# Assignment 3

- Rules same as earlier assignments e.g. to be done individually (discussing the problems with each other is fine), code to be uploaded, marks, bonus marks policy, etc.
- There is only one bonus problem - a group project. It is given in the end.
- For these problems, as before you will have to create your test data. (TAs will create their own data for testing during demos)
- If any details are missing - make reasonable assumptions and state them as comments at the top of the program.
- Any changes/clarifications to the problems will be highlighted in blue.

**Question 1.**

Print the pattern given below using recursion - no loops are allowed. You are given a value n - print a diamond which will have 2n-1 rows, and 2n columns, as shown in the figure. (In the top row print n+n stars, in the next (n-1) stars on left and right and 2 blanks in the middle, and so on till you have 1 star each on left and right; then you reverse this).

Suggestion: Have two recursive functions - one to print the top half, and other to print the bottom half.


**Question 2.**

For IIIT-D students living off-campus (we have used IP TA names for this), suppose records of whenever a student enters or exits the campus are maintained in a data file. A typical data entry will look like this: "**Student_Name**, **Crossing** (can be ENTER or EXIT), **Gate-number, Time** (in 24 hr format)**"**
If in the record, a student is shown to enter even if he had previously entered the campus, i.e., two ENTER entries before EXIT, take the first one. Similarly, there can be two consecutive EXIT entries from the campus for a student - if so, take the last one. You are given the data for one day (A sample data file can be downloaded from **File**). If for a student there is EXIT but there is

no ENTRY - it means he/she came the day before; similarly if ENTRY but no EXIT, it means that the student will leave next day. To avoid special situations, it is best if you first sort this data w.r.t time.
**Note**: Use of inbuilt libraries like datetime, etc. is not allowed.

Convert this data into a ***nested dictionary.*** The keys should be the name and value should be another dictionary containing a list of gate no, crossing type, time. Use this dictionary to answer the following queries (for querying, you can write a small loop and ask for a number between 1 and 3 - nothing given can be the end).

1) Given a **student name** (as input), show the record of students moving in/out of campus (as a list of tuples) in the day (in a output text file), and whether currently present in campus or not. Take another **input for current time** as well.
2) Given the **start time** and the **end time** (in 24hr format, both inclusive) as input, determine all the students who entered the campus during this, and all students who exited the campus during this time. Save the result into an output text file, with the format similar as the input data file.
3) Given the **gate number** (as input), determine the number of times students have entered the campus through that gate, and the number of times students have exited the campus from that gate.

**Question 3.**

As a TA for Communication Skills course, you have been requested to automate evaluating and scoring assignments (which are writing assignments for which each student submit a text file). You are to automatically evaluate each answer file.

For names of the input files - you can hard code it in a list, and then process them one by one. Or you can name the files as FILE1.txt to FILEn.txt, and take an integer input regarding the number of files and then read them. You need to write out the output files in this directory itself.

To score an answer file first determine:

F1 factor = (Unique words/Total words)
F2 factor = (total occurrences the top 5 most occurring words)/Total words
F3 factor = (number of sentences >35 words or < 5 words)/Total sentences
F4 factor = (Frequency of consecutive [comma+full-stop+colon+semicolon]/Total words)
F5 factor = 1 if (Total word count > 750 words), else 0.

Net score is: 4 + F1*6 + F2*6 -F3 - F4 - F5.

Slight change in the score formula - but older one is also fine.

Some clarifications:

1. The total sentence count needs to be computed after discounting repeated full stops. For example - I am a student of IIIT Delhi… Here, there are a few consecutive full-stops. We will, however, consider this as a single sentence. And increase the count of consecutive full-stops/ commas/ semi-colons/ colons /hyphens by one regardless of how many full-stops there are (.. is one and ….. is also one, so is .,.,.,).
2. Consider words of the same spelling but different cases as the same word.

Output is to be written in the file scores.txt , as follows:

1. Write the filename on one line.
2. On the next line, write the student's score with the text “score: ”[without quotes].
3. On the next line, write out the five most used words in descending order of usage.
4. On the next line, write out five randomly selected words from the submission.

(Points 3 and 4 help to see whether the assignment submission is valid, by checking whether the words are actual English words or not. Your program does not have to check whether these are English words - this explanation is given just to tell you why points 3 and 4 are to be done.)

***Extension (voluntary - i.e. you decide if you want to do it)***.

After a splendid and speedy evaluation of the Assignment submissions, your Instructor further piles more work on you. They ask you to check for plagiarism in the student’s submissions by looking at the similarity between their submissions.

Using the data of unique words and the total words of each submission, calculate the similarity score of each submission with every other submission, and if the similarity score exceeds 50%, then write the name of the submission file that is being evaluated for plagiarism, along with the name of the file with which plagiarism had been detected, and the similarity score between them.

For similarity score computation of File1 with File2, first:

1. Determine fCW: common words, with the common frequency in both (i.e. minimum of frequency in the two files)
2. Use fCW to determine nSW - which is the sum of all the frequencies of common words - to get the total number of same words
3. Use fCW to get the number of unique words common to both submissions(nSUW).

Similarity score for File1 sim_index (with File2 is):

sim _index = 100% x [(nSW/total words in File1)+(nSUW/total unique words in File 1)]/2

For eg, if the two submissions are:
File1:
My name is Sheldon Is.
File2:

My name is Howard Is.(We are taking fCW as a dictionary, you may choose a data type of your choice from
those taught in the class)

fCW = {“My”:1, “Name”:1, “is”:2 }
nSW = 1 + 1 + 2 = 3
nSUW = 3 (“My”, “Name” and “is”)
sim_index for File1 = 100% x (⅘ + ⅗ )/2 = 70%

Output into a file named plag.txt. One line needs to be printed for each submission with a similarity score of >50%, structure of which is:

curfile targetfile sim-index separated by spaces

Where:
The file that is currently being evaluated -> curFile
The file against which the highest similarity has been detected -> targetFile
Similarity score of curFile against targetFile -> sim_index
For example:
2023224.txt 2023343.txt 67%
2023236.txt 2023168.txt 54%
.
.
.

**Questions 4, 5, 6 -** This statement is for the next three problems.

Goal is to develop an application for grading in courses at IIIT-Delhi. The application should provide for (i) creating a course for grading with its name, credits, list of assessments with percentage weight for each (e.g. [("quiz", 5), ("mid-sem", 15), …], a grading policy as a list percent for different grades (we will assume that only grades are: A, B, C, D, and F, so this can be [80, 65, 50, 40, 30] (i.e. A above 80, B between 80 and 65, …, F below 30). However, the final cutoff for each grade will be within +/- 2 of the percent specified (i.e. for A it will be between 78 and 82) - it will be the higher of two consecutive marks within this range which have the highest difference. (So, if the marks are 81.9, 81.8, 81.7, 81, 80.9, 80.8…, then the cutoff will be midpoint of 81.7 and 81, which has a largest gap (ii) adding students' marks for all assessments given in a file - these will be rollno followed by marks for each. Once marks are given, it should do the grading - apply the policy to determine the grade for each student.

Change to: [80, 65, 50, 40] (i.e. A above 80, B between 80 and 65, …, F below 40)

It should then ask the professor what he wants to do: 1. Generate a summary - which should print the course info (name, credits), assessments and their weight, cutoffs for different grades, and grading summary (how many As, how many Bs, etc) 2. Print the grades of all the students in a file as: rollno, total marks, grade (one line for each student). 3. Search for a student record - given the roll no, show marks in different assessments, total marks, and final grade.

Develop this program and use it for grading the "IP" course. The assessments and the grading policy is given below (in the main program steps)

The main program can be something like (rough sketch):

1. cname, credits = "IP", 4
2. assessments = [("labs", 30), ("midsem", 15), ("assignments", 30), ("endsem", 25)]
3. policy = [80, 65, 50, 40, 30]
4. create-IP course (cname, credits, assessments, policy)
5. Upload-marks data - call a function/method with input as marks.txt
6. doGrading - call a function/method
7. Loop asking for what operation (1, 2, 3); perform the operation till no input given

**Question 4.** Write this program using class and objects - it seems that "Course" and "Student" will be natural classes. Write this program using classes and objects. You must have at least two main classes for course and student.

**Question 5**. Write this program without using classes and objects. (You may find dictionaries as suitable data structure.) Do the same problem without using oop.

**Question 6**. Compare the two approaches/code: (A): In your view what are the two pros and cons of the two approaches. (B Compare the performance of these two in terms of time taken. For this, use the package time. For measuring time taken, run the grading operation N (say 1000) number of times, and run searching for student N number of times (you can pick up a roll no randomly, or just give a few roll nos in a list and go through them repeatedly) and then compute the time for the two, summarizing which one is faster for (i) grading operation, and which one for (ii) search operation and by how much (takes what fraction of time of the other)

The answer to this question should be be in a .txt file, containing:
f
Advantages of using OO
1.
2.
Advantages of using Dictionaries
1.
2.

Disadvantage of using OO
1.
2.

Disadvantage of using dictionaries
1.
2.

Performance comparison for grading operation (give in each line: value of N; time taken by two approaches in seconds, and which is faster and by how much).
1. N:
2. Time by OO:
3. Time by dictionary:
4. X is faster; fraction of time x took is:

Performance comparison for search operation (give in each line: value of N; time taken by two approaches in seconds, and which is faster and by how much).
1. N:
2. Time by OO:
3. Time by dictionary:
4. X is faster; fraction of time x took is:

**====**
**Bonus Problem - a group project.**

Develop an application of your choice in a group of 2 or 3 students. You can use whatever libraries/packages you want. The project should require a reasonable amount of python code with a few functions and/or a few classes. Your group will have to demonstrate the working of the project separately from the rest of the assignment. These demos will probably be scheduled after the final exam - and you will be allowed to demo them remotely or in person.