Open Spyder. [windows button, Anaconda2, Spyder]

Go to https://www.edmodo.com/home#/group?id=25708867 for everything you'll need.

`Toy_data.txt` is a text file with (FAKE) data from air quality facilities around Pittsburgh over several years. It contains concentrations (Molar) of several oxides [CO, O3, SOx, and NOx]. It also contains 2D coordinates of the locations of the sites from which data was collected.

You are tasked with translating this data into a meaningful format for a presentation to the water authorities of Pittsburgh. Examples of my best ideas are saved under [compound].gif

These instructions will be directed towards use with Python 3.2 and the libraries numpy and matplotlib, but you are free to use whatever tools you choose (just know I can only offer limited help). This packet contains very specific instructions, while the other is much more free-form. Feel free to go back and forth as you see fit.

1. Your BEST FRIENDS will be stackexchange.com and the documentation for matplotlib, numpy. Other useful sites include pythontutor.com and ME! Come at me, y'all.
2. To access the powerful built-in tools we'll use in this project, go ahead and import the necessary libraries. I've included aliases [e.g. `as np`] so you don't have to type as much (I'm super lazy, so I do this every time, it's also pretty standard).
    a. `Import numpy as np`
    b. `Import matplotlib.pyplot as plt`
3. As we're using the numpy library, the logical choice for data manipulation is a numpy array. These are especially useful as they can contain all of your data in a single variable, while still allowing you to access certain parts of the data via a process called 'slicing'.
    a. `Arr = Np.loadtxt('toy_data.npy')`
        i. will get you an array (saved under the variable name "arr") of size (5,6,100). The first dimension is years. E.g. `arr[0]` = 2000, `arr[1]` = 2001, etc. The second is data type. E.g. `arr[0,0]` = Carbon Monoxide concentration in 2000, `arr[0,1]` = Ozone in 2000, while `arr[0,4:5]` will give you the x-y coordinates of all 100 data points. The third dimension is data points: e.g. `arr[0,0,0]` = CO concentration in 2000 at the first location. This bracketing process is called 'slicing'. It will not change the contents of your original array, but will give you specific portions of the data.
    b. `Print(arr)` and click the green play button on the top toolbar. This will print out the entire array in the bottom right corner of your window.
        i. You can do this with any slices of this array by adding the bracketed segments I mentioned in 2a.
4. The clearest to represent our data while also showing our data collection sites would be to overlay the data on a map.
    a. I've included a map in the files you see on Edmodo. This file is called "map.png"

i. (I stole it from https://i.pinimg.com/736x/9c/9b/58/9c9b58769061fe4fe4f101fc12d6ffb7--vintage-prints-print-poster.jpg)

b. To import this image, a special feature of matplotilb called implot will come in handy.

i. `im = plt.imread('map.png')`

1. This step reads and saves the image under the variable name 'im'.
2. You'll also note we've called the matplotlib.pyplot library to access this function by using the 'plt.__' syntax.

ii. `implot = plt.imshow(im,zorder=1,cmap=plt.cm.gray)`

1. This step plots the image, puts it on the bottom (zorder=1; higher zorders will be plotted above the map) and I've colored the image gray for clarity's sake. You can find plenty of other color maps at https://matplotlib.org/examples/color/colormaps_reference.html

5. When you this function, the origin of this imported image is at the top left corner, yet our data is centered at Point State Park. How can we convert one system of coordinates to the other?

a. This bit requires a bit more knowledge of data structures. As we have it, our data exists in a numpy array.

b. Shown here is an example where I've made a list, called 'xs' that will store x-y data. As data was collected at the same spots for all years and compounds, we need only do this once. To convert to the new coordinate system, we first move our origin to the map's origin. Then, we multiply by the scaling factor that will take us to the appropriate step size.

```
xs = []
for x in arr[0,4]:
    x += 4.3
    x *= 12.5
    xs.append(x)
```

c. Go ahead and try to do the same for the y-values. The origin conversion is 5.9.

6. Okay, we have x-y coordinates and a lovely little map, onto which we can plot these points. But how can we show the different compounds and their corresponding concentrations?

a. My best idea is to change the size of the dots we put on the map of Pittbsurgh. As these concentrations are tiny, I've included a scaling factor for visualization (50000)

b. Matplotilb has a very lovely built-in tool for just this purpose in its scatterplot tool called, appropriately enough, 'size'.

i. For 2000 [CO],

```
size = arr[0,0]*50000
plt.scatter(xs,ys,s=size,alpha=0.5,zorder=2)
```

        ii. You'll recall our friends xs and ys, along with zorder. Added to this is 'alpha', which reflects the opacity of the points. In order to see the map, it's helpful to make the dots a little bit transparent.
7. But how can we do this for each data point for every year? A for loop of course!
    a. For loops are super helpful in repetitive tasks such as this. It requires quite a lot of forethought, though. An example of my for loop is in the Edmodo directory saved as "example.py"
        i. I chose to make 20 individual plots, one for each data point of each year. Alternatively, you could make 5 plots, one for each year with data points of different color (https://matplotlib.org/users/colors.html)
    b. Pythontutor.com is a valuable resource for mapping out exactly what your for loop is doing. It takes you step-by-step through what's happening in your code.
8. Put it all together!
    a. Make it a movie!
    b. Ezgif.com is a superfast way to accomplish this. I took each compound's plots and made them into lovely movies. They can be seen in the directory as [compound].gif