

Programming Fundamentals (COSC2531)

Final Coding Challenge

| | |
|------------------------|--|
| Assessment Type | Individual assessment (no group work). Submit online via Canvas/Assignments/Final Coding Challenge. Marks are awarded per rubric (please see the rubric on Canvas). Clarifications/updates may be made via announcements. Questions can be raised via the Canvas discussion forum. |
| Due Date | End of Week 14 (exact time is shown in Canvas/Assignments/Final Coding Challenge) Deadline will not be advanced nor extended. Please check Canvas/Assignments/Final Coding Challenge for the most up to date information regarding the assignment. As this is a major assignment, a university standard late penalty of 10% (i.e., 3 marks) per each day applies for up to 5 days late, unless special consideration has been granted. |
| Weighting | 30 marks out of 100 |

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, please ask via the relevant Canvas discussion forums in a general manner; for example, you should replicate your problem in a different context in isolation before posting, and you must not post your code on the Canvas discussion forum.

2. Assessment Criteria

This assignment will determine your ability to:

- i. Follow coding, convention, and behavioural requirements provided in this document and in the course lessons;
- ii. Independently solve a problem by using programming concepts taught in this course;
- iii. Design an OO solution independently and write/debug in Python code;
- iv. Document code;
- v. Provide references where due;
- vi. Meet deadlines;
- vii. Seek clarification from your "supervisor" (instructor) when needed via the Canvas discussion forums; and
- viii. Create a program by recalling concepts taught in class, understand and apply concepts relevant to solution, analyse components of the problem, evaluate different approaches.

3. Learning Outcomes

This assignment is relevant to the following Learning Outcomes:

1. Analyse simple computing problems.
2. Devise suitable algorithmic solutions and code these algorithmic solutions in a computer programming language (i.e., Python).
3. Develop maintainable and reusable solutions using object-oriented paradigm.

4. Assessment Details

Please ensure that you have read Sections 1-3 of this document before going further.

Problem Overview: In this final coding challenge, you are asked to develop a Python program with the Object-Oriented Programming paradigm, named **my_school.py**, that can read data from files and perform some operations. You are required to implement the program following the below requirements. Note that we will give you some files for you to run with your developed program, BUT you should change the data in these files to test your program. **During the marking, we will use different data/files to test the behavior of your program.**

Requirements: Your code must meet the following **functionalities**, **code**, and **documentation** requirements. Your submission will be graded based on the **rubric** published on Canvas. Please ensure you read all the requirements and the rubric carefully before working on your assignment.

A - Functionalities Requirements:

There are **4 levels**, please ensure you only attempt one level after completing the previous level.

----- **PASS LEVEL (15 marks)** -----

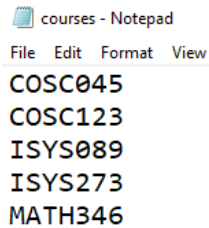
Your project is to implement the required functionalities in the Object-Oriented (OO) style with at least three classes: **Results**, **Course**, and **Student**. You need to design appropriate static/instance variables, constructors, and static/instance methods in these classes. The class related info should be encapsulated inside the corresponding class.

At this level, your program can read data from three files specified in the command line. The files store the course information (*course file*), student information (*student file*), and the results of students in these courses (*result file*). Your program should create a list of *Course* objects, a list of *Student* objects, and a variable (you can think carefully about which data type to use) to store the students' scores. You should design the classes properly so that these actions can be encapsulated within the appropriate classes. Note that, at this level, we only know the IDs of the students and the courses. These IDs are all unique.

In the main class, your program should create a *Results* object, call its *read_courses(course_file_name)* method to load data from the course file, *read_students(student_file_name)* method to load data from the student file, and *read_results(result_file_name)* method to load data from the result file, and finally call the *display_results()* method to display the results of the students in the required format (as specified in the following).

Below are examples of the three files: *course file*, *student file*, and *result file*.

In this level, the *course file* only consists of the course IDs with each course ID being in one line.

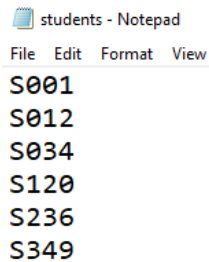


```

File Edit Format View
COSC045
COSC123
ISYS089
ISYS273
MATH346

```

Similarly, in this level, the *student file* also only consists of the student IDs with each student ID being in one line.

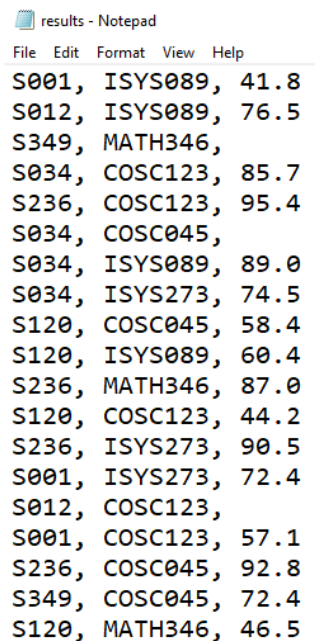


```

File Edit Format View
S001
S012
S034
S120
S236
S349

```

Finally, for the *result file*, the data fields in this file are separated by commas and new lines. Each row contains the student ID, the course ID, and the corresponding result of the student for that course. The format of each row is always *student_ID, course_ID, score*. For example, in our result file example, the student S001 enrolled in the course ISYS089 and got a score of 41.8. Or the student S012 also enrolled in the course ISYS089 and got a score of 76.5. Besides, there will be rows where the scores are empty, meaning the student enrolled in the course but haven't got the result of that course yet (on-going). You can assume there are no duplicate or redundant rows. In this level, you can assume the format of the data in the file is always correct.



```

File Edit Format View Help
S001, ISYS089, 41.8
S012, ISYS089, 76.5
S349, MATH346,
S034, COSC123, 85.7
S236, COSC123, 95.4
S034, COSC045,
S034, ISYS089, 89.0
S034, ISYS273, 74.5
S120, COSC045, 58.4
S120, ISYS089, 60.4
S236, MATH346, 87.0
S120, COSC123, 44.2
S236, ISYS273, 90.5
S001, ISYS273, 72.4
S012, COSC123,
S001, COSC123, 57.1
S236, COSC045, 92.8
S349, COSC045, 72.4
S120, MATH346, 46.5

```

Your program should print a message indicating how to run the program if an incorrect number of data files is passed in as command line arguments. Otherwise, it can display a table showing the scores of all students for all the courses, and two sentences showing the total number of students and courses, and the pass rate (in percent). The score of a student for each course is displayed as a number with 1 digit after the decimal point. In the table, if a student does not enroll in a particular course, then the data field at that location is empty. On the other hand, if a student already enrolled in a course but the

result is not available yet, then the data field at that location is shown as a double dash (--). Also, the data in the columns in the table are aligned, in particular, the student IDs are aligned to the left whilst the scores of all the courses are aligned to the right. Finally, the pass rate is computed as the percentage of passed scores (equal to or larger than 49.5 marks) over the total of available scores. For example, in the example result file above, the total of available scores is 16, and the total of pass scores is 13, therefore, the pass rate is $13/16 * 100\% = 81.25\%$.

The printed messages corresponding to two scenarios need to be exactly as below:

1. This is an example when a wrong number of arguments is passed in as command line arguments. Note there could be different scenarios for a wrong number of command line arguments, your program needs to handle all.

```
>python my_school.py
[Usage:] python my_school.py <result file> <course file> <student file>
```

2. This is when a result file, course file, and student file are passed in as command line arguments. Note that users can specify different file names, not necessarily the names *results.txt*, *courses.txt*, *students.txt*. The first file is always the result file, the second file is the course file, and the last file is the student file.

```
>python my_school.py results.txt courses.txt students.txt
```

RESULTS

| Student IDs | COSC045 | COSC123 | ISYS089 | ISYS273 | MATH346 |
|-------------|---------|---------|---------|---------|---------|
| S001 | | 57.1 | 41.8 | 72.4 | |
| S012 | | -- | 76.5 | | |
| S034 | -- | 85.7 | 89.0 | 74.5 | |
| S120 | 58.4 | 44.2 | 60.4 | | 46.5 |
| S236 | 92.8 | 95.4 | | 90.5 | 87.0 |
| S349 | 72.4 | | | | -- |

RESULTS SUMMARY

There are 6 students and 5 courses.
 The average pass rate is 81.25%.

----- CREDIT LEVEL (3 marks, you must only attempt this level after completing the PASS level) -----

At this level, your program can support more information about courses. Now, apart from the ID, each course will have a name, credit point, and offered semester(s). All IDs, names, credits, and offered semesters can be modified. There are two types of courses: *Core Course* and *Elective Course*. Core courses can have different credit points whilst all elective courses have the same credit point, which is 6 by default. All core courses are offered in all semesters whilst the elective courses can have different offered semesters. You should define appropriate variables, getters, and setters for courses.

Also, a course should have a method to compute some statistics, e.g., the number of students finished the course, the number of students that are doing the course (on-going), the average score, etc. (it is your choice to define the necessary statistics to compute). You can define extra methods if necessary.

At this level, the course file will have more information. The course file includes the course IDs, the course types, the course names, the credit points, and the offered semester (for elective courses). You

can assume there are no duplicate or redundant courses. You can assume all the courses available in this file are also available in the result file (in the PASS level), and vice versa.

```
courses - Notepad
File Edit Format View Help
COSC045, C, Java, 12
COSC123, E, Data, 6, Sem2
ISYS089, C, Computing, 24
ISYS273, C, System, 12
MATH346, E, Analytics, 6, Sem1
```

At this level, your program can now print course information tables on screen (in addition to the result information table) and save these tables into a file name *reports.txt*. For example, given the above result file (in the PASS level) and the course file (in this level), the course information tables should look like below. Note the content within the *reports.txt* file should also look the same. Also, users can specify different file names, not necessary the names *results.txt*, *courses.txt*, *students.txt*. The first file is always the result file, the second file is the course file, and the last file is the student file.

| COURSE INFORMATION | | | | | | | |
|---|-----------|------|--------|----------|---------|---------|----------|
| CourseID | Name | Type | Credit | Semester | Average | Nfinish | Nongoing |
| COSC045 | Java | C | 12 | All | 74.53 | 3 | 1 |
| ISYS089 | Computing | C | 24 | All | 66.92 | 4 | 0 |
| ISYS273 | System | C | 12 | All | 79.13 | 3 | 0 |
| CourseID | Name | Type | Credit | Semester | Average | Nfinish | Nongoing |
| COSC123 | Data | E | 6 | Sem2 | 70.60 | 4 | 1 |
| MATH346 | Analytics | E | 6 | Sem1 | 66.75 | 2 | 1 |
| COURSE SUMMARY | | | | | | | |
| The most difficult core course is ISYS089 with an average score of 66.92. | | | | | | | |
| The most difficult elective course is MATH346 with an average score of 66.75. | | | | | | | |

The Course information message displays two tables: one table for the core courses and one table for the elective courses. In each table, the CourseID column shows the courses' IDs, the Name column shows the course names, the Type column shows the types of the courses (C for core courses, and E for elective courses), the Semester column shows the semesters the courses offered (All for all semesters, Sem1 for Semester 1, Sem2 for Semester 2). The Average column displays the average score of all students who finished the course – note the average scores are shown with 2 digits after the decimal point. The Nfinish column displays the number of students who already finished the courses (already received marks), and the Nongoing column displays the number students who are still taking the courses (results not yet available).

Note apart from these two tables, your program should also display two messages indicating the most difficult core course and elective course. The most difficult course is the course with the lowest average mark. In each course type, if there are multiple courses with the lowest average marks, you can choose either to display one course or display all courses.

----- DI LEVEL (3 marks, you must only attempt this level after completing the CREDIT level) -----

At this level, your program can support two types of students: *Undergraduate Students* and *Postgraduate Students*. Undergraduate students always need to enroll as full-time students whilst

postgraduate students can enroll as either full-time or part-time students. The minimum number of enrolled courses for a full-time student is 4 whilst the minimum number of enrolled courses for a part-time student is 2. A student should have a method to compute some useful statistics for the student, e.g., the number of courses they finished, the average GPA, the number of ongoing courses, etc. (it is your choice to define the necessary statistics to compute). It should also have a method to check if a student satisfies the enrollment requirement (enrolling in equal to or more than the minimum number of courses required – note these can be courses that students already finished or are currently taking). You can define extra methods for the students if necessary.

At this level, the student file will have more information. The file includes the student IDs, the student names, the student types, and the mode for postgraduate students (full-time or part-time). You can assume there are no duplicate or redundant students. You can assume all students in the result file (in the PASS level) appeared in this file and vice versa. An example of the student file is as follows.

```
students - Notepad
File Edit Format View Help
S001, Jack, PG, FT
S012, Sarah, PG, PT
S034, Liam, PG, FT
S120, Charlie, UG
S236, Sophia, UG
S349, Alex, PG, PT
```

At this level, your program can now print student information tables on screen (in addition to the result information table and course information tables) and store these tables in the text file *reports.txt* (from the CREDIT level). Given the above result file (in the PASS level), the course file (in the CREDIT level), and the student file (in this level), the student information tables should look like below. Note the content within the *reports.txt* file should also look the same. Also, in the command line, users can specify different file names, not necessary the names *results.txt*, *courses.txt*, *students.txt*. The first file is always the result file, the second file is the course file, and the last file is the student file.

| STUDENT INFORMATION | | | | | | | |
|---|----------|------|------|----------|--------|---------|----------|
| StudentID | Name | Type | Mode | GPA(100) | GPA(4) | Nfinish | Nongoing |
| S001 | Jack (!) | PG | FT | 57.10 | 1.33 | 3 | 0 |
| S012 | Sarah | PG | PT | 76.50 | 3.00 | 1 | 1 |
| S034 | Liam | PG | FT | 83.07 | 3.67 | 3 | 1 |
| S349 | Alex | PG | PT | 72.40 | 3.00 | 1 | 1 |
| StudentID | Name | Type | Mode | GPA(100) | GPA(4) | Nfinish | Nongoing |
| S120 | Charlie | UG | FT | 52.38 | 0.75 | 4 | 0 |
| S236 | Sophia | UG | FT | 91.42 | 4.00 | 4 | 0 |
| STUDENT SUMMARY | | | | | | | |
| The best PG student is S034 with a GPA score of 3.67. | | | | | | | |
| The best UG student is S236 with a GPA score of 4.00. | | | | | | | |

There are two tables, one for Postgraduate students and one for Undergraduate students. The StudentID column shows the student IDs, the Name column shows the student names, the Type column shows the student types (PG for Postgraduate students and UG for Undergraduate students), and the Mode column shows the enroll mode of the students (FT for full-time and PT for part-time). The Nfinish column shows the number of courses the students already finished. The Nongoing column shows the number of courses the students are currently taking. The GPA(100) column displays the average GPAs of the students on a scale of 100. The GPA(4) column displays the average GPAs of the students on a

scale of 4: a result ≥ 79.5 receives 4 points, a result ≥ 69.5 and < 79.5 receives 3 points, a result ≥ 59.5 and < 69.5 receives 2 points, a result ≥ 49.5 and < 59.5 receives 1 point.

In the tables, the data in the *StudentID* and *Name* columns are aligned to the left whilst the data in all other columns are aligned to the right. Besides, in the tables, if a student doesn't satisfy the requirement of minimum courses enrolled then an exclamation mark (!) will be added next to the student's name. For example, in the table above, the student Jack is a full-time student but currently only enrolled in 3 courses, therefore, there is a (!) appears next to the student's name.

Apart from the student information tables, two sentences indicating the best Postgraduate student and Undergraduate student, along with their GPA(4) scores, are also shown. The best students are the students with the highest GPA (scale of 4). In each student type, if there are multiple students with the highest scores, you can choose to display one or all these students.

----- HD LEVEL (6 marks, you must only attempt this level after completing the DI level) -----

At this level, your program can handle some variations in the files using built-in/custom exceptions:

1. When any of the file cannot be found, then your program should print a message indicating the names of the files that cannot be found and then quit gracefully. You can assume users always type the file names in the right order in the command line, e.g., the result file first, the course file second, and the student file third.
2. The program will exit and indicate the corresponding error when one of these errors occurs:
 - a. The result file is empty;
 - b. Any of the scores in the result file is not a valid number;
 - c. The student ID doesn't start with the letter S;
 - d. The course ID doesn't start with either COSC, ISYS or MATH.

The program will have some additional requirements (some might be challenging):

1. In this level, the student information tables now have one additional column, WGPA(4), which is the weighted GPA(4) based on the course credits. The WGPA(4) of each student is computed by multiplying the scores the student obtained for the courses they took with the credits of those courses, and then dividing by the total credits. For example, the student S034 has a score of 85.7 (equivalent to 4) for COSC123 (with 6 credit points), a score of 89.0 (equivalent to 4) for ISYS089 (with 24 credit points), and a score of 74.5 (equivalent to 3) for ISYS273 (with 12 credit points). Therefore, the WGPA(4) of this student can be computed as $(4 \times 6 + 4 \times 24 + 3 \times 12) / (6 + 24 + 12) = 3.71$. An example of the student information tables with the WGPA(4) columns is shown in the below.

| STUDENT INFORMATION | | | | | | | | |
|---|----------|------|------|----------|--------|---------|---------|----------|
| StudentID | Name | Type | Mode | GPA(100) | GPA(4) | WGPA(4) | Nfinish | Nongoing |
| S001 | Jack (!) | PG | FT | 57.10 | 1.33 | 1.00 | 3 | 0 |
| S012 | Sarah | PG | PT | 76.50 | 3.00 | 3.00 | 1 | 1 |
| S034 | Liam | PG | FT | 83.07 | 3.67 | 3.71 | 3 | 1 |
| S349 | Alex | PG | PT | 72.40 | 3.00 | 3.00 | 1 | 1 |
| StudentID | Name | Type | Mode | GPA(100) | GPA(4) | WGPA(4) | Nfinish | Nongoing |
| S120 | Charlie | UG | FT | 52.38 | 0.75 | 1.25 | 4 | 0 |
| S236 | Sophia | UG | FT | 91.42 | 4.00 | 4.00 | 4 | 0 |
| STUDENT SUMMARY | | | | | | | | |
| The best PG student is S034 with a GPA score of 3.67. | | | | | | | | |
| The best UG student is S236 with a GPA score of 4.00. | | | | | | | | |

2. In this level, the course information tables are sorted (from high to low) based on the Average column (for each course type). Note that this is also reflected in the *reports.txt* file.
3. In this level, the student information tables are sorted (from high to low) based on the WGPA(4) column (for each student type). Note that this is also reflected in the *reports.txt* file.
4. The *reports.txt* is accumulated, which means when the program runs, it will not overwrite the previous report, but instead, it places the new report on top of the file (i.e., the newest report is always at the top of the *reports.txt* file). In addition, the date and time when the report was generated (in the format *dd/mm/yyyy hh:mm:ss*, e.g. *01/06/2023 09:45:00*) are also saved in the text file for each report.

B - Code Requirements:

You must demonstrate your ability to program in Python by yourself, i.e., you should not use any Python packages/libraries that can do most of the coding for you. **The only Python libraries allowed in this assignment is sys, os, and datetime.** If other packages/libraries are used, you will get a 0 mark or a heavy penalty.

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.

You should test/verify the program with different text files (not just run with our text files) to ensure your program satisfies all the required functionalities.

Your code needs to be formatted consistently. You must not include any unused/irrelevant code (even inside the comments). What you submitted must be considered the final product.

You should design your classes carefully. You may use a class diagram to assist with the design. **In the DI and HD levels, you are required to provide a detailed class diagram to show your class design.** An example of a class diagram is shown below on the next page – Figure 1 (note that this is a class diagram of a different programming assignment). In the diagram, the variables and methods of each class are shown. Note that if your code is at the PASS or CREDIT level, you do not need to submit any diagram; a diagram at these levels would NOT result in any mark.

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.**

Finally, note that in places where this specification may not tell you how exactly you should implement a certain feature, you need to use your judgment to choose and apply the most appropriate concepts from our course materials. You should follow answers given by your "client" (or "supervisor" or the teaching team) under Canvas/Discussions/Discussion on Final Coding Challenge.

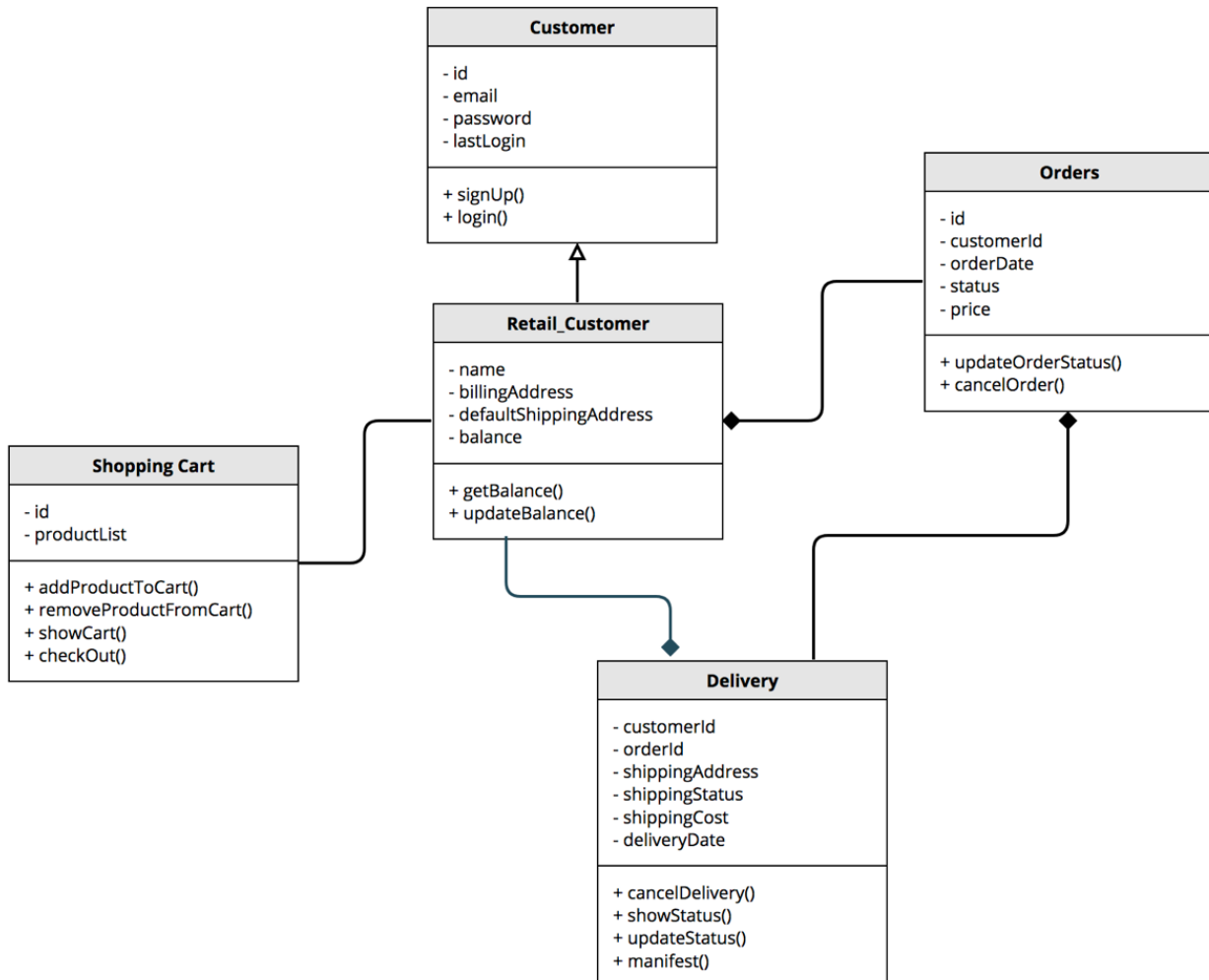


Figure 1. An example of a class diagram

C - Documentation Requirements:

You are required to write comments (documentation) as a part of your code. Writing documentation is a good habit in professional programming. It is particularly useful if the documentation is next to the code segment that it refers to. NOTE that you don't need to write an essay, i.e., you should keep the documentation succinct.

Your comments (documentation) should be in the same Python file. Please DO NOT write a separate file for comments (documentation).

At the beginning of your Python file, your code must contain the following information:

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

(we will ignore the DI and HD levels, so please make sure you fully finish one level before moving to the next one).

3. **Any problems of your code and requirements that you have not met.** For example, scenarios that might cause the program to crash or behave abnormally, the requirements your program does not satisfy. Note that you do not need to handle or address errors that are not covered in the course.

Besides, the comments in this final coding challenge should serve the following purposes:

- Explain your code in a precise but succinct manner. It should include a brief analysis of your approaches instead of simply translating the Python code to English. For example, you can comment on why you introduce a particular function/method, why you choose to use a while loop instead of other loops, why you choose a particular data type to store the data information. These comments can be placed before the code blocks (e.g., functions/methods, loops, if) and important variable declarations that the comments refer to.
- Document some analysis/reflection as a part of your code. Here, you need to write some paragraphs (could be placed at the end or at the beginning of your code) to explain in detail your design process, e.g., how you came up with the design of the program, how you started writing the code after the design process, the challenges you met during the code development.
- Document the references, i.e., any sources of information (e.g., websites, tools) you used other than the course contents directly under Canvas/Modules, you must give acknowledgement of the sources, explaining how you use the sources in this assignment. More detailed information regarding the references can be found in Section 5.

D - Rubric:

Overall:

| Level | Points |
|--|--------|
| PASS level | 15 |
| CREDIT level | 3 |
| DI level | 3 |
| HD level | 6 |
| Others (code quality, modularity, comments/analysis) | 3 |

More details of the rubric of this assessment can be found on Canvas ([here](#)). Students are required to look at the rubric to understand how the assessment will be graded.

4. Submission

As mentioned in the Code Requirements, **you must submit only one zip file with the name ProgFunFinal_<Your Student ID>.zip** via Canvas/Assignments/Final Coding Challenge. The zip file contains:

- The main Python code of your program, named **my_school.py**
- A diagram **in one of the formats: jpg, gif, png, pdf** (if you attempt the DI and HD levels)
- Other Python files written by you to be used by your main application.

It is your responsibility to correctly submit your file. Please verify that your submission is correctly submitted by downloading what you have submitted to see if the file includes the correct contents. The final zip file submitted is the one that will be marked. You do not need to submit the text files (result, course, student) we provide you – we will mark based on our text files. **NOTE, your code must be able to run under command-line as in the specification.**

Late Submission

All assignments will be marked as if submitted on time. Late submissions of assignments without special consideration or extension will be automatically penalised at a rate of 10% of the total marks available per day (or part of a day) late. For example, if an assignment is worth 30 marks and it is submitted 1 day late, a penalty of 10% or 3 marks will apply. This will be deducted from the assessed mark. Assignments will not be accepted if more than five days late unless special consideration or an extension of time has been approved.

Special Consideration

If you are applying for extensions for your assessment within five working days after the original assessment date or due date has passed, or if you are seeking extension for more than seven days, you will have to apply for Special Consideration, unless there are special instructions on your Equitable Learning Plan.

In most cases you can apply for special consideration online [here](#). For more information on special consideration, visit the university website on special consideration [here](#).

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 acknowledgement of the sources, explaining in detail how you use the sources in this assignment, and give references using the [IEEE referencing format](#).**

Where: You can add a code comment near the work (e.g., code block) to be referenced and include the detailed reference in the IEEE style.

How: To generate a valid IEEE style reference, please use the [citethisforme](#) tool if you're unfamiliar with this style.

6. Academic Integrity and Plagiarism (Standard Warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others whilst 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), summarized, paraphrased, discussed, or mentioned in your assessment through the appropriate referencing methods.
- Provided a reference list of the publication details so your readers can locate the source if necessary. This includes material taken from the 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 behaviors, including:

- Failure to properly document a source

- Copyright material from the internet of databases
- Collusion between students

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

7. Assessment Declaration:

When you submit work electronically, you agree to the assessment declaration:

<https://www.rmit.edu.au/students/student-essentials/assessment-and-results/how-to-submit-your-assessments>