In this Lab, we'll demonstrate how to use the Anaconda Prompt to run an **IPython/Juptyer notebook** within its corresponding **.ipynb** file on your local PC. You'll also work on "regular" .**py** files within PyCharm. Download the zipped folder containing starting code for this lab (**lab\_6\_starting.zip)** and place it on your local drive where you have read/write permissions.

**L6\_1\_stringops.ipynb** is a Jupyter notebook, included in this starting code folder. Unzip the folder and open it as a PyCharm project. It contains this notebook, along with several **.py** and **.txt** files for this and later Labs and Homework. Submit as many of the following problems as we have time for in class. If you can get your notebook to run, submit your final modified notebook file **L6\_1\_stringops.ipynb** files to the **L6 Assignment** area on Canvas. If you have problems, just submit the original notebook file as is.

**[L6-1]** (**L6\_1\_stringops.ipynb**) Together, we'll see how to launch this small Jupyter notebook \*outside\* of PyCharm, using the Anaconda Prompt. Then you'll complete the provided starting code, exploring the behavior of Python's **str** class methods. The notebook runs inside a browser, launching a Python interpreter that runs the Python code within the browser page. This is similar to our HTT textbook's "interactive browser" approach, but much more sophisticated.

This notebook leads you through completing a complete Python application that demonstrates a number of important Python **str** (string) class methods.

Also provided in the starting PyCharm project is a collection of Jupyter notebooks that include the book *Python Data Science Handbook*, by Jake VanderPlas, O'Reilly and Associates, 2016. The book itself is available for online browsing at: [**https://jakevdp.github.io/PythonDataScienceHandbook/**](https://jakevdp.github.io/PythonDataScienceHandbook/)

**If we have time**, we'll see how to view a Jupyter notebook that contains this book's content, which is contained in the project folder. We will browse and run the content of several of the Chapter 1 sections. Later in the course, we will again work with this book and additional Jupyter notebooks.

The remaining problems ask you to write Python code within PyCharm. Submit as many as we have time for.

**[L6-2]** (**patterns.py**) Write a program that reads an **int N >= 0**, then prints out each of the following patterns of \*, each of which depends on the value of **N**.

Do each of these in two different ways: (a) using the **\*** string (**str**) repetition operator, and (b) using nested **for**-loops, where the inner loop variable's range may depend upon the value of the outer loop's variable, but NO use of the repetition operator.

When you run your program, it should print out each pattern twice: once as performed via (a) and again via (b).

Examples diagrams for **N==4** are given, with explanations of how the displayed diagram reflects this value of **N**. Your code should be more general and work correctly for all **N >= 0,** printing the given pattern twice.

(a) (single line of N stars)

**\*\*\*\***

(b) (N lines of N stars each)

**\*\*\*\*  
\*\*\*\*  
\*\*\*\*  
\*\*\*\***

(c) (N lines, first with 1, second with 2, ... Nth with N stars)

**\*  
\*\*  
\*\*\*  
\*\*\*\***

**[L6-3]** (**readpos.py**) Write a function **read\_pos()** which prompts the user to enter an **int N >= 0**, then returns it. However, if the user enters a negative number, print an error message then prompt the user to reenter. Continue with this "bad input" defense until the user enters a valid **N >= 0** integer, then return it. Finally, write your program that calls **read\_pos()** and prints out the returned **int**. **Hint:** use a **while** loop body which reads **N**, then executes **break** if a valid **N** is entered. Otherwise print an error message and try again.

The following questions are **optional**. Try them to improve your Python programming! But... no extra credit, here.

**[L6-4]** (**bool\_practice.py**) Write a complete Python program sets initial **int** values to variables **start\_value** and **stop\_value**. For each of the items (a)-(f), write a separate **while**-loop (no **for**-loops!) that prints out each value from **start\_value** to **stop\_value**, inclusive, that satisfies the given condition. Use **print(num,end=' ')** in your loop body to keep your output on the same line, adding **print()** after the loop to complete your output ("newlines flush the print buffer").

**[L6-5]** (**wordstats.py**) The **wordstats.py** file contains code that opens and reads the provided text file **words.txt** into one single (long) string **bigline**. Add code that does the following, using **str** methods and functions:

(a) Print out the total length of the file in characters, including newlines (**'\n'**) (already done!)

(b) Print out the total number of words in the file (hint: each word ends with a newline)

(c) Print out the average length of a word (don't include newlines)

(d) Print out the count of **'e'** in all words, as well as fractional frequency of **'e';** that is, the count of **'e'** divided by the total number of letters in the file, not counting newlines.

(e) Same as (d) but print the count and fractional frequency for all letters.

Hint: calculate (d) for each character in **"abcdefghijklmnopqrstuvwxyz".** How about iterating over this string? How about defining a function?