Project 2:

Submission deadlines: 5:00 pm, Friday 27th May 2022
Value: **20%** of CITS1401.

To be completed individually.

You should construct a Python 3 program containing your solution to the following problem and submit your program electronically on Moodle. The name of the file containing your code should be your student ID e.g. `12345678.py`. No other method of submission is allowed. Your program will be automatically run on Moodle for sample test cases provided in the project sheet if you click the "check" link. However, your submission will be tested thoroughly for grading purposes after the due date. Remember you need to submit the program as a single file and copy-paste the same program in the provided text box. You have only one attempt to submit so don't submit if you are not satisfied with your attempt. All open submissions at the time of the deadline will be automatically submitted. There is no way in the system to open the closed submission and reverse your submission.

You are expected to have read and understood the University's guidelines on academic conduct. Following this policy, you may discuss with other students the general principles required to understand this project, but the work you submit must be the result of your own effort. Plagiarism detection, and other systems for detecting potential malpractice, will therefore be used. Besides, if what you submit is not your own work then you will have learned little and will, therefore, likely, fail the final exam.

You must submit your project before the submission deadline listed above. Following UWA policy, a late penalty of 5% will be deducted for each day (24 hours), after the deadline, that the assignment is submitted. No submissions will be allowed after 7 days following the deadline except approved special consideration cases.

Overview

Congratulations!! The researchers at UWA were very impressed by your 3D facial analysis skills in Project-1. They have decided to seek your help in another exciting project. They analyse 3D distances between significant facial landmarks to perform face recognition. They also analyse the relationship of these distances with certain syndromes, like Autism. For example, last month they published a research paper which found that the parents of autistic children had larger 3D facial distances compared to normal adult population. You can find out more about their work here and here.

The researchers now want to analyse ten 3D Euclidean distances between 15 significant facial landmarks. These distances on one face can then be used to calculate similarity with other faces in the data set to see which faces are closer to (or look like) the reference face. Table 1 provides the details of each landmark while Figure 1 shows their location on the face. Table-2 gives you the details of the distances to be calculated. Remember these distances are between the landmarks mentioned in Table-1.

In this project, you are required to write a computer program that can read the data from a CSV (comma separated values) file provided to you. The file contains the 3D coordinates in X, Y and Z axes for the 15 facial landmarks mentioned in Table 1 for each adult. For simplicity, the landmark abbreviations are provided in the CSV file instead of their full names. Your task is to write a program which fulfills the following requirements.

Table 1: *Facial landmarks' identification numbers and their details.*

Landmark	**Location**
Ex_L	Left Outer eye corner
En_L	Left Inner eye corner
N	Midline of nasal root @ nasofrontal structure
En_R	Right Inner eye corner
Ex_R	Right Outer eye corner
Prn	Nose Tip
Al_L	Lateral-most point of left nose contour
Al_R	Lateral-most point of right nose contour
Sbal_L	Base of left nose contour
Sbal_R	Base of right nose contour
Ch_L	Left outer mouth corner
Ch_R	Right outer mouth corner
Sn	Root of nasal midline
FT_L	Left forehead
FT_R	Right forehead

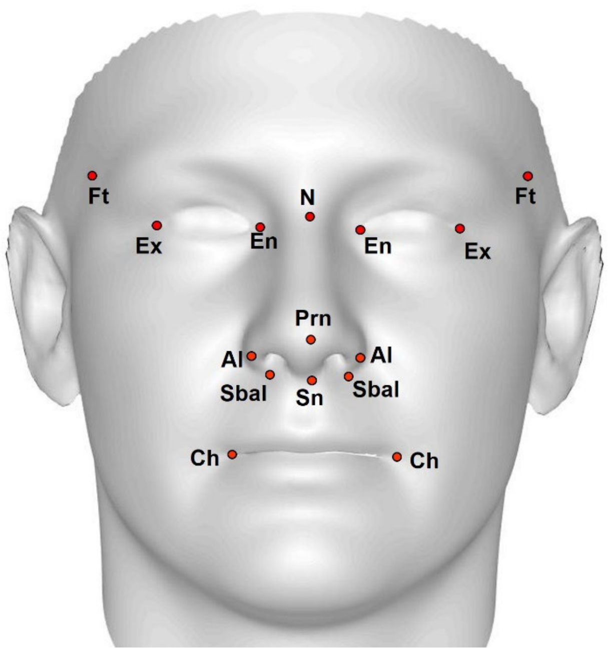

Figure 1: *Facial landmarks' locations on the face.*

#	**Facial distance**	**Abvn**	**LM1**	**LM2**
1	Forehead width	FW	FT_L	FT_R
2	Outer-canthal width	OCW	Ex_L	Ex_R
3	Left Eye fissure length	LEFL	Ex_L	En_L
4	Right Eye fissure length	REFL	En_R	Ex_R
5	Inter canthal width	ICW	En_L	En_R
6	Nose width	NW	Al_L	Al_R
7	Alar-base width	ABW	Sbal_L	Sbal_R
8	Mouth width	MW	Ch_L	Ch_R
9	Nasal bridge length	NBL	N	Prn
10	Nose height	NH	N	Sn

Table 2: List of distances to be calculated in this project. For example, Forehead width is abbreviated as 'FW' and is the 3D Euclidean distance between Left Forehead (FT_L) and Right Forehead (FT_R). Similarly Inner canthal width, abbreviated as 'ICW' is the distance between the two inner eye corners: En_L and En_R.

Specification: What your program is required to do

Input:

Your program must define the function `main` with the following syntax:

```
def main(csvfile, adultIDs):
```

The input arguments to this function are:

- `csvfile`: The name of the CSV file containing the facial record which needs to be analysed. Below are the first two rows of the sample file.

Adult ID	Landmark	X	Y	Z
A0001	Ex_L	-32.8506	-39.073	4.64672

-

The first row of the CSV file contains the following headers:

- Adult ID: The de-identified ID of an adult.
- Landmark: The facial landmark as mentioned in Table 1.
- "X", "Y" and "Z": The 3D location of the landmark in X, Y and Z axes respectively.

  We do not have prior knowledge about the number of adults we have to analyse (i.e. the number of rows) that the CSV file contains. Also we are not aware of the order of the columns, so your program needs to check for the column heading to retrieve respective information. The columns 'Adult ID' and 'Landmark' are strings data while the remaining data is numeric.

  Note: The X, Y and Z coordinates are in millimetres and needs to be within the bounds [-200,200]**.**

- `adultIDs`: A list containing two adult IDs, which need to be analysed. Remember that the ID is a string and is case insensitive.

Output:

The function is required to return the following outputs in the order provided below. For ease of description, we will refer to the input adult IDs as ***"F1"*** and ***"F2"***.

- **OP1:** A list of two dictionaries containing the facial distances (as mentioned in Table-2) for each face F1 and F2 respectively. The keys in the dictionaries are the abbreviations (case-sensitive) of the distances (e.g. FW, ICW etc.) and their values contain the 3D Euclidean distance between the corresponding landmarks (see last two columns of Table-2). The formula to calculate the Euclidean distance between two 3D landmarks is given at the end of this project sheet.
- **OP2:** The cosine similarity between faces F1 and F2 based on the ten distances calculated above. The formula to calculate cosine similarity is provided at the end of this project sheet.
- **OP3:** A list of two Tuple sequences. The first sequence contains the cosine similarity between face F1 and the five faces closest/most similar to F1 excluding face F2 (based on cosine similarity calculated using the ten distances). The second sequence has the same information for face F2 excluding face F1. In each tuple, the first member of each tuple is the "Adult ID" of the face while the second member is the cosine distance between this face and the reference face. The sequences must be arranged in the decreasing order of cosine similarity but if the cosine distance for two faces is exactly same then they should be arranged in alphabetical order of their Adult ID.
- **OP4:** A list of two dictionaries containing the average of each of the ten facial distances (See Table-2) of the closest five faces (Output 3) for the reference faces F1 and F2. The keys in the dictionaries are the abbreviations of the distances (e.g. FW, ICW etc) and their values contain the average of 3D Euclidean distance of all five closest faces.

  All returned numeric outputs (both in lists and individual) must contain values rounded to four decimal places (if required to be rounded off). Do not round the values during calculations and round them only at the time that you save them into the final output variables.

Examples:

Download `sample_face_data.csv` file from the folder of Project 2 on LMS or Moodle. Some examples of how you can call your program from the Python shell (and examine the results it returns) are:

```
>>> OP1, OP2, OP3, OP4 = main('sample_face_data.csv',['R7033', 'P1283'])
```

The output variables returned are:

```
>>> OP1 = [{'FW': 128.0695, 'OCW': 93.9636, 'LEFL': 33.092, 'REFL': 32.6327,
'ICW': 32.1305, 'NW': 31.7842, 'ABW': 18.3002, 'MW': 46.8504, 'NBL':
40.0941, 'NH': 62.7722}, {'FW': 123.9306, 'OCW': 100.081, 'LEFL': 34.4401,
'REFL': 32.3075, 'ICW': 37.3563, 'NW': 43.815, 'ABW': 28.1968, 'MW': 63.467,
'NBL': 38.416, 'NH': 60.9737}]

>>> OP2
0.9932

>>> OP3
[[('L8682', 0.9995), ('C9721', 0.9993), ('J0951', 0.9993), ('K5219',
0.9992), ('N0889', 0.9991)], [('A5474', 0.9991), ('D2742', 0.9987),
('H1286', 0.9982), ('G8293', 0.9976), ('U0216', 0.9975)]]

>>> OP4
[{'FW': 125.5241, 'OCW': 91.6592, 'LEFL': 31.7918, 'REFL': 31.2151, 'ICW':
31.2196, 'NW': 32.9661, 'ABW': 19.4147, 'MW': 48.1833, 'NBL': 39.1743,
'NH': 61.2486}, {'FW': 123.5339, 'OCW': 98.5101, 'LEFL': 31.9901, 'REFL':
32.1118, 'ICW': 36.7311, 'NW': 42.4031, 'ABW': 27.3566, 'MW': 57.6451,
'NBL': 42.074, 'NH': 63.3527}]
```

Additional requirements:
There are few more requirements for your program.

- Your program needs to validate the inputs to the main() function and gracefully terminate if invalid inputs are provided.
- You program needs to terminate *gracefully* if the file cannot be found or opened. For graceful terminations, you need to print the message related to the problem and return None for each unavailable output.
- Your program needs to validate the input data from the file. The X,Y or Z coordinate (or all coordinates) of a landmark could be corrupted or missing. In that case the value in the cell would be empty or out of bounds. If data is not correct, then entire row needs to be discarded.
- If a particular landmark of a face is missing or corrupted, then consider the entire adult's record corrupt.
- It is possible that the Research Assistant marking these landmarks on the faces could have forgotten to mark/ record a landmark. In this case, that particular landmark will be missing from the CSV file. You should follow the protocol given above in such cases.
- Your program needs to consider that record of the landmarks for a particular Adult may not have any specific order or can be in any order (excluding header row).
- The columns in the CSV file can be in any order and the headings are case insensitive i.e. AdultID, Landmark, X, Y and Z.
- Your program needs to convert all text data in the csv to be upper order alphabets to ensure that program is case insensitive.

Important grading instruction:

Note that you have not been asked to write specific functions. This task has been left to you. However, it is essential that your program defines the top-level function *main(csvfile, adultIDs)* (hereafter referred to as "main()" in the project sheet to save space when writing it. Note that when "main()" is written it still implies that it is defined with its two input arguments). The idea is that within main(),the program calls the other functions. (Of course, these functions may then call further functions.) This is important because when your code is tested on Moodle, the testing program will call your main() function. So, if you fail to define main(), the testing program will not be able to test your code and your submission will be graded zero. Don't forget the submission guidelines provided at the start of the project sheet.

Things to avoid:

There are a few things for your program to avoid.

- **You are not allowed to import any Python module.** While use of many of these modules, e.g. csv or math is a perfectly sensible thing to do in production setting, it takes away much of the point of different aspects of the project, which is about getting practice opening text files, processing text file data, and use of basic Python structures and sequences.
- Do not assume that the input file names will end in .csv. File name suffixes such as .csv and .txt are not mandatory in systems other than Microsoft Windows. Do not enforce that within your program that the file must end with a .csv or any other extension (or try to add an extension onto the provided csvfile argument).
- Ensure your program does NOT call the `input()` function at any time. Calling the input() function will cause your program to hang, waiting for input that automated testing system will not provide (in fact, what will happen is that if the marking program detects the call(s), it will not test your code at all which may result in zero grade).
- Your program should also not call the `print()` function at any time except for the case of graceful termination. If your program has encountered an error state and is exiting gracefully then your program needs to return as mentioned in additional requirements and print an appropriate message.

Disclaimer: Although this project addresses a real-world problem, the files provided to you contain synthetically generated data.

Submission:

Submit your solution on Moodle before the deadline as per details provided at the start of the project sheet.

You need to contact unit coordinator if you have special considerations or making a submission after the mentioned due date.

Marking Rubric:

Your program will be marked out of 30 (later scaled to be out of 20% of the final mark).

22 out of 30 marks will be awarded based on how well your program completes a number of tests, reflecting normal use of the program, and also how the program handles various states including, but not limited to, different numbers of rows in the input file, missing asymmetry record and / or any error states. You need to think creatively what your program may face. <u>Your submission will be graded by data files other than the provided data file. Therefore, you need to be creative to look into corner or worst cases.</u> I have provided few guidelines from ACS Accreditation manual at the end of the project sheet which will help you to understand the expectations.

8 out of 30 marks will be awarded on *style* (5/8) "the code is clear to read" and *efficiency* (3/8) "your program is well constructed and runs efficiently". For style, think about use of comments, sensible variable names, your name at the top of the program, etc. (Please watch the lectures where this is discussed).

Style Rubric:

0	Gibberish, impossible to understand
1-2	Style is really poor or fair
3-4	Style is good or very good, with small lapses
5	Excellent style, really easy to read and follow

Your program will be traversing text files of various sizes (possibly including large csv files) so you need to minimise the number of times your program looks at the same data items.

Efficiency Rubric:

0	Code too incomplete to judge efficiency, or wrong problem tackled
1	Very poor efficiency, additional loops, inappropriate use of `readline()`
2	Acceptable or good efficiency with some lapses
3	Excellent efficiency, should have no problem on large files, etc.

Automated testing is being used so that all submitted programs are being tested the same way. Sometimes it happens that there is one mistake in the program that means that no tests are passed. If the marker is able to spot the cause and fix it readily, then they are allowed to do that and your - now fixed - program will score whatever it scores from the tests, minus 4 marks, because other students will not have had the benefit of marker intervention. Still, that's way better than getting zero. On the other hand, if the bug is hard to fix, the marker needs to move on to other submissions.

Extract from Australian Computing Society Accreditation manual 2019:

As per Seoul Accord section D,

A complex computing problem will normally have some or all of the following criteria:

- involves wide-ranging or conflicting technical, computing, and other issues;
- has no obvious solution, and requires conceptual thinking and innovative analysis to formulate suitable abstract models;
- a solution requires the use of in-depth computing or domain knowledge and an analytical approach that is based on well-founded principles;
- involves infrequently-encountered issues;
- is outside problems encompassed by standards and standard practice for professional computing;
- involves diverse groups of stakeholders with widely varying needs;
- has significant consequences in a range of contexts;
- is a high-level problem possibly including many component parts or sub-problems;
- identification of a requirement or the cause of a problem is ill defined or unknown.

Formulas:

3D Euclidean Distance:

The 3D distance (d) between two points A = (x1, y1, z1) and B = (x2, y2, z2) in cartesian plain is calculated as,

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2})$$

You can find more details at https://en.wikipedia.org/wiki/Euclidean_distance

Cosine Similarity Score

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Suppose we have two dictionaries A and B that contains the 3D distances between landmarks, A = {"FW": 5, "OCW":2, "LEFL":1, "REFL":0, "ICW": 3, "NW": 5, "ABW":2, "MW":1, "NBL":0, "NH": 3} and B = {"FW": 4, "OCW":0, "LEFL":12, "REFL":9, "ICW": 7, "NW": 4, "ABW":0, "MW":12, "NBL":9, "NH": 7}

The cosine similarity of A and B can be calculated using the below formula and the answer will be:

$$similarity(A,B) = \frac{(5*4)+(2*0)+(1*12)+(0*9)+(3*7)+(5*4)+(2*0)+(1*12)+(0*9)+(3*7)}{\sqrt{5^2+2^2+1^2+0^2+3^2+5^2+2^2+1^2+0^2+3^2} * \sqrt{4^2+0^2+12^2+9^2+7^2+4^2+0^2+12^2+9^2+7^2}} =$$

$$\frac{106}{\sqrt{78} * \sqrt{580}} = \frac{106}{212.7} = 0.4984$$

You can find more details at https://en.wikipedia.org/wiki/Cosine_similarity