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 management system for a pharmacy. The pharmacists are the ones that use this system to process and print out receipts of the 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 pharmacists):

1. Display a message asking the user to enter the customer’s name. In this part, you can assume the customer name to be entered only consists of alphabet characters.
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. The total cost is equal to the product's unit price multiplying with the product quantity. For example, if the product is *vitaminC*, the unit price of this product is *12$*, and if the product quantity is *3*, then the total cost is *36$*.
5. Calculate the reward points earned by the purchase. For each 1$, there will be 1 reward point. The reward points will be rounded. For example, if the purchase is 35.5$, the corresponding reward point is 36. If the purchase is 35.4$, the corresponding reward point is 35.
6. All the purchase information will be displayed as a formatted message to the user as follows. Note that the product's unit price and total cost are all displayed with two digits after the decimal point.

```
---------------------------------------------------------
                     Receipt
---------------------------------------------------------
Name:               <customer_name>
Product:            <product_name>
Unit Price:         <price> (AUD)
Quantity:           <quantity>
---------------------------------------------------------
Total cost:         <total_cost> (AUD)
Earned reward:      <reward_points>
```

7. In the program, you should have some lists (or dictionaries or other data types) to store the names of all customers, the accumulated reward points of the customers, the available products, the unit prices of the products. You can assume the customer names and the product names are all unique and case sensitive.
8. When a new customer finishes a purchase, your program will automatically add the customer's name to the customer list and the earned reward points to the customer profile. When an existing customer finishes a purchase, your program will add the earned reward points to the customer profile.
9. Your program needs to be initialized with the following existing customers: *Kate* and *Tom*, with the reward points being 20 and 32, respectively. Your program will also be initialized with the following products: *vitaminC*, *vitaminE*, *coldTablet*, *vaccine*, and *fragrance* with the corresponding prices: *12.0*, *14.5*, *6.4*, *32.6*, and *25.0*.
10. Note: in the requirements No. 7, we use the term 'list' when describing the customer list, the product 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 (5 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 have the following features:

i. Each product can belong to one of the two categories: can be purchased without a doctor's prescription or can only be purchased with a doctor's prescription. During the purchase, if the product requires a doctor's prescription, the program will ask the customer if they have a doctor's prescription for the product. If the customer answers *n* (meaning no), then the program will display a message saying that this product can't be purchased and finish the purchase. If the customer answers *y* (meaning yes), then the program will proceed as usual as in PART 1. Note that among all the initial products in PART 1 (bullet 9), only *vaccine* requires a doctor's prescription.

ii. 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 product entered by the user is not a valid product. When this error occurs, the user will be given another chance, until a valid product is entered.
   c. Display an error message if the quantity entered is 0, negative, or not an integer. When this error occurs, the user will be given another chance, until a valid quantity is entered.
   d. Display an error message if the answer by the user is not *y* or *n* when asking if the customer has a doctor's prescription. 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: make a purchase, add/update information of a product, display existing customers, display existing products, 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. *Make a purchase*: this option includes all the requirements from 1 to 10 in PART 1 and the requirements i) to ii) in the first part of PART 2.
2. *Add/update information of a product*: this option displays a message asking the user for the information of a product (name, price, doctor's prescription requirement) to be added or updated. The product name, price and doctor's prescription requirement will be separated by white spaces and must be entered with the following format: *product price dr_prescription*. For example, the user can enter *toothpaste 5.2 n* to add/update the product *toothpaste* with the price *5.2* and the requirement for the doctor's prescription is *no*. If the product is an existing product, the newly entered price will replace the existing price, and the new requirement of a doctor's prescription will replace the existing one. If the product is new, then it, its price and the requirement of a doctor's prescription will be added to the data collection of the program. You can assume users always enter the correct formats of the product, price, and the requirement of the doctor's prescription, but note they can enter multiple white spaces. In this part, you can assume the *price* entered is always valid and is a positive number; you can also assume the value of *dr_prescription* entered is always valid and is either *y* or *n* (meaning yes or no). In this part, you can assume the user will only enter one product.
3. *Display existing customers*: this option displays on screen all existing customers and their accumulated reward points. The messages are flexible (your choice).
4. *Display existing products*: this option displays on screen all existing products with their prices, and information whether they need a doctor's prescription. The messages are flexible (your choice).
5. *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 *"Make a purchase"* option, your program will allow customers to enter multiple products and quantities. For the product names, this information will be passed via a list separating by commas with the format: *product_1*, *product_2*, ... For the quantities, this information will be passed via a list separating by commas with the format *quantity_1, quantity_2,* ... For example, if the customers want to buy 2 vitaminC, 1 toothpaste and 6 coldTablet, they will enter the lists: *vitaminC, toothpaste, coldTablet* and *2, 1, 6*. The program should handle invalid products and invalid quantities. If there exists an invalid product or quantity, the program will ask the users to enter the corresponding list again until a valid list is entered. The program should also handle the case when some products require a doctor's prescription. In that case, if the customer doesn't have a doctor prescription, then that product will not be able to purchase (other products can still be purchased as usual). You can assume one doctor's prescription is enough for multiple products in the list. The formatted message for the receipt is as follows.

```
---------------------------------------------------------
                    Receipt
---------------------------------------------------------
Name:                 <customer_name>
Product:              <product_name>
Unit Price:           <price>
Quantity:             <quantity>
Product:              <product_name>
Unit Price:           <price>
Quantity:             <quantity>
                ......
---------------------------------------------------------
Total cost:           <total_cost> (AUD)
Earned reward:        <reward_points>
```

2. Furthermore, in this part, in the option *"Make a purchase"*, your program will check the current reward points of a customer before finishing a purchase, if they are larger than 100, then the program can deduct some amount of money (corresponding to the reward points) from the total cost and update the remaining reward points of the customer. Specifically, every 100 reward points can be converted to 10$ and be deducted from the purchase. Note the earned reward points from the purchase are still based on the total cost before the deduction. For example, if the total cost of the purchase is 45$, the customer currently has 120 reward points (before the purchase), then the total cost will be 35$ (as the customer can convert 100 reward points to 10$), the earned reward points of this purchase are still 45, and the customer will now have 20 + 45 = 65 reward points after the purchase.
3. In this part, the option "*Add/update information of a product*" will become "*Add/update information of products*". This means, this option now can take a list of multiple products and prices separating by white spaces and commas. The format of the list is as following: *product_1 price_1 dr_prescription_1, product_2 price_2 dr_prescription_2, ...* For example, the user can enter *toothpaste 5.2 n, shampoo 8.2 n* to add/update the products *toothpaste* and *shampoo*. You can assume the users always enter the list in the right format, but there could be multiple spaces around the colon and commas. In this part, your program will check the prices entered in this option, if one of them is not a valid number, or negative number, or 0, the program will ask the user to enter the whole list again, until a valid list (contain all valid prices) is entered. The same with the doctor's description requirement, if the list contains an invalid value of

dr_prescription, the program will ask the user to enter the whole list again, until a valid list is entered.

4. The menu now has an option *"Display a customer order 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 orders of that customer, including the information of the products and quantities of their orders, the total cost, and the earned rewards. 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 made 3 previous purchases, one order with 1 *vitaminC*, one order with 1 *fragrance* and 2 *vitaminE*, and one order with 3 *coldTablet* and 1 *vitaminC*, then the program will display the formatted message as follows.

```
This is the order history of Tom.
                Products                      Total Cost     Earned Rewards
Order 1         1 x vitaminC                  12.0           12
Order 2         1 x fragrance, 2 x vitaminE   54.0           54
Order 3         3 x coldTablet, 1 x vitaminC  31.2           31
```

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 in detail 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	6
Part 2	5
Part 3	6
Others (code quality, modularity, comments/reflection)	3

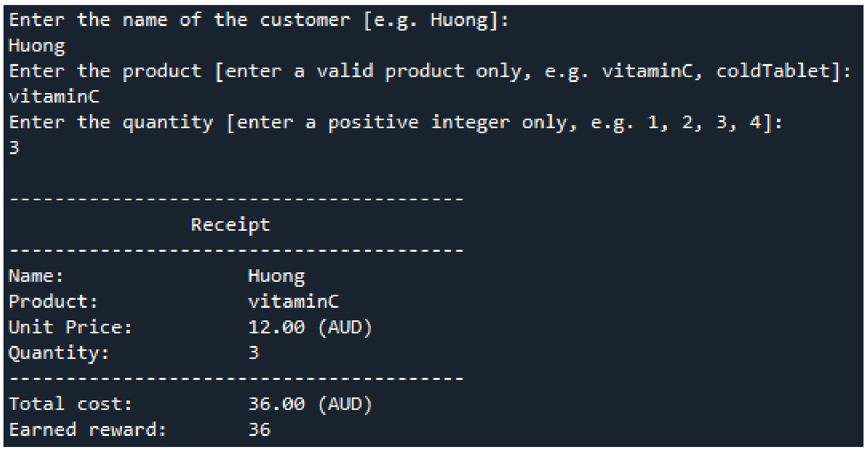
More details of the rubric of this assignment can be found on Canvas (here). Students are required to look at the rubric to understand how the assignment will be graded.

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

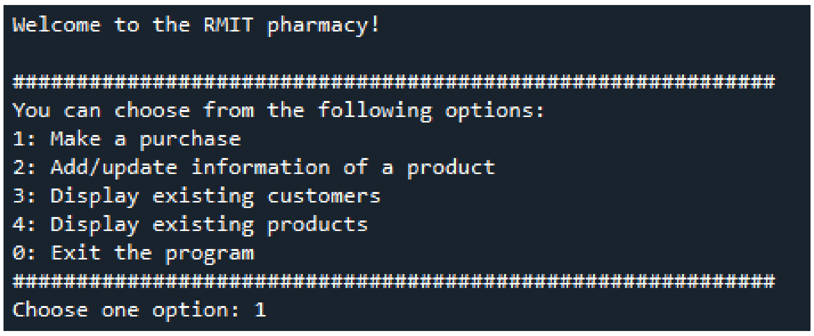
As an example, this is how the output screen of our sample program looks like for PART 1. In this example, the customer has the name *Huong*, making a purchase with the product being *vitaminC* and the quantity being *3*. Note that in this test, we use the values *Huong*, *vitaminC,* and *3* as examples only. You should test your program with different test cases, e.g., when an existing customer (e.g., Kate or Tom) makes a purchase with another product and quantity, to make sure your program satisfies 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. vitaminC, coldTablet]:
vitaminC
Enter the quantity [enter a positive integer only, e.g. 1, 2, 3, 4]:
3

----------------------------------------
                Receipt
----------------------------------------
Name:                Huong
Product:             vitaminC
Unit Price:          12.00 (AUD)
Quantity:            3
----------------------------------------
Total cost:          36.00 (AUD)
Earned reward:       36
```

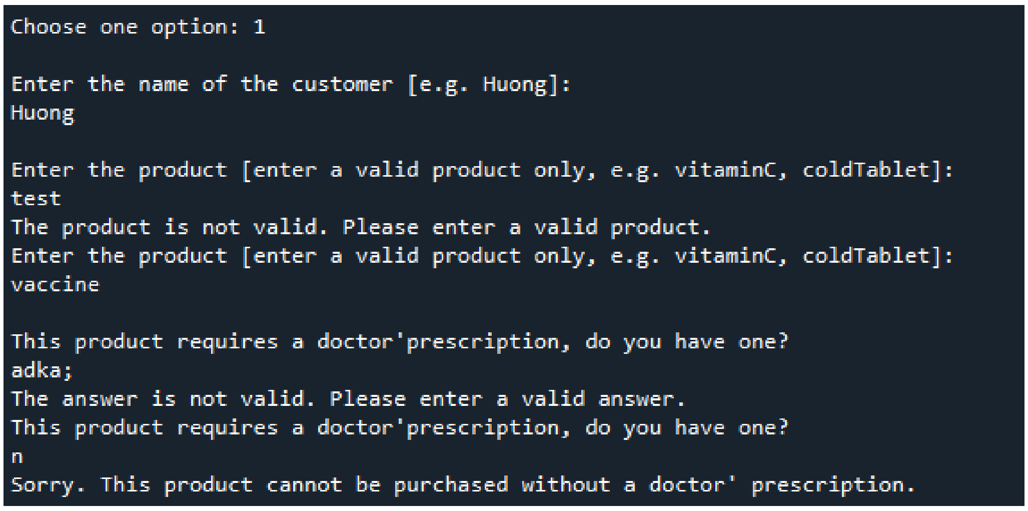
5.2. PART 2

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 pharmacy!

############################################################
You can choose from the following options:
1: Make a purchase
2: Add/update information of a product
3: Display existing customers
4: Display existing products
0: Exit the program
############################################################
Choose one option: 1
```

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

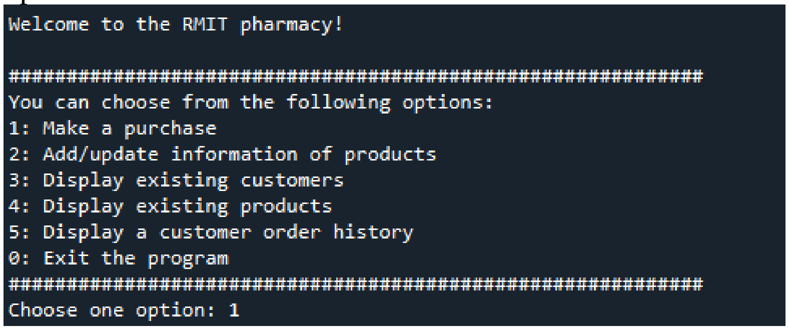


```
Choose one option: 1

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

Enter the product [enter a valid product only, e.g. vitaminC, coldTablet]:
test
The product is not valid. Please enter a valid product.
Enter the product [enter a valid product only, e.g. vitaminC, coldTablet]:
vaccine

This product requires a doctor'prescription, do you have one?
adka;
The answer is not valid. Please enter a valid answer.
This product requires a doctor'prescription, do you have one?
n
Sorry. This product cannot be purchased without a doctor' prescription.
```

Other requirements in PART 2 (add/update information of a product, display existing customers, display existing products) can also be displayed in a similar manner.

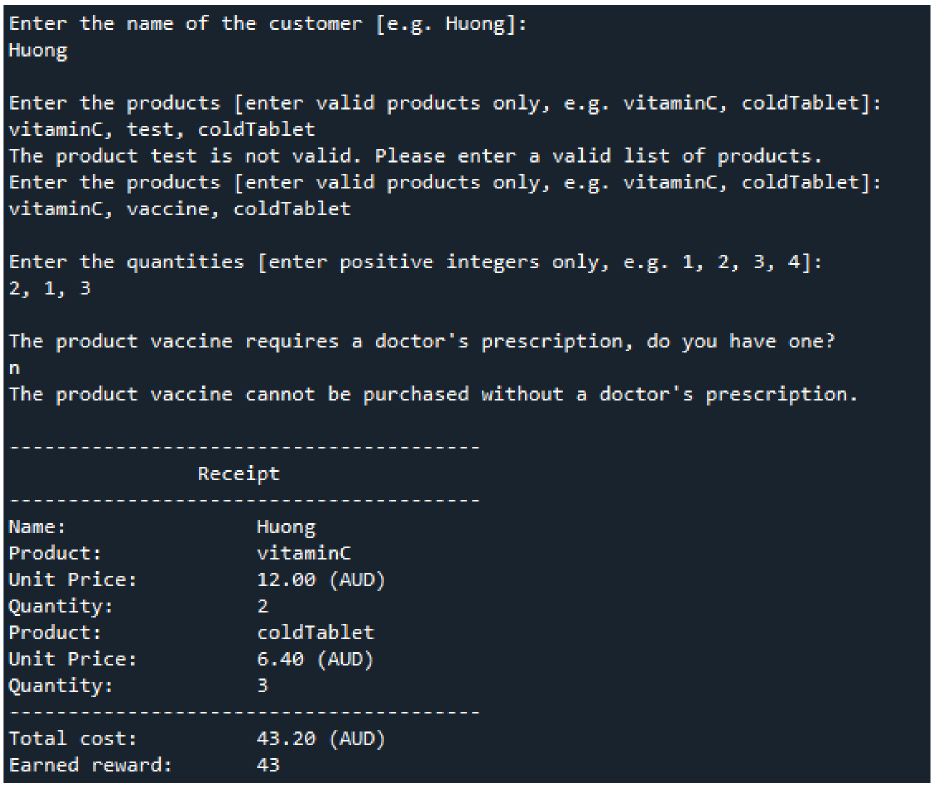
5.3. PART 3

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 pharmacy!

############################################################
You can choose from the following options:
1: Make a purchase
2: Add/update information of products
3: Display existing customers
4: Display existing products
5: Display a customer order history
0: Exit the program
############################################################
Choose one option: 1
```

This is an example showing the output screen of our sample program when we select option 1 and make a purchase with multiple products.



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

Enter the products [enter valid products only, e.g. vitaminC, coldTablet]:
vitaminC, test, coldTablet
The product test is not valid. Please enter a valid list of products.
Enter the products [enter valid products only, e.g. vitaminC, coldTablet]:
vitaminC, vaccine, coldTablet

Enter the quantities [enter positive integers only, e.g. 1, 2, 3, 4]:
2, 1, 3

The product vaccine requires a doctor's prescription, do you have one?
n
The product vaccine cannot be purchased without a doctor's prescription.

----------------------------------------
                Receipt
----------------------------------------
Name:                Huong
Product:             vitaminC
Unit Price:          12.00 (AUD)
Quantity:            2
Product:             coldTablet
Unit Price:          6.40 (AUD)
Quantity:            3
----------------------------------------
Total cost:          43.20 (AUD)
Earned reward:       43
```

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