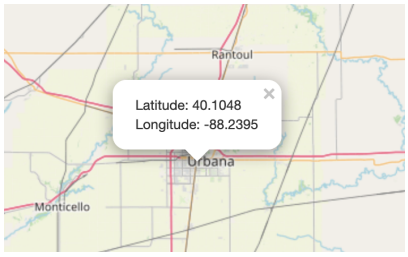
Out[8]:



Part 4: Easily Get Coordinates

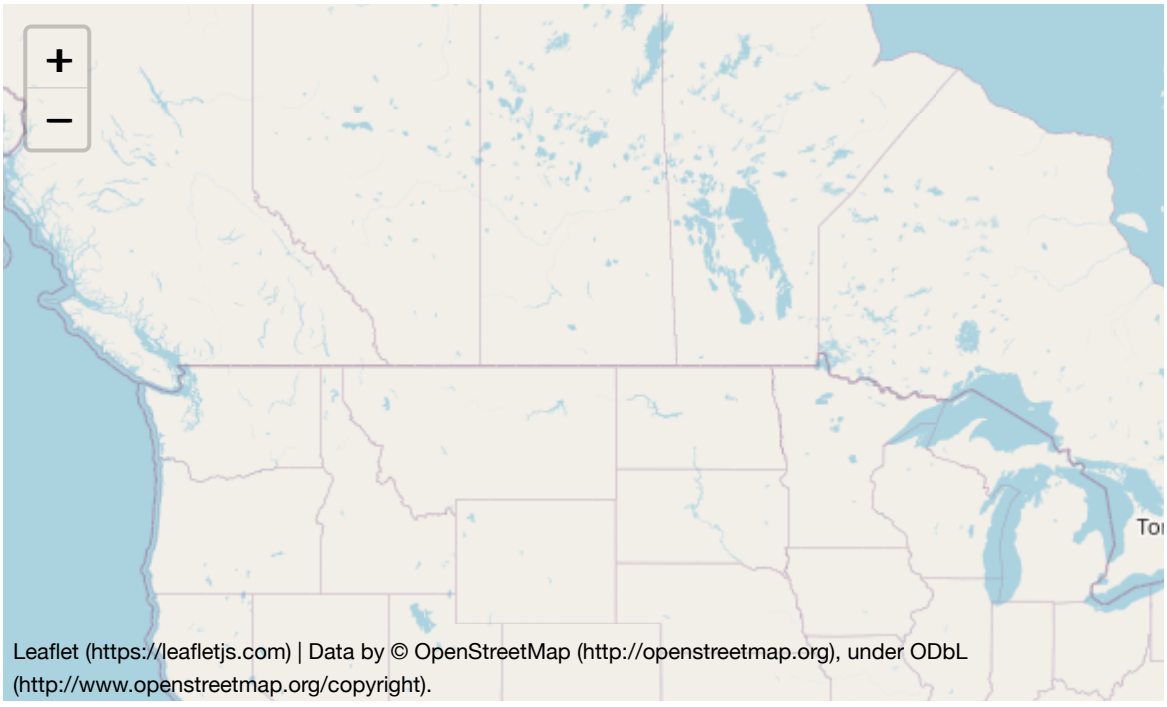
Now we pause really quick. The next section is going to involve many different locations. In the previous notebook on Finding Coordinates (Finding%20Coordinates.ipynb) we discussed how to find coordinates using Google Maps. Now that we know about Folium, let me show you another way!

Run the next cell. In the map in the next cell, click on the map to see a window pop up with the coordinates displayed in the window like in the image below. You can select the text with your mouse and copy and paste to easily find coordinates for things you want to map in the next section!



In [9]:
```
lat_long_map = folium.Map(location=[41.8746, -87.6240], zoom_start=4)
lat_long_map.add_child(folium.LatLngPopup())
lat_long_map
```

Out[9]:

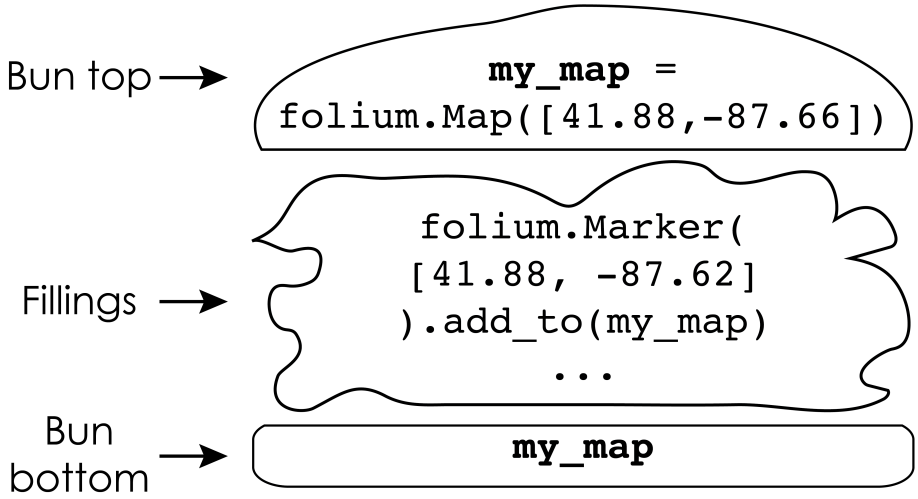


Part 5: Looking up more options

At this point in the tutorial you may want to look to see what different types of kewords and extra things Folium has that you can use! A master list can be found here: https://python-visualization.github.io/folium/modules.html (https://python-visualization.github.io/folium/modules.html) . You can search for the name of what you want to edit. Note that things like `folium.Map()` and `folium.Marker()` are called **classes**.

Note that this is written for people who code and it's OK if it is a little difficult to understand at first. Using what you have learned you should be able to try out new combinations.

Part 6: Adding and Expanding Different Types of Content in the Map



On to more **filling**, what you've been waiting for! You have already seen a **marker**. We will now show you more things you can do with a marker, and introduce some other things you can add to your map.

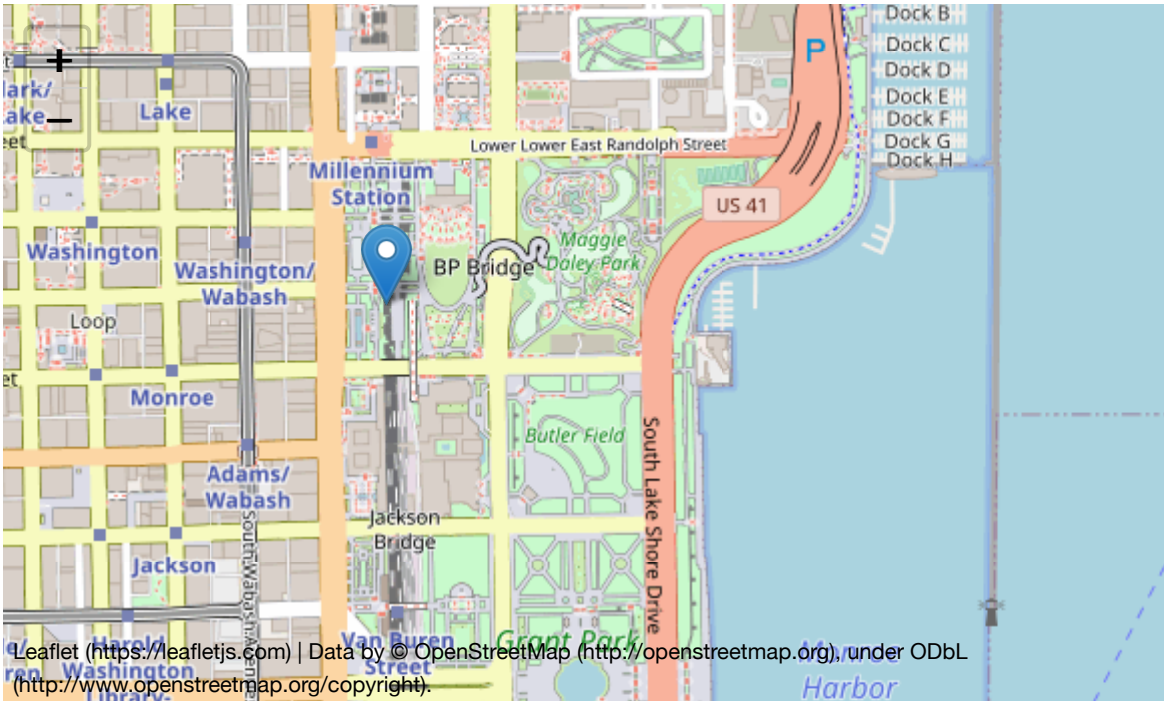
Each of these items needs to be placed at a specific location, so keep your coordinates in mind as you go on!

Adding content to a Marker

Look at the code before you **run** this next cell. What do you think the new lines might do?

In [10]:
```
my_map = folium.Map(location=[41.8772, -87.6102],
                    zoom_start=15)

folium.map.Marker(location=[41.881832,-87.623177],
                  popup="Hello, I'm a popup",
                  tooltip='This is a tooltip').add_to(my_map)
my_map
```

Out[10]:



We've added a few more keywords!

- `popup` controls the text that you see when you **click on** the marker.
- `tooltip` contains the text that you see when you **hover your mouse over** the marker.

Try both of these actions if you haven't yet!

Note that the red text in the code cell is called a **string** of text or a **text string**. We've seen these before with the map `tiles` **keyword** content. Strings need to be surrounded by either single or double quotation marks (`'` or `"`). If your text you want to enter contains one of these quotation marks, you need to use the other mark to surround the string. If you use both, you can use a set of three quotation marks (you can see an example below).

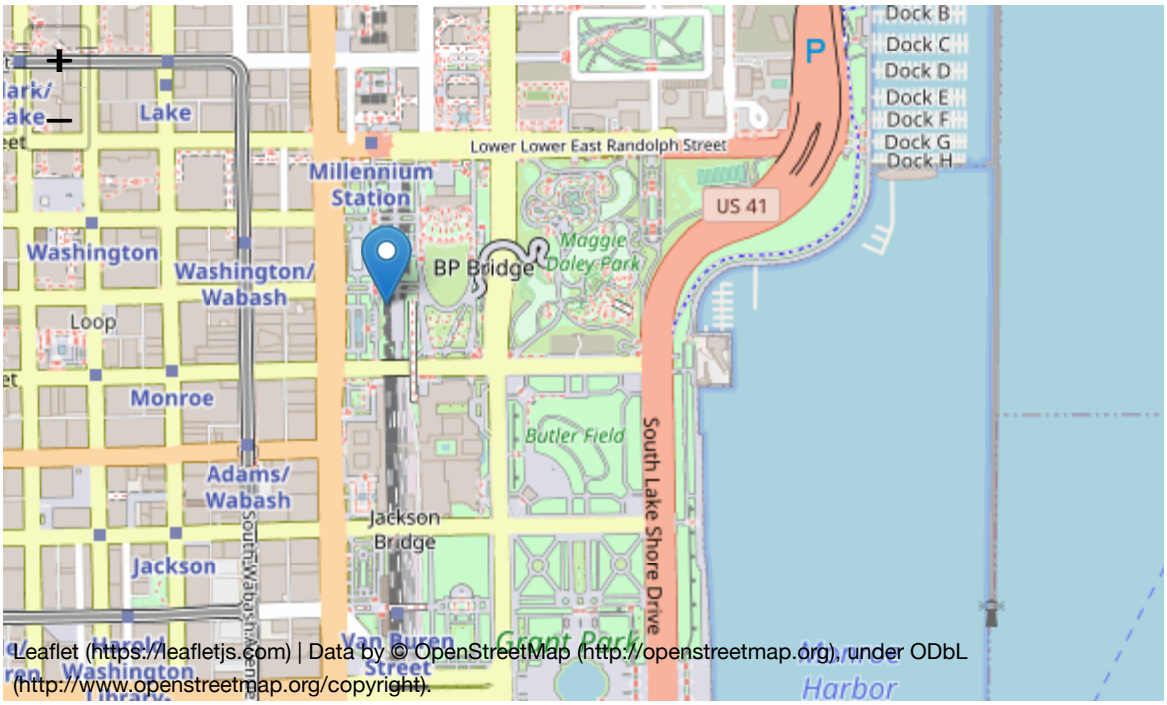
You can also store a **text string** as a variable just like you have been storing your map in a variable with the name `my_map` . I've stored some text in a variable called `new_tooltip_text` in the next cell. Remember you have to **run** the cell for Python to capture your code and be able to use the variable later.

In [11]:
```
new_tooltip_text = """The example's code says:<br>"this is<br>a tooltip"! """
```

In the next map, instead of typing the text for the `tooltip` **keyword**, we typed the variable name for the **text string** I just created in the previous cell. `<br>` is an example of an **HTML tag** (learn more about HTML (https://www.w3schools.com/html/html_intro.asp)). Make a wild guess: How do you think the `<br>` tags will be displayed in the tooltip?

In [12]:
```
my_map = folium.Map(location=[41.8772, -87.6102], zoom_start=15)

folium.map.Marker(location=[41.881832,-87.623177],
                  tooltip=new_tooltip_text).add_to(my_map)
my_map
```

Out[12]:



Adding Multiple Markers

The `<br>` **HTML tag** tells Folium to switch to the next line at the point that the `<br>` tag is entered. You can use **HTML** to format some of the text you display in your map, including in the popup window.

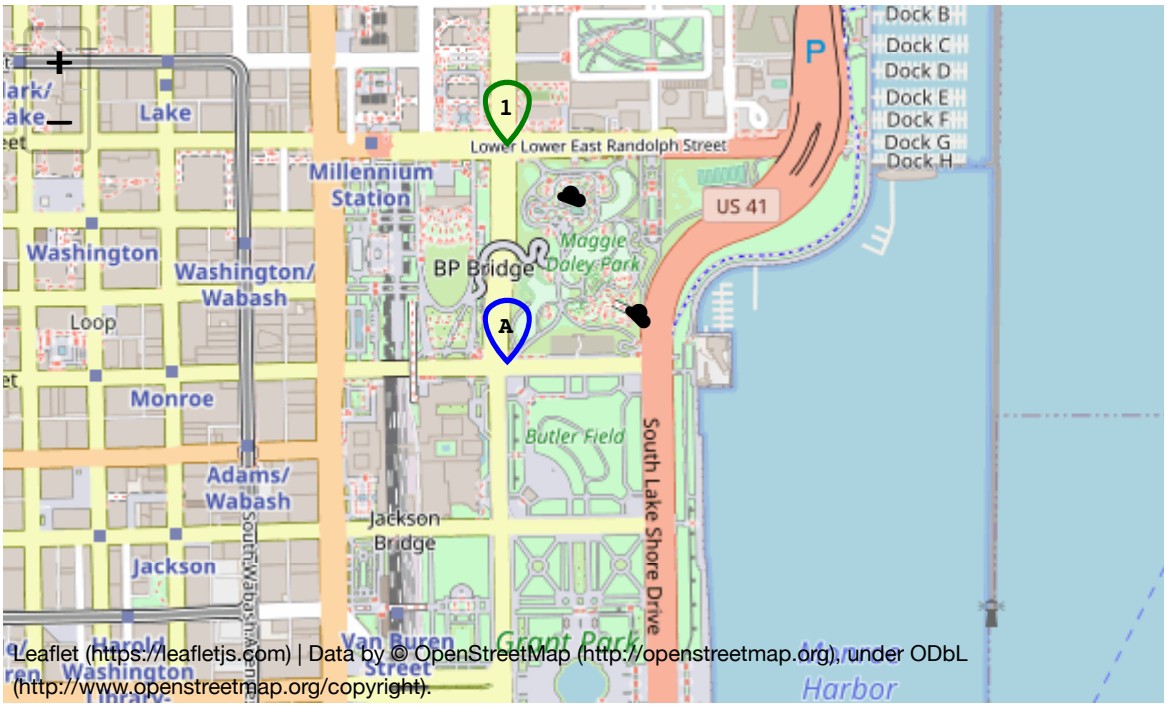
We can also change the icon used for the marker, as in the cell below. Note here that we are providing a `folium.Icon()` to the `icon` **keyword**. This is a specific format you need to use to provide more information to Folium about how you want the icon displayed. You need to use this format because Folium has already put together code as a shortcut so you don't have to do a ton of work to change the icon around.

You can add multiple markers by copying and pasting the whole `folium.map.Marker(...).add_to(my_map)` code and placing the second one after the first. Use this same technique to add multiple of other types of fillings.

Remember the markers are **filling** so your bottom bun (`my_map`) must always be the last line!

In [13]:
```
from folium import plugins # this line is needed for BeautifyIcon

my_map = folium.Map(location=[41.8772, -87.6102], zoom_start=15)

folium.map.Marker(location=[41.880932,-87.617777],
                  icon=folium.Icon(color='red', icon='cloud', angle=4
5),
                  tooltip="red cloud"
                 ).add_to(my_map)

# The coordinates for this marker are slightly changed!
folium.map.Marker(location=[41.882832,-87.619177],
                  icon=folium.Icon(color='orange', icon='cloud', angle
=20),
                  tooltip="orange cloud"
                 ).add_to(my_map)

# We can use a slightly different icon 'recipe' to add text or numbers
to an icon
folium.map.Marker(location=[41.880832,-87.620577],
                  icon=folium.plugins.BeautifyIcon(border_color='blue
',
                                   text_color='blue',
                                   number='A',
                                   icon_shape='marker'),
                  tooltip="blue A"
                 ).add_to(my_map)

folium.map.Marker(location=[41.884332,-87.620577],
                  icon=folium.plugins.BeautifyIcon(border_color='Green
',
                                   text_color='green',
                                   number=1,
                                   icon_shape='marker'),
                  tooltip="green 1"
                 ).add_to(my_map)

my_map
```

Out[13]:



Adding Text to the Map

You may want to add some **text** to your map! This is useful for labeling things!

You can add text using `folium.DivIcon()` with the `icon` keyword instead of `folium.Icon()`. This allows you to display **HTML** directly on your map. `div` is an **HTML tag**. This type of **HTML tag** has two parts, the first part, `<div>`, that marks the beginning of the tag, and the second part `</div>` that marks the end. The text in the middle between the two tags can be thought of as the contents of the tag. The `<br>` tag didn't need two parts because a line break can not contain anything.

It can be easier to work with **HTML** if you use three quotation marks to surround the **text string**. That way Python won't get mad if you divide the text in multiple lines.

In [14]:
```
my_map = folium.Map(location=[41.8772, -87.6102],  tiles='CartoDB positron', zoom_start=15)

folium.Marker(location=[41.881832,-87.620177],
              icon=folium.DivIcon(
                  icon_size=(100, 100),
                  html='''<div>This is some text on the map</div>''')
             ).add_to(my_map)

my_map
```

Out[14]:

The text is there but it can be hard to see on a regular map background.

This next example is slightly more complicated. The extra values in the `div style` are used to control more aspects of how the text is displayed. `style` is a **keyword** in **HTML** that provides more information about the `div` tag.

Note that these keyword groups are slightly different than the ones we've seen before. Instead of parenthesis, all the keyword groups are contained in double quotation marks. So the entire keyword container structure looks like `style="  "` . In the container, each keyword group contains a keyword, such as `font-size` , a colon (`:`), and more information. Here keyword groups are separated by a semi-colon (`:`).

In [15]:
```
my_map = folium.Map(location=[41.8772, -87.6102], zoom_start=15)

folium.Marker(location=[41.881832,-87.623177],
              icon=folium.DivIcon(
                  icon_size=(180, 20),
                  html='''<div style="
                  font-size: 18pt;
                  font-family: serif;
                  color: black;
                  text-align: center;
                  background-color: white;">
                  This is some text on the map
                  </div>''')
              ).add_to(my_map)

my_map
```

Out[15]:

Adding an Image

You can use `folium.raster_layers.ImageOverlay()` to add an image.

The hardest part is telling Folium where the corners of your image are. The `bounds` **keyword** expects a set of two coordinates like this: `[[lat_min, lon_min], [lat_max, lon_max]]`, where `lat` stands for latitude and `lon` stands for longitude. In practice though it's fairly flexible.

You will probably need to try different coordinates for the `bounds` keyword to place the image where you want it.

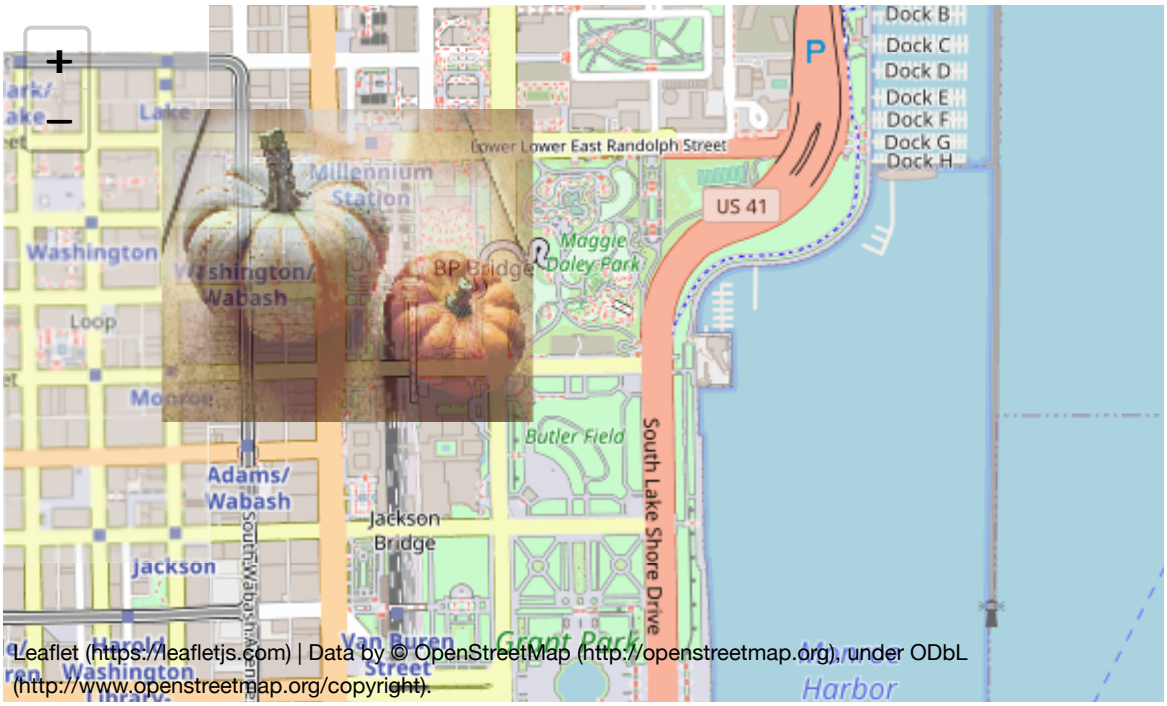
The `opacity` keyword can be used to make the image more or less transparent (see-through).

Note that if you move the map the image moves too so that it stays in the same spot on the map.

Also note that having too many images can make your map load slowly!

In [16]:
```
my_map = folium.Map(location=[41.8772, -87.6102], zoom_start=15)

folium.raster_layers.ImageOverlay(
    image='map photos/pumpkins.png',
    bounds=[[41.885, -87.620], [41.880, -87.628]],
    opacity=0.6).add_to(my_map)

my_map
```

Out[16]:

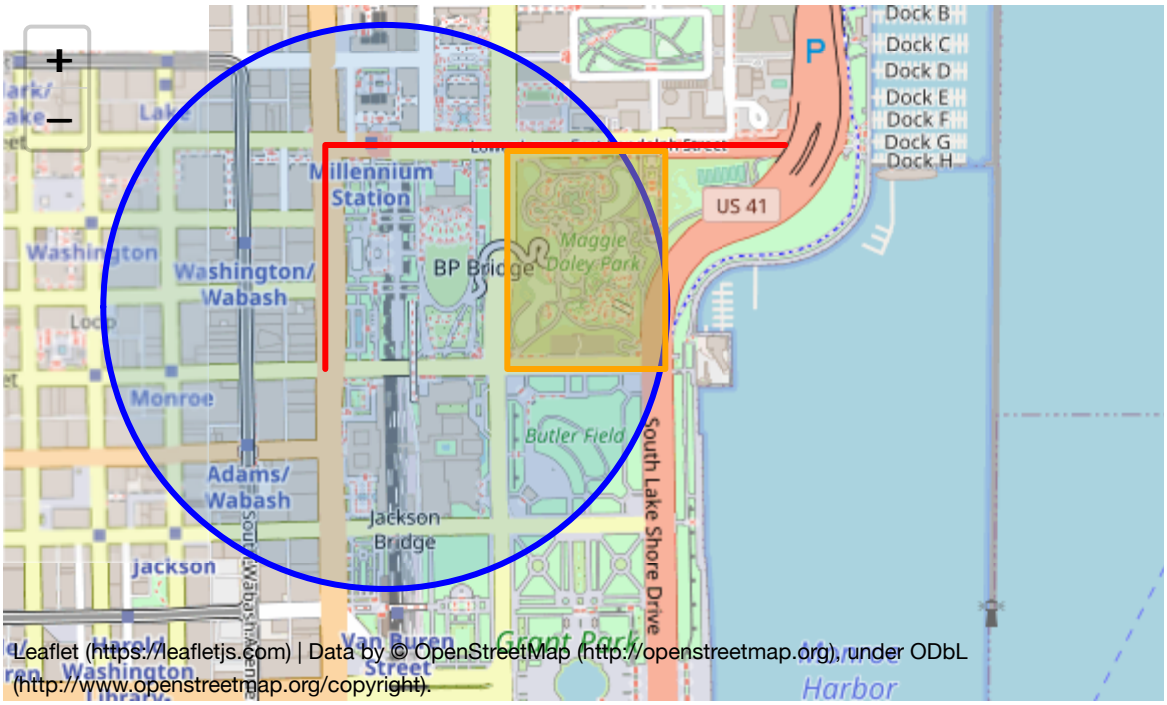


Adding Shapes to the Map

Shapes work similarly to adding an image, but you use a different name for each shape. For example, `folium.Circle()` adds a circle to the map. Remember you can look at https://python-visualization.github.io/folium/modules.html (https://python-visualization.github.io/folium/modules.html) for more explanation about what each shape needs, but you can easily copy and paste these shapes to get you started!

In [17]:
```
my_map = folium.Map(location=[41.8772, -87.6102], zoom_start=15)

# notice the order: first circle, then line, then rectangle
folium.Circle(
    radius=500,
    location=[41.881832,-87.623177],
    popup='Downtown',
    color='blue',
    fill=True,
    fill_color='#3186cc'
).add_to(my_map)

folium.PolyLine(
    locations=[(41.880832,-87.624477),
               (41.884432,-87.624477),
               (41.884432,-87.614577)],
    popup='My Path',
    color='red',
).add_to(my_map)

folium.Rectangle(
    bounds=[(41.880832,-87.617177), (41.884332,-87.620577)],
    popup='Maggie Daley Park',
    color='orange',
    fill=True,
    fill_color='#ccb031',
    fill_opacity='0.4'
).add_to(my_map)

my_map
```

Out[17]:



Part 7: Creating the Example Map from the Start of this Notebook

Finally, we will use what we have learned to create the map you saw at the beginning! After this, you can start creating your own similar map!

In [18]:
```
import folium # if you use this map for reference you need this line
from folium import plugins # this line is needed for BeautifyIcon

my_map = folium.Map(location=[40.0656, -88.2810], tiles="Stamen Waterc
olor", zoom_start=15)

folium.raster_layers.ImageOverlay(
    image='map photos/pumpkins.png',
    bounds=[[40.0650, -88.2920], [40.0620, -88.2870]]).add_to(my_map)

folium.Marker(location=[40.063868, -88.293814],
                  icon=folium.Icon(icon='glyphicon-leaf', color='orang
e')).add_to(my_map)

folium.map.Marker(location=[40.069467, -88.285145],
                  icon=folium.plugins.BeautifyIcon(border_color='brown
',
                                   text_color='brown',
                                   number=1,
                                   icon_shape='marker'),
                  tooltip="stop 1"
                 ).add_to(my_map)

folium.Rectangle(
    bounds=[(40.065749, -88.294934), (40.061983, -88.285556)],
    color='orange',
    fill=True,
    fill_color='#ccb031',
    fill_opacity='0.3'
).add_to(my_map)

folium.PolyLine(
    locations=[
                (40.069067, -88.249264),
                (40.069467, -88.295145),
                (40.063925, -88.294993),
                (40.063898, -88.294112)],
    color='brown',
    weight=9
).add_to(my_map)

folium.Marker(location=[40.0670, -88.290145],
               icon=folium.DivIcon(
                   icon_size=(180, 20),
                   html='''<div style="
                   font-size: 18pt;
                   font-family: serif;
                   color: black;
                   text-align: center;
                   background-color: orange;">
                   Curtis Orchard Pumpkin Patch
                   </div>''')
              ).add_to(my_map)

folium.Marker(location=[40.070867, -88.290145],
```

```
                icon=folium.DivIcon(
                    icon_size=(110, 10),
                    html='''<div style="
                    font-size: 18pt;
                    font-family: serif;
                    color: white;
                    text-align: center;
                    background-color: brown;">
                    My Route
                    </div>''')
                ).add_to(my_map)
```

```
my_map
```

Out[18]:

Final Note - Saving your Map

If you want to save a map with interactive parts, you can save it using the code line in the cell below. To use a map for a report, it's easiest to take a screen shot once the map on the screen looks like how you want it.

In [19]:
```
my_map.save('final_map.html')
```

This notebook was created by R. Vandewalle for the NOBEL project, 2020