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 1) 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 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, abstract classes, method overloading, 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 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 bus booking system based on Assignment 1 using object-oriented approach. As specified in Assignment 1, this bus booking system is for a travel agency which delivers bus services from Melbourne to other destinations in Australia. The agents from the travel agency are the ones that use this system to book trips for customers. You are required to implement the program following the below requirements. Note the requirements in this assignment maybe more complex compared to those in Assignment 1.

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 parts**, please ensure you only attempt one part after completing the previous part.

---------- PART 1 (12 marks) ----------

In this part, 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 PART 1, your program should be able to run with a menu described as in the Operations.

Customers:

1. Class Customer

All customers have a unique **ID**, unique **name** (a name will not include any digit). You are required to write the class named **Customer** to support the following:

i. Attributes **ID** and **name**
ii. Attribute **value** for total money the customer spent to date. A new customer will have the value of **value** to be 0 before booking any trip.
iii. Constructor takes the values of **ID**, **name**, and **value** as arguments
iv. Appropriate getter/accessor methods for the attributes of this class
v. A method **get_discount(self, cost)** which should be an empty super method
vi. A method **displayCustomer** that prints the ID, name and value of a customer

2. Class Member

A member is a customer with a membership. When booking a trip, a member will be offered a discount for the booking cost (discount applied to ticket cost only). All members are offered a discount of a flat rate (i.e. all members have the same discount rate). The class **Member** should have the following components:

i. An attribute for **rate of discount**, by default it is 5%.
ii. Constructor takes the appropriate parameters/arguments (be careful)
iii. A method **get_discount(self, cost)** which takes the booking cost (ticket cost only) and returns both the discount rate and the discount of the booking. For example, this method returns (0.05, 50) when the discount rate is 5% and the cost is 1000$.
iv. Appropriate getter/accessor methods for the attributes of this class
v. A method **displayCustomer** that prints the ID, name, value and discount rate of a member.
vi. A method **setRate** to adjust the flat rate of discount. This affects all members.

3. Class VIPMember

A VIP member is a special member (i.e. a customer with a VIP membership). All VIP members are offered a discount based on two rates: the first rate applies when the booking cost (ticket cost only) is smaller or equal than a threshold ($1000 by default), and the second rate applies when the booking cost (ticket cost only) exceeds this threshold. For example, with the threshold being 1000$, then, when a VIP member named Sarah books a trip that costs 800$ (ticket cost), the discount rate for this booking is the 1st discount rate; when Sarah books a trip that costs 1200$ (ticket cost), the discount rate for this booking is the 2nd discount rate.

NOTE for all VIP members, the 2nd discount rate is always 5% more than the 1st discount rate. The discount rates might be different among the VIP members. If not specified, the first and second discount rates are set as 10% and 15%, respectively. On the other hand, the threshold applies to all VIP members, i.e. all VIP members have the same threshold.

The class **VIPMember** should support the following components

i. Attributes to support the two discount rates
ii. Necessary constructors
iii. A method **get_discount(self, cost)** which takes the cost and returns both the discount rate and the discount offered.
iv. Appropriate getter/accessor methods for the attributes of this class
v. A method **displayCustomer** that prints the ID, name, value, two discount rates and the threshold limit of a VIP member.
vi. A method **setRate** to adjust the discount rates of each individual VIP member.
vii. A method **setThreshold** to adjust the threshold limit. This affects all VIP members.

Destinations

4. Class Destination

This class is to keep track of information of different destinations that the travel agency supports. This class supports the following information:

- **ID**: A unique identifier for the destination (e.g. D1, D2)
- **Name**: The name of the destination (you can assume the destination names are unique and they do not include any digit)
- **Price**: The ticket price of the destination
- **SeatAvailable**: the number of available seats of the destination

You need to define appropriate variables and methods to support the class **Destination**. Note that SeatAvailable obviously will be changed. The ticket price of a destination may also be changed by the user.

Bookings

5. Class Booking

This class is to handle the customer booking. This class supports the following information of a booking:

- **Customer**: the one who made the booking (can be a normal customer, a customer with a normal membership, or a customer with a VIP membership). Note you need to think/analyse if this should be an ID, name, or something else.
- **Destination**: the destination of the booking. Note you need to think/analyse if this should be an ID, name, or something else.
- **Quantity**: the number of tickets booked by customers.
- You need to think if there are any extra variables/information you want to store in the Booking class
- You need to think if there are any extra methods need to be in this class

Note that this class can update information in the corresponding customer and destination if necessary. For example, an object from the class Booking can update the information of the corresponding customer (e.g. value, discount rate) or/and the destination (e.g. seat available). Therefore, you need to define appropriate variables and methods to support this class.

Also, note that you may need to find a way to manage multiple bookings. It is up to you how that can be handled, e.g. by a dedicated class, or by a part of an existing class.

Records

6. 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 (ID, name, or something else?)
- **A list of existing destinations** – you need to think what you should store in this list (ID, name, or something else?)
- This class has a method named **readCustomers** that can read a comma separated file called *customers.txt* and add the customers in this file to the customer list of the class. See an example of the *customers.txt* file as below.

```
C1, James, 0, 100
C2, Lily, 0, 30
M3, Tom, 0.05, 500
V5, Harry, 0.125, 1000
C8, Annie, 0, 60
V10, Sarah, 0.15, 2000
M11, Wilson, 0.05, 300
```

In this file, customers are always in this format: *ID, name, discount rate*, and *value* (value of all the bookings made in the past). For example, in the 1st line, the *ID* is C1, the *name* is James, the *discount rate* is 0, and the *value* is 100. A normal customer has ID starting with the letter *"**C**"*. A member (customer with a normal membership) has ID starting with the letter *"**M**"*. A VIP member (customer with a VIP membership) has ID starting with the letter *"**V**"*. The numbers in the ID after these characters (C, M, V) are all unique (i.e. 1, 2, 3, 5… are unique). In this assignment, you can assume there will be no error in this file *customers.txt* (the formats of the data are always correct, and the values are always valid).

- This class has another method named **readDestinations** that can read another comma separated value file called "destinations.txt" and add the destinations stored in that file to the destination list of the class. See an example of the "destinations.txt" file as below.

```
D1, Sydney, 150, 30
D2, Geelong, 20, 20
D3, Adelaide, 120, 35
D4, Brisbane, 250, 40
```

In this file, destinations are always in this format: *ID, name, ticket price* (price per each ticket), and the *number of seats available*. The IDs of all destinations always start with the letter "***D***". In this assignment, you can assume there will be no error in this file (the formats of the data are always correct, and the values are always valid).

- This class also has two methods **findCustomer** and **findDestination**. These two methods are to search through the list of customers and destinations to find out whether a given customer or a given destination exists or not. If found, the corresponding customer and destination will be returned, otherwise, None will be returned. Note that both the customer destination can be searched using either ID or name.

- This class also has two methods **listCustomers** and **listDestinations**. These two methods can list all the existing customers and destinations on screen. The display format is the same as in the .txt files shown above. These methods can be used to validate the reading from the .txt files *customers.txt* and *destinations.txt.*

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

Operations

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

i. *Book a new trip*: this option allows user to book a trip for a customer. Detailed requirements for this option are in the paragraph below.
ii. *Display the current customer list*: this option can display the information of all the existing customers with the format as in the file *customers.txt* (as specified in **Records**).
iii. *Display the current destination list:* this option can display the information of all the existing destinations with the format as in the file *destinations.txt* (as specified in **Records**).
iv. *Exit the program:* this option allows users to exit the program.

Other requirements of the menu program are as follows:

v. When the program starts, it looks for the files *customers.txt* and *destinations.txt* in the local directory (the directory that stores the .py file). If found, the data will be read into the program accordingly, the program will then display a menu with the 4 options described above. If any file is missing, the program will quit gracefully with an error message indicating the files are missing.
vi. Your menu program will allow the user to manually enter a booking as specified in PART 1 of Assignment 1. Note that in this case, the customer can choose to get a normal membership or a VIP membership (a VIP membership will cost 100$ more). More detailed information regarding the membership choice is in section vii below. Also, note that you do not need to handle errors in input in this part. For example, similar to PART 1 of Assignment 1, you can assume user always enter valid destination, valid ticket price, valid *"y"* or *"n"* answer. You can also assume the user enter the membership type correctly, for example, *"M"* for a normal membership, and *"V"* for a VIP membership.
vii. When a customer finishes a booking,
    a. If the customer is a new customer, you need to add the information of that customer into your data collection (think/analyse carefully which information you need to add to your data collection). If the customer answers *"n"* for the question of becoming a member, then the customer is just a normal customer. If the customer answers *"y"*, then the program will ask what type of member the customer wants. If the answer is *"M"*, then the customer will become a member (a customer with a normal membership). If the answer is *"V"*, then the customer will become a VIP member (a customer with a VIP membership). Note, the customer will need to pay extra 100$ for becoming a VIP member. Discount is NOT applied on this 100$ membership fee. Again, you can assume the user enter the membership type correctly ("*M*" or "*V*").
    b. If the customer is an existing customer, you need to update the information of that customer into your data collection (think/analyse carefully which information you

need to update). Note, for existing customers, you DO NOT need to ask if they want a membership (normal or VIP membership). This is slightly different compared to the requirements in Assignment 1, so please be careful.

c. Note that, be careful when you add a new customer to your data collection. The ID of a customer must be unique. You can assume the customer names are also unique.
d. After each booking, the number of seats available for the chosen destination will be reduced by the number of tickets in the booking. Note in this part, you do not need to handle error when the number of tickets in the booking is larger than the number of available seats for the destination.

viii. The total cost of a booking can be displayed as a formatted message as below (for existing customers, or, new customers who are normal customers or customers with normal membership):

<customer name > books <quantity> tickets to <destination>.
<customer name> gets a discount of <discount percentage>%.
Unit price: <the price of each ticket> (AUD)
Total price: <the total price> (AUD)

The formatted message is as below for new customers who register to be a VIP member:

<customer name > books <quantity> tickets to <destination>.
<customer name> gets a discount of <discount percentage>%.
Unit price: <the price of each ticket> (AUD)
Membership price: <the price of VIP membership> (AUD)
Total price: <the total price> (AUD)

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

x. In this assignment, you can assume any customer names do not include any digits. And you DO NOT need to handle issues for this requirement.

- PART 2 (Please do not attempt this part before completing PART 1 – 5 marks) -

In this part, you need to handle exceptions compared to PART 1. Now your program can:

1. Display an error message if the destination entered by the user does not exist in the destination list. When this error occurs, the user will be given another chance, until a valid destination is entered.
2. Display an error message if the ticket quantity is 0, negative, not an integer, or larger than the number of available seats of the destination. When this error occurs, the user will be given another chance, until a valid ticket quantity is entered.
3. Display an error message if the answer by the user is not *y* or *n* when asking if the customer wants a membership. When this error occurs, the user will be given another chance, until a valid answer (i.e. *y*, *n*) is entered.
4. Display an error message if the answer by the user is not *M* or *V* when asking the membership type. When this error occurs, the user will be given another chance, until a valid answer (i.e. *M*, *V*) is entered.

5. Display an error message when reading data from any file, e.g. *customers.txt*, *destinations.txt*, *services.txt*. When this error occurs, the program will display a message indicating something wrong with the files, and then exit.

Besides, in this part, your menu program has three more main features: (1) add extra service/bundle to the *Book a new trip* option, (2) automatically load the services/bundles when the program starts, and (3) a new *Display the current service/bundle list* option, and (4) support both names and IDs of the destinations and services. These features might be challenging. Details about these features are as below.

Firstly, in the *Book a new trip* option: now your program will also display a message asking if customer wants to order extra service/bundle (details about service/bundle are in the next paragraph). If yes, display a message asking which extra service/bundle the customer wants to order. For each booking, the customer can only order one extra service/bundle. Note:

a. The total price is the tickets price and the extra service price (and the membership price if necessary).
b. The discount rates of members and VIP members are only applied to the ticket price.
c. Only customers with membership (normal members or VIP members) can order free extra services (i.e. services with price of 0). A message will be displayed if a normal customer (customer without membership) orders a free extra service.
d. Any customers can order any bundle (including bundle with free services)
e. Display an error message if the answer by the user is not *y* or *n* when asking if customer wants to order extra service. When this error occurs, the user will be given another chance, until a valid answer (i.e. *y*, *n*) is entered. You can assume the user always enters correct extra service name (*Internet, Snack, Drink, Entertainment, SeatSelection,* etc.).
f. Modify the cost formatted message (in PART 1) to be as follows (note new customers with VIP membership will have another line: Membership price: <the price of VIP membership> (AUD) as specified in PART 1).

```
<customer name > books <quantity> tickets to <destination>.
<customer name> gets a discount of <discount percentage>%.
Unit price:                    <the price of each ticket> (AUD)
Extra service price:    <the price of the extra service> (AUD)
Total price:                          <the total price> (AUD)
```

To support the above feature (i.e. order extra service/bundle), you need to add two more classes to your program and modify an existing class (note that you can modify more classes if you'd like).

7. Class Service

This class is to keep track of information of different extra services (*Internet, Snack, Drink, Entertainment, SeatSelection,* etc.) that the travel agency supports. This class supports the following information:

- **ID**: A unique identifier for the service (e.g. S1, S2, etc.)
- **Name**: The name of the service (e.g. *Internet, Snack, Drink, Entertainment*, etc.). You can assume the names of the services are unique. And you can assume the names can not include any digit.
- **Price**: The price of the service

The ID of a service always starts with letter "S". All the IDs and names are unique. You need to define the appropriate variables and methods to support the class Service. Note that the

price of a service may be changed by the user. You are allowed to add extra attributes/methods if these attributes/methods make your program to be more efficient.

8. Class Bundle

This is a special kind of service. It means multiple services can be offered together as one service. For example, a *Business* bundle consists of *Entertainment, LoungeAccess, Internet, Snack,* and *Drink*. You can assume all parts of a bundle are existing services in the system (i.e. the services are combined together to make a bundle).

The price of a bundle is 80% of the total price of all individual services. For example, if *Entertainment* costs 5$, *LoungeAccess* costs 10$, *Internet* costs 0$, *Snack* costs 10$, *Drink* costs 5$, then the price of the *Business* bundle is 80% x (5 + 10 + 0 + 10 + 5) = 24$.

Each bundle has a unique **ID** and **name** (as with Service). You need to define the appropriate variables and methods to support the class **Bundle**.

Also, the **class Records** will store more information:

- **a list of existing services**
- an additional method named **readService**, which can read a comma separated file called *services.txt* and add services to the service list. See an example of the *services.txt* file as below.

```
S1, Internet, 0
S2, Entertainment, 5
S3, SeatSelection, 5
S4, TripCancellation, 10
S5, LoungeAccess, 10
S6, Snack, 10
S7, Drink, 5
B8, StarterMax, S1, S2, S3, S4
B9, StarterPlus, S3, S6, S7
B10, Business, S2, S5, S1, S6, S7
B11, Starter, S3, S4
```

  In this file, the services are always in this format: *ID, name* and the *price of the service*. On the other hand, the data of a bundle consists of all the services in the bundle. For example, the bundle *StarterMax* consists the services *S1, S2, S3, S4*. The ID of a service always start with the letter "***S***" whilst the ID of a bundle always starts with the letter "***D***". The IDs/ names of the services and bundles are all unique. The data format of a bundle is different compared to a service. You can assume all the services in a bundle are unique (no duplicates) and existing services. You can assume bundles are always stored at the end of a file, after all normal services. You can again assume no errors in the file *services.txt.*

- Additional methods named **findService** and **listServices** to find a service (including bundles) and list all existing services (including bundles). The specifications are same as the **findDestination** and **listDestinations** methods.

Secondly, when your program starts, it looks for the files *services.txt* in the local directory (the directory that stores the .py file). If found, the data will be read into the program accordingly, the program will then display a menu. If any file is missing, the program will quit gracefully with an error message indicating the files are missing.

Thirdly, your menu program should now have an option *Display the current service list* to list all the existing services/bundles with the format as in the file *services.txt* (as specified in the Records class).

Finally, your program should support both IDs and names of the destinations and services for any functions with destination names and service names. For example, instead of entering the names like in PART 1, now, the user can enter the IDs.

- PART 3 (Please do not attempt this part before completing PART 2 – 3 marks) -

In this part, 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.

Class Booking: This class now supports **date** information, i.e. it now has an attribute name **date** that stores which date and time the booking is made (you can use an external Python module for this feature).

Your program now can:

i. Automatically load previous bookings that stored in a comma separated file named *bookings.txt* that is located in the local directory (same directory with the .py file). Below is an example of the *bookings.txt* file:

```
James, Sydney, 2, None, 0, 01/09/2020 10:10:00
Lily, Adelaide, 1, S6, 0, 05/09/2020 14:00:00
M3, Sydney, 4, Internet, 0.05, 12/10/2020 09:05:00
Harry, D4, 2, Business, 0.125, 20/12/2020 15:20:00
Tom, Sydney, 2, Internet, 0.05, 04/01/2021 09:10:00
C8, Geelong, 1, None, 0, 25/01/2021 20:00:00
Sarah, Adelaide, 2, StarterPlus, 0.15, 01/03/2021 09:45:00
Wilson, Sydney, 2, Entertainment, 0.05, 20/05/2021 16:30:00
```

Each line in the file is a booking. The format is always *customer name, destination, ticket quantity, service/bundle, discount rate* (discount rate applied to the booking), and *date*. If there are no extra services/bundles, then the services/bundles are recorded as *None*. You can assume all the customers in the *bookings.txt* are existing customers (their names are always inside the *customers.txt* file). You can assume all other information (destinations, ticket quantity, service/bundle, discount rate, date) are always valid. All *customers*, *destinations and services* can be referred by their ID or name in the bookings.txt file.

- Errors when loading the *bookings.txt* file should also be handled. When there is any error reading the file, the program will print message saying "*Cannot load the booking file. Run as if there is no booking previously.*" and running as if there is no booking previously.
- Your menu program should have an option *"Adjust the discount rate"* to adjust the discount rates of the members (customers with normal membership). This adjustment will affect all members in all the future bookings. Invalid input (non-number or negative number discount rate) should be handled. The user will be given another chance until the valid input is entered.
- Your menu program should have an option "*Adjust the threshold limit*" to adjust the threshold limit of the VIP members (customers with VIP membership). This adjustment will affect all members with VIP membership in all the future bookings. Invalid input (non-number of 0 or negative threshold) should be handled. The user will be given another chance until the valid input is entered.

- Your menu program should have an option "*Display all the bookings*" to display all the existing booking. The format is same as in the *bookings.txt* file.
- Finally, your menu program should have an option "*Add a new destination*" to add a new destination to your program. The specifications are as in Assignment 1. The default value of the number of tickets available is always 50, for any new destination. You should be careful when generating the ID of a new destination. All the IDs of all the destinations are unique.

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

- PART 4 (Please do not attempt this part before completing PART 3 – 4 marks) -

In this part, 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 can:

- Your program can use command line arguments to accept the four file names (first being the customer file name, second being the destination file name, third being the service file name, and fourth being the booking file name). Note the first three files are mandatory, the fourth file is optional. If no file names are provided, your program will look for *customers.txt, destinations.txt, services.txt* and *bookings.txt* in the local directory. If a wrong number of arguments are provided, the program will display a message indicating the correct usage of arguments and exit.
- In this part, the data in the booking file maybe invalid. If this happens, the particular booking (the corresponding line in the booking file that has invalid data) will be ignored. An error message is displayed to indicate that booking is ignored. Other valid bookings in the same file will be processed as usual.
- The format of the "*Display all the bookings*" option now becomes:

Name	*Destination*	*Ticket*	*Service*	*Date*
<name>	*<destination>*	*<ticket_number>*	*<service>*	*<date-time>*
<name>	*<destination>*	*<ticket_number>*	*<service>*	*<date-time>*
<name>	*<destination>*	*<ticket_number>*	*<service>*	*<date-time>*

- The menu now has an option "*Display all the bookings of a customer*" to reveal all the bookings of a particular customer. The user needs to enter the customer name, and the program will provide the booking history is in the format as above.
- The menu also has an option "*Display the most valuable customer*" to reveal the most valuable customer, that is the customer with the highest money spent to date.
- The menu also has an option "*Display the most popular destination*" to reveal the destination with the highest number of bookings (not based on tickets, but bookings).
- When your program terminates, it will update the all the files (customer, destinations, services, 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 as ProgFunA2_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 external Python libraries allowed in this assignment are sys, datetime, copy, 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, before the code blocks (e.g. functions/methods, loops, if, etc.) and important variable declarations that the comments refer to. Please DO NOT write a separate file for comments (documentation).

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

D - Rubric:

Overall:

Part	Points
Part 1	12
Part 2	5
Part 3	3
Part 4	4

Others (code quality, modularity, comments)	3
Others (weekly submission)	3

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

4. Submission

As mentioned in the Code Requirements, **you must submit only one 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 are required to submit your code every week starting from Week 9 to Week 11, 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 30 lines of code per week). We will be awarded marks for the weekly submissions. If your code in the weekly submissions are related to the assignment and satisfy the condition of at least 30 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). If your code in the weekly submission are 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 particular 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 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.

5. Referencing Guidelines

What: This is an individual assignment, and all submitted contents must be your own. If you have used sources of information other than the contents directly under Canvas/Modules, you must give acknowledgement of the sources, and give references using the IEEE referencing format.

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

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

6. Academic Integrity and Plagiarism (Standard Warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others whilst developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarized, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods.
- Provided a reference list of the publication details so your readers can locate the source if necessary. This includes material taken from the internet sites.

If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviors, including:

- Failure to properly document a source
- Copyright material from the internet of databases
- Collusion between students

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

7. Assessment Declaration:

When you submit work electronically, you agree to the assessment declaration: https://www.rmit.edu.au/students/student-essentials/assessment-and-results/how-to-submit-your-assessments