Assignment 1 – Mini-BearTracks

Due Date: Friday, October 13th 2023 at 23:55 (Edmonton time)
Percentage overall grade: 6.66%
Penalties: No late assignments allowed
Maximum marks: 100

Goal: refresher of Python and hands-on experience with file input/output, dictionaries, and string manipulation.

Assignment Specifications

You are tasked with creating a Python program that is a simplified version of the University of Alberta's BearTracks registration system. Your Mini-BearTracks will be flexible enough that other educational institutions can use it (not just the University of Alberta). That's because your Mini-BearTracks must be able to read in information about students and courses at that institution, and use that information to allow students to enroll into new courses or drop existing courses. Your Mini-BearTracks must also be capable of displaying a given student's timetable.

Input

You will be provided with **THREE** text files. These text files will be updated over time, so your program must be able to read in the most up-to-date information every time your program starts.

The first text file is called *students.txt*, and it contains information about each student registered at the educational institution. Each line contains the student's unique student ID, followed by the abbreviation for the faculty that they are registered in (e.g. SCI = Science), followed by the student's complete name. All of these items are separated by a comma (",") and varying amounts of whitespace, as can be seen in the sample *students.txt* provided with the assignment description.

The second text file is called *courses.txt*, and it contains information about the courses offered at the educational institution. Specifically, each line contains:

- The course name, which is a course abbreviation code, followed by a single space, followed by a three-digit whole number (e.g. CMPUT 175). The course abbreviation codes will have varying numbers of capital letters. Please note that your Mini-BearTracks will only be able to handle unique course names. So, for example, only one section of CMPUT 175 can be offered with this system. (This simplification is part of what makes it "mini".) Therefore, you can assume that all of the course names in *courses.txt* only appear once. (Your program does not need to check this.)

- The initials of the days that the course runs, followed by a single space, followed by the course's starting time (shown in the 24-hour time format). For example, if a course runs Mondays, Wednesdays, and Fridays and starts at 2pm, it would be shown as MWF 14:00. You can assume that there are only two formats for the courses:
    - All MWF (Monday, Wednesday, Friday) courses are 60 minutes long, and
    - All TR (Tuesday, Thursday) courses are 90 minutes long.

  There are no labs or seminars that only occur once a week.

- The maximum number of students who can register for the course.
- The name of the lecturer teaching the course.

All of these items are separated by a semi-colon (";") and varying amounts of whitespace, as can be seen in the sample *courses.txt* file provided with the assignment description. The lines in the file are not arranged in any particular order.

The third text file is called *enrollment.txt*, and it contains information about which students are registered in each course. Each line, **except for the very last line**, contains the course name, followed by a colon (":"), followed by varying amounts of whitespace, followed by the student's ID. The last line of the file will be empty. You can assume that only course names that appear in *courses.txt* will appear in this file. You can also assume that only student IDs that appear in *students.txt* will appear in this file, and the enrollment information for a given student in a particular course only occurs once. (Your program does not need to verify these assumptions.)

Output

Menu:

Your program should display a menu on the screen which allows the user to keep selecting options until s/he chooses to quit the program. When you first start the program, it should look exactly like this:

```
==========================
Welcome to Mini-BearTracks
==========================

What would you like to do?
1. Print timetable
2. Enroll in course
3. Drop course
4. Quit
>
```

The only valid options for the user to enter are the numbers 1, 2, 3, or 4. If the user enters an invalid choice, your program should display an error message and then re-prompt the user to enter a valid choice. It should continue to re-prompt until a valid choice is entered. For example:

```
What would you like to do?
1. Print timetable
2. Enroll in course
3. Drop course
4. Quit
> 5
Sorry, invalid entry. Please enter a choice from 1 to 4.
> abc
Sorry, invalid entry. Please enter a choice from 1 to 4.
>
```

Option 1:
When option 1 is chosen, the user should be prompted to enter a student ID. If this student ID is not valid (i.e. it is not an ID in *students.txt*), an error message should be displayed and the main menu is displayed again for the user to make another choice. For example, in the sample *students.txt* provided, 000000 is not a valid student ID because it does not exist in the file:

```
What would you like to do?
1. Print timetable
2. Enroll in course
3. Drop course
4. Quit
> 1

Student ID: 000000
Invalid student ID.  Cannot print timetable.

What would you like to do?
1. Print timetable
2. Enroll in course
3. Drop course
4. Quit
>
```

However, if the user enters a valid student ID, the program will display the student's name (all in capital letters), and faculty abbreviation (all in capital letters) on the screen. It will then display either the simple or challenging version of the student's timetable of enrolled classes, as described below.

<u>Simple Print Timetable Option (worth maximum of 15/20)</u>:
In this simple version, the columns will represent the times, and the rows will represent the days, as shown below. (The timetable is shown in a smaller font just to fit on this page. Please also refer to **sampleOutput_simple.txt**.) We recommend that you start out with this simpler version, so that you can easily test the rest of your program and don't get stuck. If you have time, you can do the challenging option to get the extra 5 marks.

```
What would you like to do?
1. Print timetable
2. Enroll in course
3. Drop course
4. Quit
> 1

Student ID: 123456
Timetable for MARY LOU SOLEIMAN, in the faculty of SCI
```

	8:00	9:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00
Mon/Wed/Fri	STAT 151 229	CMP* 175 114			MATH 101 71			ENGL 125 0	
Tues/Thurs		MATH 209 139			CMP* 272 157				

Notice that the schedule will always start at 8am, and end at 5pm (17:00). Monday/Wednesday/Friday classes will always be 60 minutes, and Tuesday/Thursday classes will always be 90 minutes (as shown). The borders of the timetable should look exactly as shown, and the column widths should also match the sample output exactly. If the student is registered in a course in a given time slot, the course name should be displayed (centered in the column), as should the number of **open** seats (also centered in the column). If the course abbreviation code is longer than 4 characters, it should be truncated to 3 letters with an asterisk (`*`) appended to the end. For example, `CMPUT` should be displayed as `CMP*`.

Challenging Print Timetable Option (worth maximum of 20/20):
In this challenging version, the columns will correspond to each day (Monday – Friday), and the rows will correspond to times (8:00 – 17:00), as shown below. (Please also refer to **sampleOutput_challenge.txt**.)

```
Student ID: 123456
Timetable for MARY LOU SOLEIMAN, in the faculty of SCI

        Mon        Tues       Wed       Thurs      Fri
    +----------+----------+----------+----------+----------+
8:00 | STAT 151 |          | STAT 151 |          | STAT 151 |
     |   229    |          |   229    |          |   229    |
     |          |          |          |          |          |
8:30 |          |          |          |          |          |
     |          |          |          |          |          |
     |----------|          |----------|          |----------|
9:00 | CMP* 175 |          | CMP* 175 |          | CMP* 175 |
     |   114    |          |   114    |          |   114    |
     |          |----------|          |----------|          |
9:30 |          | MATH 209 |          | MATH 209 |          |
     |          |   139    |          |   139    |          |
     |----------|          |----------|          |----------|
10:00|          |          |          |          |          |
     |          |          |          |          |          |
     |          |          |          |          |          |
10:30|          |          |          |          |          |
     |          |          |          |          |          |
    +----------+----------+----------+----------+----------+
11:00|          |          |          |          |          |
     |          |          |          |          |          |
     |          |          |          |          |          |
11:30|          |          |          |          |          |
     |          |          |          |          |          |
     |----------|          |----------|          |----------|
12:00| MATH 101 |          | MATH 101 |          | MATH 101 |
     |    71    |          |    71    |          |    71    |
     |          |----------|          |----------|          |
12:30|          | CMP* 272 |          | CMP* 272 |          |
     |          |   157    |          |   157    |          |
     |----------|          |----------|          |----------|
13:00|          |          |          |          |          |
     |          |          |          |          |          |
     |          |          |          |          |          |
13:30|          |          |          |          |          |
     |          |          |          |          |          |
    +----------+----------+----------+----------+----------+
14:00|          |          |          |          |          |
     |          |          |          |          |          |
     |          |          |          |          |          |
14:30|          |          |          |          |          |
     |          |          |          |          |          |
     |----------|          |----------|          |----------|
15:00| ENGL 125 |          | ENGL 125 |          | ENGL 125 |
     |    0     |          |    0     |          |    0     |
     |          |----------|          |----------|          |
15:30|          |          |          |          |          |
     |          |          |          |          |          |
     |----------|          |----------|          |----------|
16:00|          |          |          |          |          |
     |          |          |          |          |          |
     |          |          |          |          |          |
16:30|          |          |          |          |          |
     |          |          |          |          |          |
    +----------+----------+----------+----------+----------+
```

The borders of the timetable should look exactly as shown, and the column widths should also match the sample output exactly. As with the simple version, if the student is registered in a course in a given time slot,

the course name should be displayed (centered in the column), as should the number of open seats (also centered in the column). If the course abbreviation code is longer than 4 characters, it should be truncated to 3 letters with an asterisk (*) appended to the end. **To complete this challenging version well, it's very important that you break the table down into lines and look for patterns among those lines.**

Regardless of whether you are printing the simple or challenging timetable (only display one), after the appropriate output has been displayed, the main menu is displayed again for the user to make another choice.

Option 2:
When the user chooses option 2, the user should be prompted to enter a student ID. If this student ID is not valid (i.e. it is not an ID in *students.txt*), an error message should be displayed and the main menu is displayed again for the user to make another choice. (Please refer to **sampleOutput_challenge.txt** for exact error message.)

If a valid student ID is entered, the user should then be prompted to enter a course name. If that course name is:

- invalid (i.e. it is not in *courses.txt*), or
- the student is already registered in a course at the same time, or
- if the selected course is full (i.e. has zero open seats),

the appropriate error message should be displayed and the main menu is displayed for the user to make another choice. (Again, please refer to **sampleOutput_challenge.txt** for exact error messages.)

If all entries are valid, the appropriate data structures should be updated, the number of open seats for that course should be decreased by one, and a confirmation message including the student's name (exactly as entered in *students.txt*), the course name, and the day/time of the course should be displayed, as shown below:

```
What would you like to do?
1. Print timetable
2. Enroll in course
3. Drop course
4. Quit
> 2

Student ID: 123456
Course name: CMPUT 174
Mary Lou Soleiman has successfully been enrolled in CMPUT 174, on TR 8:00
```

Once the student has been successfully enrolled in the new course, the main menu should be displayed for the user to make another choice.

Option 3:
When the user enters option 3, the user should be prompted to enter a student ID. If this student ID is not valid (i.e. it is not an ID in *students.txt*), an error message should be displayed and the main menu is displayed again for the user to make another choice. (Please refer to **sampleOutput_challenge.txt** for exact message.)

If a valid student ID is entered, an alphabetical list of all of the courses that the student is currently registered in should be displayed, along with a prompt to enter the course to drop. If the entered course name is invalid (i.e. it is not among the courses that the student is registered in), an error message is displayed, and the main menu is displayed for the user to make another choice. (Again, please refer to **sampleOutput_challenge.txt** for exact error message.)

However, when a valid course name is entered, the student is removed from that course and a confirmation message is displayed. In other words, the appropriate data structures are updated and the number of open seats for the course increases by one.

```
What would you like to do?
1. Print timetable
2. Enroll in course
3. Drop course
4. Quit
> 3

Student ID: 123456
Select course to drop:
- CMPUT 174
- CMPUT 175
- CMPUT 272
- ENGL 125
- MATH 101
- MATH 209
- STAT 151
> CMPUT 174
Mary Lou Soleiman has successfully dropped CMPUT 174
```

Once the student has successfully dropped the course, the main menu should be displayed for the user to make another choice.

Option 4:
When the user chooses option 4, *enrollment.txt* must be updated to reflect all of the successful course enrollments (i.e. via option 2) and successful course drops (i.e. via option 3). Remember that enrollment information about a given student in a particular course should only appear once in the file. Once that information is saved, a goodbye message is displayed and the program will end:

```
What would you like to do?
1. Print timetable
2. Enroll in course
3. Drop course
4. Quit
> 4
Goodbye
```

Testing

Use the sample input text files to test your code. However, keep in mind that the markers will test your code with DIFFERENT input data in the input text files. You should test your code by adding at least one new student to *students.txt*, an additional course to *courses.txt*, and additional enrollments to *enrollment.txt*. You should also create additional tests to be confident that your program is working correctly.

Assessment

In addition to making sure that your code runs properly, we will also check that you follow good programming practices. For example, divide the problem into smaller sub-problems, and write functions to solve those sub-problems so that each function has a single purpose; use the most appropriate data structures for your algorithm (**HINT: use Python dictionaries instead of parallel lists**); use concise but descriptive variable names; define constants instead of hardcoding literal values throughout your code; include meaningful comments and docstrings to document your code; and be sure to acknowledge any collaborators/references in a header comment at the top of your Python file.

Restrictions for this assignment: **you cannot use break/continue, and you cannot import any modules. Doing so will result in deductions.**

Rubric

- Code quality and adherence to the specifications: 20%
- File handling (input/output) and appropriate use of dictionaries: 20%
- Program control and menu input validation: 10%
- Option 1: 20%
- Option 2: 15%
- Option 3: 15%

Hints for getting started

- Figure out your algorithms BEFORE trying to write your code.
- Break tables down into single lines of strings.
- There's a lot of different data here that forms various relationships – dictionaries will be your friend when solving this problem.

Submission Instructions

Please follow these instructions to correctly submit your solution:

- All of your code should be contained in a single Python file: **assignment1.py**.
- Make sure that you include your name (as author) in a header comment at the top of assignment1.py, along with an acknowledgement of any collaborators/references.
- Please submit your assignment1.py file via eClass before the due date/time.
- Do not include any other files in your submission.

- Note that late submissions ***will not be accepted***. You can make as many submissions as you like before the deadline – only your last submission will be marked. So submit early, and submit often.

REMINDER: Plagiarism will be checked for

Just a reminder that, as with all submitted assessments in this course, we use automated tools to search for plagiarism. In case there is any doubt, you **CANNOT** post this assignment (in whole or in part) on a website like Chegg, Coursehero, StackOverflow or something similar and ask for someone else to solve this problem (in whole or in part) for you. Similarly, you cannot search for and copy answers that you find already posted on the Internet. You cannot copy someone else's solution, regardless of whether you found that solution online, or if it was provided to you by a person you know. **YOU MUST SUBMIT YOUR OWN WORK.**