Prelab 7

So far you have been exposed to some of the standard features of Python. Some of these features are string and list manipulation, decision structures, loop structures and some basics of graphics. This Prelab aims to introduce you a completely new feature of Python needed to complete Lab07. In this Prelab you will learn how to plot graphs given a dataset using the `matplotlib.pyplot` library. This library provides you a set of functions to plot several types of graphs such as line, bar, pie and scatter charts. Because of this, we encourage you to carefully study the definition of each function in the library and understand very well the functionality of their parameters. These parameters are used to control the appearance, shape and size of the graphs.

Through this Prelab and Lab07 you will be required to do some manipulation of the data before plotting it. Thus, please read through chapters 6 and 8 in your textbook. It is assumed that you already master these previously covered topics. These topics will be required to manipulate the given datasets.

The `matplotlib.pyplot` library

The library needed to plot graphs is `matplotlib.pyplot`. This library is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure. For example, create a figure, create a plotting area in a figure, plot some lines in a plotting area, decorate the plot with labels and so on. The library is stateful, namely, it keeps track of the current figure and plotting area, and the plotting functions are directed to the current axes.

This Prelab walks you through some basic examples of the use of `matplotlib.pyplot`. Firstly, you will see how to plot a line graph of a function $y = f(x)$. You will see that the library by default treats each variable $x$ and $y$ as a list. That is, $x$ and $y$ are two lists with corresponding values. Then you will see how the code can be modified to have control of what you want to plot. Secondly, you will learn how to plot a bar chart, a pie chart and a scatter chart with a slight modification to the code used to plot the line graph of the function $y = f(x)$. Finally, you will be shown a complete code that plot a graph after calculating the desired values of $x$ and $y$.

About Lab07

Lab07 will challenge you requesting for the plot of several types of graphs. The Lab itself will contain some useful examples that will help you to complete every task. We strongly encourage you to work in this PreLab and review the links we have provided as external resources here. These resources will definitely help you to have a better understanding of the library.

A good starting point is the website below. There you can find a description of all the functions available in `matplotlib.pyplot` library. In addition, every example provided below contains a reference link to a comprehensive documentation about the function used to plot that particular graph. Please review these links (doing a right click and open them in a new tab) in order to understand the parameters of each function and their functionalities.
Summary of available functions: http://matplotlib.org/api/pyplot_summary.html

**Plotting with some functions**

In order to create 2D plots in python, you need to install the following libraries:

matplotlib: [http://www.lfd.uci.edu/~gohlke/pythonlibs/#matplotlib](http://www.lfd.uci.edu/~gohlke/pythonlibs/#matplotlib)

In case you are Mac user execute “pip3 install matplotlib”.

**Plotting a line graph**

Reference: [http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot)

```python
import matplotlib.pyplot as pyplot
pyplot.plot([1,2,3,4])
pyplot.ylabel('some numbers')
pyplot.show()
```

You may be wondering why the x-axis ranges from 0-3 and the y-axis from 1-4. If you provide a single list or array to the plot() command, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you. Since python ranges start with 0, the default x vector has the same
length as y but starts with 0. Hence the x data are [0,1,2,3]. You can control the values of x from which a particular y takes its value by just adding an extra list of the same size. Also you can customize the title, axis labels and legend. See the example:

```python
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], label='Line1')
plt.title('Line Graph Example')
plt.ylabel('Values of the dependent variable')
plt.xlabel('Values of the independent variable')
plt.legend()
plt.show()
```

What's wrong here? Look at the scale of the x variable. We can fix this through the function `matplotlib.pyplot.axis([min X, max X, min Y, max Y])`, in our particular case `plotter.axis([min X, max X, min Y, max Y])` since we imported `matplotlib.pyplot` as `plotter`.

```python
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], '-r', label='Line1')
plt.ylabel('Line Graph Example')
plt.xlabel('Values of the independent variable')
plt.legend()
plt.axis([0, 6, 0, 20])
plt.show()
```
As you can see it is possible control the details of the graph. Notice how we can change the shape and color of the line by adding '-r' to the plot() function (visit the reference given above to learn more about the options to change the shape and color of a line).

The best part of these new library is that you are not limited to the plot() function. The matplotlib.pyplot library includes other functions that allow you have other types of graphs. Let's see an example of a bar chart in the next section, but before continuing, please, get familiar with the plot and try to change the color and shape of lines. Even more, try to plot several lines in the same graph by yourself.

**Plotting a bar chart**

Reference: [http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.bar](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.bar)

```python
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]
color = ['yellow', 'red', 'blue', 'green']
for i in range(len(x)):
    plt.bar(x[i], y[i], width=0.5, color=color[i], align='center')
plt.axis([0, 6, 0, 18])
plt.show()
```
Notice here that every bar is plotted independently in each iteration of the loop. Also notice that the characteristics of the bars can be customized. In this simple example we set the width of each bar to be 0.5 centered in the corresponding value of x, which corresponds to 0.25 to the left and 0.25 to the right of that x value. Also we change the color of each bar. Spend some time getting familiar with this function. Change the values of the parameters and observe the results. In order to know the available parameters visit the website indicated in the reference above.

**Plotting a pie chart**

Reference: [http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.pie](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.pie)

```python
import matplotlib.pyplot as pyplot
labels = ['Frogs', 'Hogs', 'Dogs', 'Logs']
sizes = [15, 30, 45, 10]
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
pyplot.pie(sizes, labels=labels, colors=colors)
pyplot.show()
```
Notice that this pie chart is not very good looking. We could modify some of its parameters and use some extra functions of the library to improve its appearance (see the reference specified above).

```python
import matplotlib.pyplot as plt
labels = ['Frogs', 'Hogs', 'Dogs', 'Logs']
sizes = [15, 30, 45, 10]
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
explode = (0, 0.1, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=90)
plt.axis('equal')
plt.show()
```
Notice how we separate the portion corresponding to 'Hogs' using the explode parameter. Notice as well that pyplot.axis('equal') gives a more symmetric appearance.

**Plotting scatter graph**

Reference: [http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.scatter](http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.scatter)

```python
import numpy as np
import matplotlib.pyplot as plt

N = 50
# Create arrays of random numbers for the coordinates
x = np.random.rand(N)
y = np.random.rand(N)
# Create array of random numbers for coloring the circles
colors = np.random.rand(N)
# Create an array of area values, note that np.random.rand(N) returns an array
# The arithmetic operations are performed on each array element independently
area = np.pi * (15 * np.random.rand(N))**2  # 0 to 15 point radii

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```
Here we use an numpy array to store the data. You can think of a numpy array as a collections of elements of the same type. You can create unidimensional arrays (called vectors in other programming languages) and multidimensional arrays (called matrices). The usefulness of numpy arrays is that there are procedures that will manipulate the entire array, something you cannot do with tuples (another structure available in Python).

For our purposes, just focus on the fact that a numpy array is sequence of values of the same type. In this particular piece of code, those values are randomly chosen floating point numbers. Then, the variable $x$ will be a list of random numbers. The variable $y$ will also be a list of random numbers and the scatter function plots for every value of $x$ a corresponding value $y$.

http://cs231n.github.io/python-numpy-tutorial/#numpy

**Extra exercise**

In mathematics, the Leibniz formula for $\pi$, states that:

\[
1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}.
\]

Using summation notation:
\[ \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4}. \]

```python
import matplotlib.pyplot as pyplot

def Leibniz(nterm):
    # Initial values
    s = 0
    x = []
    y = []

    for i in range(nterm):
        # Append each iteration
        x.append(i)
        # Summation of Leibniz
        s += ((-1)**i) / ((2.0*i)+1)
        # Append approximation
        y.append(s)

    print("Leibniz Sum = ", s)
    # s*4 = 3.14...
    print("PI approximation = ", s*4)
    return x, y

# Plotting Leibniz Formula
def Leibniz_graph(x, y):
    pyplot.plot(x, y)
    pyplot.title('Leibniz Formula')
    pyplot.xlabel('Number of Terms ')
    pyplot.ylabel('Terms ')
    pyplot.show()

def main():
    n = 119
    # Function Calls
    x, y = Leibniz(n)
    Leibniz_graph(x, y)

main()
```