

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_competition.py**, that can read data from files and perform some operations. You are required to implement the program following the below requirements. Note we will give you some files for you to run with your developed program, BUT you should change 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: **Competition**, **Student**, and **Challenge**. 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.

In this level, your program can read data from a file specified in the command line. The file stores the students' results in a competition. Your program should create a list of *Student* objects, a list of *Challenge* objects, and a 2D-list (or other data types) to store the students' results. 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 IDs of the challenges.

In the main class, your program should create a *Competition* object, call its *read_results(file_name)* method to load data from the file (*file_name*), and then call its *display_results()* method to display the results in the required format (as specified in the following).

Below is an example of the file that stores the students' results – see the next page (in the sequel, we will call this file as the result file). The data fields are separated by commas and new lines. The first row contains the challenges' IDs, and the first column contains the students' IDs. The first field in the data, the top left corner, is always empty. The file stores all students' results in all the challenges. These are the amount of time each student needs to solve each challenge. A result of *"-1"* means the student does not participate in the challenge, and a result of *"444"* means the student is working on the challenge (ongoing) and the result is not available yet. In this level, you can assume the real results

(not the -1 nor 444) are all positive floating-point numbers. You can assume there are no duplicate or redundant rows or columns. And you can assume the format of the data in the file is always correct.

```

results - Notepad
File Edit Format View Help
, C03, C04, C09, C12, C15
S001, 12.5, 6.8, 17.6, 444, -1
S052, 10.6, 7.0, 20.1, -1, 444
S125, 9.4, 6.2, 18.2, -1, -1
S098, 13.8, -1, 19.5, 25.3, 444
S246, 9.9, 5.9, 17.9, 20.1, -1
S012, 11.2, 444, 19.5, -1, 10.4

```

Your program should show usage if no result file is passed in as a command line argument. Otherwise, it can display a table showing the students' results, and a message showing the student with the fastest (smallest) average time. Note this average time is computed over all the finished results only. In the printed table, if a student does not participate in a challenge (i.e., when the result is -1), the data field at that location of challenge and student is empty. If a result is ongoing (i.e., when the result is 444), a double dash (--) is shown at that data field. The printed message needs to be exactly as below:

1. This is when no result file is passed in as a command line argument.

```

>python my_competition.py
[Usage:] python my_competition.py <result file>

```

2. This is when the result file is passed in as a command line argument. Note, users can specify a different file name, not necessary the name *results.txt*.

```

>python my_competition.py results.txt

COMPETITION DASHBOARD
+-----+-----+-----+-----+-----+
| Results | C03 | C04 | C09 | C12 | C15 |
+-----+-----+-----+-----+-----+
| S001 | 12.5 | 6.8 | 17.6 | -- |  |
| S052 | 10.6 | 7.0 | 20.1 |  | -- |
| S125 | 9.4 | 6.2 | 18.2 |  |  |
| S098 | 13.8 |  | 19.5 | 25.3 | -- |
| S246 | 9.9 | 5.9 | 17.9 | 20.1 |  |
| S012 | 11.2 | -- | 19.5 |  | 10.4 |
+-----+-----+-----+-----+-----+

There are 6 students and 5 challenges.
The top student is S125 with an average time of 11.27 minutes.

```

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

In this level, your program can support more information of challenges. Now, apart from the ID, each challenge will have a name and a weight; all IDs, names and weights can be modified. There are two types of challenges: one is *Mandatory Challenge*, and one is *Special Challenge*. All the mandatory challenges have a same weight, by default, it is 1.0. Each special challenge will have its own weight and the weights of the special challenges are required to be larger than 1.0. You should define appropriate private/hidden variables, getters, and setters for challenges.

Also, a challenge should have a method to compute some statistics: number of students finished the challenge, number of on-going students, average time of each challenge, etc. (it is your choice to compute the necessary statistics). You can define extra methods for the challenges if necessary.

Your program now can read one more file specified in the command line. This file stores the information of the challenges (see an example below, in the sequel, we will call this file as the challenge file). The file includes the challenge IDs, the challenge names, the types of the challenges ("M" means mandatory and "S" means special), and the weight of each challenge. You can assume there are no duplicate or redundant challenges. You can assume all challenges available in the competition appeared in this file and in the previous result file (in the PASS level).

```

challenges - Notepad
File Edit Format View Help
C03, M, Search, 1
C15, S, Select, 1.2
C04, M, Sort, 1
C09, S, Compression, 1.5
C12, S, Vote, 2
  
```

Your program now can print the challenge summary on screen and save that summary into a file named *competition_report.txt*. For example, given the above *challenges.txt* file and the *results.txt* file as in the PASS level, the displayed message should look like below (note the content within the *competition_report.txt* should also look the same). Also, users can specify different file names, not necessary the names *results.txt* and *challenges.txt*. The first file is always the result file, and the second file is the challenge information file.

```

>python my_competition.py results.txt challenges.txt
  
```

COMPETITION DASHBOARD						
Results	C03	C04	C09	C12	C15	
S001	12.5	6.8	17.6	--		
S052	10.6	7.0	20.1		--	
S125	9.4	6.2	18.2			
S098	13.8		19.5	25.3	--	
S246	9.9	5.9	17.9	20.1		
S012	11.2	--	19.5			10.4

```

There are 6 students and 5 challenges.
The top student is S125 with an average time of 11.27 minutes.

CHALLENGE INFORMATION
  
```

Challenge	Name	Weight	Nfinish	Nongoing	AverageTime
C03	Search(M)	1.0	6	0	11.23
C04	Sort(M)	1.0	4	1	6.47
C09	Compression(S)	1.5	6	0	18.8
C12	Vote(S)	2.0	2	1	22.7
C15	Select(S)	1.2	1	2	10.4

```

The most difficult challenge is Vote (C12) with an average time of 22.70 minutes.
Report competition_report.txt generated!
  
```

In the above message, the *Weight* column displays the weights with 1 digit after the decimal point. The *Nfinish* column displays the number of students who already finished the challenge (need to be integers). The *Nongoing* column displays the number of students who started working on the challenge but haven't finished yet (need to be integers). The *AverageTime* column displays the average time (2 digits after the decimal point) the challenge is solved (based on finished students only). In the *Name* column, apart from the names of the challenges, your program should also display the challenge types ("M" for mandatory challenges and "S" for special challenges).

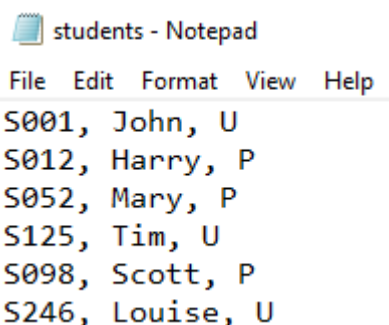
Note, apart from a challenge information table, your program should also display a message indicating the most difficult challenge. The most difficult challenge is the one with the highest average time. If there are multiple challenges with the highest average time, you can choose to either display one challenge or display multiple challenges.

Finally, note that the printed message includes the printed message from the PASS level (and the *competition_report.txt* file should also include the table and the message from the PASS level).

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

In this level, your program can support two types of students: *Undergraduate Students* and *Postgraduate Students*. All students need to participate in all the mandatory challenges. An undergraduate student needs to participate in at least 1 special challenge whilst a postgraduate student needs to participate in at least 2 special challenges. The classes defined for students should have methods to check whether a student meets these requirements. Also, a student should have a method to compute some useful statistics for the student, e.g., number of finished challenges, number of on-going challenges, average time, etc. You can define extra methods for the students if necessary.

Your program now can read the student information from a file specified in the command line. This file stores information about the students (in the sequel, we will call this file as the student file). The file includes the students' IDs, the students' names, and the types of the students ("U" for undergraduate students and "P" for postgraduate students). You can assume there are no duplicate or redundant students. You can assume all students in the competition appeared in this file and in the previous result file (in the PASS level). An example of the student file is as follows.



```
students - Notepad
File Edit Format View Help
S001, John, U
S012, Harry, P
S052, Mary, P
S125, Tim, U
S098, Scott, P
S246, Louise, U
```

Your program now can print a summary of students on screen and store that summary in the text file *competition_report.txt* (from the CREDIT level). Given the *students.txt* file above, the *challenges.txt* file in the CREDIT level, and the *results.txt* file in the PASS level, the displayed message should look like below (and so does the content in the *competition_report.txt* file) – see the next page. Note in the command line, users can specify different file names, not necessary the names *results.txt*, *challenges.txt* and *students.txt*. You can assume users always type the file names in the right order in the command line, e.g., the result file first, the challenge file second, and the student file third.

The *Type* column stores the types of students ("U" for undergraduate students and "P" for postgraduate students). The *Nfinish* column stores the number of challenges each student finished, the *Nongoing* column stores the number of challenges each student started working but has not finished yet. The *AverageTime* column stores the average time (with two digits after the decimal point) the students spent working on the challenges (computed based on finished challenges only). If a student hasn't had any result yet, then the value in the *AverageTime* column is a double dash (--).

In addition, in the *Name* column, if a student doesn't satisfy the requirements on the number of mandatory challenges and minimum special challenges, an exclamation mark (!) is added at the beginning of their names (e.g., "!Mary", "!Scott" and "!Harry" in the screenshot below).

Finally, a message indicating the student with the fastest average time will also be displayed. Note, the content in the *competition_report.txt* needs to be the same as the printed message.

```
python my_competition.py results.txt challenges.txt students.txt
```

COMPETITION DASHBOARD

Results	C03	C04	C09	C12	C15
S001	12.5	6.8	17.6	--	--
S052	10.6	7.0	20.1	--	--
S125	9.4	6.2	18.2	25.3	--
S098	13.8		19.5	20.1	--
S246	9.9	5.9	17.9		
S012	11.2	--	19.5		10.4

There are 6 students and 5 challenges.

The top student is S125 with the average time of 11.27 minutes.

CHALLENGE INFORMATION

Challenge	Name	Weight	Nfinish	Nongoing	AverageTime
C03	Search(M)	1.0	6	0	11.23
C04	Sort(M)	1.0	4	1	6.47
C09	Compression(S)	1.5	6	0	18.8
C12	Vote(S)	2.0	2	1	22.7
C15	Select(S)	1.2	1	2	10.4

The most difficult challenge is Vote (C12) with an average time of 22.70 minutes.

Report competition_report.txt generated!

STUDENT INFORMATION

StudentID	Name	Type	Nfinish	Nongoing	AverageTime
S001	John	U	3	1	12.3
S052	!Mary	P	3	1	12.57
S125	Tim	U	3	0	11.27
S098	!Scott	P	3	1	19.53
S246	Louise	U	4	0	13.45
S012	!Harry	P	3	1	13.7

The student with the fastest average time is S125 (Tim) with an average time of 11.27 minutes.

Report competition_report.txt generated!

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

In this level, your program can handle some variations in the files using exceptions:

1. When the result file is empty, your program will exit and display a message indicating no results are available for the competition.
2. Results in the result file might be characters (e.g., "NA", "x"). In these cases, they will be treated as same as -1, except "TBA" or "tba", which means the challenge is ongoing (same as 444).
3. When there are any formatting issues in any line of the files (e.g., other delimiters are used instead of commas, IDs are not in the right format, more columns compared to the required number of columns), the program will exit and display a message indicating there is a problem with the corresponding file.
4. When the files are missing or cannot be found, then your program should print a message indicating the names of the files are missing 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 challenge file second, and the student file third.

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

1. The *competition_report.txt* is accumulated, which means when the program is run, 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 *competition_report.txt* file). In addition, the date and time when the report was generated (in the format *dd/mm/yyyy hh:mm:ss*, e.g. *01/03/2021 09:45:00*) are also saved in the text file for each report.
2. The student information table (produced from the DI level) now has two new columns: *Score* and *Wscore*.
 - a. The *Score* column is computed based on the scores the students obtained if they come first, second, third or last in a challenge. A student obtains 3 pts if they come first, 2 pts if they come second, 1 pts if they come third, and -1 pts if they come last in a challenge. Students come after the third place and before the last place will not have any points. The total score of a student is the total score of all the challenges they finished. A challenge only has scores when it is finished, i.e., there are no on-going students (note a challenge can still finish when there exist students who do not participate on it). If a challenge has less than 4 students participating, then the students are still awarded the points when they come first, second, or third even in these cases, a student comes third/second/first might also come last. For example, if there are only 3 students participating in a challenge, they are still awarded 3 pts, 2 pts, and 1 pts respectively.
 - b. The *Wscore* column is computed based on the scores the student obtained in each challenge and the weight of that challenge. For example, if the student comes first in a challenge with the weight being 1.2, then they will obtain a weighted score of $3 \times 1.2 = 3.6$ pts for that challenge. The total weighted score is the total weighted score of all the challenges the student finished. The weighted scores in the *Wscore* column have at most 1 digit after the decimal point.

See the below screenshot an example of how the new student information table with the *Score* and *Wscore* columns look like.

3. Messages indicating students with the highest scores and weighted scores are also be displayed (see the below screenshot).

STUDENT INFORMATION							
StudentID	Name	Type	Nfinish	Nongoing	AverageTime	Score	Wscore
S001	John	U	3	1	12.3	3	4.5
S052	!Mary	P	3	1	12.57	0	-0.5
S125	Tim	U	3	0	11.27	4	4.5
S098	!Scott	P	3	1	19.53	-1	-1.0
S246	Louise	U	4	0	13.45	4	5.0
S012	!Harry	P	3	1	13.7	0	0

The student with the fastest average time is S125 (Tim) with an average time of 11.27 minutes.
 The student with the highest score is S125 (Tim) with a score of 4.
 The student with the highest weighted score is S246 (Louise) with a weighted score of 5.0.
 Report competition_report.txt generated/updated!

- The challenge information table (produced from the CREDIT level) is sorted (from low to high) based on average time.
- The student information table is sorted (from high to low) based on the weighted scores.

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 are sys and datetime.** If other packages/libraries are used, you will get 0 mark or very 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 satisfy all the required functionalities.

In this challenge, your program has no interaction with users during execution. It simply runs the code, read the files, and display the desired message and/or generate the report text file.

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 as the final product.

You should design your classes carefully. You may use a class diagram to assist 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 in the next page (note this is a class diagram of a different programming assignment). In the diagram, 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 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.

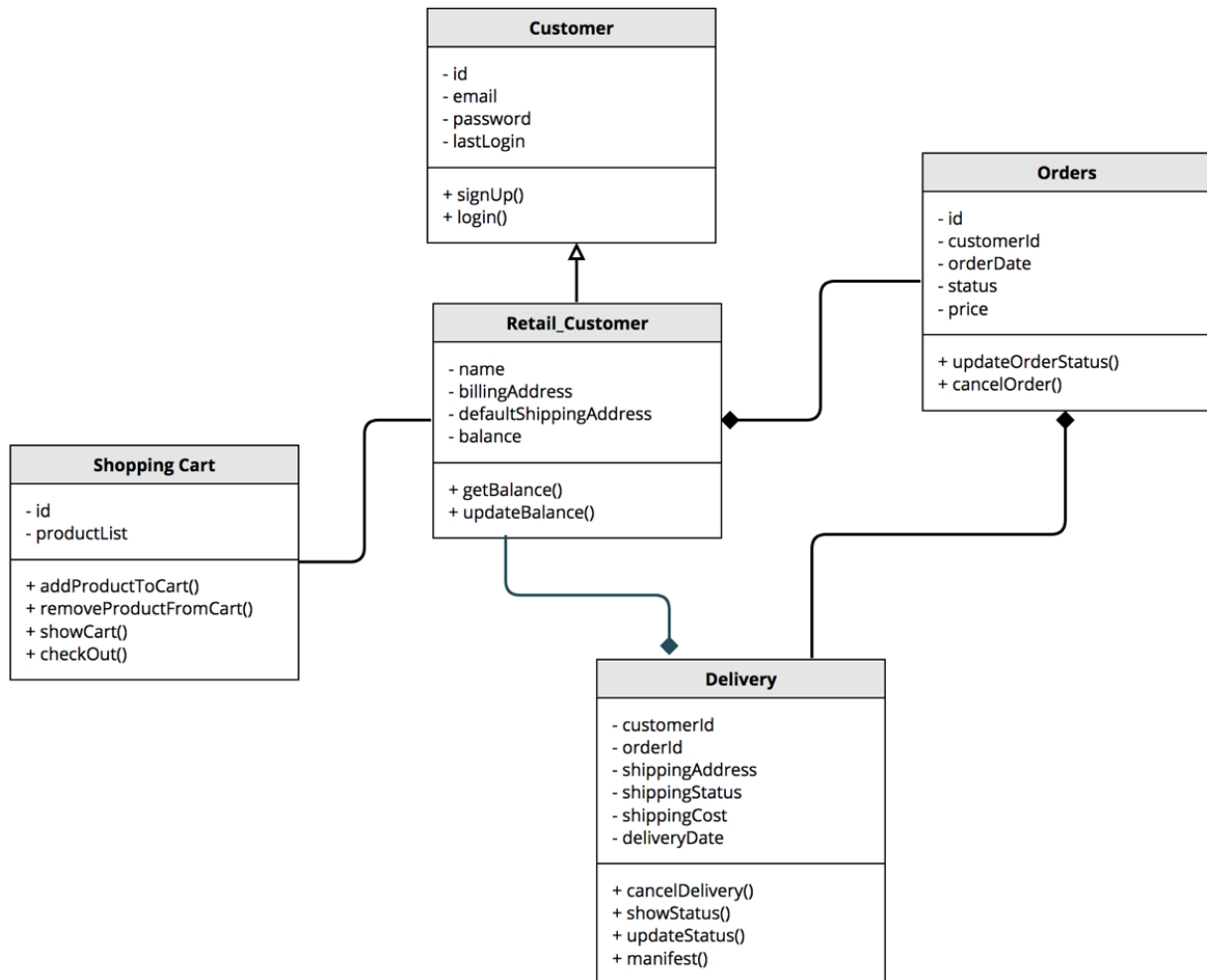


Figure 1. An example of a class diagram

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, before the code blocks (e.g., functions/methods, loops, if, etc.) and important variable declarations that the comments refer to. 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. Note, 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.
- Document any problems of your code and requirements that you have not met, e.g., the situations that might cause the program to crash or behave abnormally, the requirements your program do not satisfy. Note that you do not need to handle or address errors that are not covered in the course material yet.
- Document some analysis/discussion/reflection as a part of your code, e.g., how your code could be improved if you have more time, which part you find most challenging, etc.

D - Rubric:

Overall:

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

More details of the rubric of this assignment can be found on Canvas. Students are required to look at the rubric to understand how the assignment 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_competition.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, challenge, student) we provide you. **NOTE, your code must be able to run under command-line. Specifically, it should be able to run under the following command lines:**

1. `python my_competition.py`
2. `python my_competition.py results.txt`
3. `python my_competition.py results.txt challenges.txt`
4. `python my_competition.py results.txt challenges.txt students.txt`

If you attempt the PASS level, your program should be able to run the first and second command lines. If you attempt until the CREDIT level, your program should be able to run the first, second, and third command lines. If you attempt until the DI or HD level, your program should be able to run all the four command lines. **NOTE, the filenames specified in the command lines can be different, not**

necessary results.txt, challenges.txt, students.txt. You can assume users always specify the correct order of file names, i.e., the result file first, the challenge file second, and the student file third.

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, 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 or 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>