

# Programming Fundamentals (COSC2531)

## Assignment 2

<b>Assessment Type</b>	<b>Individual assignment</b> (no group work). Submit online via Canvas/Assignments/Assignment 2.  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.
<b>Due Date</b>	<b>End of Week 12</b> (exact time is shown in Canvas/Assignments/Assignment 2) Deadline will not be advanced nor extended. Please check Canvas/Assignments/Assignment 2 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., 3 marks/day) applies for up to 5 days late, unless special consideration has been granted.
<b>Weighting</b>	<b>30 marks out of 100</b>

### 1. Overview

The main objective of this assignment is to familiarize you with object-oriented design and programming. Object-oriented programming helps to solve complex problems by coming up with a number of domain classes and associations. However, identifying meaningful classes and interactions requires a fair amount of design experience. Such experience cannot be gained by classroom-based teaching alone but must be gained through project experience. This assignment is designed to introduce different concepts such as inheritance, method overriding, and polymorphism.

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 Assessment 2). 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:

- 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 duration 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 as in Assignment 1 using the object-oriented programming (OOP) paradigm. Same as in Assignment 1, the box office cashiers or managers from a cinema are the ones that use this system to process customers' bookings. You are required to implement the program following the below requirements. Note the requirements in this assignment are sometimes slightly different and more complex compared to those in Assignment 1. Also, we will provide you with some sample .txt files (download on Canvas), but you should change the data in these files to test your program as during the marking, we will use different text files to test 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 (12 marks)** -----

At this level, your program will have some basic classes with specifications as below. You may need to define methods wherever appropriate to support these classes. At the end of the PASS level, your program should be able to run with a menu described in the class Operations.

## Customers:

### 1. Class Customer

Each customer has a unique **ID**, unique **name** (a name will not include any digit nor white space). You are required to write the class named **Customer** to support the following:

- i. Attributes **ID** and **name**
- ii. Constructor takes the values of **ID**, **name** as arguments
- iii. Appropriate getter methods for the attributes of this class
- iv. A method **get\_discount(self, cost)** which takes the ticket cost and returns 0.
- v. A method **get\_booking\_fee(self, ticket\_quantity)** which takes the ticket quantity and returns the booking fee. The booking fee is fixed at 2\$ per each ticket.
- vi. A method **display\_info(self)** that prints the values of the **Customer** attributes.

### 2. Class RewardFlatCustomer

RewardFlat customers are customers that are in the rewards program and have a flat discount rate when purchasing tickets. All RewardFlat customers will have the same discount rate. The class RewardFlatCustomer should have the following components:

- i. An attribute for the **discount rate**, by default it is 20%.
- ii. Constructor takes the appropriate parameters/arguments (be careful)
- iii. Appropriate getter methods for the attributes of this class
- iv. A method **get\_discount(self, cost)** which takes the ticket cost and returns the discount. For example, this method returns 20 when the discount rate is 20% and the ticket cost is 100\$.
- v. A method **display\_info(self)** that prints the values of the **RewardFlatCustomer** attributes.
- vi. A method **set\_discount\_rate** to adjust the discount rate. This affects all RewardFlat customers.

### 3. Class RewardStepCustomer

RewardStep customers are customers that are in the rewards program and have a different discount rate structure. In particular, the RewardStep customers only receive a discount if they spend equal to or more than a threshold. For example, if the threshold is 50\$, then when the RewardStep customer named Mary purchases tickets that cost 60\$, the discount will be applied; when Mary only spends 40\$, then there will be no discount.

The discount rate might be different among the RewardStep customers. If not specified, the discount rate is set as 30%. In contrast, the threshold applies to all RewardStep customers, i.e., all RewardStep customers have the same threshold. The default threshold is 50\$.

The class **RewardStepCustomer** should have the following components:

- i. Constructor takes the appropriate parameters/arguments (be careful)
- ii. Appropriate getter methods for the attributes of this class

- iii. A method **get\_discount(self, cost)** which takes the ticket cost and returns the discount offered.
- iv. A method **display\_info(self)** that prints the values of the **RewardStepCustomer** attributes.
- v. A method **set\_discount\_rate** to adjust the discount rate of each individual RewardStep customer.
- vi. A method **set\_threshold** to adjust the threshold limit. This affects all RewardStep customers.

## Movies:

### 4. Class Movie

This class is to keep track of information on different movies that the cinema offers. This class supports the following information:

- **ID**: a unique identifier of the movie
- **name**: the name of the movie (you can assume the movie names are unique and they do not include any digit)
- **seat\_available**: the number of available seats for the movie
- A method **display\_info** that prints the values of the **Movie** attributes.
- Extra attributes and methods if you want to define.

## Tickets:

### 5. Class Ticket

This class is to keep track of information on different ticket types that the cinema offers. This class supports the following information:

- **ID**: a unique identifier of the ticket type
- **name**: the ticket type (you can assume the ticket types are unique and they do not include any digit)
- **price**: the price corresponding to the ticket type
- A method **display\_info** that prints the values of the **Ticket** attributes.
- Extra attributes and methods if you want to define.

## Bookings

### 6. Class Booking

This class is to store a customer's booking information. This class supports the following information of a booking:

- **customer**: the one who purchases the tickets (can be a customer, RewardFlat customer, or RewardStep customer). You need to think/analyse carefully if this should be an ID, name, or something else.
- **movie**: the movie the customer chooses. You need to think/analyse carefully if this should be an ID, name, or something else.
- **ticket**: the ticket type that the customer chooses. You need to think/analyse carefully if this should be an ID, name, or something else.
- **quantity**: the quantity of the ticket ordered by the customer.
- A method **compute\_cost** that returns the ticket cost (the ticket cost without the discount and booking fee), the booking fee, and the discount. For example, if the order is of the

customer Tom (RewardFlat customer with discount rate 20%), and the movie chosen is Avatar, ticket type is adult (unit price: 25.0) and quantity 2, then this method will return (50, 4, 10).

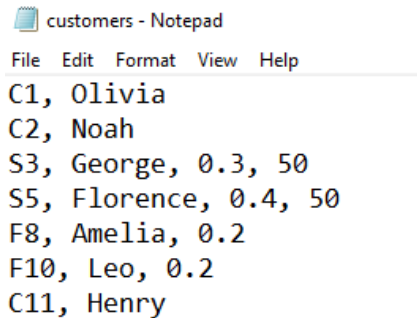
- Extra attributes and methods if you want to define.

## Records

### 7. Class Records

This class is the central data repository of your program. It supports the following information:

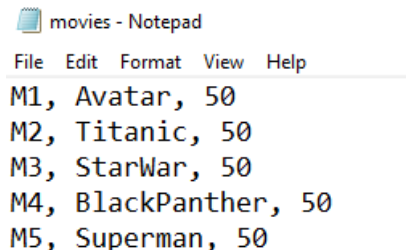
- **a list of existing customers** – you need to think what you should store in this list (customer ID, customer name, or something else?)
- **a list of existing movies the cinema offers** – you need to think about what you should store in this list (movie ID, movie name, or something else?)
- **a list of existing ticket types that the cinema offers** - you need to think about what you should store in this list (ticket type ID, ticket type's name, or something else?)
- This class has a method named **read\_customers**. This method takes in a file name and then read and add the customers in this file to the customer list of the class. In the sequel, we call this the *customer file*. See an example of the customer file below.



```
customers - Notepad
File Edit Format View Help
C1, Olivia
C2, Noah
S3, George, 0.3, 50
S5, Florence, 0.4, 50
F8, Amelia, 0.2
F10, Leo, 0.2
C11, Henry
```

In this file, the customers are always in this format: *customer\_ID*, *customer\_name*, *discount\_value* (if that is a RewardFlat/RewardStep customer), *threshold* (if that is a RewardStep customer). For example, in the 1<sup>st</sup> line, the *customer\_ID* is C1, the *name* is Olivia. In the 3<sup>rd</sup> line, the *customer\_ID* is S3, the *name* is George, the *discount* is 0.3 (i.e., 30%), and the *threshold* is 50. A normal customer has ID starting with the letter "C". A RewardFlat customer has the ID starting with the letter "F". A RewardStep customer has ID starting with the letter "S". The numbers in the ID after these characters (C, F, S) are all unique (i.e., 1, 2, 3, 5... are unique). In this part, you can assume there will be no error in this customer file (e.g., the data format is always correct, and the discount values and thresholds are always valid).

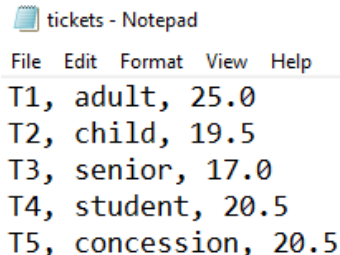
- This class has another method named **read\_movies**. This method takes in a file name and can read and add the movies stored in that file to the movie list of the class. In the sequel, we call this the *movie file*. See an example of the movie file below.



```
movies - Notepad
File Edit Format View Help
M1, Avatar, 50
M2, Titanic, 50
M3, StarWar, 50
M4, BlackPanther, 50
M5, Superman, 50
```

In this file, the movies are always in this format: *movie\_ID*, *movie\_name*, *number\_available\_seats*. The movie ID always starts with the letter "M". The movie IDs and names are all unique. You can assume there will be no error in this file (e.g., the data format is always correct, and the values are always valid).

- This class has another method named **read\_tickets**. This method takes in a file name and can read and add the ticket types stored in that file to the ticket type list of the class. In the sequel, we call this the *ticket file*. See an example of the ticket file below.



```

tickets - Notepad
File Edit Format View Help
T1, adult, 25.0
T2, child, 19.5
T3, senior, 17.0
T4, student, 20.5
T5, concession, 20.5

```

In this file, the data are always in this format: *ticket\_ID*, *ticket\_name*, *ticket\_unit\_price*. The ticket ID always starts with the letter "T". The IDs and the names are all unique. You can assume there will be no error in this file (e.g., the data format is always correct, and the prices are always valid).

- This class also has three methods **find\_customer**, **find\_movie**, and **find\_ticket**. These methods take in a search value (can be either a name or an ID of a customer, movie, or ticket type), search through the list of customers/movies/ticket types and then return the corresponding customer, movie, ticket type if found or return None if not found.
- This class also has three methods **display\_customers**, **display\_movies**, and **display\_tickets**. These three methods can display the information of existing customers, movies, and ticket types on screen. The method **display\_customers** will display the customer ID, name, the discount rate (only for RewardFlat/RewardStep customers), and the threshold (only for RewardStep customers). The method **display\_movies** will display the movie ID, name, and the number of available seats. The method **display\_tickets** will display the ticket type ID, name, and the unit price. Note the three methods can be used to validate the reading from the three .txt files associated with the customers, movies, and ticket types.

NOTE you are allowed to add extra attributes and methods in this class if these attributes and methods make your program more efficient.

## Operations

### 8. Class Operations

This can be considered the main class of your program. It supports a menu with the following options:

- Purchase a ticket*: this option allows users to purchase a ticket for a customer. Detailed requirements for this option are below (Requirements vii-ix).
- Display existing customers' information*: this option can display all the information of all existing customers: ID, name, discount rate (only for RewardFlat/RewardStep customers), and threshold (only for RewardStep customers).
- Display existing movies' information*: this option can display all the information of all existing movies: ID, name, and the number of available seats.



- iv. *Display existing ticket types' information:* this option can display all the information of all ticket types: ID, name, and the unit price.
- v. *Exit the program:* this option allows users to exit the program.

Other requirements of the menu program are as follows:

- vi. When the program starts, it looks for the files *customers.txt* (the customer file), *movies.txt* (the movie file), and *tickets.txt* (the ticket file) in the local directory (the directory that stores the .py file of the program). If found, the data will be read into the program accordingly, the program will then display a menu with the 5 options described above. If any file is missing, the program will quit gracefully with an error message indicating the corresponding file is missing.
- vii. Your menu program will allow the user to purchase a ticket as specified in PART 1 of Assignment 1. Note that in this assignment, the customer can choose to register for the rewards program as a RewardFlat or a RewardStep customer. More detailed information regarding the registration for the rewards program is in section viii below. In this part, like in PART 1 of Assignment 1, you can assume users always enter valid movies, valid ticket types, valid ticket quantities, and valid "y" or "n" answers. You can also assume users always enter the status for the rewards program accurately, for example, "F" for a RewardFlat customer registration, and "S" for a RewardStep customer registration.
- viii. When a customer finishes purchasing a ticket,
  - a. If the customer is a new customer, the program will add the information of that customer into the data collection (think/analyse carefully which information needs to be added). If the customer answers "n" for the question of registering for the rewards program, then the customer is just a standard customer. If the customer answers "y", then the program will ask what type of rewards they want. If the answer is "F", then the customer will become a RewardFlat customer. If the answer is "S", then the customer will become a RewardStep customer. The discount is applied immediately after the rewards program registration. Again, in this level, you can assume the users enter the rewards type correctly ("F" or "S").
  - b. If the customer is an existing customer, the program will print out a message showing the customer type (e.g., standard/RewardFlat/RewardStep customer), then proceed with the purchase and display the receipt. Also, for existing customers, you DO NOT need to ask if they want to register for the rewards program. This is slightly different compared to the requirements in Assignment 1, so please be careful.
- ix. The total cost of a booking can be displayed as a formatted message as below.

```

-----
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>

```

- x. When a task is accomplished, the menu will appear again for the next task. The program always exits gracefully from the menu.

----- CREDIT level (4 marks, please do not attempt this level before completing the PASS level) -----

### Operations

At this level, your program needs to handle exceptions compared to the PASS level. At this level, you are required to define various custom exceptions to handle the below issues:

- a. Display an error message if the movie entered by the user is not a valid movie or the movie is sold out (0 available seats). 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.
- e. Display an error message if the answer by the user is not *F* or *S* when asking the type of the rewards program. When this error occurs, the user will be given another chance, until a valid answer (i.e., *F*, *S*) is entered.

Besides,

- i. In this level, in the "*Purchase a ticket*" option, your program will allow ordering a group ticket, which is a ticket set that contains multiple tickets. For example, a group ticket can consist of two adult tickets and one child ticket. You can assume all components of a Group ticket are existing ticket types in the program.


The price of a Group ticket is 80% of the total price of all component tickets. For example, if a Group ticket consists of two adult tickets and one child ticket, and if an adult ticket costs 25.0\$, a child ticket costs 19.5\$, then the price of this Group ticket is  $0.8 \times (25.0 \times 2 + 19.5) = 55.6\$$ . Note the price of each Group ticket needs to be equal to or more than 50\$.

To support this feature, you need to add one more class named **GroupTicket** to your program.

9. **Class GroupTicket:** Each group ticket has a unique **ID** and **name** (as with **Ticket**). You need to define the appropriate attributes and methods to support the class **GroupTicket**. With this modification, the ticket file at this level may look like in the next page.

The type of a Group ticket is indicated with the letter "**G**" in the ID. Note that the data format of a Group ticket is different compared to a normal ticket type. Its format is as follows: *group\_ticket\_ID*, *group\_ticket\_name*, *ticket\_1*, *quantity\_1*, *ticket\_2*, *quantity\_2* ... The IDs/names of all ticket types (including the group ticket types) are all unique. The components of a group ticket can be shown as names or IDs. In the below example, the group ticket *Family3* consists of one adult ticket and two child tickets. You can assume all the ticket types in a group ticket are existing ticket types and unique (no duplicates). You can assume group tickets are always stored at the end of a file, after all normal ticket types. You can assume the ticket types' names never have any commas in the middle.



 tickets - Notepad  
File Edit Format View Help  
T1, adult, 25.0  
T2, child, 19.5  
T3, senior, 17.0  
T4, student, 20.5  
T5, concession, 20.5  
G6, Family3, adult, 1, child, 2  
G7, Family4, adult, 2, child, 2  
G8, GrandFamily, adult, 2, T3, 2, child, 1  
G9, Friend4, T1, 4  
G10, Student4, student, 4  
G11, Test, adult, 2

NOTE that as each group ticket needs to cost equal to or more than 50\$, so when your program reads a group ticket from the ticket file, and if it does not satisfy this requirement, it will display a message saying something wrong with this group ticket and ignore it. Note your program still processes other tickets/group tickets in the file as normal.

- ii. At this level, for the option "*Display existing ticket types' information*", when displaying the group tickets, your program will display the ID, name of the group ticket, and the price. On the other hand, the information of the standard ticket types is the same as in the PASS level.
- iii. At this level, your program should support both customers' IDs and names when purchasing a ticket. That is, your program will let users to enter either their IDs or their names. Your program should also support the movies' IDs and names, and tickets' IDs and names when purchasing a ticket.

### ----- DI Level (3 marks, please do not attempt this level before completing the CREDIT level) -----

In this level, there are some additional main features for some classes in your program. Some features might be challenging. Details of these features are described as follows.

#### Operations

Your program now has the following new features.

- i. In this level, in the "*Purchase a ticket*" option, your program will allow customers to purchase multiple ticket types in one booking. The requirements are as in Assignment 1 for this option (requirement 1 of Part 3). You can modify the existing classes or design extra classes to support this requirement.
- ii. Your program now has an option "*Add movies*" to add movies. The specification is same as Assignment 1 for this option (requirement 2 in Part 2).
- iii. Your program now has an option "*Adjust the discount rate of all RewardFlat customers*" to adjust the discount rate of all RewardFlat customers. This adjustment will affect all RewardFlat customers in all future orders. Invalid inputs (non-number or 0 or negative rate) should be handled via exceptions; the user will be given another chance until a valid input is entered.
- iv. Your program now has an option "*Adjust the discount rate of a RewardStep customer*". The option will ask for the name or ID of the RewardStep customer, then ask for a new discount rate (e.g., 0.2 which corresponds to 20% discount rate). Invalid customers (non-existent or non-RewardStep customers) needs to be handled, i.e., your program will give the user another

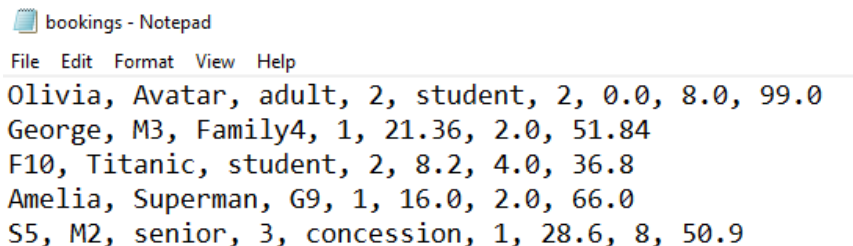
chance until a valid RewardStep customer is entered. Invalid inputs (non-number or 0 or negative values) should also be handled via exceptions, and the user will be given another chance until a valid input is entered. Also, your program should support both customers' IDs and names in this option, i.e., users can type either the customer's name or the ID.

- v. Note, in this part, you need to analyse the requirements and update some classes so that your program can satisfy the requirements listed above.

### ----- HD level (5 marks, please do not attempt this level before completing the DI level) -----

At this level, there are some additional features for some classes in your program. Note that some of them are very challenging (require you to optimize the class design and add components to support the features). Your program now will have the following features.

- i. The program now can automatically load the previous bookings which are stored in a comma separated file named *bookings.txt* that is located in the same directory with the .py file. In the sequel, we will call this the *booking file*. Below is an example of the booking file:



```

bookings - Notepad
File Edit Format View Help
Olivia, Avatar, adult, 2, student, 2, 0.0, 8.0, 99.0
George, M3, Family4, 1, 21.36, 2.0, 51.84
F10, Titanic, student, 2, 8.2, 4.0, 36.8
Amelia, Superman, G9, 1, 16.0, 2.0, 66.0
S5, M2, senior, 3, concession, 1, 28.6, 8, 50.9
  
```

Each line in the booking file is the information of a booking. The format is: *customer\_name/ID, movie\_name/ID, ticket\_type\_name/ID, ticket\_quantity, ticket\_type\_name/ID, ticket\_quantity ..., discount, booking\_fee, total\_cost*. You can assume all the customers in the booking file are existing customers (they are in the customer file). You can assume all movies in the booking file are existing movies (they are in the movie file). You can assume all ticket types in the booking file are existing ticket types (they are in the ticket file). The tickets can be normal tickets or group tickets. You can assume the ticket types are all distinguished. Customers, movies, and ticket types can be referred by IDs or names in this booking file. You can assume all other information (quantity, discount, booking fee, total cost) in this booking file is always valid, and the file entered by the user is always in the same directory with the .py file.

Note that errors when loading the booking file should also be handled. When there are any errors loading the file, your program will print a message saying "*Cannot load the booking file, run as if there is no previous booking file*".

- ii. Your program now has an option "*Display all bookings*" to display all bookings' information including the customer's name, movie's name, ticket types, ticket quantities, discount, booking fee, total cost. The printed message is flexible.
- iii. The program now can use command line arguments to accept four file names (the first being the customer file name, the second being the movie file name, the third being the ticket file name, and the fourth being the booking file). The first three files are mandatory, the fourth file is optional (i.e., if the booking file is not supplied, the program runs as if there was no previous bookings). If no file names are provided, your program will look for *customers.txt*, *movies.txt*, *tickets.txt*, and *bookings.txt* in the local directory. If a wrong number of arguments is provided, the program will display a message indicating the correct usage of arguments and exit.

- iv. The menu now 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.
- v. Your program will now have 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.

	<i>adult</i>	<i>child</i>	<i>senior</i>	<i>student</i>	<i>concession</i>	<i>Family3</i>	<i>...</i>	<i>Revenue</i>
<i>Titanic</i>	12	3	8	10	4	0	...	720.5
<i>Avatar</i>	15	7	5	14	6	2	...	924.6
<i>StarWar</i>	14	8	6	11	5	3	...	860.8

- vi. When your program terminates, it will update the three files: customers, movies, and bookings, based on the information when the program executes.

### B - Code Requirements:

The program **must be entirely in one Python file named ProgFunA2\_<Your Student ID>.py**. For example, if your student ID is s1234567, then the Python file must be named ProgFunA2\_s1234567.py. Other names will not be accepted.

Your code needs to be formatted consistently. You must not include any unused/irrelevant code. 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 libraries allowed in this assignment are sys and os.**

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 Assessment 2.

### 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.
3. **Any problems of your code and requirements that your program has not met.** For example, scenarios that might cause the program to crash or behave abnormally, requirements your program does not satisfy. Note that you don't 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:

Level	Points
PASS level	12
CREDIT level	4
DI level	3
HD level	5
Others (code quality, modularity, comments)	3
Others (weekly submission)	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. PASS Level

As an example, this is how the output screen of our sample program looks like for the PASS level, when we choose option 1, which is to purchase tickets with a customer named *Huong*, purchasing 2 adult tickets, and registering for the rewards program as a RewardFlat customer. You should test your program with different test cases, e.g., when customers choose *n* (no) for the rewards program registration option or when customers choose the RewardStep option, to make sure your program

satisfy the requirements of this level. Note that here, the program is implemented with the object-oriented paradigm and the classes described in the PASS level of Section 4.

```
Welcome to the RMIT movie ticketing system!

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

Choose one option: 1

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

What kind of rewards the customer wants?
F
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. CREDIT Level

As an example, this is what the output screen of our sample program looks like for the CREDIT level, when we choose option 1, which is to purchase tickets with a customer named *Huong*, ordering 1 *Friend4* group ticket and registering as a *RewardStep* customer. Here, we also test if the program can handle some types of invalid inputs. You should test your program with different test cases to make sure your program satisfies all the requirements of this level.

Besides, you should notice that at the beginning, there is a message displaying "*Group ticket G11 is not valid! The price of a group ticket type needs to be equal to or higher than 50!*" as the ticket type G11 (Test) in the ticket file has only two adult tickets, making its price to be  $0.8 \times (25.0 \times 2) = 40$ , thus, this ticket type is invalid.

```

Group ticket G11 is not valid! The price of a group ticket type needs to be equal to or higher than 50$!

Welcome to the RMIT movie ticketing system!

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

Choose one option: 1

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

Enter the name of the movie [enter a valid name only, e.g. Avatar]:
da;lk;a
Movie is invalid. Please enter a valid movie!

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

Enter the ticket type [enter a valid type only, e.g. adult, child, senior]:
Friend4

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

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

What kind of rewards the customer wants?
5
Successfully add the customer to the rewards program.

-----
Receipt of Huong
-----
Movie:                Titanic
Ticket type:          Friend4
Ticket unit price:    80.0
Ticket quantity:      1
-----
Discount:             24.0
Booking fee:          2
Total cost:           58.0
  
```

### 5.3. DI Level

As an example, this is what the output screen of our sample program looks like for the DI level, when we choose option 5 to add movies. After that, we choose option 3 to display all movies to check if the program successfully adds movies to the program. It shows that the program indeed adds movies successfully.

Note that you should test your program with different test cases to make sure your program satisfies all the requirements of this level.



```
Group ticket G11 is not valid! Each group ticket needs to cost more than 50$!
```

```
Welcome to the RMIT movie ticketing system!
```

```
#####
```

```
You can choose from the following options:
```

- 1: Purchase a ticket
- 2: Display existing customers' information
- 3: Display existing movies' information
- 4: Display existing ticket types' information
- 5: Add movies
- 6: Adjust the discount rate of all RewardFlat customers
- 7: Adjust the discount rate of a RewardStep customer
- 0: Exit the program

```
#####
```

```
Choose one option: 5
```

```
Enter the list of movies to be added (separating by commas):
```

```
Frozen , Avatar , Avenger
```

```
Adding the movie Frozen ...
```

```
The movie Avatar is an existing movie, will not do anything.
```

```
Adding the movie Avenger ...
```

```
Press enter to go back to the menu!
```

```
#####
```

```
You can choose from the following options:
```

- 1: Purchase a ticket
- 2: Display existing customers' information
- 3: Display existing movies' information
- 4: Display existing ticket types' information
- 5: Add movies
- 6: Adjust the discount rate of all RewardFlat customers
- 7: Adjust the discount rate of a RewardStep customer
- 0: Exit the program

```
#####
```

```
Choose one option: 3
```

```
ID: M1, name: Avatar, seats available: 50
```

```
ID: M2, name: Titanic, seats available: 50
```

```
ID: M3, name: StarWar, seats available: 50
```

```
ID: M4, name: BlackPanther, seats available: 50
```

```
ID: M5, name: Superman, seats available: 50
```

```
ID: M6, name: Frozen, seats available: 50
```

```
ID: M7, name: Avenger, seats available: 50
```

### 5.3. HD Level

As an example, this is what the output screen of our sample program looks like for the HD level, when the program takes command line arguments, and we pass 4 text files.

```
python assignment2_sample_hd_sem12023.py customers.txt movies.txt tickets.txt bookings.txt
```

```
Group ticket G11 is not valid! Each group ticket needs to cost more than 50$!
```

```
Welcome to the RMIT movie ticketing system!
```

```
#####
```

```
You can choose from the following options:
```

- 1: Purchase a ticket
- 2: Display existing customers' information
- 3: Display existing movies' information
- 4: Display existing ticket types' information
- 5: Add movies
- 6: Adjust the discount rate of all RewardFlat customers
- 7: Adjust the discount rate of a RewardStep customer
- 8: Display all bookings
- 9: Display the most popular movie
- 10: Display all movie record
- 0: Exit the program

```
#####
```

```
Choose one option:
```

Below is what the program looks like when we choose option 8 to display all bookings and option 10 to display all movie record. Note that you should test your program with different test cases to make sure your program satisfies all the requirements of this level.

```

Group ticket G11 is not valid! Each group ticket needs to cost more than 50$!

Welcome to the RMIT movie ticketing system!

#####
You can choose from the following options:
1: Purchase a ticket
2: Display existing customers' information
3: Display existing movies' information
4: Display existing ticket types' information
5: Add movies
6: Adjust the discount rate of all RewardFlat customers
7: Adjust the discount rate of a RewardStep customer
8: Display all bookings
9: Display the most popular movie
10: Display all movie record
0: Exit the program
#####

Choose one option: 8
Olivia, Avatar, adult, 2, student, 2, 0.0, 8.0, 99.0
George, StarWar, Family4, 1, 21.36, 2.0, 51.84
Leo, Titanic, student, 2, 8.2, 4.0, 36.8
Amelia, Superman, Friend4, 1, 16.0, 2.0, 66.0
Florence, Titanic, senior, 3, concession, 1, 28.6, 8.0, 50.9
  
```

```

#####
You can choose from the following options:
1: Purchase a ticket
2: Display existing customers' information
3: Display existing movies' information
4: Display existing ticket types' information
5: Add movies
6: Adjust the discount rate of all RewardFlat customers
7: Adjust the discount rate of a RewardStep customer
8: Display all bookings
9: Display the most popular movie
10: Display all movie record
0: Exit the program
#####

Choose one option: 10

```

	adult	child	senior	student	concession	Family3	Family4	GrandFamily	Friend4	Student4	Revenue
Avatar	2	0	0	2	0	0	0	0	0	0	99.00
Titanic	0	0	3	2	1	0	0	0	0	0	87.70
StarWar	0	0	0	0	0	0	1	0	0	0	51.84
BlackPanther	0	0	0	0	0	0	0	0	0	0	0.00
Superman	0	0	0	0	0	0	0	0	1	0	66.00

## 6. Submission

As mentioned in the Code Requirements, **you must submit only one file named ProgFunA2\_<Your Student ID>.py** via Canvas/Assignments/Assignment 2. **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.

### Weekly Submission

You can submit your code every week starting from Week 9 to Week 11 (you will be awarded marks for the weekly submissions), and the final version before the due date in Week 12. In each weekly submission, you need to write some code demonstrating some parts of your program (at least 50 lines of code per week – not include comments). If your code in the weekly submissions is related to the assignment and satisfy the condition of at least 50 lines of code per week, then you are awarded full 1

mark per each weekly submission (maximum 3 marks for 3 weeks, excluding the final submission in Week 12). If your code in the weekly submission is not satisfied the criteria mentioned in the previous sentence, you are awarded 0.5 marks for each weekly submission. If you do not submit any file, you are not awarded any mark for that week.

### 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](#).

## 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 or 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>