Programming Fundamentals (COSC2531)
Assignment 1

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

1. Overview

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

You should develop this assignment in an iterative fashion (as opposed to completing it in one sitting). You can and should get started now (when this assignment specification is posted on Canvas) as there are concepts from previous lessons that you can employ to do this assignment. If there are questions, you can ask via the lectorial, practical sessions or the Canvas discussion forum (Canvas/Discussions/Discussion on Assignment 1). Note that the **Canvas discussion forum is preferable** as it allows other students to see your questions as well. Also, you should ask questions in a general manner, for example, you should replicate your problem in a different context in isolation before posting, and **you must not post your code on the Canvas discussion forum**.

2. Assessment Criteria

This assignment will determine your ability to:

i. Follow coding, convention and behavioural requirements provided in this document and in the course lessons;
ii. Independently solve a problem by using programming concepts taught over the first several weeks of the course;
iii. Write and debug Python code independently;

iv. Document code;
v. Provide references where due;
vi. Meet deadlines;
vii. Seek clarification from your "supervisor" (instructor) when needed via the Canvas discussion forums; and
viii. Create a program by recalling concepts taught in class, understand, and apply concepts relevant to solution, analyse components of the problem, evaluate different approaches.

3. Learning Outcomes

This assignment is relevant to the following Learning Outcomes:

1. Analyse simple computing problems.
2. Devise suitable algorithmic solutions and code these algorithmic solutions in a computer programming language.
3. Develop maintainable and reusable solutions.

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

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

4. Assessment Details

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

Problem Overview: In this assignment, you are developing a taxi management system. 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.

Requirements: Your code must meet the following **functionalities**, **code**, and **documentation** requirements. Your submission will be graded based on the **rubric** published on Canvas. Please ensure you read all the requirements and the rubric carefully before working on your assignment.

A - Functionalities Requirements:

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

---------- **PART 1 (7 marks)** ----------

In this part, your program can perform some simple interactions with users (i.e., the taxi drivers):

1. Display a message asking the user to enter the customer's name. In this part, you can assume a customer name only consists of alphabet characters.
2. Display a message asking the user to enter the departure location. In this part, you can assume the location to be entered is always a valid location.
3. Display a message asking the user to enter the destination location. In this part, you can assume the location to be entered is always a valid location and it needs to be different compared to the departure location.

4. Display a message asking the distance (in km) between the departure and the destination locations. In this part, you can assume the distance entered is always a positive valid number, e.g., 10.2, 20.5.
5. Display a message asking the user to enter the rate type. There are 4 types of rates: *standard, peak, weekends, holiday*. The prices per 1 km for these types are *1.5, 1.8, 2, 2.5*, respectively. In this part, you can assume the rate type entered by the user is a valid value (within the list of rate types).
6. Calculate the total cost for the customer including the discount. The total cost is equal to the distance fee + basic fee – discount. The basic fee is always 4.2$. The distance fee is computed based on the distance and the price per km, for example, if the distance is 10km, the rate is standard, then the distance fee is 1.5x10 = 15$. For the discount, see No. 7 below.
7. An existing customer will have a discount of 10% over the distance fee (not apply to basic fee). For example, if the distance fee is 15$, then the discount will be 1.5$. There is no discount for new customers.
8. All the booking information will be displayed as a formatted message to the user as follows. Note that the rate, distance, basic fee, distance fee, discount, and total cost are all displayed with two digits after the decimal point.

```
------------------------------------------------------
                    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)
```

9. In the program, you should have some lists (or dictionaries or other data types) to store the names of all customers, the available locations, the available rate types, the prices of those rate types. You can assume the customer names, locations, and the rate types are all unique and case sensitive.
10. When a new customer finished booking a taxi trip, your program will automatically add the customer's name to the customer list. When any customer books a trip, your program will check if they are an existing customer. If yes, the discount fee is applied.
11. Your program needs to be initialized with the following existing customers: *Louis* and *Ella*, and the following locations: *Melbourne, Chadstone, Clayton, Brighton, Fitzroy*.
12. Note: in the requirements No. 9 & 10, we use the term 'list' when describing the customer list, the locations, etc, but you can use other data types to store this information such as dictionaries and other data types. Make sure you think and analyse the requirements in detail so that you can choose the most appropriate/suitable data types.

---------- PART 2 (4 marks, please do not attempt this part before completing PART 1) ----------

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

First, compared to the requirements in PART 1, now your program will be able to handle invalid inputs from users:

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.

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

1. *Book a trip*: this option includes all the requirements from 1 to 12 in PART 1 and the requirements a to d in the first part of PART 2.
2. *Add/update rate types and prices*: this option displays a message asking if users want to add new or update existing rate types, and another message asking users to enter the prices of the entered rate types. The rate types must be entered as a list that separates by commas, e.g., *metropolitan, deluxe, premium*. The prices will also be entered as a list separating by commas, e.g., *2.0, 2.5, 2.2*. If the rate types are new, the rate types and their prices will be added to the data collection of the program. If the rate types are the existing rate types, the newly entered prices will replace existing prices. The order of the prices is the same as the order of the rate types, e.g., in our example, the prices of metropolitan, deluxe, and premium are 2.0, 2.5, and 2.2 respectively. You can assume the rate types are always unique, and each rate type is a single word with no comma but only alphabet characters. You can assume users always enter the correct formats of the rate type and price lists, but note they can enter multiple spaces before or after the commas. In this part, you can assume the prices entered are always valid and positive numbers and has no comma.
3. *Display existing customers*: this option displays on screen all existing customers. The messages are flexible (your choice).
4. *Display existing locations*: this option displays on screen all existing locations. The messages are flexible (your choice).
5. *Display existing rate types*: this option displays all the rate types with their prices. The messages are flexible (your choice), but they should show all the information required.
6. *Exit the program*: this option allows users to exit the program.

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

---------- PART 3 (6 marks, please do not attempt this part before completing PART 2) ----------

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

1. In this part, in the *"Book a trip"* option, your program will allow customers to enter multiple destinations. That is, after a destination and its corresponding distance to the departure location is entered into the system, the user will be asked if another destination is added. If the answer by the user is *y* (meaning yes), then the system will ask for the destination and the distance (in relation to the previous destination). The process will be repeated until the user answers *n* (meaning no). The cost of the trip will be based on the total distance entered. In this option, invalid inputs need to be handled. That is, the destination should be a valid destination and should be different to the departure and the previous entered destination – if not, the user will be given another chance until a valid location is entered. The answer by the user should only be *y* or *n* – if the user enters a value that is different to *y* or *n*, they will be given another chance until a valid answer is entered. The formatted message for the receipt is as follows.

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

2. In this part, your program will check the prices entered in the "*Add/update rate types and prices*" option. If one of them is not a valid number, or negative number, or 0, the program will ask the user to enter the price list again, until a valid list (contain all valid prices) is entered.
3. The menu now has an option *"Add new locations"* to add new locations to the program. This option will accept a list of locations (separating by commas) as the input. For example, users can enter the input: *Geelong, Carlton, Southbank* to add Geelong, Carlton, Southbank to the location list of the program. If a location is new, then it will be added to the program. If a location is an existing location, the program will display a message saying this is an existing location so it will not do anything. You can assume the locations are always unique, and each location is a single word with no space nor commas. You can assume users always enter the

correct formats (i.e., using commas to separate the locations), but note that users can enter multiple spaces before/after the commas.

4. The menu also has an option *"Display the most valuable 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 spent, you can display only one customer or all customers, it's your choice.
5. The menu now has an option *"Display a customer booking history"*. This option will display a message asking the user to enter the name of the customer, and the program will display all the previous bookings of that customer, including the information of the departure and destination locations of their bookings, and the total cost. Invalid customer names will be handled, and the user will be given another chance until a valid customer name is entered. For example, if a customer named Tom booked 2 previous trips, one trip from Melbourne to Chadstone then Clayton with the total cost of 40.5$, and another trip from Docklands to Clayton with the total cost of 35$, then the program will display the formatted message as follows.

This is the booking history of Tom.

	Booking 1	*Booking 2*
Departure	*Melbourne*	*Docklands*
Destination	*Chadstone, Clayton*	*Clayton*
Total cost	*40.5*	*35.0*

B - Code Requirements:

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

Your code needs to be formatted consistently. You must not include any unused/irrelevant code (even inside the comments). What you submitted must be considered as the final product.

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

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

Note that in places where this specification may not tell you how exactly you should implement a certain feature, you need to use your judgment to choose and apply the most appropriate concepts from our course materials. You should follow answers given by your "client" (or "supervisor" or the teaching team) under Canvas/Discussions/Discussion on Assignment 1.

C - Documentation Requirements:

You are required to write comments (documentation) as a part of your code. Writing documentation is a good habit in professional programming. It is particularly useful if the documentation is next to the code segment that it refers to. Note that you don't need to write an essay, i.e., you should keep the documentation succinct.

Your comments (documentation) should be in the same Python file. Please DO NOT write a separate file for comments (documentation).

At the beginning of your Python file, your code must contain the following information:

1. **Your name and student ID.**
2. **The highest part you have attempted.**
3. **Any problems of your code and requirements that you have not met.** For example, scenarios that might cause the program to crash or behave abnormally, the requirements your program do not satisfy. Note, you do no need to handle errors that are not covered in the course.

Besides, the comments (documentation) in this assignment should serve the following purposes:

- Explain your code in a precise but succinct manner. It should include a brief analysis of your approaches instead of simply translating the Python code to English. For example, you can comment on why you introduce a particular function/method, why you choose to use a while loop instead of other loops, why you choose a particular data type to store the data information. These comments can be placed before the code blocks (e.g., functions/methods, loops, if) and important variable declarations that the comments refer to.
- Document some analysis/reflection as a part of your code. Here, you need to write some paragraphs (could be placed at the end or at the beginning of your code) to explain in detail your design process, e.g., how you came up with the design of the program, how you started writing the code after the design process, the challenges you met during the code development.
- Document the references, i.e., any sources of information (e.g., websites, tools) you used other than the course contents directly under Canvas/Modules, you must give acknowledgement of the sources, explaining how you use the sources in this assignment. More detailed information regarding the references can be found in Section 7.

D - Rubric:

Overall:

Part	**Points**
Part 1	7
Part 2	4
Part 3	6
Others (code quality, modularity, comments/reflection)	3

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

5. Example Program

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

5.1. PART 1

As an example, this is how the output screen of our sample program looks like for PART 1. In this example, the customer has the name *Ella* (an existing customer), book a taxi trip from *Melbourne* to *Chadstone* with the distance being *20* (km). The rate is *standard*. Note that in this test, we use the values *Ella, Melbourne, Chadstone, 20, standard* as examples only. You should test your program with different test cases, e.g., when a new customer books a trip, to make sure your program satisfies the requirements in Section 4.

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

Enter the departure location [enter a valid location only, e.g. Melbourne]:
Melbourne

Enter the destination location [enter a valid location only, e.g. Chadstone]:
Chadstone

Enter the distance (km) [enter a positive number only, e.g. 12.5, 6.8]:
20

Enter the rate type [enter a valid type only, e.g. standard, peak, weekends, holiday]:
standard

---------------------------------------
              Taxi Receipt
---------------------------------------
Name:                Ella
Departure:           Melbourne
Destination:         Chadstone
Rate:                1.50 (AUD per km)
Distance:            20.00 (km)
---------------------------------------
Basic fee:           4.20 (AUD)
Distance fee:        30.00 (AUD)
Discount fee:        3.00 (AUD)
---------------------------------------
Total cost:          31.20 (AUD)
```

5.2. PART 2

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

```
Welcome to the RMIT taxi management system!

############################################################
You can choose from the following options:
1: Book a trip
2: Add/update rate types and prices
3: Display existing customers
4: Display existing locations
5: Display existing rate types
0: Exit the program
############################################################

Choose one option: 1
```

When the user (the taxi driver) enters an option, the corresponding functionality will appear. For example, if the user chooses option 1, which is to book a trip, then the output screen of our sample program is as follows.

```
Choose one option: 1

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

Enter the departure location [enter a valid location only, e.g. Melbourne]:
test
The departure location is not valid. Please enter a valid location.

Enter the departure location [enter a valid location only, e.g. Melbourne]:
Melbourne

Enter the destination location [enter a valid location only, e.g. Chadstone]:
Melbourne
The destination location is not valid. Please enter a valid location.

Enter the destination location [enter a valid location only, e.g. Chadstone]:
Chadstone

Enter the distance (km) [enter a positive number only, e.g. 12.5, 6.8]:
dkaa;d
Distance needs to be a valid number. Please enter a valid distance.

Enter the distance (km) [enter a positive number only, e.g. 12.5, 6.8]:
20

Enter the rate type [enter a valid type only, e.g. standard, peak, weekends, holiday]:
kad;ada
The rate type is not valid. Please enter a valid rate type.

Enter the rate type [enter a valid type only, e.g. standard, peak, weekends, holiday]:
standard

---------------------------------------
              Taxi Receipt
---------------------------------------
Name:                Ella
Departure:           Melbourne
Destination:         Chadstone
Rate:                1.50 (AUD per km)
Distance:            20.00 (km)
---------------------------------------
Basic fee:           4.20 (AUD)
Distance fee:        30.00 (AUD)
Discount fee:        3.00 (AUD)
---------------------------------------
Total cost:          31.20 (AUD)
```

Other requirements in PART 2 (add/update rate types and prices, display existing customers information, display existing locations, display existing rate types) can also be displayed in a similar manner.

5.3. PART 3

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

```
Welcome to the RMIT taxi management system!

############################################################
You can choose from the following options:
1: Book a trip
2: Add/update rate types and prices
3: Display existing customers
4: Display existing locations
5: Display existing rate types
6: Add new locations
7: Display the most valuable customer
8: Display a customer booking history
0: Exit the program
############################################################

Choose one option:
```

This is an example showing the output screen of our sample program when we select option 1 and book a trip with multiple destinations.

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

Enter the departure location [enter a valid location only, e.g. Melbourne]:
Melbourne

Enter the destination location [enter a valid location only, e.g. Chadstone]:
Chadstone

Enter the distance (km) [enter a positive number only, e.g. 12.5, 6.8]:
20

Do you want to add another destination?
y

Enter the destination location [enter a valid location only, e.g. Chadstone]:
Clayton

Enter the distance (km) [enter a positive number only, e.g. 12.5, 6.8]:
15.2

Do you want to add another destination?
n

Enter the rate type [enter a valid type only, e.g. standard, peak, weekends, holiday]:
standard

---------------------------------------
              Taxi Receipt
---------------------------------------
Name:                Ella
Departure:           Melbourne
Destination:         Chadstone
Distance:            20.00 (km)
Destination:         Chadstone
Distance:            15.20 (km)
Rate:                1.50 (AUD per km)
Total Distance:      35.20 (km)
---------------------------------------
Basic fee:           4.20 (AUD)
Distance fee:        52.80 (AUD)
Discount fee:        5.28 (AUD)
---------------------------------------
Total cost:          51.72 (AUD)
```

6. Submission

As mentioned in the Code Requirements, **you must submit only one file named ProgFunA1_<Your Student ID>.py** via Canvas/Assignments/Assignment 1. It is your responsibility to correctly submit your file. Please verify that your submission is correctly submitted by downloading what you have submitted to see if the file includes the correct contents. The final .py file submitted is the one that will be marked.

Late Submission

All assignments will be marked as if submitted on time. Late submissions of assignments without special consideration or extension will be automatically penalised at a rate of 10% of the total marks available per day (or part of a day) late. For example, if an assignment is worth 20 marks and it is submitted 1 day late, a penalty of 10% or 2 marks will apply. This will be deducted from the assessed mark.

Special Consideration

If you are applying for extensions for your assessment within five working days after the original assessment date or due date has passed, or if you are seeking extension for more than seven days, you will have to apply for Special Consideration, unless there are special instructions on your Equitable Learning Plan.

In most cases you can apply for special consideration online here. For more information on special consideration, visit the university website on special consideration here.

7. Referencing Guidelines

What: This is an individual assignment, and all submitted contents must be your own. If you have used any sources of information (e.g., websites, tools) other than the course contents directly under Canvas/Modules, you must give acknowledgement of the sources, explaining in detail how you use the sources in this assignment, and give references using the IEEE referencing format.

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

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

8. Academic Integrity and Plagiarism (Standard Warning)

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

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

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

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

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

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

9. Assessment Declaration:

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