




COSC2531 Programming Fundamentals

Final Challenge

	<p>Assessment Type: Individual assignment; no group work.</p> <p>Submit online via Canvas → Assignments → Final Challenge.</p> <p>Marks awarded for meeting requirements as closely as possible.</p> <p>Clarifications/updates may be made via announcements/relevant discussion forums.</p>
	<p>Due date: end of Week 14; The deadline will not change.</p> <p>Please check Canvas → Assignments → Final Challenge for the most up to date information.</p> <p>As this is a major assignment, a university standard late penalty of 10% per each working day applies for up to 5 working days late, unless special consideration has been granted.</p>
	<p>Weighting: 40 marks out of 100</p>

1. Overview

The main objective of this final project is to assess your capability of program design and implementation for solving a non-trivial problem. You are to solve the problem by designing a number of classes, methods, code snippets and associating them towards a common goal. If you have questions, ask via the relevant Canvas discussion forums in a general manner.

2. Assessment Criteria

This assessment will determine your ability to:

1. Follow coding, convention and behavioral requirements provided in this document and in the lessons.
2. Independently solve a problem by using programming concepts taught in this course.
3. Design an OO solution independently and write/debug in Python code.
4. Document code.
5. Ability to provide references where due.
6. Meeting deadlines.
7. Create a program by recalling concepts taught in class, understanding and applying concepts relevant to solution, analysing components of the problem, evaluating different approaches.

3. Learning Outcomes

1. Analyse computing problems.
2. Devise suitable algorithmic solutions and code these algorithmic solutions in Python.
3. Develop maintainable and reusable solutions using the modular or object oriented paradigm.

4. Assessment details

my_school.py is a Python application for a school. It reads data from files. **IMPORTANT:** you should change data in these files to verify your program. We will use different files during marking.

Section 1: PASS Level

Your project is to implement the required functionalities in OO style with at least three classes, **School**, **Student** and **Course**. Design the appropriate instance variables, constructor(s) and methods for these classes. Class related info should be encapsulated inside of these classes.

At this level, your program can read from a file specified in command line, create **a list of Student objects** and **a list of Course objects**. Note at this level, we only know the ID of each course in the form of CXXX, e.g. 'C081', 'C082', and the ID of each student in SYYYY, e.g. 'S2123', 'S2024'.

Your program should create a School object, call its `read_scores(file_name)` method to load data from a text file specified in the command line. The data, student scores, should be stored in a 2D array. You may define instance variables and methods appropriate to support the functionalities.

scores.txt

```
34      C081  C082  C083  C084
S2123   99    75    85    62
S2025   -1    92    67    52
S1909   100   83    45    -1
```

Above is an example file that stores student score data. Data fields are separated by spaces and new lines. The first row contains course IDs and the first column contains student IDs. The first field in the data, the top left corner, shows the number of rows and the number of columns in one integer. For example '34', the first digit 3 means there are 3 students in this table. The second digit 4 means there are 4 courses. **You can assume that the total number of courses and students will never be more than 9.**

The table stores every student's final results in those courses. Results are all strictly integers. A result '-1' means not enrolled in that course. A '0' means the student did enrol but failed to receive any mark. No mark can go beyond 100, except '888', which means the student is enrolled but the result is not available yet for some reason. Such a course is NOT counted for result average.

Your program should show usage if no file is supplied. Otherwise it can show the students and list the student with the highest average score and display on the command line **exactly** as below:

```
> Python my_school.py
[Usage:] Python my_school.py <scores file>
```

```
> Python my_school.py scores.txt
```

	C081	C082	C083	C084
S2123	99	75	85	62
S2025		92	67	52
S1909	100	83	45	

```
3 students, 4 courses, the top student is S2123, average 80
```

Pending results will be shown as double dash -- in the above table.

Section 2: CREDIT Level --- You must ONLY attempt this level after you complete the PASS level

At this level, your program can support two types of courses. One is Compulsory Course. Compulsory courses may have different credit points. Another type is Elective Course. All elective courses have the same credit point. By default that is 6 points. Course title, credit points can be modified. Also it is possible to change type for a compulsory course, e.g. from C1 to C3. No compulsory course can be changed into an elective. No elective can change type. Private instance variables, appropriate getters and setters should be defined for courses.

A course should have a method to calculate its average score. The method should be able to handle situations like no enrolment, some results pending and all results pending.

Your program now can read one more file which stores the information of courses (see example below). Info includes course ID, course title (all titles are in one word), type of the course and credit points. C1, C2 are both compulsory courses. E means an elective course. You can assume all courses available in the school appear in this file and in the first file (student scores file). There are no duplicate or redundant courses.

courses.txt

```
C081 Mathematics C1 12
C082 Science C1 12
C083 English C2 24
C084 Technologies E 6
```

At this level your program can print a course summary on screen and save that summary into a file named as **course_report.txt**. Given the above **courses.txt**, your program output should look like below. The content of **course_report.txt** should also be the same, except the last line. Course names in the second column use * to indicate a compulsory course and - to indicate an elective course. The fourth column is the number of students enrolled in that course. The fifth column is the average score of the course.

```
> Python my_school.py scores.txt courses.txt
```

```
CID      Name          Pt. Enl. Avg.
-----
C081 * Mathematics  12   2   99
C082 * Science     12   3   83
C083 * English     24   3   65
C084 - Technologies  6    2   57
-----
```

```
The worse performing course is C084 with an average 57
```

```
courses_report.txt generated!
```

When there is no enrolment or all results pending, the last column of that course should show double dash -- . Such a course is EXCLUDED in the consideration of the worse performing course.

Section 3: DI Level --- You must ONLY attempt this level after you complete the CREDIT level

At this level, your program can support two types of Students, full time students (FT) and part time students (PT). A full time student is required to enrol in at least 3 compulsory courses. A part time student is required to enrol in at least 2 compulsory courses. These classes should have methods to check whether a student is meeting the minimum enrolment requirement.

Student information can be read from a file from command line as another argument. That file stores information about students, that includes student ID, name (no space between first name and last name, but an underscore) and study mode (FT or PT). You can assume all students appear in this file as well as in the first file (student results file). There is no duplicate records or empty records. See the example below.

students.txt

```
S2123 Sue_Vaneer FT
S2025 Robin_Smith FT
S1909 Barry_Banks PT
```

Your program can print a report of students on screen and store that report in a text file named as **student_report.txt**. Given the above **students.txt**, your program output should look like below. The content of **student_report.txt** should be the same, just without the last line.

```
> Python my_school.py scores.txt courses.txt students.txt
```

```
SID      Name           Mode  Enl.  GPA
-----
S2123 Sue_Vaneer    FT    4     3.25
S2025 Robin_Smith  FT    3 !   2.33
S1909 Barry_Banks PT    3     2.66
```

```
student_report.txt generated!
```

In the above report, the fourth column is the number of courses that student enrolled in. If the student did not meet the minimum requirement, then a **!** will appear next to that figure. See the second line Robin Smith in the above example. He only enrolled in 2 compulsory courses.

The fifth column is the average GPA. A course result of 80+ receives 4 GPA points. A result of 70-79 receives 3 points. A result in between 60-69 is 2 points. 50-59 gets 1 points. Under 50 has 0 points. For example Sue Vaneer has 2 HD, 1 DI and 1 CR. So her GPA is $(4 \times 2 + 3 + 2) / 4 = 3.25$.

Your program should be able handle situations like a student has not enrolled in any courses as yet or all results are pending. For these cases, a double dash **--** should be displayed under column GPA in the student report shown above.

Section 4: HD Level --- You must ONLY attempt this level after you complete the DI level

At this level, your program can handle some variations in the files.

- (1) ' new school scenario' where the scores file has no course, no student, just an empty file.
- (2) characters in sources.txt will be treated as -1, except 'TBA', which means enrolled but result not available as yet, same as '888'.
- (3) decimal numbers will be treated as integers, ignoring the decimal part, e.g 99.5 -> 99
- (4) The order of lines in all files, **scores.txt**, **students.txt** and **courses.txt** does not matter. (You can assume that the order of columns does not change.)

scores.txt

SID	C081	C082	C083	C084
34				
S2123	99.5	75	85	62
S1909	100	83.2	45	TBA
S2025	x	92	67	52

students.txt

SID	Name	Mode
S1909	Barry_Banks	PT
S2025	Robin_Smith	FT
S2123	Sue_Vaneer	FT

[Hint] You may find exception useful.

At this level your student report and the corresponding file **student_report.txt** is more advanced, taking credit points of each course into consideration. See below. The fourth column is now the total credit points that the student has completed. For example Sue Vaneer, she has done all four courses, so she earned $12 + 12 + 24 + 6 = 54$ credit points. The fifth column is the adjusted GPA. So that for Sue is $(4 \times 12 + 3 \times 12 + 4 \times 24 + 2 \times 6) / 54 = 3.55$, which is more accurate than that in the DI level.

A full time student now has an additional criteria, the minimum credit points of 50. Similarly, a part time student now need to earn at least 30 point and enrol in no less than 2 compulsory courses. These thresholds can be adjusted respectively. When adjusted, it will apply on all FT students or PT students. Your student classes should have appropriate private variables and methods to support the above. If a student does not meet the requirement, a ! will appear in the fourth column (see Robin Smith below, his total credit point is less than 50 points).

```
> Python my_school.py scores.txt courses.txt students.txt
```

SID	Name	Mode	CrPt	GPA
S2123	Sue_Vaneer	FT	54	3.55
S2025	Robin_Smith	FT	42 !	2.42
S1909	Barry_Banks	PT	48	2.0

```
student_report.txt generated!
```

In additional, students are listed in descending order of GPA.

[Maybe challenging]

At this level, your course report and student report are both accumulative, meaning a new report will not overwrite the existing ones but be placed on the top of the file. The latest report is always on the top. In addition, the date and time when the report was generated are also saved in the text files, just above the report, in the format "%d/%m/%Y %H:%M" .

[Maybe challenging]

Section 5: For All Levels

You should not use modules other than `sys` and `datetime`. The required functionalities must be solely created by yourself.

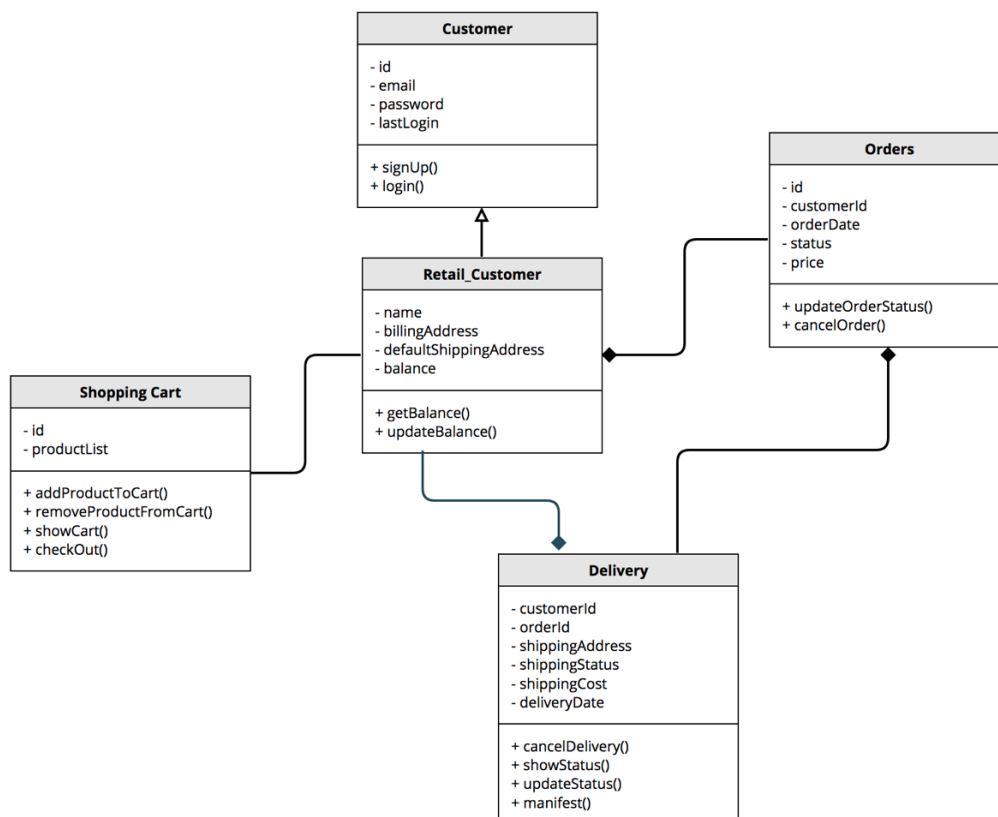
To verify the calculations, you can import the files, especial the provided test files, into a spreadsheet tool, e.g. Excel, Google Spreadsheet, Numbers, which can easily compute average, max, min etc.

Your program may have no interaction with users during execution. Simply run the code, read the files, display output and/or generate file(s).

You can assume user always type file names in the right order in command line, e.g. score file first, then course file, then student file. However it is possible that file is missing or cannot be found. Your program should quit gracefully in these circumstances.

Your program at all levels should **be fully OO**, e.g. no variables, methods or code snippets dangling outside a class. Your main program should simply create an object and run its methods to invoke methods from other classes to perform the required functionalities.

Design your classes carefully. You may use a diagram to assist the design. At DI level, you are required to provide a diagram to show your class design. At HD level, you are required to provide a detailed diagram similar to the example below, which is a class diagram for a different task. Variables and methods of each class are shown. Note if your code is at DI level, a detailed diagram would NOT result in more marks.



You could use tools like Powerpoint, Keynotes or online tools like moqups.com to draw the diagram. The diagram needs to be submitted in jpg, gif, png or PDF format.

Documentation requirements

Write comments in the same Python file, before code blocks (e.g. before methods, loops, ifs, etc) and important variable declarations. DO NOT write a separate file.

The comments at the beginning of your main code **MUST** contain the following information:

1. Your name and student ID.

2. The highest level you have attempted. That means you have completed all requirements of the levels below. Mark will be only given at the lowest level of partial completion. For example, you completed the PASS level, tried 50% of the CREDIT level, 30% of the DI level and 10% of the HD level, then your submission will be marked at the CREDIT level.

3. Any problems of your code and requirements that you have not met. For example, situations that might cause the program to crash or behave abnormally, or ideas/attempts for completing a certain functionality. Write these in the approximate locations within your code.

No need to handle or address errors that are not covered in the course.

5. Referencing guidelines

What: This is an individual assignment and all submitted contents must be your own. If you have used sources of information other than the contents directly under Canvas→Modules, you must give acknowledge the sources and give references.

Where: Add a code comment near the work to be referenced and include the reference in the IEEE style.

How: To generate a valid IEEE style reference, please use the [citethisforme tool](#) if unfamiliar with this style. Add the detailed reference before any relevant code (within code comments).

6. Submission format

Submit **one file** `ProgFunFinal_<Your Student ID>.zip`, which is the zipped file of all your files (Python code and a digram), via [Canvas→Assignments→Final Challenge](#). It is the responsibility of the student to correctly submit their files. Please verify that your submission is correctly submitted by downloading what you have submitted to see if the files include the correct contents.

1. Your final code submission should be clean, neat and abide by the formatting guidelines.
2. Identifiers should be named properly in consistent styles, e.g. upper camel case e.g. `UsedCar`, for classes and snake case for others, e.g. `get_price()`, `check_enrol_status()`.
3. You must include adequate meaningful code-level comments in your program.
4. **IMPORTANT:** your code must be able to run under command-line.

7. Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods,
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites.

If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the [University website](#).

8. Assessment declaration

When you submit work electronically, you agree to the [assessment declaration](#).

Code must be runnable under command line with no error,

```
> Python my_school.py
```

and runnable under command line

```
> Python my_school.py scores.txt
```

```
> Python my_school.py scores.txt courses.txt
```

```
> Python my_school.py scores.txt courses.txt students.txt
```

Submission failed to run would receive heavy mark deduction.

Rubric

Assessment Task	Marks
PASS Level	20 marks
CREDIT Level	4 marks
DI Level	4 marks
HD Level	8.5 marks
Others: Code quality and style; proper use of methods & arguments; good comments	3.5 marks