Introductory Exercise

November 22, 2016 by Admin (Edit)

You may want to create a free account on https://cloud.sagemath.com or use your local Jupyter Notebook installation for completing this task during workshop

In this section, we're going to cover creating a line graph with legends, titles, and labels within Matplotlib. A lot of times, graphs can be self-explanatory, but having a title to the graph, labels on the axis, and a legend that explains what each line is can be necessary.

To start:

```
import matplotlib.pyplot as plt

x = [1,2,3]
y = [5,7,4]

x2 = [1,2,3]
y2 = [10,14,12]
```

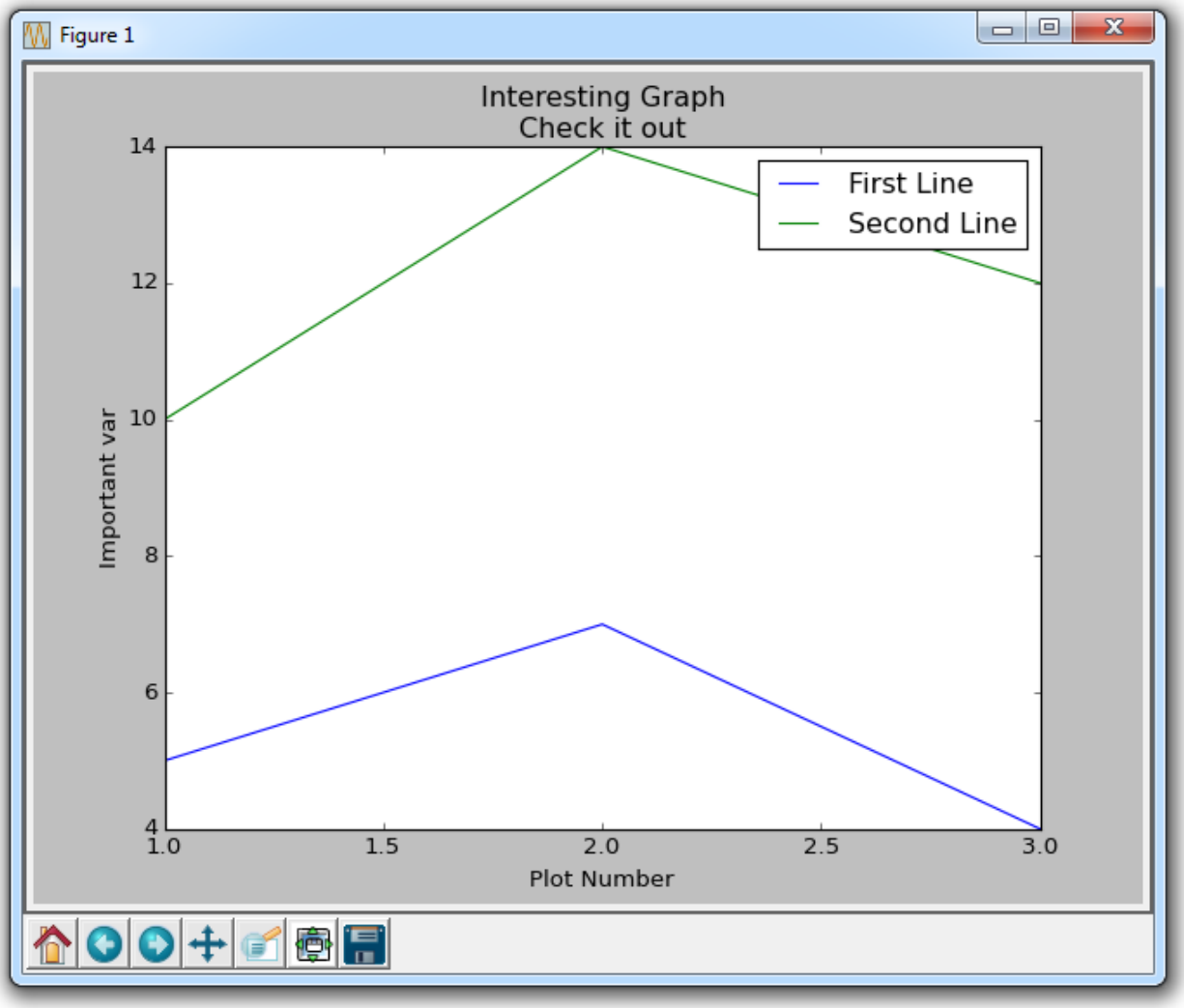
This way, we have two lines that we can plot. Next:

```
plt.plot(x, y, label='First Line')
plt.plot(x2, y2, label='Second Line')
```

Here, we plot as we've seen already, only this time we add another parameter "label." This allows us to assign a name to the line, which we can later show in the legend. The rest of our code:

```
plt.xlabel('Plot Number')
plt.ylabel('Important var')
plt.title('Interesting Graph\nCheck it out')
plt.legend()
plt.show()
```

With plt.xlabel and plt.ylabel, we can assign labels to those respective axis. Next, we can assign the plot's title with plt.title, and then we can invoke the default legend with plt.legend(). The resulting graph:

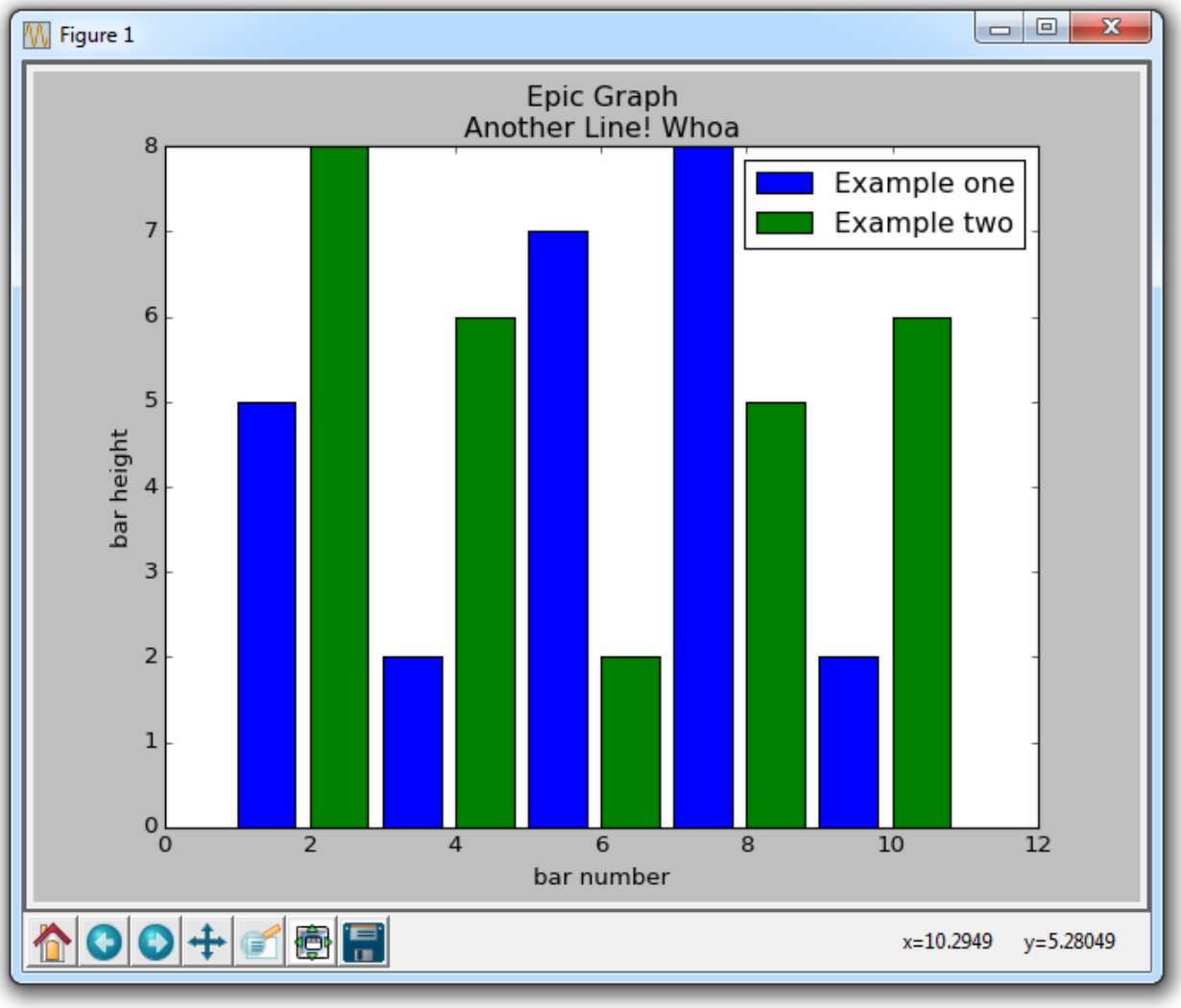


Bar Charts and Histograms with Matplotlib

Lets now create bar charts and histograms with Matplotlib. First, let's cover a bar chart.

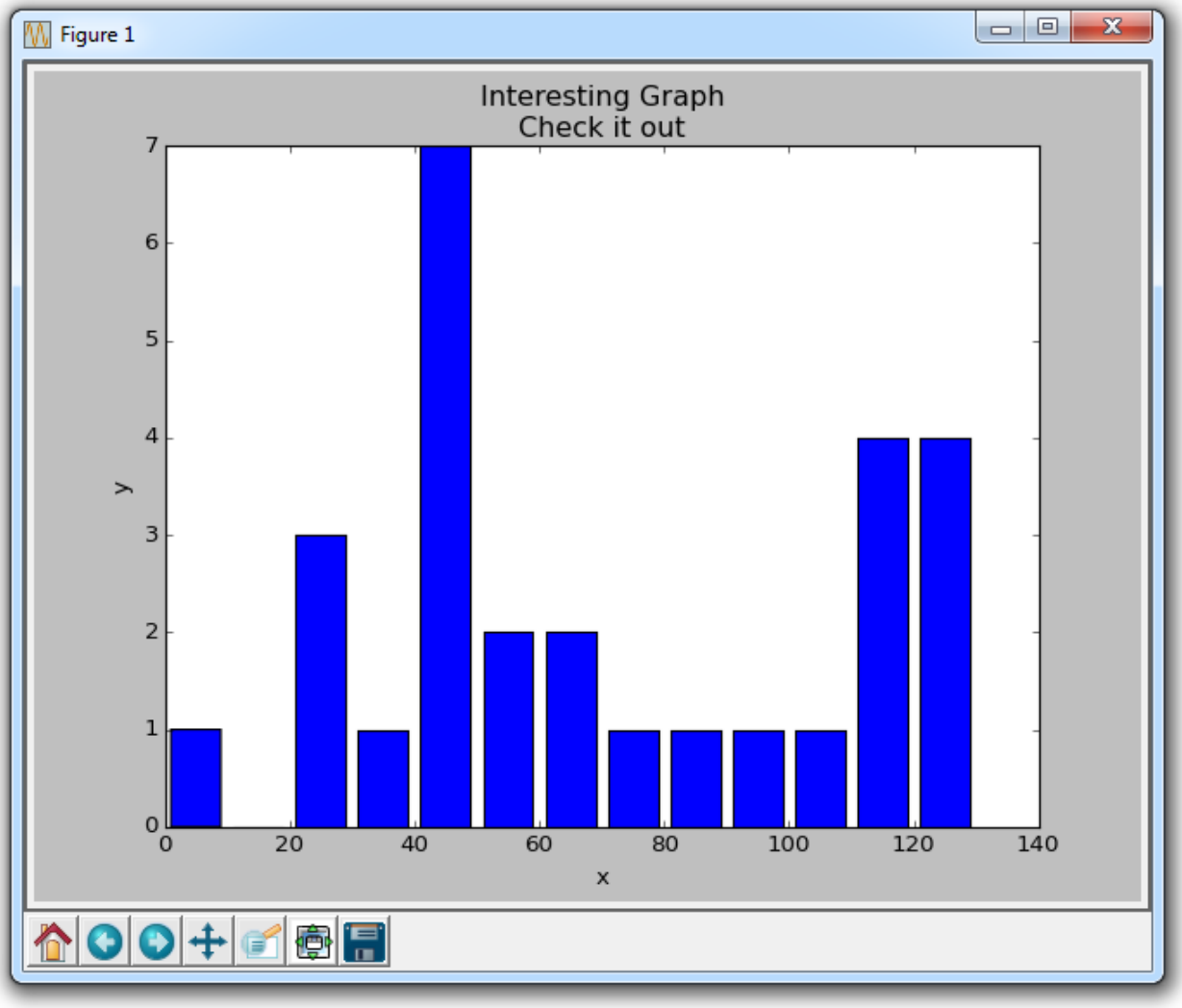
```
import matplotlib.pyplot as plt
plt.bar([1,3,5,7,9],[5,2,7,8,2], label="Example
plt.bar([2,4,6,8,10],[8,6,2,5,6], label="Exampl
plt.legend()
plt.xlabel('bar number')
plt.ylabel('bar height')

plt.title('Epic Graph\nAnother Line! Whoa')

plt.show()
```

The plt.bar creates the bar chart for us. If you do not explicitly choose a color, then, despite doing multiple plots, all bars will look the same. This gives us a change to cover a new Matplotlib customization option, however. You can use color to color just about any kind of plot, using colors like g for green, b for blue, r for red, and so on. You can also use hex color codes, like #191970



Next, we can cover histograms. Very much like a bar chart, histograms tend to show distribution by grouping segments together. Examples of this might be age groups, or scores on a test. Rather than showing every single age a group might be, maybe you just show people from 20-25, 25-30… and so on. Here's an example:

```
1  import matplotlib.pyplot as plt
2
3  population_ages = [22,55,62,45,21,22,34,42,42,4,99,102,110,120,121,122,130,111,1
4
5  bins = [0,10,20,30,40,50,60,70,80,90,100,110,120,130]
6
7  plt.hist(population_ages, bins, histtype='bar', rwidth=0.8)
8
9  plt.xlabel('x')
10 plt.ylabel('y')
11 plt.title('Interesting Graph\nCheck it out')
12 plt.legend()
13 plt.show()
```



For plt.hist, you first put in all of the values, then you specify into what “bins” or containers you will place the data into. In our case, we are plotting a bunch of ages, and we want to display them in terms of increments of 10 years. We give the bars a width of 0.8, but you can choose something else if you want to make the bars thicker, or thinner.

Scatter Plots with Matplotlib

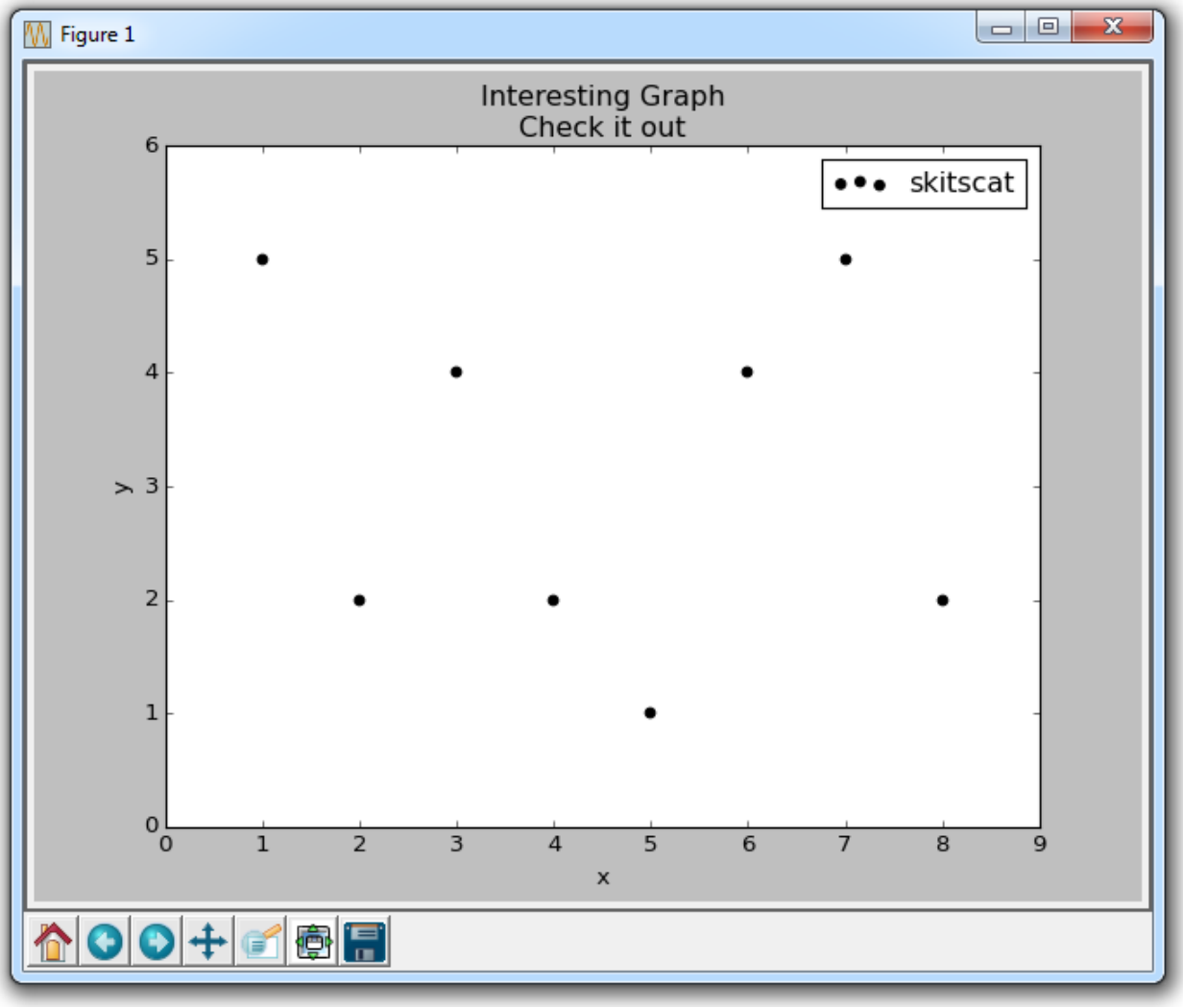
Next up, we cover scatter plots! The idea of scatter plots is usually to compare two variables, or three if you are plotting in 3 dimensions, looking for correlation or groups.

Some sample code for a scatter plot:

```
1  import matplotlib.pyplot as plt
2
3  x = [1,2,3,4,5,6,7,8]
4  y = [5,2,4,2,1,4,5,2]
5
6  plt.scatter(x,y, label='skitscat', color='k', s=25, marker="o")
7
8  plt.xlabel('x')
9  plt.ylabel('y')
10 plt.title('Interesting Graph\nCheck it out')
```

```
plt.legend()
plt.show()
```

The result:



The plt.scatter allows us to not only plot on x and y, but it also lets us decide on the color, size, and type of marker we use. There are a bunch of marker options, see the Matplotlib Marker Documentation for all of your choices.

Stack Plots with Matplotlib

In this part, we cover how to create stack plots. The idea of stack plots is to show "parts to the whole" over time. A stack plot is basically like a pie-chart, only over time.

Let's consider a situation where we have 24 hours in a day, and we'd like to see how we're spending out time. We'll divide our activities into: Sleeping, eating, working, and playing.

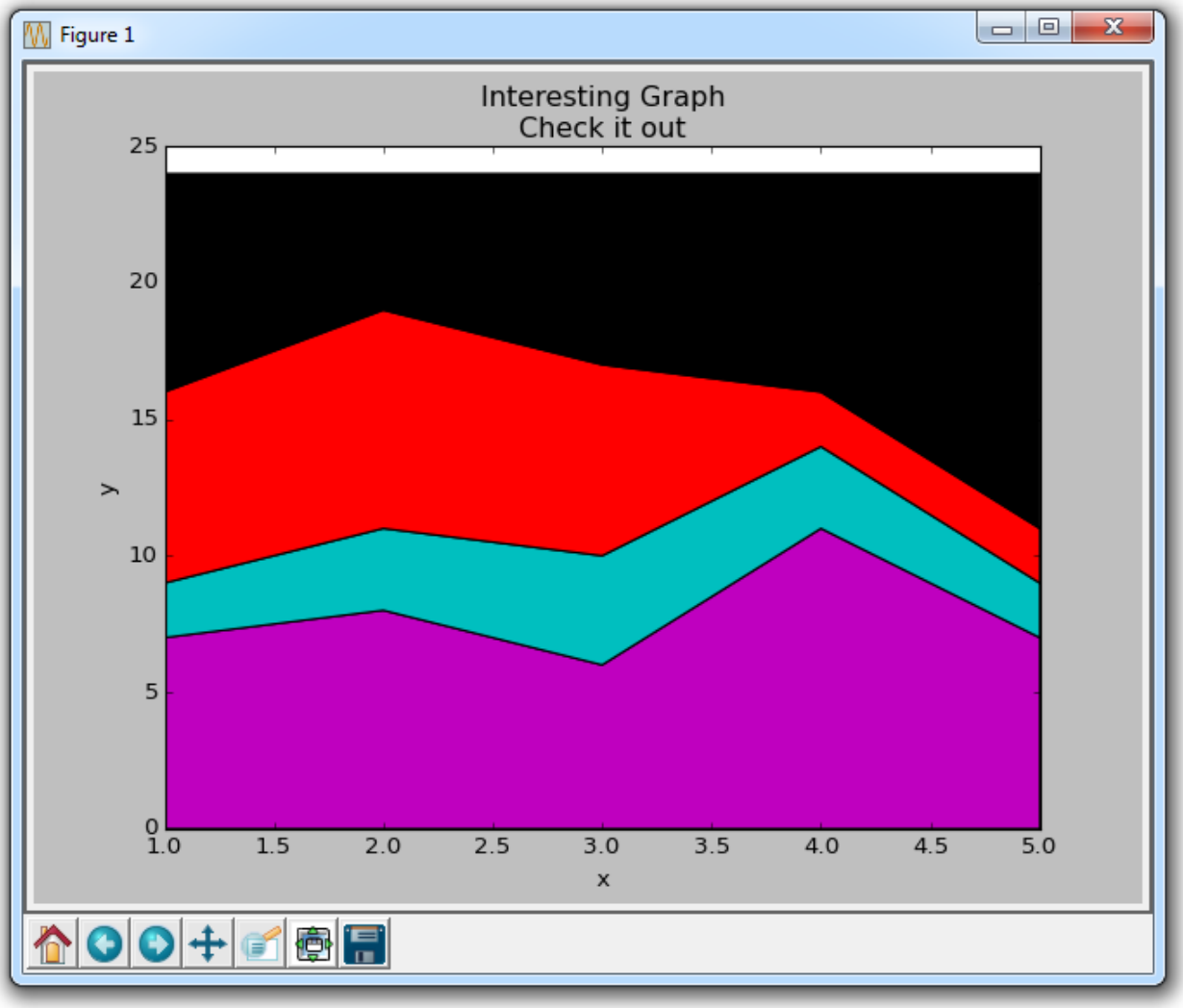
We're going to assume that we're tracking this over the course of 5 days, so our starting data will look like:

```
```

```
import matplotlib.pyplot as plt

days = [1,2,3,4,5]

sleeping = [7,8,6,11,7]
eating = [2,3,4,3,2]
working = [7,8,7,2,2]
playing = [8,5,7,8,13]
```

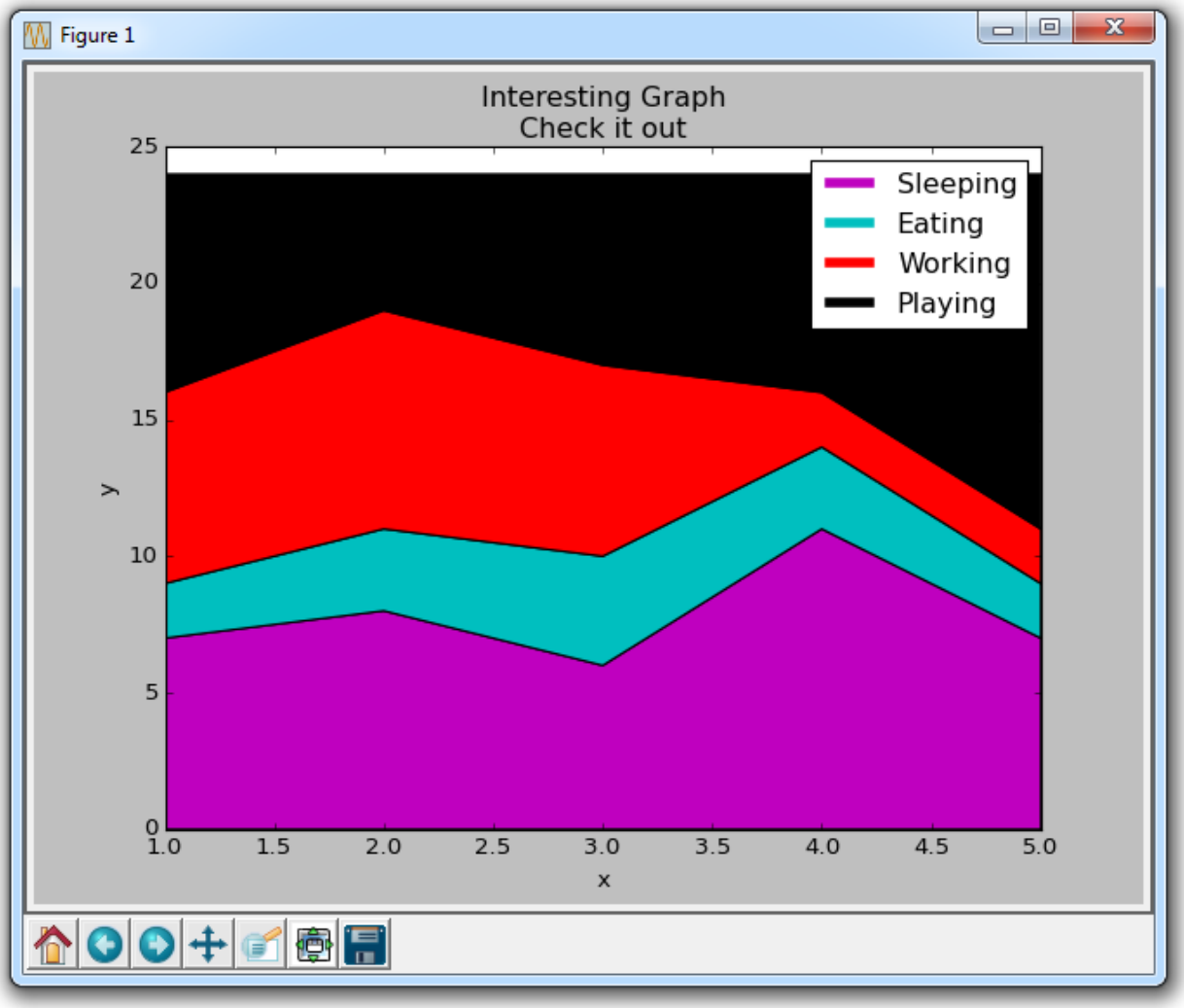
So, our “x” axis will consist of the days variable, which is 1,2,3,4, and 5. Then, our constituents for the days are held in their respective activities. To plot them:

```
plt.stackplot(days, sleeping,eating,working,pla

plt.xlabel('x')
plt.ylabel('y')
plt.title('Interesting Graph\nCheck it out')
plt.show()
```



Here, we can see, at least in colors, how we’re spending our data. The problem is, we don’t really know which color is which without looking back at the code. The next problem is, with polygons, we cannot actually have “labels” for the data. So anywhere where there is more than just a line, with things like fills or stackplots like this, we cannot label the specific parts inherently. That shouldn’t stop a programmer however. We can hack our way around this:

```
import matplotlib.pyplot as plt

days = [1,2,3,4,5]

sleeping = [7,8,6,11,7]
eating = [2,3,4,3,2]
working = [7,8,7,2,2]
playing = [8,5,7,8,13]

plt.plot([],[],color='m', label='Sleeping', linewidth=5)
plt.plot([],[],color='c', label='Eating', linewidth=5)
plt.plot([],[],color='r', label='Working', linewidth=5)
plt.plot([],[],color='k', label='Playing', linewidth=5)

plt.stackplot(days, sleeping,eating,working,playing, colors=['m','c','r','k'])

plt.xlabel('x')
plt.ylabel('y')
plt.title('Interesting Graph\nCheck it out')
plt.legend()
plt.show()
```



All we did here was plot some empty lines, giving them the same colors, and the correct labels in accordance with our stack plot. We also gave them a linewidth of 5, to make the lines a bit thicker in the legend. Now, we can easily see how we're spending our days!

Pie Charts with Matplotlib

Pie charts are a lot like the stack plots, only they are for a certain point in time. Typically, a Pie Chart is used to show parts to the whole, and often a % share. Luckily for us, Matplotlib handles the sizes of the slices and everything, we just feed it the numbers.

```
1  import matplotlib.pyplot as plt
2
3  slices = [7,2,2,13]
4  activities = ['sleeping','eating','working','playing']
5  cols = ['c','m','r','b']
6
7  plt.pie(slices,
8  labels=activities,
9  colors=cols,
10 startangle=90,
11 shadow= True,
12 explode=(0,0.1,0,0),
13 autopct='%1.1f%%')
14
15 plt.title('Interesting Graph\nCheck it out')
16 plt.show()
```