Programming Fundamentals (COSC2531) Assignment 2

Assessment Type	**Individual assignment** (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.
Due Date	**End of Week 12** (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.
Weighting	**30 marks out of 100**

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:

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 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 book rental system as in Assignment 1 using the Object-Oriented Programming (OOP) paradigm. Same as in Assignment 1, the receptionists are the ones that use this system to process and print out receipts of the customers' rentals. 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 (10 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 only contain alphabet characters). 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 **display_info** that prints the values of the **Customer** attributes

2. **Class Member**

   All members have a discount when renting a book and all members will have the same discount rate. By default, the discount rate is 10%. The class Member should have the following components:

   i. An attribute for the **discount rate**
   ii. Constructor takes the appropriate parameters/arguments
   iii. Appropriate getter methods for the attributes of this class
   iv. A method **get_discount** that takes the rental cost and returns the discount. For example, when the rental cost is 5$ and the discount rate is 10%, then the method returns 0.5.
   v. A method **display_info** that prints the values of the **Member** attributes.
   vi. A method **set_discount_rate** to adjust the discount rate. This method affects all members.

3. **Class GoldMember**

   A Gold member not just receives the discount but also gets a reward for the rental. The discount rate is the same for all Gold members, and by default, it is set as 12%. The reward rate is different for each Gold member and if not specified, the reward rate is set as 100%.

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

   i. Appropriate attributes to support the **discount rate**, **reward rate**, and **reward**
   ii. Constructor takes the appropriate parameters/arguments
   iii. Appropriate getter methods for the attributes of this class
   iv. A method **get_discount** which takes the rental cost and returns the discount offered
   v. A method **get_reward** which takes the rental cost after the discount and returns the reward. Note, the reward is always rounded. For example, when the rental cost after the discount is 4.8 and the reward rate is 100%, then the method will returns the reward which is *round(4.8 x 100%)* = 5.
   vi. A method **update_reward** which takes a value and increase the attribute **reward** with that value.
   vii. A method **display_info** that prints the values of the **GoldMember** attributes
   viii. A method **set_discount_rate** to adjust the discount rate of Gold members. This method affects all Gold members.
   ix. A method **set_reward_rate** to adjust the reward rate of each individual Gold member.

Products:

4. Class Book

This class is to keep track of information on different books that can be rented. This class supports the following information:

- **ID**: a unique identifier of the book
- **name**: the name of the book (you can assume the book names are unique, and they do not include any digit)
- **category**: the book category of the book (you need to think carefully if this should be an ID, name, or something else)
- A method **display_info** that prints the values of the **Book** attributes
- A method **get_price** which takes the number of borrowing days and returns the price of the book
- Extra attributes and methods if you want to define

5. Class BookCategory

This class is to keep track of information on different book categories. This class supports the following information:

- **ID**: a unique identifier of the book category
- **name**: the name of the book category (you can assume the book category names are unique and they do not include any digit)
- **price_1**: the first-tier rental price per day
- **price_2**: the second-tier rental price per day
- **books:** a list of books in the book category (you need to think/analyse carefully about what information in the list, e.g. if they should be IDs, names, or something else – see the method **add_book** for a hint)
- A method **display_info** that prints the values of the **BookCategory** attributes
- A method **add_book** which takes a Book object and then append the Book object into the variable **books**
- A method **get_price** which takes the number of borrowing days and returns the price of the book category
- Extra attributes and methods if you want to define

Orders

6. Class Rental

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

- **customer**: the one who makes the rental (can be a Customer, Member or Gold Member). You need to think/analyse carefully if this should be an ID, name, or something else.
- **book**: the rented book. You need to think/analyse carefully if this should be an ID, name, or something else.
- **number of borrowing days**: the number of borrowing days ordered by the customer
- A method **compute_cost** that returns the original cost (the cost before the discount), the discount, the total cost (the cost after the discount), and the reward (for Gold members only). For example, if the original cost of a rental of the customer Noah (Gold

member with discount rate of 12% and reward rate of 100%) is 5, then this method will return (5.0, 0.6, 4.4, 4). For a member with the same discount rate, then this method will return (5.0, 0.6, 4.4).

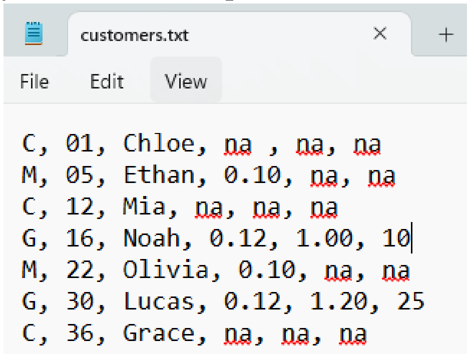
- Extra attributes and methods if you want to define.

Records

7. Class Records

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

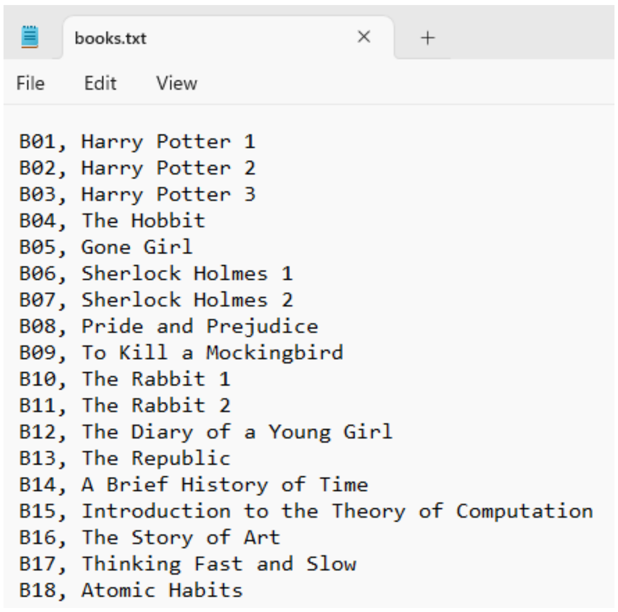
- **a list of existing customers** – you need to think what you should store in this list (customer ID, customer name, or something else?)
- **a list of existing book categories** – you need to think about what you should store in this list (book category ID, book category name, or something else?)
- **a list of existing books -** you need to think about what you should store in this list (book ID, book 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.



```
C, 01, Chloe, na , na, na
M, 05, Ethan, 0.10, na, na
C, 12, Mia, na, na, na
G, 16, Noah, 0.12, 1.00, 10
M, 22, Olivia, 0.10, na, na
G, 30, Lucas, 0.12, 1.20, 25
C, 36, Grace, na, na, na
```

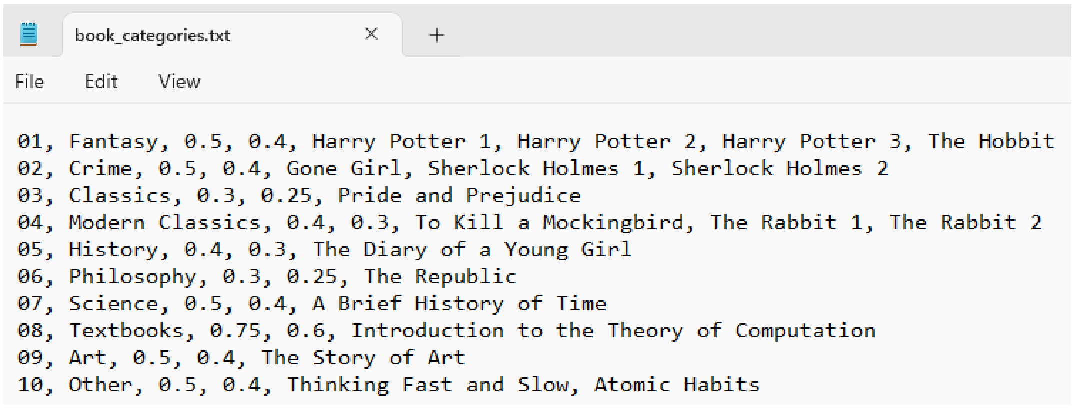
In this file, the customers are always in this format: *customer type, customer ID, customer_name, discount_rate, reward_rate* and *reward points*. For example, in the 1st line, the *customer type* is *C* – meaning a normal customer, the *customer ID* is 01, the *customer name* is Chloe, the *discount rate* is na – meaning this customer has no discount, the *reward rate* is na – meaning this customer has no reward rate, and the *reward points* is na – meaning this customer has no reward points. In the 4th line, the *customer type* is *G* – meaning a Gold member, the *customer ID* is 16, the *customer name* is Noah, the *discount rate* is 0.12, the *reward rate* is 1.00, and the *reward points* is 10. A normal customer has the customer type being the letter "*C*", a member has a customer type being the letter "*M*" whilst a Gold member has a customer type being "*G*". The IDs are all unique (i.e., 01, 05, 12, 16… are unique). For the fields that do not have values, e.g. discount rate of a customer, there will be the value *na*. 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 and reward values are always valid).

- This class has another method named **read_books_and_book_categories**. This method takes in *two file names* and can read and add the books and book categories stored in those two files to the list of books and the list of book categories. In the sequel, we call these the *book file* and *book category file*. See an example of these two files below.



```
B01, Harry Potter 1
B02, Harry Potter 2
B03, Harry Potter 3
B04, The Hobbit
B05, Gone Girl
B06, Sherlock Holmes 1
B07, Sherlock Holmes 2
B08, Pride and Prejudice
B09, To Kill a Mockingbird
B10, The Rabbit 1
B11, The Rabbit 2
B12, The Diary of a Young Girl
B13, The Republic
B14, A Brief History of Time
B15, Introduction to the Theory of Computation
B16, The Story of Art
B17, Thinking Fast and Slow
B18, Atomic Habits
```

In the book file, the books are always in this format: *book_ID, book_name*. The book IDs and names are all unique. In this level, you can assume there are no format errors in this file (e.g., the data format is always correct, and the values are always valid).



```
01, Fantasy, 0.5, 0.4, Harry Potter 1, Harry Potter 2, Harry Potter 3, The Hobbit
02, Crime, 0.5, 0.4, Gone Girl, Sherlock Holmes 1, Sherlock Holmes 2
03, Classics, 0.3, 0.25, Pride and Prejudice
04, Modern Classics, 0.4, 0.3, To Kill a Mockingbird, The Rabbit 1, The Rabbit 2
05, History, 0.4, 0.3, The Diary of a Young Girl
06, Philosophy, 0.3, 0.25, The Republic
07, Science, 0.5, 0.4, A Brief History of Time
08, Textbooks, 0.75, 0.6, Introduction to the Theory of Computation
09, Art, 0.5, 0.4, The Story of Art
10, Other, 0.5, 0.4, Thinking Fast and Slow, Atomic Habits
```

In the book category file, the book categories are always in this format: *book_category_ID, book_category_name, price_1, price_2, book_1, book_2, …*. The book category IDs and names are all unique. In this level, you can assume there will be no format errors in this file (e.g., the data format is always correct, and the values are always valid). All the books in this book category file will be in the book file.

- This class also has three methods **find_customer**, **find_book_category**, and **find_book**. These methods take in a search value (can be either a name or an ID of a customer or book category or book), search through the list of customers/book categories/books and then return the corresponding customer/book category/book if found or return None if not found.
- This class also has three methods **list_customers**, **list_books**, and **list_book_categories**. These methods can display the information of existing customers, books, and book categories on screen. The method **list_customers** will display the customer ID, name, the discount rate, the reward rate and the reward. The method **list_books** will display the book ID, book name, and book category name. The method **list_book_categories** will display the book category ID, book category name, and the two prices per day, and the list of book names. Note these three methods can be

used to validate the reading from the three text files associated with the customers, books, and book categories.

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:

i. *Rent a book*: this option allows users to rent a book for a customer. Detailed requirements for this option are below (Requirements vi-ix).
ii. *Display existing customers*: this option can display all the information of all existing customers: ID, name, discount rate, reward rate and reward.
iii. *Display existing book categories:* this option can display all the information of all existing book categories: ID, name, the two prices per day, and the list of book names.
iv. *Display existing books:* this option can display all the information of all existing books: ID, name, and book category.
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), *books.txt* (the book file), and *book_categories.txt* (the book category 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 not found, the program will quit gracefully with an error message indicating the corresponding file is missing.
vii. Your menu program will allow the user to rent a book as specified in PART 1 of Assignment 1. In this part, like in PART 1 of Assignment 1, you can assume users always enter valid customer names, books, and number of borrowing days. Besides, note that for Gold members, there will be the computation of the reward points, and they are computed based on the reward rate and the total cost (which is = the original cost – discount).
viii. When a customer finishes renting a book, if the customer is a new customer, the program will also ask if the customer wants to be a member of just a customer, then add the information of that customer into the data collection (think/analyse carefully which information needs to be added). Note that the Gold members will only be from the customer file (i.e. no registration during the renting a book process).
ix. The receipt of a rental of a Customer or a Member can be displayed as a formatted message as below.

```
---------------------------------------------------------------------------------
Receipt for <customer_name>
---------------------------------------------------------------------------------
Books rented:
   - <book_name> for <number_days> days (<price_per_day> AUD/day)
---------------------------------------------------------------------------------
Original cost:        <original_cost> (AUD)
Discount:             <discount> (AUD)
Total cost:           <total_cost> (AUD)
```

The receipt of a rental of a Gold Member can be displayed as a formatted message as below.

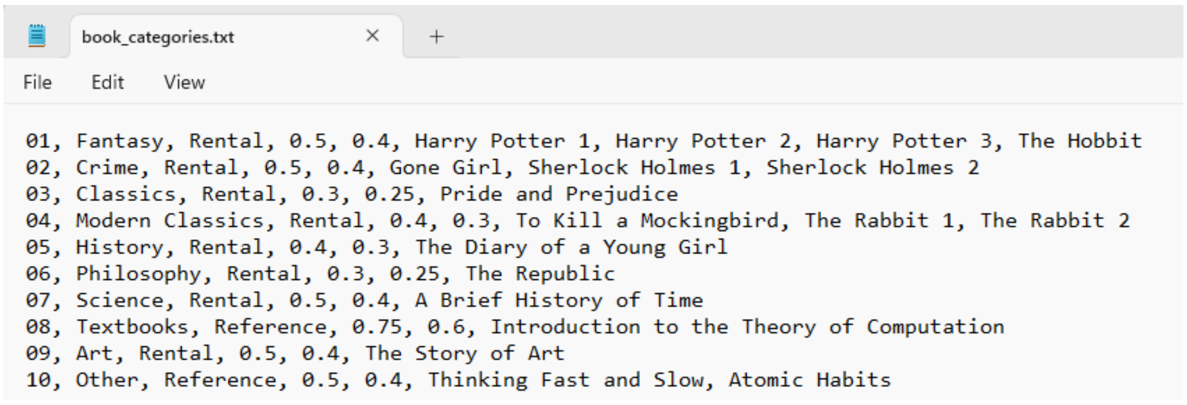
```
-------------------------------------------------------------------------------------
Receipt for <customer_name>
-------------------------------------------------------------------------------------
Books rented:
    - <book_name> for <number_days> days (<price_per_day> AUD/day)
-------------------------------------------------------------------------------------
Original cost:      <original_cost> (AUD)
Discount:           <discount> (AUD)
Total cost:         <total_cost> (AUD)
Reward:             <reward_points>
```

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

i. At this level, your program will need to have the feature d) specified in PART 2 of Assignment 1 (i.e., each book category will belong to one of the two types: *Rental* and *Reference*). In this level, the format of the book category file *book_categories.txt* will look as follows:



```
01, Fantasy, Rental, 0.5, 0.4, Harry Potter 1, Harry Potter 2, Harry Potter 3, The Hobbit
02, Crime, Rental, 0.5, 0.4, Gone Girl, Sherlock Holmes 1, Sherlock Holmes 2
03, Classics, Rental, 0.3, 0.25, Pride and Prejudice
04, Modern Classics, Rental, 0.4, 0.3, To Kill a Mockingbird, The Rabbit 1, The Rabbit 2
05, History, Rental, 0.4, 0.3, The Diary of a Young Girl
06, Philosophy, Rental, 0.3, 0.25, The Republic
07, Science, Rental, 0.5, 0.4, A Brief History of Time
08, Textbooks, Reference, 0.75, 0.6, Introduction to the Theory of Computation
09, Art, Rental, 0.5, 0.4, The Story of Art
10, Other, Reference, 0.5, 0.4, Thinking Fast and Slow, Atomic Habits
```

Each line in the book category file has the following format: *book_category_ID, book_category_name, book_category_type, price_1, price_2, book_1, book_2, …* The value of *book_category_type* could be *Rental* or *Reference*. You will need to think of which parts of your program and which classes to be updated for your program to satisfy this feature.

ii. Besides, 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 customer's name entered by the user contains non-alphabet characters. When this error occurs, the user will be given another chance, until a valid name (names contain only alphabet characters) is entered.
   b. Display an error message if the book entered by the user is not a valid book. When this error occurs, the user will be given another chance, until a valid book is entered.

c. Display an error message if the number of borrowing days entered is 0, negative, or not an integer. When this error occurs, the user will be given another chance, until a valid quantity is entered.
d. Display an error message saying this book can't be borrowed for more than 14 days and ask the user to enter the number of borrowing days again when the number of borrowing days is larger than 14 for books from the *Reference* type.

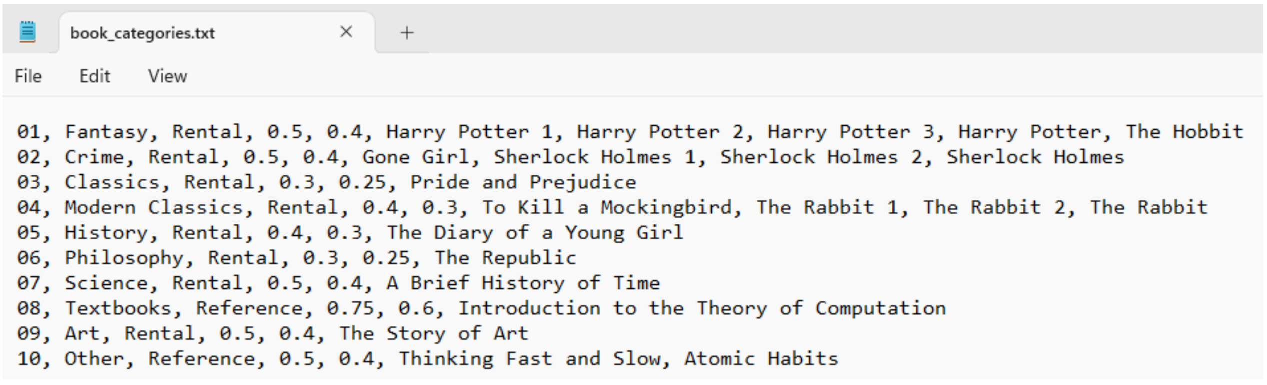
iii. Furthermore, in this level, in the "*Rent a book*" option, your program will allow ordering a BookSeries, which contain multiple books. For example, a book series can consist of Harry Potter 1, Harry Potter 2, Harry Potter 3. You can assume all books of a book series are existing books in the system. And all books from a book series belong to the same book category.

The rental price of a book series is 50% of the total rental prices of all individual books. For example, if then rental price of each Harry Potter book is 0.5, then the price per day of all the books from the book series Harry Potter (which consists of Harry Potter 1, Harry Potter 2, and Harry Potter 3) is 50% x (0.5 x 3) = 0.75$.

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

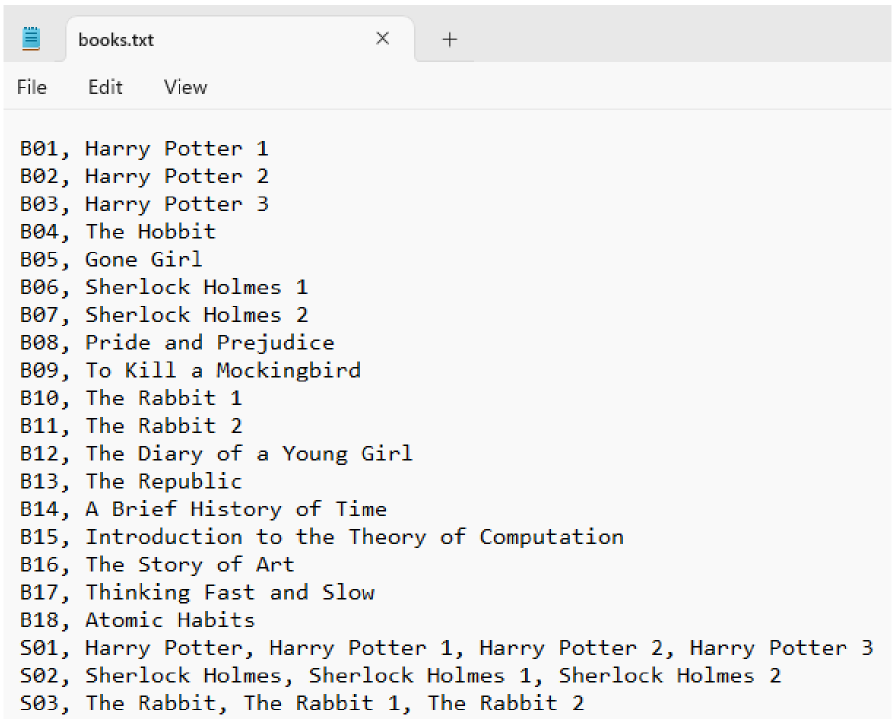
9. Class BookSeries: Each book series has a unique **ID**, **name**, and **book category** (as with **Book**). The book category of a book series is the book category of the component books. You need to define the appropriate variables and methods to support the class **BookSeries**.

With this modification, the book category file *book_categories.txt* at this level now may look like this:



```
01, Fantasy, Rental, 0.5, 0.4, Harry Potter 1, Harry Potter 2, Harry Potter 3, Harry Potter, The Hobbit
02, Crime, Rental, 0.5, 0.4, Gone Girl, Sherlock Holmes 1, Sherlock Holmes 2, Sherlock Holmes
03, Classics, Rental, 0.3, 0.25, Pride and Prejudice
04, Modern Classics, Rental, 0.4, 0.3, To Kill a Mockingbird, The Rabbit 1, The Rabbit 2, The Rabbit
05, History, Rental, 0.4, 0.3, The Diary of a Young Girl
06, Philosophy, Rental, 0.3, 0.25, The Republic
07, Science, Rental, 0.5, 0.4, A Brief History of Time
08, Textbooks, Reference, 0.75, 0.6, Introduction to the Theory of Computation
09, Art, Rental, 0.5, 0.4, The Story of Art
10, Other, Reference, 0.5, 0.4, Thinking Fast and Slow, Atomic Habits
```

And the book file *books.txt* at this level now may look like this:



```
B01, Harry Potter 1
B02, Harry Potter 2
B03, Harry Potter 3
B04, The Hobbit
B05, Gone Girl
B06, Sherlock Holmes 1
B07, Sherlock Holmes 2
B08, Pride and Prejudice
B09, To Kill a Mockingbird
B10, The Rabbit 1
B11, The Rabbit 2
B12, The Diary of a Young Girl
B13, The Republic
B14, A Brief History of Time
B15, Introduction to the Theory of Computation
B16, The Story of Art
B17, Thinking Fast and Slow
B18, Atomic Habits
S01, Harry Potter, Harry Potter 1, Harry Potter 2, Harry Potter 3
S02, Sherlock Holmes, Sherlock Holmes 1, Sherlock Holmes 2
S03, The Rabbit, The Rabbit 1, The Rabbit 2
```

The ID of a BookSeries always starts with the letter "*S*". Note that the data format of a book series is different compared to a book; it includes the names of the book components. The IDs/names of the book series are all unique (no duplicates). You can assume book series are always stored at the end of a file, after all books.

iv. At this level, the option "*Display existing books*" will also be able to display information of the book series. When displaying books, your program will display the book IDs, the book name, and the book category. When displaying book series, your program will display the book series' ID, name, the component books, and the book category.

v. Finally, your program should support both IDs and names of the customers, book categories, books/book series for any functions that use this information. For example, instead of entering the customer names, book category names, book names, book series names, now, the user can enter the customer IDs and book category/book/book series IDs, respectively.

---------- DI Level (5 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 "*Rent a book*" option, your program will allow customers to enter multiple books and number of borrowing days in one rental. 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. In this level, in the option "*Rent a book*", your program will check the current reward points of a Gold member before finishing a rental, if they are larger than 20, then the program can deduct some amount of money (corresponding to the reward points) from the total cost and update the remaining reward points of the Gold member. Specifically, every 20 reward points can be converted to 1$ and be deducted from the total cost. Note the earned reward points from the rental are based on the total cost before the reward deduction. For example, if the total cost of the rental (after the discount) is 6.2$, the customer currently has 25 reward points (before the rental), then the total cost will be 5.2$ (as the customer can convert 20 reward points to 1$), the earned reward points of this purchase are still 6, and the customer will now have 5 + 6 = 11 reward points after the rental.

iii. Your program now has an option "*Update information of a book category*" to update information of a book category including the type and the two rental prices per day. The specification is same as Assignment 1 for this option (requirement 2 in PART 2 and requirement 2 in PART 3).

iv. Your program now has an option "*Update books of a book category*" to update books (including book series) of a book category. The specification is same as Assignment 1 for this option (requirement 3 in PART 2 and requirement 3 in PART 3), however, note that the format of the list of books to be added/removed will be: *book_category, book_1: book_1_ID, book_2: book_2_ID, ...* Note that for the books removed from a book category, they will also be removed from the data collection of the program.

v. Your program now has an option *"Adjust the discount rate of all members"* to adjust the discount rate of all members. This adjustment will affect all members in all future rentals. 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. Note that in this option, if the user enters 0.2, this means the discount rate is 20%.

vi. Your program now has an option "*Adjust the reward rate of a Gold member*". The option will ask for the name or ID of a Gold member, then ask for a new reward rate. Invalid customers (non-existent or non-Gold members) needs to be handled, i.e., your program will give the user another chance until a valid Gold member 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. Note that 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. Besides, note that in this option, if the user enters 1, this means the reward rate is 100%.

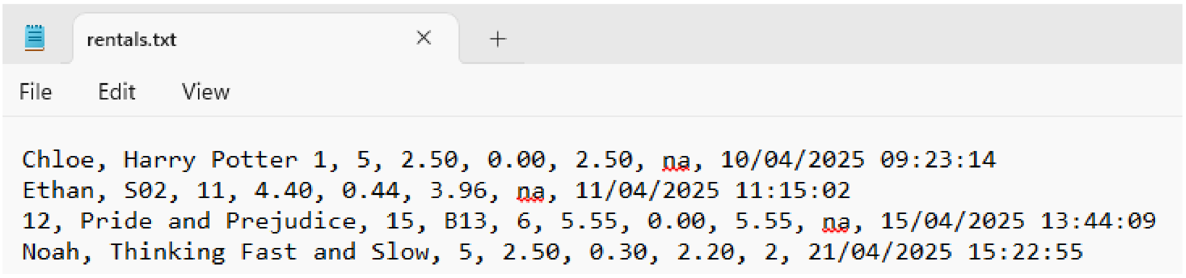
Note, in this part, you need to analyse the requirements and update some classes so that your program can satisfy the requirements listed above.

------------ 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 option "*Rent a book*" will then now be able to save the timestamp when a rental is made. The format of the timestamp is *dd/mm/yyyy hh:mm:ss*. For example, a valid timestamp with this format is *10/04/2025 10:05:12*.

ii. The program now has an option "*Rent books via a file*" to accept rental via a file. This option will ask the user to enter the name of the file and then add all the rentals in the file into the data collection. Below is an example of the file:



```
Chloe, Harry Potter 1, 5, 2.50, 0.00, 2.50, na, 10/04/2025 09:23:14
Ethan, S02, 11, 4.40, 0.44, 3.96, na, 11/04/2025 11:15:02
12, Pride and Prejudice, 15, B13, 6, 5.55, 0.00, 5.55, na, 15/04/2025 13:44:09
Noah, Thinking Fast and Slow, 5, 2.50, 0.30, 2.20, 2, 21/04/2025 15:22:55
```

Each line in the order file is the information of an order. The format is: *customer_name/ID, book1_name/ID, number_borrowing_days1, book2_name/ID, number_borrowing_days2, ..., original_cost, discount, total_cost, earned_rewards, date.* You can assume all the customers in the rental file are existing customers (they are in the customer file). You can assume all books/book series in the rental file are existing books/book series (they are in the book file) and are valid. Customers and books/book series can be referred by IDs or names in this rental file. You can assume all other information (number of borrowing days, original cost, discount, total cost, earned reward points) in this rental file is always valid. Note that after loading this rental file, the reward points of the Gold members will also be updated based on the reward points from this rental file. For the customers and members, the values of reward points will be *na*.

Note that errors when loading this file should also be handled. When the file cannot be found in the directory, your program will print a message saying: "*Cannot find the rental file.*".

iii. Your program now has an option "*Display all rentals*" to display all rentals' information: the customer's name, books/book series, number of borrowing days, original cost, discount, total cost, earned rewards (for Gold members), and the rental time. The printed message is flexible.

iv. The program now can use command line arguments to accept three file names (the first being the customer file name, the second being the book file name, and the third being the book category file name). These three files are mandatory. If no file names are provided, your program will look for *customers.txt, books.txt,* and *book_categories.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.

v. Your program will now have an option *"Display the most valuable customer"*. The specification of this option is as in Assignment 1.

vi. Your program will now have an option *"Display a customer rental history"*. The option will show a table displaying all the previous rentals of a particular customer. The specification is as in Assignment 1 but also include the reward points. Note that here, the total cost is the final total cost (after the discount).

This is the rental history of Chloe.

	Books & Borrowing days	*Original Cost*	*Discount*	*Final Cost*	*Rewards*
Rental 1	*Harry Potter 1: 7 days, Gone Girl: 12 days*	*8.30*	*0.83*	*7.47*	*na*
Rental 2	*A Brief History of Time: 20 days*	*8.00*	*0.80*	*7.20*	*na*
Rental 3	*The Republic: 30 days, The Hobbit: 5 days*	*10.00*	*1.00*	*9.00*	*na*

vii. When your program terminates, it will update the three files: *customer, book,* and *book category*. Furthermore, it will also save all rentals to a rental file (*rentals.txt*), based on the information when the program executes. The format of the rental file is similar to the format of the file used for the option "*Rent books via a file*".

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, datetime 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) 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	10
CREDIT level	4
DI level	5
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. 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.

6. 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.

7. 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).

8. 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