New York University
School of Continuing and Professional Studies
Division of Programs in Information Technology

Introduction to Python
Homework Discussion, Session 3

3.1 Calculate the average Mkt-RF value for a given year. The chief new challenge you are facing here is correlating the script's logic and output with the data the script is parsing. With a large file the output can be copious and confusing. This is why it's wise to start with the shortened FF_abbreviated.txt file, because it's easier to correlate a small dataset with the output you're seeing. With this shortened file, you should be able to mentally calculate the count, sum and average to confirm that your program is doing the right thing.

Begin with your while True: and input() loop. If you successfully used this technique in the previous week's homework, you should be able to require a 4-digit year without too much extra work. (Your test needs to both check to see that the len() of the user's input string is 4 as well as to confirm that the string is all digits. The most concise way to do this is to place the two tests in the same if expression, connected by and; in other words an expression that says):

```
# if len of user_input is 4 and user_input is all digits
```

Make sure to test this part of the program before proceeding. You should test it against bad input as well as good input. Try to figure out a way to screw it up -- your program should successfully reject any input that is not 4 digits.

Next, declare two variables: a float this_sum set to 0.0, and an integer count set to 0.

You can then open the file and loop through it with for. Inside the loop, for each line in the file:

- slice out the year (test this by printing out the year -- you should see an output of 4-digit years, one per line)
- if the year is equal to the user's input year,
    - str.split the line into elements (split() without arguments splits on whitespace.) (Test this step by printing out the split list -- you should see each line of the file split into list elements, 5 per line; and you should see this only for the selected year.)
    - convert the 2nd element from the split (the MktRF value) into a float (test this by printing out the float element -- you should see an output of float values, one for each line that matches the test year
    - add the float value to this_sum (test this by printing out the sum after each time it is augmented; you should see the value grow after each line, and it should be increasing by the amount in each line of the file)
    - increment the integer count by 1 (test this by printing out the count; you should see this value grow by one for each line, and only as many as there are lines that contain the user's year)

Once the loop of the file is complete, you will have a count of the lines that matched the selected year and a variable called this_sum containing a sum of the values encountered for the selected year.

Harking back to the "fog of code", the reason I suggest you use the FF_abbreviated.txt file is that you can very clearly see what the program is doing. If you use the larger file, you'll get a lot of output and it will be difficult to see whether your output is correct (call it the "fog of data"). With the smaller file, you can visually check the results.

Finally, print out the count, float_sum and average (float_sum / count) in a formatted string using str.format().

Pseudocode:

```
# establish a while True loop

    # take user input
    # if the input has a length of 4 and is all digits:
        # break
    # print an error message:  input was incorrect
    # loop will automatically return to the top of the block

# set this_sum to a float
# set a count to 0

# open a file
# loop through the file line by line

    # slice out the year from the line using string slicing
    # if the year is equal to the user's input:
        # split the line on whitespace (str.split() without arguments)
        # convert the 2nd element from the split (the MktRF value) to a float
        # add the float to this_sum
        # increment count by 1

# calculate the average of the values by dividing this_sum by count

# report the count, sum and average in a formatted string
```

3.2 wc emulation. The overall purpose of the assignment is to get you to think of a file as a list, a file as a string, and a string as a list of strings. To determine the number of characters in the file, you need only think of the file as a string. To determine the number of lines in the file, you need only think of the file as a list of lines. To determine the number of words in the file, you need only think of the file as a list of words.

How can we think of the file as a string? file.read() returns the file contents as a string. We can use len() on a string to see how many characters are in the string.

How can we think of the file as a list of lines? str.splitlines() returns the string split on the newline character, i.e. a list of lines. We can use len() on a list to see how many elements are in the list.

How can we think of the file as a list of words? str.split() returns a string split on whitespace, i.e. a list of words. We can use len() on the list to see how many words are in the list.

Pseudocode:

```
# use raw_input() to take a filename

# open the file

# read the entire file text into a single string

# split the string into a list of lines

# split the string into a list of words

# print the len of the list of lines, the len of the list of words, and the len of the string itself
```