Programming Fundamentals (COSC2531)
Assignment 1

Assessment Type	**Individual assignment** (no group work). Submit online via Canvas/Assignments/Assignment 1. Marks are awarded per rubric (please see the rubric on Canvas). Clarifications/updates may be made via announcements. Questions can be raised via the lectorial, practical sessions or Canvas discussion forum. Note the Canvas discussion forum is preferable.
Due Date	**End of Week 6** (exact time is shown in Canvas/Assignments/Assignment 1) Deadline will not be advanced, but they may be extended. Please check Canvas/Assignments/Assignment 1 for the most up to date information regarding the assignment. As this is a major assignment, a university standard late penalty of 10% per each day (e.g., 2 marks/day) applies for up to 5 days late, unless special consideration has been granted.
Weighting	**20 marks out of 100**

1. Overview

The objective of this assignment is to develop your programming and problem-solving skills in a step-by-step manner. The different stages of this assignment are designed to gradually introduce different basic programming concepts.

You should develop this assignment in an iterative fashion (as opposed to completing it in one sitting). You can and should get started now (when this assignment specification is posted on Canvas) as there are concepts from previous lessons that you can employ to do this assignment. If there are questions, you can ask via the lectorial, practical sessions or the Canvas discussion forum (Canvas/Discussions/Discussion on Assignment 1). Note that the **Canvas discussion forum is preferable** as it allows other students to see your questions as well. Also, you should ask questions 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 over the first several weeks of the course;
iii. Write and debug Python code independently;

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.
3. Develop maintainable and reusable solutions.

Specifically, upon the completion of this assignment, you will be able to:

- Demonstrate knowledge of basic concepts, syntax, and control structures in programming
- Devise solutions for simple computing problems under specific requirements
- Encode the devised solutions into computer programs and test the programs on a computer
- Demonstrate understanding of standard coding conventions and ethical considerations in programming

4. Assessment Details

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

Problem Overview: In this assignment, you are developing a movie ticketing system. The box office cashiers or managers from a cinema are the ones that use this system to process customers' ticket purchases. You are required to implement the program following the below requirements.

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 3 parts; please ensure you only attempt one part after completing the previous part.

---------- **PART 1 (8 marks)** ----------

In this part, your program can perform some simple interactions with users (i.e., the box office cashiers or the managers):

1. Display a message asking the user to enter the customer's name.
2. Display a message asking the user to enter the movie's name. In this part, you can assume the movie to be entered is always a valid movie.
3. Display a message asking the user to enter the ticket type the customer chooses. In the system, there are 5 ticket types for customers to choose from: *adult*, *child*, *senior*, *student*, *concession*. In this part, you can assume the ticket type to be entered is always a valid ticket type.

4. Display a message asking the quantity of the tickets ordered by the customer. In this part, you can assume the quantity to be entered is always a positive integer, e.g., 1, 2, 3 …
5. Calculate the total cost for the customer including the discount fee and booking fee (see No. 6, 7 and 8 below). The total cost is equal to the total ticket cost – discount fee + the booking fee.
6. The unit price for the ticket types *adult*, *child*, *senior*, *student*, *concession* are *25.0*, *19.5*, *17.0*, *20.5*, and *20.5*, respectively. All the movies have the same ticket price structure. The total ticket cost for a particular ticket type is equal to the price of that ticket type multiplying with the ticket quantity. For example, if the quantity ordered by the customer is 2, the ticket type is *adult*, then the total ticket cost is 25 x 2 = 50$.
7. For customers that are in the rewards program, 20% discount will apply. No discount for customers that are not in the rewards program. The discount fee is computed based on the total ticket cost. For example, if the total ticket cost is 50$, then the discount fee is 20% x 50 = 10$.
8. For all customers, for all ticket types, the booking fee is 2 x the ticket quantity. For example, if the quantity ordered by the customer is 2, then the booking fee is 2 x 2 = 4$.
9. The total cost will be displayed as a formatted message to the user, e.g.,

---

Receipt of <customer_name>

Movie:	*<movie_name>*
Ticket type:	*<ticket_type>*
Ticket unit price:	*<ticket_unit_price>*
Ticket quantity:	*<ticket_quantity>*

Discount:	*<discount_fee>*
Booking fee:	*<booking_fee>*
Total cost:	*<total_cost>*

10. In the program, you should have some lists (or dictionaries or other data types) to store the names of all customers, the names of the customers that are in the rewards program, the available movies, the available ticket types, the prices of those ticket types. You can assume the customer names, the movie names, and the ticket types are all unique and case sensitive.
11. When a new customer orders a ticket(s), your program will add the customer's name to the customer list. Also, when any customer orders a ticket, your program will check if they are in the rewards program; if they are not in the rewards program, your program will display a message asking if they want to register to be in the rewards program. If yes, then your program will add the customer's name to the rewards program. The discount fee is applied immediately after the customer is added to the rewards program. In this part, you can assume the answer entered is always either *y* or *n* (meaning yes or no, respectively).
12. Your program needs to be initialized with the following customers: *Mary* (a customer in the rewards program), and *James* (a customer not in the rewards program). Your program also needs to be initialized with the following movies: *Avatar*, *Titanic*, and *StarWar*.
13. Note: in the requirements No. 10 & 11, we use the term 'list' when describing the customer list, the customer in the rewards program list, etc. But you can use other data types to store this information such as dictionaries and other data types. Make sure you think and analyse the requirements in detail so that you can choose the most appropriate/suitable data types.

---------- PART 2 (5 marks, please do not attempt this part before completing PART 1) ----------

In this part, your program can: (a) perform some additional requirements compared to PART 1, and (b) be operated using a **menu**.

First, compared to the requirements in PART 1, now your program will have the following features:

i. Each movie is associated with a number of seats. After each purchase, the number of available seats for the purchased movie needs to be updated accordingly. Your program needs to be initialized with 50 tickets per each movie.
ii. Handle invalid inputs from users:
    a. Display an error message if the movie entered by the user is not a valid movie. When this error occurs, the user will be given another chance, until a valid movie is entered.
    b. Display an error message if the ticket type entered by the user is not a valid ticket type. When this error occurs, the user will be given another chance, until a valid ticket type is entered.
    c. Display an error message if the ticket quantity is 0, negative, not an integer, or exceed the number of available seats. When this error occurs, the user will be given another chance, until a valid quantity is entered.
    d. Display an error message if the answer by the user is not *y* or *n* when asking if the customer wants to join the rewards program. When this error occurs, the user will be given another chance, until a valid answer (i.e., *y*, *n*) is entered.

Second, your program will be operated using a **menu**. A menu-driven program is a computer program in which options are offered to the users via the menu. Your program will have the following options: Purchase a ticket, add movies, display existing customers information, display existing movies information, exit the program (please see Section 5 in this document regarding an example of how the menu program might look like). Below are the specifications of the options:

1. *Purchase a ticket*: this option includes all the requirements from 1 to 13 in PART 1 and the requirements i to ii in the first part of PART 2.
2. *Add movies*: this option displays a message asking if users want to add a list of movies. The movies must be entered as a list that separates by commas with the following format: *movie_1, movie_2, movie_3, …* For example, users can enter the input: *Titanic, Avenger, Frozen* to add the movies Titanic, Avenger, and Frozen. If a movie is new, then the movie and its initial numbers of seats (50) will be added to the data collection of the program. If a movie is an existing movie, your program will print a message saying this is an existing movie, so it does not do anything. You can assume the movie names are always unique, and each movie name is a single word with no space nor commas. You can assume users always enter the correct formats (i.e., using commas to separate the movie names), but note that users can enter multiple spaces before/after the commas.
3. *Display existing customers information*: this option displays on screen all existing customers and their rewards program information (whether they're in or not in the rewards program). The messages are flexible (your choice), but they should show all the information required.
4. *Display existing movies information*: this option displays all the movies with their numbers of available seats. The messages are flexible (your choice), but they should show all the information required.
5. *Exit the program*: this option allows users to exit the program.

Note that in your program, when a task (option) is accomplished, the menu will appear again for the next task.

---------- PART 3 (4 marks, please do not attempt this part before completing PART 2) ----------

In this part, your menu program is equipped with some advanced features. Note, some features maybe very challenging.

1. In this part, in the *"Purchase a ticket"* option, your program will allow customers to purchase multiple ticket types in one order. To do so, your program will ask users to input a list of ticket types, and then a list of the ticket quantities that correspond to the ticket types in the previous list. The items in these two lists are separated by commas. For example, users can enter a list of ticket types: *adult*, *child*, *senior* and then a list of ticket quantities: *2*, *1*, *1*, meaning that users want to purchase 2 adult tickets, 1 child ticket, and 1 senior ticket. You can assume the number of ticket types entered is the same as the number of ticket quantities entered. Note that, in this option, if the user only enters one ticket type and one ticket quantity as in the previous 2 parts, then your program will still work as normal (as in PART 2). Besides, your program needs to handle invalid inputs. Specifically, if the lists contain any invalid ticket type or ticket quantity (0, negative, not integers, or total quantities in the list exceeds the number of available seats), then your program will display a message saying the corresponding list is not valid, and the user will be given another chance (re-enter the lists), until all valid ticket types and ticket quantities are entered. The formatted message for the receipt is as follows.

```
--------------------------------------------------------------------------
Receipt of <customer_name>
--------------------------------------------------------------------------
Movie:                                                    <movie_name>
Ticket type:                                              <ticket_type>
Ticket unit price:                                        <ticket_unit_price>
Ticket quantity:                                          <ticket_quantity>
                                ..............
Ticket type:                                              <ticket_type>
Ticket unit price:                                        <ticket_unit_price>
Ticket quantity:                                          <ticket_quantity>
--------------------------------------------------------------------------
Discount:                                                 <discount_fee>
Booking fee:                                              <booking_fee>
Total cost:                                               <total_cost>
```

2. The menu also has an option *"Display the most popular movie*" to display the movie with the maximum money (total cost) purchased by users to date and the amount of money it was purchased. If there are multiple movies with the same maximum money purchased, you can display only one movie or all movies, it's your choice.
3. The menu now has an option *"Display all movie record"*. The option will display a table displaying all the previous purchases of all movies, including the ticket types and the quantity for each ticket type they purchased and the total cost (including all the fees). An example table is as follows.

	adult	*child*	*senior*	*student*	*concession*	*Revenue*
Titanic	*12*	*3*	*8*	*10*	*4*	*720.5*
Avatar	*15*	*7*	*5*	*14*	*6*	*924.6*
StarWar	*14*	*8*	*6*	*11*	*5*	*860.8*

B - Code Requirements:

The program **must be entirely in one Python file named ProgFunA1_<Your Student ID>.py**. For example, if your student ID is s1234567, then the Python file must be named as ProgFunA1_s1234567.py. Other names will not be accepted.

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 use appropriate data types and handle user inputs properly. You must not have any redundant parts in your code.

You must demonstrate your ability to program in Python by yourself, i.e., you should not attempt to use external special Python packages/libraries/classes that can do most of the coding for you. **The only Python library allowed in this assignment is the sys module.**

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 Assignment 1.

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 part you have attempted.**
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 do not satisfy. Note, you do no need to handle errors that are not covered in the course.

Besides, the comments (documentation) in this assignment 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 7.

D - Rubric:

Overall:

Part	Points
Part 1	8
Part 2	5
Part 3	4
Others (code quality, modularity, comments/reflection)	3

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

5. Example Program

We demonstrate a **sample program** that satisfies the requirements specified in Section 4. Note that this is just an example, so it is okay if your program looks slightly different, but you need to make sure that **your program satisfies the requirements listed in Section 4**.

5.1. PART 1

As an example, this is how the output screen of our sample program looks like for PART 1. In this example, the customer has the name *Huong*, purchase *2* tickets of type *adult* for the movie *Avatar*. The customer is not in the rewards program, and then choose *y* (yes) for the rewards program registration option. Note that in this test, we use the values *Huong*, *Avatar*, *adult*, *2*, *y* as examples only. You should test your program with different test cases, e.g., when users choose *n* (no) for the rewards program registration options, to make sure your program satisfy the requirements in Section 4.

```
Enter the name of the customer [e.g. Huong]:
Huong

Enter the name of the movie [enter a valid name only, e.g. Avatar]:
Avatar

Enter the ticket type [enter a valid type only, e.g. adult, child, senior]:
adult

Enter the ticket quantity [enter a positive integer only, e.g. 1, 2, 3]:
2

The customer is not in the rewards program. Does the customer want to join the rewards program [enter y or n]?
y
Successfully add the customer to the rewards program.

---------------------------------------
Receipt of Huong
---------------------------------------
Movie:                          Avatar
Ticket type:                     adult
Ticket unit price:                25.0
Ticket quantity:                     2
---------------------------------------
Discount:                         10.0
Booking fee:                         4
Total cost:                       44.0
```

5.2. PART 2

As an example, this is how the output screen of our sample program looks like for a menu with all the options described in PART 2.

```
Welcome to the RMIT movie ticketing system!

############################################################
You can choose from the following options:
1: Purchase a ticket
2: Add new movies
3: Display existing customers information
4: Display existing movies information
0: Exit the program
############################################################

Choose one option:
```

When the user (the box office cashier or manager from a cinema) enters an option, the corresponding functionality will appear. For example, if the user chooses option 1, which is to purchase a ticket, then the output screen of our sample program is as follows.

```
Choose one option: 1

Enter the name of the customer [e.g. Huong]:
Huong

Enter the name of the movie [e.g. Avatar]:
test
The movie is not valid. Please enter a valid movie.

Enter the name of the movie [e.g. Avatar]:
Titanic

Enter the ticket type [e.g. adult, child, senior]:
test
The ticket type is not valid. Please enter a valid ticket type.

Enter the ticket type [e.g. adult, child, senior]:
adult

Enter the ticket quantity [e.g. 1, 2, 3]:
100
Ticket quantity must be less than the number of available seats. Please enter a smaller ticket quantity.

Enter the ticket quantity [e.g. 1, 2, 3]:
2

The customer is not in the rewards program. Does the customer want to join the rewards program [enter y or n]?
test
Please only enter y or n

The customer is not in the rewards program. Does the customer want to join the rewards program [enter y or n]?
y
Successfully add the customer to the rewards program.

---------------------------------------
Receipt of Huong
---------------------------------------
Movie:                          Titanic
Ticket type:                      adult
Ticket unit price:                 25.0
Ticket quantity:                      2
---------------------------------------
Discount:                          10.0
Booking fee:                          4
Total cost:                        44.0
```

Other requirements in PART 2 (add movies, display existing customers information, display existing movies information) can also be displayed in a similar manner.

5.3. PART 3

As an example, this is how the output screen of our sample program looks like for a menu with all the options described in PART 3.

```
Welcome to the RMIT movie ticketing system!

############################################################
You can choose from the following options:
1: Purchase a ticket
2: Add new movies
3: Display existing customers information
4: Display existing movies information
5: Display the most popular movie
6: Display all movie record
0: Exit the program
############################################################

Choose one option:
```

This is an example showing the output screen of our sample program when we select option 1, and order multiple ticket types in one purchase.

```
Choose one option: 1

Enter the name of the customer [e.g. Huong]:
Huong

Enter the name of the movie [e.g. Avatar]:
Avatar

Enter a list of ticket types [e.g. adult, child, senior]:
adult  , child, senior

Enter a list of ticket quantities [e.g. 2, 1, 1]:
2, 1, 1

The customer is not in the rewards program. Does the customer want to join the rewards program [enter y or n]?
y
Successfully add the customer to the rewards program.

---------------------------------------
Receipt of Huong
---------------------------------------
Movie:                          Avatar
Ticket type:                     adult
Ticket unit price:                25.0
Ticket quantity:                     2
Ticket type:                     child
Ticket unit price:                19.5
Ticket quantity:                     1
Ticket type:                    senior
Ticket unit price:                17.0
Ticket quantity:                     1
---------------------------------------
Discount:                         17.3
Booking fee:                         8
Total cost:                       77.2
```

And this is an example showing the output screen of our sample program when we select option 6, which is to display all movie records. Other options can also be displayed in a similar manner.

```
############################################################
You can choose from the following options:
1: Purchase a ticket
2: Add new movies
3: Display existing customers information
4: Display existing movies information
5: Display the most popular movie
6: Display all movie record
0: Exit the program
############################################################

Choose one option: 6

             adult     child     senior    student   concession    Revenue
Avatar         15         8          0         12          0         691.6
Titanic        16         0         10         14          0         765.6
StarWar        24        18          0          0          8         992.0
```

6. Submission

As mentioned in the Code Requirements, **you must submit only one file named ProgFunA1_<Your Student ID>.py** via Canvas/Assignments/Assignment 1. 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 .py file submitted is the one that will be marked.

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 20 marks and it is submitted 1 day late, a penalty of 10% or 2 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.

7. Referencing Guidelines

What: This is an individual assignment, and all submitted contents must be your own. If you have used any sources of information (e.g., websites, tools) other than the course 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.

8. 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).

9. 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