IF100
Files & Dictionary - Practice #1

Introduction

The aim of this example is to practice file operations and dictionaries in Python. The use of file operations is due to the nature of the problem; that is, you cannot finish this practicing assignment without using file operations. You may implement the solution without using a dictionary, but that would be less efficient.

Description

In this exercise, you will implement a program which generates a sorted student ranking table according to the GPAs of the students after reading letter grades from a file. This table will consist of the following information: names of the students, abbreviations of the course names and letter grades of the students for specific courses. Your program will read letter grades from an input file, and by processing these grades, for each student, your program will compute the average GPA earned. Then, your program will generate a sorted list and print it. The details of the input file format and the rules of processing is explained below.

Input File

Your program will read grades from an input file. There will be several students and their grades in the same file, where grades begins with a constant title:

Names/Courses IF100 SPS101 NS101 MATH101 HIST191 TLL101

After this line, there comes the grades list of the students for that term, line by line. There is only one student name and his/her letter grades for each course on each line of this file. Between the letter grades and the names, there is a single **"\t"** character which represents a **TAB**. At the end of the grades file, there is an empty line.

Yahsi, Fatih Taha	A-	B-	B-	D	B+	A
Karabaglı, Batuhan	D+	C-	B	B	D+	F
Karaev, Timur	A	C+	D+	C	A-	C+
Egeli, Rukiye-ayshe	A-	B-	B+	D	C-	D+

For a sample input file, you may check out the `studentGrades.txt` provided with this assignment. Please note that you cannot make any assumptions on the number of lines in the file (i.e. the number of students) and also the amount of courses in a term, they are not fixed. However, you may assume that credits of courses are **equal** to each other.

The name of the input file will be a *keyboard* input. In other words, your program will prompt for this input.

Processing

For each student in the list, you will calculate the average GPAs with the help of the table given below:

Letter Grades	Points
A	4.00
A-	3.70
B+	3.30
B	3.00
B-	2.70
C+	2.30
C	2.00
C-	1.70
D+	1.30
D	1.00
F	0.00

For each student, you will find the total points earned in that term by adding points according to the letter grades, and divide the result by the number of courses taken by the student in that term. As an example, if a student gets A, B+ and C- from 3 courses, his/her GPA will be (4.00 + 3.30 + 1.70) / 3 = 3.00. You can use a dictionary to store the data for the students.

After your program finishes reading and processing all of the grades from the input file and storing the necessary data in a dictionary, it should print the list that includes (i) rankings, (ii) names of the students, and (iii) earned GPAs, in descending order. Thus, students with higher GPA must be at higher positions in the table. If multiple students have the same GPA, then your program should display them in the ascending order of the alphabet with respect to their names.

Hint: You can use built-in ***len()*** and ***max()*** functions together with the ***pop()*** method on dictionary objects and the ***sort()*** method on list objects.

Output

After the students are sorted, your program should display the point table. The format of this table is as described below.

The first line is the header of the table. This line is fixed and given below.

```
RANK - NAME - GPA
```

After this header line, the point table will be displayed in rank order (from top to bottom). Rank values should start from 1. You should display separate student data in separate lines.

For each student, 3 pieces of information should be displayed at the very end: (1) rank of the student in the term, (2) name of the student, and (3) GPA of the student. These 3 pieces of information must be displayed in this order, there must be spaces and dashes in between them, and GPA must have exactly 2 digits after the decimal point.

Together with this assignment document, we provide a sample input file. Please see "Sample Runs" section in order to understand the flow of the program, the inputs and the outputs in a better way.

We highly recommend the use of dictionaries.

We are very aware that this homework can also be done by:

1) Using lists instead, which makes the implementation much complicated.

2) Using files only by parsing several times, which is very inefficient.

Also, we would like to remind that this practice example is given for your preparation for the final exam and dictionaries will be an important part of the exam. Doing this practice example by avoiding using dictionaries will not help you much in this direction.

We also highly recommend the use of functions.

Sample Runs

Below, we provide some sample runs of the program that you will develop. The *italic* and **bold** phrases are inputs taken from the user.

Sample Run

```
Please enter the filename: studentGrades.txt
RANK - NAME - GPA
1 - Aya, Ecem Meva - 3.33
2 - Boy, Navruz Yusuf - 3.33
3 - Pekel, Ege Deniz - 3.33
4 - Tuzlen, Serra - 3.27
5 - Elyakubi, Emin - 3.23
6 - Karabulut, Eylul Berfin - 3.13
7 - Dokgoz, Alp - 3.12
8 - Yetki, Tan Doruk - 3.12
9 - Aga, Firuze Alinda - 3.10
10 - Akant, Zeynep - 3.10
11 - Gonenc, Sude Sena - 3.10
12 - Onar, Eylul - 3.10
13 - Sakallı, Irem - 3.07
14 - Okanoglu, Bora - 3.02
15 - Ozerli, Ahmet sevki - 3.02
16 - Akalın, Cahit Batu - 3.00
17 - Yılmaz, Mert - 3.00
18 - Ozturk, Zeynep Serra - 2.97
19 - Mete, Sabanur - 2.95
20 - Ogmen, Ayse Esra - 2.95
21 - Zengin, Burcu - 2.95
...
```

Programming and Coding Advice

It would be easier for you to implement the algorithm of this problem if you **first** try to **draw the flowchart** or **write the pseudocode** so that you can go over your solution to see if there are any errors.

Additionally, this homework is not short to implement and it will be very difficult and problematic to code the solution without decomposing the process into smaller pieces. Indeed, it is always a good idea to **use decomposition and pattern recognition** in programming, regardless of the length of the problem. Just define subproblems in the whole problem, and try to solve these subproblems before.