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 for up to 5 days late, 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 Assessment 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 retail management system for a department store. The cashiers or the store managers from the department store are the ones that use this system to process customers' purchases. 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 (6 marks)** ----------

In this part, your program can perform some simple interactions with users (i.e., the cashiers or the store managers):

1. Display a message asking the user to enter the name of the customer.
2. Display a message asking the user to enter the name of the product the customer chooses. In this part, you can assume the product to be entered is always a valid product.

3. Display a message asking the quantity of the product ordered by the customer that was entered earlier. In this part, you can assume the quantity to be entered is always a positive integer, e.g., 1, 2, 3 …
4. Calculate the total cost for the customer including the discount (see No. 5 below).
5. For customers with membership, 5% discount will apply. No discount for customers without membership.
6. The total cost will be displayed as a formatted message to the user, e.g.

   <customer name > purchases <quantity> x <product>.
   Unit price: <the price of the product> (AUD)
   <customer name> gets a discount of <discount percentage>%.
   Total price: <the total price> (AUD)

7. In the program, you should have some lists (or dictionaries or other data types) to store the names of all customers, the names of customers with membership, the available products, the prices of those products. You can assume the customer names and the product names are all unique and case sensitive.
8. When a new customer purchases an item, your program will add the customer's name to the customer list. Also, when any customer purchases an item, check if they have a membership; if they currently don't have a membership, display a message asking if they want to have a membership. If yes, then your program will add the customer's name to the customers with membership list. The discount is applied immediately after the customer is added to the customers with membership list. In this part, you can assume the answer entered is always either *y* or *n* (meaning yes or no, respectively).
9. Note: in the requirements No. 7 & 8, we use the term 'list' when describing the customer list, customer with membership list, 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 (6 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 can:

1. Display an error message if the product entered by the user does not exist in the product list. When this error occurs, the user will be given another chance, until a valid product is entered.
2. Display an error message if the product quantity is 0, negative or not an integer. When this error occurs, the user will be given another chance, until a valid positive integer 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.

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: place an order, add/update products and prices, display all existing customers, display all customers with membership, display all existing products with their prices, 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. *Place an order*: this option includes all the requirements from 1 to 8 in PART 1 and the requirements 1 to 3 in the first part of PART 2.
2. *Add/update products and prices*: this option displays a message asking users to add new products or update products, and another message asking users to enter the prices of the entered products. The products must be entered as a list that separates by commas, e.g., *shirt, dress, oven, towel, kettle*. The prices will also be entered as a list separating by commas, e.g., *50.0, 60.0, 300.0, 20.0*. If the products are new, the products and their prices will be added to the data collection of the program. If the products are existing products, the newly entered prices will replace existing prices. The order of the prices is the same as the order of the products, e.g., in our example, the prices of shirt, dress, oven, and towel are 50, 60, 300, 20 respectively. A product may have no price, e.g., in our example, kettle does not have any price. You can assume the product names are always unique, and each product name is a single word with no comma but only alphanumeric characters. You can assume users always enter the correct formats of the product and price lists. In this part, you can assume the prices entered are always valid and positive numbers and has no comma. And in this part, you can also assume users never purchase products with no price.
3. *Display existing customers*: this option displays the names of all existing customers on screen.
4. *Display existing customers with membership*: this option displays the names of all customers with membership on screen.
5. *Display existing products*: this option displays all the products on screen, with their prices. Products with no price will also be displayed.
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 (5 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 challenging.

1. In this part, in the "*Place an order*" option, your program will allow customers to purchase multiple items in one order. That is, after each item is entered into the system, the user will be asked if another item is being purchased, if the answer by the user is *y* (meaning yes), then the system will ask for the product name and the quantity. The process will be repeated until the user answers *n* (meaning no). Note that, in this requirement, if the answer by the user is not *y* or *n* then requirement 3 in PART 2 will be activated. The formatted message for the order cost will be as follows.

<customer name > purchases <quantity> x <product>.
Unit price: <the price of the product> (AUD)
<customer name > purchases <quantity> x <product>.
Unit price: <the price of the product> (AUD)
……
<customer name> gets a discount of <discount percentage>%.
Total price: <the total price> (AUD)

2. Besides, in the "*Place an order*" option, a product with no price cannot be sold and a message will be shown if customers attempt to order such products. When this error occurs, your program will skip this product, and continue to process the next product. If the customer

decides to not order any product, then your program will directly go back to the main menu, without adding the customer's name to the data collection (for new customer) or checking the customer membership.

3. In this part, your program will check the prices entered in the "*Add/update products and prices"* option. If they are not valid numbers, or negative numbers, or 0, they will be treated as no price.
4. The menu also has an option "*Reveal the most valuable customer*" to display the customer with maximum total money spent to date and the total money they've spent. If there are multiple customers with the same maximum money spent, you can just display only one customer (or all customers, it's your choice).
5. The menu now has an option "*Display a customer order history*". The option will display a message asking users to enter the name of the customer, and the program will display all the previous orders of that customer, including the products they purchased and the corresponding quantities. For example, if a customer named Lily purchased 2 times previously, the first time with 2 dresses, 1 shirt, and the second time with 1 oven, 1 shirt and 2 towels, then the program will display the formatted message as follows.

This is the order history of Lily.

	dress	*shirt*	*oven*	*towel*
Purchase 1	*2*	*1*		
Purchase 2		*1*	*1*	*2*

Note that for this option, if the user enters non-existing customer, then your program needs to give the user another chance, until an existing customer is entered. Also, you can assume that in each order (via the "*Place an order*" option), the product to be entered each time is different, e.g., in one order, a user can buy dress, then shirt, then oven, but the user will never buy dress, then oven, then dress again.

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 Assessment 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, 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	6
Part 2	6
Part 3	5
Others (code quality, modularity, comments)	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 *Huong*, purchasing an *oven* with the quantity of 1, no existing membership and then choose *y* (yes) for the membership registration option. Note that in this sample program, we use the values *Huong, shirt, towel, oven, 1, 2, 3, etc* as examples only. In your program, you can, and you should use different values. Also, you should test your program with different test cases, e.g., customers choose *n* (no) for the membership registration options, to make sure your program satisfy the requirements in Section 4.

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

Enter the product [enter a valid product only, e.g. shirt, towel, oven, kettle]:
oven

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

Customer does not have a membership. Does the customer want to have a membership [enter y or n]?
y

---------------------------------------
Huong purchases 1 x oven.
Unit price:         300.0 (AUD)
Huong gets a discount of 5.0 %.
Total price:        285.0 (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 retail management system!

############################################################
You can choose from the following options:
1: Place an order
2: Add/update products and prices
3: Display existing customers
4: Display existing customers with membership
5: Display existing products
0: Exit the program
############################################################

Choose one option:
```

When the user (a cashier or the store manager at the department store) enters an option, the corresponding functionality will appear. For example, if the user chooses option 1, which is to make an order, then the output screen of our sample program is as follows.

```
Welcome to the RMIT retail management system!

############################################################
You can choose from the following options:
1: Place an order
2: Add/update products and prices
3: Display existing customers
4: Display existing customers with membership
5: Display existing products
0: Exit the program
############################################################

Choose one option: 1

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

Enter the product [enter a valid product only, e.g. shirt, towel, oven, kettle]:
oven

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

Customer does not have a membership. Does the customer want to have a membership [enter y or n]?
n

---------------------------------------
Huong purchases 1 x oven.
Unit price:         300.0 (AUD)
Huong gets a discount of 0 %.
Total price:        300.0 (AUD)
---------------------------------------

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

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

5.2. 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 retail management system!

############################################################
You can choose from the following options:
1: Place an order
2: Add/update products and prices
3: Display existing customers
4: Display existing customers with membership
5: Display existing products
6: Reveal the most valuable customer
7: Display a customer order history
0: Exit the program
############################################################

Choose one option:
```

And this is an example showing the output screen of our sample program when we select option 7, which is to display a customer order history. Other options can also be displayed in a similar manner.

```
############################################################
You can choose from the following options:
1: Place an order
2: Add/update products and prices
3: Display existing customers
4: Display existing customers with membership
5: Display existing products
6: Reveal the most valuable customer
7: Display a customer order history
0: Exit the program
############################################################

Choose one option: 7

Please enter a customer name for checking order history:
Huong

This is the order history of Huong.
                shirt     towel     oven      kettle
Purchase 1       2                  1
Purchase 2       1         1
Purchase 3                                     1
```

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. Assignments will not be accepted if more than five days late, unless special consideration or an extension of time has been approved.

Special Consideration

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

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

7. Referencing Guidelines

What: This is an individual assignment, and all submitted contents must be your own. If you have used 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.

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