Coursework 2 Assignment Specification
(60 % of the module assessment)
Submission Deadline: at 9am on 17th January 2022

Student Monitoring System

1 Problem Description

1.1 Overview

Your task is to write a simple student monitoring system for a module leader/lecturer. The module leader should be able to check whether students have engaged with the formative and summative online test activities on the module VLE page. The system will use a database which contains the student results of all the tests. For each student assessment, the following information are stored: Timestamps of start and end of the assessment, grade of each question, total grade and a unique ID number which can be used to identify each student.

1.2 Test Results (20 marks)

Your program should include functionality to find out all the test results of a student. Given student ID, your program should return a complete list of all the student's assessments with all their associated marks, and appropriately visualises the list by using the Matplotlib module.

1.3 Student Performance (20 marks)

The lecturer should be able to perform an analysis of any test in the database to find out student performance on each question of the assessment. Both relative and absolute performance of the student need to be calculated and suitably visualised for each question.

- Absolute performance is a percentage grade (i.e., student grade out of 100).
- Relative performance is a distance number between student's grade and test average. A distance number could be simply a subtraction of the student grade and average grade. Alternatively, the number could be calculated by using a relevant statistical function.

1.4 Underperforming Students (30 marks)

The module leader would like to support underperforming students. Your program should use all the test results in the database to find out those students and display their test results. Student IDs must be sorted by their grades of the summative online test. The lowest grade of their formative tests should be highlighted in the output for each student.

1.5 Hardworking Students (30 marks)

Students were asked to rate their prior knowledge and skills regarding to the module contents at the start of a module (e.g., rate your programming knowledge skills) and their initial rate were saved in the StudentRate.csv file. You assume that the student self-rates are correct and utilise the file to create a list of hardworking students. The list includes the student IDs, their grades of the summative test and their rates. The program should display the list sorted in descending order of their grades.

2 How to Structure Your Program

The following structure is needed as a final submission for your project:

Data Files: Stores all the data (see previous section).

Resultdatabase.db stores the student results of all the tests in an SQLite database. This database must be the same as the database submitted as a part of CW 1.

StudentRate.csv stores survey data which include the student self-rates. This file is given on Learn. You are allowed to modify the data to test your program functionality.

testResults.py: A Python module which contains functions used to ask for student ID and display all test results of the student as described in the previous section.

studentPerformance.py: A Python module which contains functions used to ask for student ID and the title of the test, the lecturer wishes to analyse. Then, after performing the validity checks (i.e., correct ID and valid test title) and carry out the functionality as described in the previous section.

underperformingStudent.py: A Python module which contains functions used to allow the lecturer to see test results of underperforming students as described in the previous section. You should come up with the details of the criteria to identify underperforming students. Your criteria must exclude the disengaged students, who didn't attempt many formative tests. For example, students are underperforming if their grades of the summative test and all formative tests are between 1 and 50.

hardworkingStudents.py: A Python module which contains functions used to list hardworking students as described in the previous section. The program should classify the students as hardworking if their grade of the summative online test is more than 60, and their rates are "Below Benninger" or "Beginner" in the file.

DAFunction.py: A Python module which contains common functions that the Test Results, Performance, Underperforming and Hardworking Students modules use to carry out their functionalities.

menu.py: A python main program which provides the required menu options to the module leader for the program functionalities. The menu could be based on the Python **G**raphical **U**ser **I**nterface (namely the tkinter python module).

3 What to Submit

In addition to the files mentioned in Section 2 you may write a short text file called README (max 500 words). This is to provide any special instructions or warnings to the user (or assessor!), or to draw attention to any aspects of the program that you are particularly proud of (please don't waste time by writing an excessive amount).

All the files (including sample data files) should be compressed into a zip file and submitted electronically as directed on the module Learn page.

4 Notes on Expectations:

You will be marked according to the Assessment Matrix that will be provided to you separately from this document. However below follows a qualitative description of some general expectation that may help you understand the general level of expectation associated with this piece of coursework.

Technical mastery of Python Your programs should show mastery of what you have been taught.

Design Your programs should be well structured for the task in hand so that it is as easy as possible for:

- a user to use the program for any likely purpose,
- a programmer to understand the code structure and be able to develop it further,
- a programmer to be able to re-use as much as possible of the code in a related application.

Clarity and Self-Documentation Given the structure of your programs, they should be as easy to read and understand as possible. *Lay your code out* so that it can be listed sensibly on a variety of devices: avoid having any lines longer than 80 characters as these may wrap (to reduce the number of "problem lines" you should use 4 spaces for indentation rather than tabs). *Sensible names* should be chosen for all variables, methods etc. *Documentation strings* should be included for each:

Program Fully explain what the program does and how it should be used. Also state who wrote it and when.

Function State what each function does and explain the roles of its parameters.

In addition, you should include occasional comments in your code; these may be (a) to introduce a new section in the code, or (b) to explain something that is not obvious. Bear in mind that pointless comments make your code harder to read, not easier.

Implementation requirements

1) Your code must store data into the SQLite Database
2) See a copy of a Conda environment (yml file) on the module Learn page. Submitted code must work in the environment which includes
    a. Python v3.8 or above.
    b. Python standard libraries
    c. Matplotlib.
    d. Numpy.
    e. Grapviz
    f. Pandas