Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [1]:
```
NAME = "Sharjil Mohsin"
COLLABORATORS = "N/A"
```

Minor Assignment 3

ENGINEER 1D04
Dr. Ashgar Bokhari
McMaster University, Fall 2017

Background

`wc` (for word count) is a well-known unix program that computes the number of lines, words, and characters in a text file. A *line* is a string of characters delimited by newline characters, and a *word* is a string of characters delimited by space characters or newline characters.

Overview

The purpose of this assignment is to write a Python version of the unix `wc` program. Design, implement, and test a program that satisfies the requirements below.

Requirements

1. Write one function that does three things:

   a. Counts the number of lines in a given text file f.

   b. Counts the number of words in f.

   c. Counts the number of characters in f. Spaces count as characters. Do not count newline characters as characters.

2. The function returns the lineCount, wordCount and charCount as output.

3. Your name, MacID, student number, and the date are given in comments at the top of the first Python cell in the notebook.

4. Your answers to the design questions and test plan are given in the appropriate Markdown cells below.

5. Your program MUST have valid Python syntax and it must run without errors. Ensure that your program runs properly by running it before you submit.

Design and Implementation Instructions

1. You will likely want to use some of Python's built in string methods (e.g. `split`, `strip`) from the Python `str` class. These can be found in the Python 3 documentation, and they were covered in Tutorial as well.

2. There is code already included to create four text files for you to use, but you can also create a new text file using notepad, you do not have to write the file using Python as in the cell below. Also, two of the four files were created using randomly generated text, so the files are supposed to look like gibberish.

In [2]:
```
# Code to generate sample text files for testing
# RUN THIS BLOCK AT THE START OF YOUR LAB
# If you modify the text files this cell creates, rerun this cell

infile1 = open("wc.txt", 'w')
infile1.write("This will enter some text\ninto the file I am testing.")
infile1.close()

infile2 = open("wc2.txt", 'w')
infile2.write("Abilities forfeited situation extremely my to he resembled.\nOld had conviction discretion underst
infile2.close()

infile3 = open("wc3.txt", 'w')
infile3.write("Sussex result matter any end see. It speedily me addition weddings vicinity in pleasure. Happiness
infile3.close()

infile4 = open("wc4.txt", 'w')
infile4.write("9")
infile4.close()
```

In [1]:
```
# DO NOT PUT AN INPUT STATEMENT IN THIS CELL

#################---minor3(f) - (lineCount: 3, wordCount: 3, charCount: 4 Marks)---#################
def minor3(f):
    #Place your code between here...
    file = open(f, 'r')
    readLines = file.readlines()
    file.close()


    file = open(f, 'r')
    list1 = file.readlines()
    count = 0
    for i in range(len(list1)):
        count = count + len(list1[i].split(' '))
    file.close()


    file = open(f, 'r')
    list2 = file.readlines()
    count2 = 0
    for i in range(len(list2)):
        count2 = count2 + len(list2[i].strip())
    file.close()

    lineCount = len(readLines) # change this value to your answer for lineCount
    wordCount = count  # change this value to your answer for wordCount
    charCount = count2  # change this value to your answer for charCount

    #...and here

    return lineCount, wordCount, charCount
# Change the string "wc.txt" to a string of another text file to test your code
result = minor3("wc.txt")
print("lineCount =", result[0])
print("wordCount =", result[1])
print("charCount =", result[2])
```

```
lineCount = 2
wordCount = 11
charCount = 52
```

In [2]:
```
###########---TESTING CELL---#############

#Run this cell to see your autograded mark

'''
Sample Input:
wc.txt

Sample Output:
lineCount = 2
wordCount = 11
charCount = 52
'''

from autograder import autograde
checkMinor3Mark = autograde()
checkMinor3Mark.marker([minor3], "minor3")
```

```
Automarked Score =  10.0  /  10.0

    NOTE: WE HAVE CHANGED FROM THE PAPER SIGN OUTS TO AN ELECTRONIC SYSTEM
          YOU WILL RECEIVE 0 IF YOU DO NOT SIGN OUT WITH AN IAI OR TA

(This mark is an estimate of your grade and may not accurately represent the mark you receive)
Use your testing plan to ensure your code is working as intended

BEFORE LEAVING THE LAB:
1. Restart your kernel and run all cells to check that your code is working as intended
        Kernel --> Restart & Run All
2. Make sure to answer the design questions and testing plan
3. Save your work
        File --> Save and Checkpoint
4. Submit your assignment
        On Jhub: Assignments --> Downloaded Assignments --> assign#_section# --> Submit
5. Sign out with your IAI or one of your TAs
```

Design Questions

1. What is the best way to compute the number of lines, words, and characters in a file: by reading the file once, or by reading the file three times?
2. When writing to a file, can you use the format `"x=", x, "\n"` instead of `"x="+str(x)+"\n"`?
3. Is it necessary that most of the test cases for your program are files having a large number of entries? Explain your answer
4. What is the difference between `read()` and `readlines()`?

Enter your answers into the Markdown cell below.

1. The best way to compute the number of lines, words, and characters in a file is by reading the file three times. This is because if the file is read only once, the first function, which is to compute the number of lines, worked perfectly but for the number of words and characters, it does not compute properly and therefore gives a result of 0 or some other wrong number instead. This is why the file must be read by the program 3 times.

2. No I cannot use that format because the write() function can only take one argument. In this case, there are three separate arguments being taken in, so it gives an error.

3. It depends on the user's choice to use a large number of entries or not, but it is recommended to use a large number of entries so that the user can see where the program might not get the correct amount of line counts, word counts, and character counts when reading a file, as large entries tend to have a higher probability to cause an error.

4. The difference between read() and readlines() is that the read() function prints out exactly what's in the txt file and prints that output here, while the readlines() takes all the in each line and puts it in an index in a list.

Testing Plan

Produce a test plan in the Markdown cell below, in the following form:

```
Test i
Input: ["test.txt"]
Expected Output: [lineCount, wordCount, charCount]
```

```
Actual Output: [lineCount_a, wordCount_a, characterCount_a]
Result: Pass/Fail
```

Note: The actual output should be what the program produces, even if your output does not match the expected output.

You must have $NO\ LESS\ THAN\ 3\ TEST\ CASES$. Have at least 1 case where your program does not output an error. For the other cases, we encourage you to try and find test cases where your program would output an error (not mandatory, just recommended). That is, where the expected output is an error.

```
Test 1
Input: ["wc2.txt"]
Expected Output: [5, 66, 407]
Actual Output: [5, 66, 407]
Result: Pass

Test 2
Input: ["test.txt"]
Expected Output: [3, 5, 31]
Actual Output: [3, 11, 23]
Result: Fail

Test 3
Input: ["sample.txt"]
Expected Output: [3, 12, 81]
Actual Output: [3, 20, 75]
Result: Fail
```

In [5]:
```
#Ignore this cell
```

In [6]:
```
#Ignore this cell
```

In [7]:
```
#Ignore this cell
```

In [8]:
```
#Ignore this cell
```

In [9]:
```
#Ignore this cell
```

In [10]:
```
#Ignore this cell
```

In [11]:
```
#Ignore this cell
```

In [12]:
```
#Ignore this cell
```

In [13]:
```
#Ignore this cell
```

In [14]:
```
#Ignore this cell
```