

# 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 taxi management system as in Assignment 1 using the object-oriented programming (OOP) paradigm. Same as in Assignment 1, the taxi drivers are the ones that use this system to process and print out receipts of the customers' trips. 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). All existing customers are offered discount. 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** which should be an empty super method
- v. A method **display\_info** which should be an empty super method

### 2. Class BasicCustomer

All Basic customers have a flat discount rate when booking a trip if they have made a booking before. The class **BasicCustomer** should have the following components:

- i. An attribute for the **discount rate**, by default it is 10%. Note that all Basic customers have the same discount rate.
- ii. Constructor takes the appropriate parameters/arguments (be careful)
- iii. Appropriate getter methods for the attributes of this class
- iv. A method **get\_discount** which takes the distance fee and returns the discount. For example, this method returns 10 when the discount rate is 10% and the distance fee is 100\$.
- v. A method **display\_info** that prints the values of the **BasicCustomer** attributes.
- vi. A method **set\_discount\_rate** to adjust the discount rate. This method affects all basic customers.

### 3. Class EnterpriseCustomer

Enterprise customers are customers from organizations that have a special deal with the taxi service provider. These customers are offered discount based on two rates: one when the distance fee is smaller than a threshold, and one when the distance fee larger than or equal to a threshold. Each Enterprise customer can have different rates, but the second rate is always 5% more than the first rate. The two rates are 15% and 20% if not specified. The threshold is the same for all Enterprise customers. The default threshold is 100\$.

For example, if the threshold is 100\$, and the two rates are 15% and 20%, then when an Enterprise customer books a trip with the distance fee being 60\$ then the discount is 9\$, and when they book a trip with the distance fee being 120\$ then the discount is 24\$.

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

- i. Appropriate attributes to support the two rates
- ii. Constructor takes the appropriate parameters/arguments (be careful)
- iii. Appropriate getter methods for the attributes of this class
- iv. A method **get\_discount** which takes the distance fee and returns the discount offered.

- v. A method **display\_info** that prints the values of the **EnterpriseCustomer** attributes.
- vi. A method **set\_discount\_rate** to adjust the discount rates of each individual Enterprise customer.
- vii. A method **set\_threshold** to adjust the threshold limit. This affects all Enterprise customers.

## Locations:

### 4. Class Location

This class is to keep track of information on different locations that the taxi service offers. This class supports the following information:

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

## Rates:

### 5. Class Rate

This class is to keep track of information on different rate types that the taxi service offers. This class supports the following information:

- **ID**: a unique identifier of the rate type
- **name**: the rate type (you can assume the rate types are unique and they do not include any digit)
- **price**: the price per km corresponding to the rate type
- A method **display\_info** that prints the values of the **Rate** 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:

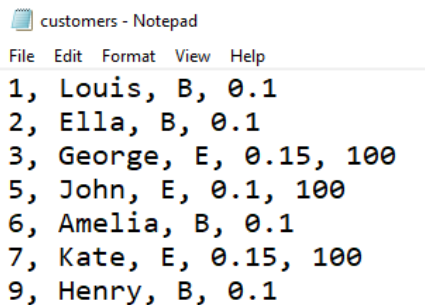
- **customer**: the one who makes the booking (can be a Basic or Enterprise customer). You need to think/analyse carefully if this should be an ID, name, or something else.
- **departure**: the departure the customer chooses. You need to think/analyse carefully if this should be an ID, name, or something else.
- **destination**: the destination the customer chooses. You need to think/analyse carefully if this should be an ID, name, or something else.
- **distance**: the distance of the trip chosen by the customer.
- **rate**: the rate type that the customer chooses. You need to think/analyse carefully if this should be an ID, name, or something else.
- A method **compute\_cost** that returns the distance fee, the basic fee, and the discount. For example, if the booking is of the customer Louis (Basic customer with discount rate 10%), the distance is 12, rate type is standard (price per 1 km: 1.5), then this method will return (18, 4.2, 1.8).
- 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 locations** – you need to think about what you should store in this list (location ID, location name, or something else?)
- **a list of existing rate types that the taxi provider offers** - you need to think about what you should store in this list (rate type ID, rate 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.

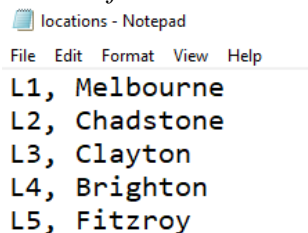


```

1, Louis, B, 0.1
2, Ella, B, 0.1
3, George, E, 0.15, 100
5, John, E, 0.1, 100
6, Amelia, B, 0.1
7, Kate, E, 0.15, 100
9, Henry, B, 0.1
  
```

In this file, the customers are always in this format: *customer\_ID*, *customer\_name*, *customer type*, *discount\_rate* (first discount rate for Enterprise customers), *threshold* (if that is an Enterprise customer). For example, in the 1<sup>st</sup> line, the *customer\_ID* is 1, the *name* is Louis, the *customer type* is B (Basic customer), the *discount rate* is 10%. In the 3<sup>rd</sup> line, the *customer\_ID* is 3, the *name* is George, the *customer type* is E (Enterprise customer), the *discount* is 15% (the first discount rate), and the *threshold* is 100. The IDs should be all 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\_locations**. This method takes in a file name and can read and add the locations stored in that file to the location list of the class. In the sequel, we call this the *location file*. See an example of the location file below.

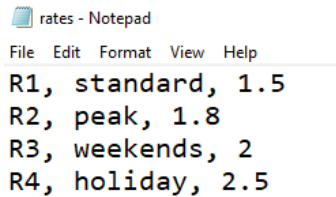


```

L1, Melbourne
L2, Chadstone
L3, Clayton
L4, Brighton
L5, Fitzroy
  
```

In this file, the locations are always in this format: *location\_ID*, *location\_name*. The location ID always starts with the letter "L". The location 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\_rates**. This method takes in a file name and can read and add the rates stored in that file to the rate list of the class. In the sequel, we call this the *rate file*. See an example of the rate file below.



```
rates - Notepad
File Edit Format View Help
R1, standard, 1.5
R2, peak, 1.8
R3, weekends, 2
R4, holiday, 2.5
```

In this file, the data are always in this format: *rate\_ID*, *rate\_name*, *price\_per\_km*. The rate ID always starts with the letter "**R**". 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\_location**, and **find\_rate**. These methods take in a search value (can be either a name or an ID of a customer, location, or rate type), search through the list of customers/locations/rate types and then return the corresponding customer, location, rate type if found or return None if not found.
- This class also has three methods **list\_customers**, **list\_locations**, and **list\_rates**. These three methods can display the information of existing customers, locations, and rate types on screen. The method **list\_customers** will display the customer ID, name, customer type, the discount rate (first discount rate for Enterprise customers), and the threshold (for Enterprise customers). The method **list\_locations** will display the location ID and name. The method **list\_rates** will display the rate type ID, name, and the price per km. Note the three methods can be used to validate the reading from the three .txt files associated with the customers, locations, and rate 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:

- Book a trip*: this option allows users to book a trip for a customer. Detailed requirements for this option are below (Requirements vi-ix).
- Display existing customers*: this option can display all the information of all existing customers: ID, name, customer type, discount rate (first discount rate for Enterprise customers), and threshold (only for Enterprise customers).
- Display existing locations*: this option can display all the information of all existing locations: ID and name.
- Display existing rate types*: this option can display all the information of all rate types: ID, name, and the price per km.
- 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), *locations.txt* (the location file), and *rates.txt* (the rate 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 book a trip 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 names, locations, rate types, and distances.
- viii. When a customer finishes booking a trip,
  - 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).
  - b. If the customer is an existing customer, the program will print out a message showing the customer type (e.g., Basic/Enterprise customer), then proceed with the booking and display the receipt.
- ix. The receipt of a booking can be displayed as a formatted message as below.

```

-----
                        Taxi Receipt
-----
Name:                  <customer_name>
Departure:             <departure>
Destination:           <destination>
Rate:                  <rate> (AUD per km)
Distance:              <distance> (km)
-----
Basic fee:             <basic_fee> (AUD)
Distance fee:          <distance_fee> (AUD)
Discount:              <discount > (AUD)
-----
Total cost:            <total_cost> (AUD)
  
```

- 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 (5 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 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 location entered by the user is not a valid location. When this error occurs, the user will be given another chance, until a valid location is entered. For the departure location, a valid location is a location in the location list. For the destination location, a valid location is a location in the location list and is different from the departure location.
- c. Display an error message if the rate type entered by the user is not a valid rate type (e.g., standard, peak, etc.). When this error occurs, the user will be given another chance, until a valid rate type is entered.



- d. Display an error message if the distance entered is 0, negative, or not a number. When this error occurs, the user will be given another chance, until a valid distance is entered.

Besides, in this level, in the "*Book a trip*" option, your menu program has three more main features: (1) add extra service/package to the *Book a trip* option, (2) automatically load the services/packages when the program starts, and (3) a new option: *Display the existing services/packages*, and (4) support both names and IDs of the customers, locations, rate types and services. These features might be challenging. Details about these features are as below.

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

- e. The discount rates of customers are only applied to the distance fee.
- f. The total cost is equal the basic fee + distance fee – discount + service fee. For example, if the basic fee is 4.2, the distance fee is 15\$, the discount is 1.5\$, and the service fee is 2\$, then the total cost is  $4.2 + 15 - 1.5 + 2 = 19.7\$$ .
- g. Display an error message if the answer by the user is not *y* or *n* when asking if customer wants to order extra service/package. 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*, etc.).
- h. The receipt now includes the information of the service (service/package name and its price)

To support the above feature (i.e., order extra service/package), 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).

## 9. Class Service

This class is to keep track of information of different extra services (*Internet*, *Snack*, *Drink*, *Entertainment*, etc.) that the taxi service provider 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*. The price of a service can be changed. You are allowed to add extra attributes/methods if these attributes/methods make your program to be more efficient.

## 10. Class Package

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

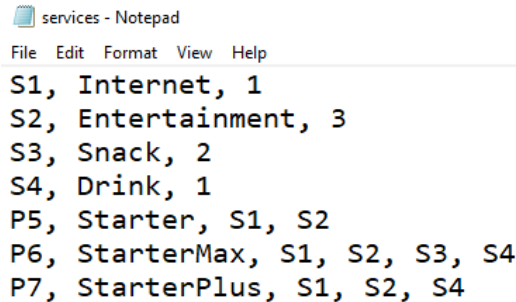
The price of a package is 80% of the total price of all individual component services. For example, if *Entertainment* costs 3\$, *Internet* costs 1\$, *Snack* costs 2\$, *Drink* costs 1\$, then the price of the *StarterMax* package is  $80\% \times (3 + 1 + 2 + 1) = 5.6\$$ . Note that a package needs to have at least 2 services.



Each package has a unique **ID** and **name** (as with service). You need to define the appropriate variables and methods to support the class **Package**.

The **class Records** will then store more information:

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



```
File Edit Format View Help
S1, Internet, 1
S2, Entertainment, 3
S3, Snack, 2
S4, Drink, 1
P5, Starter, S1, S2
P6, StarterMax, S1, S2, S3, S4
P7, StarterPlus, S1, S2, 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 package consists of all the component services of the package. For example, the package *StarterMax* consists the services *S1, S2, S3, S4*. The ID of a service always start with the letter "*S*" whilst the ID of a package always starts with the letter "*P*". The IDs/ names of the services and packages are all unique. The data format of a package is different compared to a service. You can assume all the component services in a package are unique (no duplicates) and existing services. You can assume packages are always stored at the end of a file, after all normal services. You can again assume there are no errors in the file *services.txt*.

- Additional methods named **find\_service** and **list\_services** to find a service (including packages) and list all existing services (including packages). The specifications are same as the **find\_location** and **list\_locations** methods.

Secondly, when your program starts, it also looks for the file *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 (customer, location, rate, service) 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 existing services* to list all the existing services/packages 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 customers, locations, rate types and services for any functions that use these information. For example, instead of entering the customer names, location names, rate type names, service/package names, now, the user can enter the customer IDs, location IDs, rate type IDs and service/package IDs, respectively.

### ----- 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

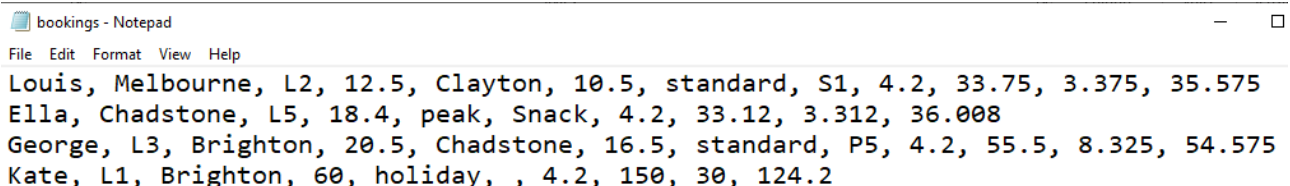
Your program now has the following new features.

- i. In this level, in the *"Book a trip"* option, your program will allow customers to enter multiple destinations 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 new locations"* to add locations. The specification is same as Assignment 1 for this option (requirement 3 in Part 3).
- iii. Your program now has an option *"Adjust the discount rate of all Basic customers"* to adjust the discount rate of all Basic customers. This adjustment will affect all Basic 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 an Enterprise customer"*. The option will ask for the name or ID of an Enterprise customer, then ask for a new first discount rate (e.g., 0.2 which corresponds to 20% discount rate). Invalid customers (non-existent or non-Enterprise customers) needs to be handled, i.e., your program will give the user another chance until a valid Enterprise 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. 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.
- 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 (6 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 load previous bookings via a file which are stored in a comma separated file named *bookings.txt* that is in the same directory with the .py file. This file will be loaded when the program starts (as with the customer, location, rate, and service files). In the sequel, we will call this the *booking file*. Below is an example of the booking file:



```

Louis, Melbourne, L2, 12.5, Clayton, 10.5, standard, S1, 4.2, 33.75, 3.375, 35.575
Ella, Chadstone, L5, 18.4, peak, Snack, 4.2, 33.12, 3.312, 36.008
George, L3, Brighton, 20.5, Chadstone, 16.5, standard, P5, 4.2, 55.5, 8.325, 54.575
Kate, L1, Brighton, 60, holiday, , 4.2, 150, 30, 124.2
  
```

Each line in the booking file is the information of a booking. The format is: *customer\_name/ID, departure\_name/ID, destination1\_name/ID, distance1, destination2\_name/ID, distance2, ..., rate\_name/ID, service\_name/ID, basic\_fee, distance\_fee, discount, 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 locations in the booking file are existing locations (they are in the location file) and are valid. You can assume all rate types in the booking file are existing rate types (they are in the rate file). The services can be normal services or packages and they are all in the service file (when there is no service, there will be an empty field). Customers, locations, rate types,

and services can be referred by IDs or names in this booking file. You can assume all other information (distance, basic fee, distance fee, discount, 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"*.

- ii. Your program now has an option *"Add/update rate types and prices"* option to add/update rate types and prices. The specification is same as Assignment 1 for this option (requirement 2 in Part 2 and requirement 2 in Part 3). Note that for the invalid input handling, your program needs to use custom exceptions.
- iii. Your program now has an option *"Display all bookings"* to display all bookings' information as in the booking file (i.e., the customer's name, locations, distances, rate type, service, basic fee, distance fee, discount, total cost). The printed message is flexible.
- iv. The program now can use command line arguments to accept five file names (the first being the customer file name, the second being the location file name, the third being the rate file name, the fourth being the service file, and the fifth being the booking file). The first four files are mandatory whilst the fifth file (booking) is optional. If no file names are provided, your program will look for *customers.txt*, *locations.txt*, *rates.txt*, *services.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.
- v. The menu now has an option *"Display the most popular customer"* to display the customer with the maximum money (total cost) spent to date and the amount of money they have spent. If there are multiple customers with the same maximum money purchased, you can display only one customer or all customers, it's your choice.
- vi. Your program will now have an option *"Display a customer booking history"*. The option will show a table displaying all the previous bookings of a particular customer. The specification is as in Assignment 1 but note it will include the information about the services/packages.

*This is the booking history of Louis.*

	<i>Booking 1</i>	<i>Booking 2</i>
<i>Departure</i>	<i>Melbourne</i>	<i>Docklands</i>
<i>Destination</i>	<i>Chadstone, Clayton</i>	<i>Clayton</i>
<i>Service</i>	<i>Internet</i>	<i>Snack</i>
<i>Total cost</i>	<i>40.5</i>	<i>35.0</i>

- vii. When your program terminates, it will update the four files: customers, locations, rates, 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	10
CREDIT level	5
DI level	3
HD level	6
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 or 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>