**IS201 Fundamentals of Computing**

**HOP10 Data Visualization**

04/17/2020 Developed by Amrutha Vaidyanathan

05/28/2020 Reviewed by Kim Nguyen

Center for Information Assurance (CIAE) @City University of Seattle (CityU)

A picture containing drawing

Description automatically generated A close up of a sign

Description automatically generated

**Before You Start**

* The directory path shown in screenshots may be different from yours.
* Some steps are not explained in the tutorial**.**If you are not sure what to do:
  + Consult the resources listed below.
  + If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

* Using Matplotlib, a mathematical plotting library
* Make simple plots, such as line graphs and scatter plots

**Resources**

Matthes, E. (2019). [Python Crash Course: A Hands-On, Project-Based Introduction to Programming, 2nd Edition](https://login.proxy.cityu.edu/sso/skillport?context=146803). No Starch Press. (ISBN 9781593279288)

**Preparation**

1. In Visual Studio Code, open the private repository generated when you accepted the HOP09 assignment (If you cannot find that repository in your machine, you might have not cloned the repo, if so, please do before proceeding).

A screenshot of a cell phone

Description automatically generated

**Installing Matplotlib**

1. Open a new terminal and type the following command to install Matplotlib using pip



After successful installation, you can see something similar to the following message in your terminal.



Note: If the above command doesn't work on macOS, try running the command again without the --user flag.

**Plotting a line graph**

Let take the dataset of multiples of 5: 0,5,10,15,20,25 and 30 for plotting in the graph.

1. Create a file **line.py** under **Module10** folder and type the following code

A screenshot of a cell phone

Description automatically generated

The first line of the code is the import of pyplot module using the alias “plt”. The reason for using alias is we don’t have to type “pyplot” repeatedly. This module contains several functions that generate charts and plots.

The multi variable holds the data set.

The Matplotlib convention - subplots() function can generate one or more plots in the same figure.

The variable fig represents the entire figure or collection of plots that are generated.

The variable ax represents a single plot in the figure and is the variable we'll use most of the time.

The plot() method will plot the data it's given in a meaningful way.

The function plt.show() opens Matplotlib's viewer and displays the plot. Let’s run the code and check how the plt.show() displays the graph.

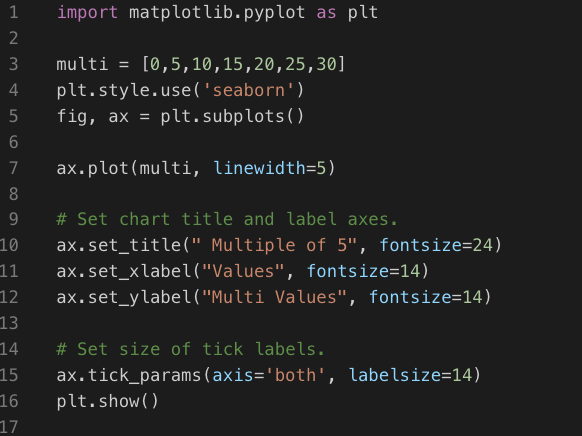
In the terminal type **python3 line.py**

A screenshot of a cell phone

Description automatically generated

In the above diagram the x and y axis are not named, and we can adjust the display type of the line and background.

1. In the same file **line.py** update the existing code with the below



The plt.style.use sets the background style.

The linewidth parameter controls the thickness of the line.

The set\_title() method sets a title for the chart.

The fontsize parameters control the size of the text in various elements on the chart.

The set\_xlabel() and set\_ylabel() methods allow you to set a title for each of the axes

The method tick\_params() styles the tick marks. The axis=‘both’ affects the tick marks on both the x and y-axes and labelsize=14 sets the font size of the tick mark labels to 14.

In the terminal type **python3 line.py**. You can find the difference between the previously generated graph and the current output.

A screenshot of a computer

Description automatically generated

**Scatter and Automatic calculating**

In the above generated graphs, the data sets were small, so we specified it in the lists. Suppose if we have to plot the multiples of 5 till 50th value, then specifying the data sets in the lists will be inefficient. Therefore, we can calculate the values automatically and plot it in the graph. In this section we are going to use points instead of line.

1. Create a file **scatter\_auto.py** under **Module10** folder and type the following code

A screen shot of a social media post

Description automatically generated

In the x\_values instead of mentioning the data set, we specify the range. In the y\_values we are calculating the values for each value in the x\_values range(i.e.) 0 to 50.

In the scatter method we pass the x and y axis values and specify the size of the scatter (i.e.) s= 30.

In the terminal type **python3 scatter\_auto.py** to check the output of the code.

A picture containing table, large, white

Description automatically generated

**Random Walks**

A random walk is a path that has no clear direction but is determined by a series of random decisions, each of which is left entirely to chance. You might imagine a random walk as the path a confused ant would take if it took every step in a random direction.

1. Create a file **random\_walk.py** and type the below code

**Note:** If you get syntax error for **“”....””**, delete the symbol and add **#** in front of the line.

A picture containing screen, black, sitting, monitor

Description automatically generated

The above code will make random decisions about which direction the walk should take. The class needs three attributes: one variable to store the number of points in the walk and two lists to store the x and y-coordinate values of each point in the walk.

We'll only need two methods for the RandomWalk class: the \_\_init\_\_() method and fill\_walk(), which will calculate the points in the walk.

The above code shows the \_\_init\_\_() functions. To make random decisions, we'll store possible moves in a list and use the choice() function, from the random module, to decide which move to make each time a step is taken.

Now let’s look at the fill\_walk().

1. In the same file **random\_walk.py**, update the code as shown below. Code to be added is highlighted in red box

**Note:** If you get syntax error for **“”....””**, delete the symbol and add **#** in front of the line.

A screenshot of a cell phone

Description automatically generated

The fill\_walk() function will fill our walk with points and determine the direction of each step. In this function we set up a loop that runs until the walk is filled with the correct number of points.

The main part of the fill\_walk() method tells Python how to simulate four random decisions: will the walk go right or left? How far will it go in that direction? Will it go up or down? How far will it go in that direction?

We use choice ([1, -1]) to choose a value for x\_direction, which returns either 1 for right movement or –1 for left.

Next, choice([0, 1, 2, 3, 4]) tells Python how far to move in that direction (x\_distance) by randomly selecting an integer between 0 and 4. (The inclusion of a 0 allows us to take steps along the y-axis as well as steps that have movement along both axes.)

After the choice we determine the length of each step in the x and y directions by multiplying the direction of movement by the distance chosen. A positive result for x\_step means move right, a negative result means move left, and 0 means move vertically. A positive result for y\_step means move up, negative means move down, and 0 means move horizontally. If the value of both x\_step and y\_step are 0, the walk doesn't go anywhere, so we continue the loop to ignore this move.

To get the next x-value for the walk, we add the value in x\_step to the last value stored in x\_values and do the same for the y-values. When we have these values, we append them to x\_values and y\_values.

Now let’s look at how to plot all the points in the walk

1. In the same file **random\_walk.py**, add the below line before the **from random import choice**. Code to be added is highlighted in red box

A close up of a sign

Description automatically generated

Below the existing last line add the below code. Code to be added is highlighted in red box.

A screenshot of a cell phone

Description automatically generated

We begin by importing pyplot and RandomWalk. We then create a random walk and store it in rw, making sure to call fill\_walk(). Then we feed the walk's x and y-values to scatter() and choose an appropriate dot size.

For reference: **random\_walk.py**

A screenshot of text

Description automatically generated

A close up of text on a black background

Description automatically generated

In the terminal type **python3 random\_walk.py** to check the output of the code.

A screenshot of a cell phone

Description automatically generated

**Push your work to GitHub**

Run the following commands to push your work to the GitHub repository:

Open the terminal from the VSCode by hitting the “control” + “~” key and type the following command:

>>> git add .

>>> git commit -m “Submission for Module 10”

>>> git push origin master