RMIT UNIVERSITY

Programming Fundamentals (COSC2531) Assignment 2

Assessment Type	**Individual assignment** (no group work). Submit online via Canvas/Assignments/Assignment 2. Marks are awarded per rubric (please see the rubric on Canvas). Clarifications/updates may be made via announcements. Questions can be raised via the lectorial, practical sessions or Canvas discussion forum. Note the Canvas discussion forum is preferable.
Due Date	**End of Week 12** (exact time is shown in Canvas/Assignments/Assignment 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.
Weighting	**30 marks out of 100**

1. Overview

The main objective of this assignment is to familiarize you with object-oriented design and programming. Object-oriented programming helps to solve complex problems by coming up with a number of domain classes and associations. However, identifying meaningful classes and interactions requires a fair amount of design experience. Such experience cannot be gained by classroom-based teaching alone but must be gained through project experience. This assignment is designed to introduce different concepts such as inheritance, method overriding, and polymorphism.

You should develop this assignment in an iterative fashion (as opposed to completing it in one sitting). You can and should get started now (when this assignment specification is posted on Canvas) as there are concepts from previous lessons that you can employ to do this assignment. If there are questions, you can ask via the lectorial, practical sessions or the Canvas discussion forum (Canvas/Discussions/Discussion on Assessment 2). Note that the **Canvas discussion forum is preferable** as it allows other students to see your questions as well. Also, you should ask questions in a general manner, for example, you should replicate your problem in a different context in isolation before posting, and you must not post your code on the Canvas discussion forum.

2. Assessment Criteria

This assignment will determine your ability to:

  i.   Follow coding, convention and behavioural requirements provided in this document and in the course lessons;

ii.   Independently solve a problem by using programming concepts taught over the 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 pharmacy management system as in Assignment 1 using the Object-Oriented Programming (OOP) paradigm. Same as in Assignment 1, 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. 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). Each customer also has some **reward** points. You are required to write the class named **Customer** to support the following:

- i. Attributes **ID**, **name**, and **reward**
- ii. Constructor takes the values of **ID**, **name**, and **reward** as arguments
- iii. Appropriate getter methods for the attributes of this class
- iv. A method **get_reward** which should be an empty super method
- v. A method **get_discount** which should be an empty super method
- vi. A method **update_reward** which should be an empty super method
- vii. A method **display_info** which should be an empty super method

2. Class BasicCustomer

All Basic customers have a flat reward rate when making a purchase (i.e., all Basic customers have the same reward rate). By default, the flat reward rate is 100%. The class BasicCustomer should have the following components:

- i. An attribute for the **reward rate**
- ii. Constructor takes the appropriate parameters/arguments.
- iii. Appropriate getter methods for the attributes of this class
- iv. A method **get_reward** which takes the total cost and returns the reward. Note, the reward is always rounded. For example, when the total cost is 14.8 and the reward rate is 100%, then the method will returns the reward which is $round(14.8 \times 100\%) = 15$.
- v. A method **update_reward** which takes a value and increase the attribute **reward** with that value.
- vi. A method **display_info** that prints the values of the **BasicCustomer** attributes.
- vii. A method **set_reward_rate** to adjust the reward rate. This method affects all Basic customers.

3. Class VIPCustomer

A VIP customer not just receives the reward but also gets a discount for the purchase. The reward rate is the same for all VIP customers and has the default value of 100%. On the other hand, the discount rate is different among VIP customers. If not specified, the discount rate is set as 8%.

Note that the reward is computed after the discount. For example, with a reward rate of 100% and discount rate of 8%, if a VIP customer makes a purchase with the original total cost of 30$, then discount will be 30 x 8% = 2.4$, the total cost after the discount is 27.6$, and thus, the reward is $round(27.6 \times 100\%) = 28$.

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

- i. Appropriate attributes to support the **reward rate** and **discount rate**

ii. Constructor takes the appropriate parameters/arguments
iii. Appropriate getter methods for the attributes of this class
iv. A method **get_discount** which takes the total cost and returns the discount offered
v. A method **get_reward** which takes the total cost and returns the reward
vi. A method **update_reward** which takes a value and increase the attribute **reward** with that value.
vii. A method **display_info** that prints the values of the **VIPCustomer** attributes
viii. A method **set_reward_rate** to adjust the reward rate. This method affects all VIP customers
ix. A method **set_discount_rate** to adjust the discount rate of each individual VIP customer

Products:
4. Class Product
This class is to keep track of information on different products that the pharmacy offers. This class supports the following information:

- **ID**: a unique identifier of the product
- **name**: the name of the product (you can assume the product names are unique and they do not include any digit)
- **price**: the unit price of the product
- A method **display_info** that prints the values of the **Product** attributes
- Extra attributes and methods if you want to define

Orders
5. Class Order
This class is to store a customer's purchase information. This class supports the following information of an order:

- **customer**: the one who makes the purchase (can be a Basic or VIP customer). You need to think/analyse carefully if this should be an ID, name, or something else.
- **product**: the product of the purchase. You need to think/analyse carefully if this should be an ID, name, or something else.
- **quantity**: the quantity of the product ordered by the customer
- A method **compute_cost** that returns the original total cost (the cost before the discount), the discount, the final total cost (the cost after the discount), and the reward. For example, if the original total cost of an order of the customer Tom (VIP customer with reward rate of 100% and discount rate of 8%) is 30, then this method will return (30, 2.4, 27.6, 28).
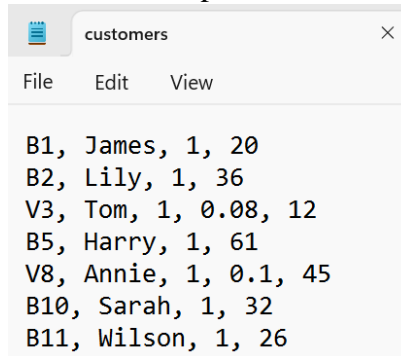- Extra attributes and methods if you want to define.

Records
6. Class Records
This class is the central data repository of your program. It supports the following information:

- **a list of existing customers** – you need to think what you should store in this list (customer ID, customer name, or something else?)
- **a list of existing products** – you need to think about what you should store in this list (product ID, product 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.
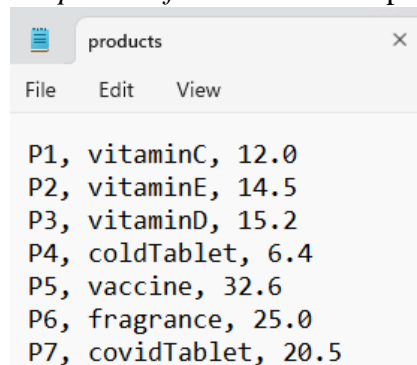
```
customers                                    ×

File    Edit    View

B1, James, 1, 20
B2, Lily, 1, 36
V3, Tom, 1, 0.08, 12
B5, Harry, 1, 61
V8, Annie, 1, 0.1, 45
B10, Sarah, 1, 32
B11, Wilson, 1, 26
```

  In this file, the customers are always in this format: *customer_ID, customer_name, reward rate, discount_rate* (if that is a VIP customer), and *reward*. For example, in the 1st line, the *customer_ID* is B1, the *name* is James, the *reward rate* is 100%, and the *reward point* is 20. In the 3rd line, the *customer_ID* is V3, the *name* is Tom, the *reward rate* is 100%, the *discount rate* is 8%, and the *reward point* is 12. A Basic customer has an ID starting with the letter "*B*" whilst a VIP customer has an ID starting with the letter "*V*". The numbers in the ID after these characters (B, V) are all unique (i.e., 1, 2, 3, 5… are unique; for example, if there is a customer with the ID of B1, there won't be a customer with the ID of V1). 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 reward and discount values are always valid).

- This class has another method named **read_products**. This method takes in a file name and can read and add the products stored in that file to the product list of the class. In the sequel, we call this the *product file*. See an example of the product file below.

```
products                                    ×

File    Edit    View

P1, vitaminC, 12.0
P2, vitaminE, 14.5
P3, vitaminD, 15.2
P4, coldTablet, 6.4
P5, vaccine, 32.6
P6, fragrance, 25.0
P7, covidTablet, 20.5
```

  In this file, the products are always in this format: *product_ID, product_name, unit_price*. The product ID always starts with the letter *"P"*. The product 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 also has two methods **find_customer** and **find_product**. These methods take in a search value (can be either a name or an ID of a customer or product), search through the list of customers/products and then return the corresponding customer/product if found or return None if not found.

- This class also has two methods **list_customers** and **list_products**. These methods can display the information of existing customers and products on screen. The method **list_customers** will display the customer ID, name, the reward rate, the discount rate

(for VIP customers), and the reward. The method **list_products** will display the product ID, product name, and the unit price. Note the two methods can be used to validate the reading from the two .txt files associated with the customers and products.

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

Operations

7. Class Operations

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

i. *Make a purchase*: this option allows users to make a purchase for a customer. Detailed requirements for this option are below (Requirements v-ix).

ii. *Display existing customers*: this option can display all the information of all existing customers: ID, name, reward rate, discount rate (only for VIP customers), and reward.

iii. *Display existing products:* this option can display all the information of all existing products: ID, name, and unit price.

iv. *Exit the program:* this option allows users to exit the program.

Other requirements of the menu program are as follows:

v. When the program starts, it looks for the files *customers.txt* (the customer file) and *products.txt* (the product 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 4 options described above. If any file is missing, the program will quit gracefully with an error message indicating the corresponding file is missing.

vi. Your menu program will allow the user to make a purchase 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 customer names, products, and quantities. Note that for VIP customers, the final total cost = the original cost – discount, and the earned reward points are computed by the final total cost.

vii. When a customer finishes making a purchase,

   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). Any new customer will be registered as a Basic customer.

   b. If the customer is an existing customer, the program will print out a message showing the customer type (e.g., Basic/VIP customer), then proceed with the purchase and display the receipt.

viii. The receipt of a purchase of a Basic customer can be displayed as a formatted message as below.

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

The receipt of a purchase of a VIP customer can be displayed as a formatted message as below.

```
---------------------------------------------------------
                        Receipt
---------------------------------------------------------
Name:                   <customer_name>
Product:                <product_name>
Unit Price:             <price> (AUD)
Quantity:               <quantity>
---------------------------------------------------------
Original cost:          <original_cost> (AUD)
Discount:               <discount> (AUD)
Total cost:             <total_cost> (AUD)
Earned reward:          <reward_points>
```
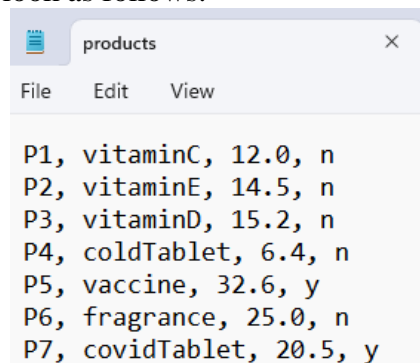
ix.   When a task is accomplished, the menu will appear again for the next task. The program always exits gracefully from the menu.

---------- CREDIT level (4 marks, please do not attempt this level before completing the PASS level) ------------

Operations

i.   At this level, your program will need to have the feature i) specified in PART 2 of Assignment 1 (i.e., each product will 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). In this level, the format of the product file *products.txt* will look as follows:

Each line in the product file has the following format: *product_ID, product_name, unit_price, dr_prescription*. The value of *dr_prescription* could be *n* (meaning no) or *y* (meaning yes). You will need to think of which parts of your program and which classes to be updated for your program to satisfy this feature.

ii.   Besides, 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.
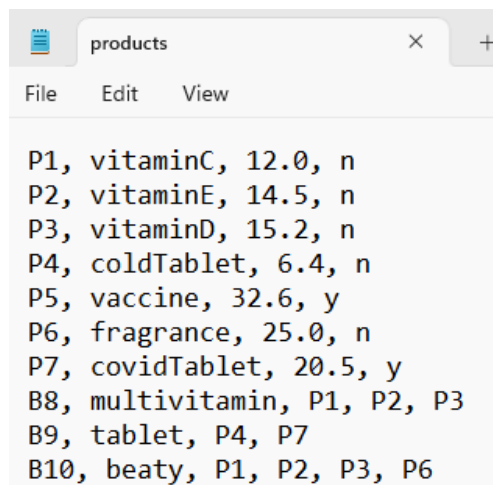
b. Display an error message if the product entered by the user is not a valid product. When this error occurs, the user will be given another chance, until a valid product is entered.

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.

iii. Furthermore, in this level, in the "*Make a purchase*" option, your program will allow ordering a Bundle, which is a special product. It means multiple products can be offered together as one product. For example, a bundle can consist of vitaminC, vitaminE, and vitaminD. You can assume all parts of a bundle are existing products in the system. If in the bundle, there exists a product that requires a doctor's prescription, then the bundle will require a doctor's prescription.

The price of a bundle is 80% of the total price of all individual products. For example, if vitaminC costs 12.0$, vitaminE costs 14.5$, and vitaminD costs 15.2$ then the price of this bundle is 80% x (12.0 + 14.5 + 15.2) = 33.36$.

To support this feature, you need to add one more class to your program.

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

With this modification, the product file *products.txt* at this level now may look like this:

```
products                    ×    +

File    Edit    View

P1, vitaminC, 12.0, n
P2, vitaminE, 14.5, n
P3, vitaminD, 15.2, n
P4, coldTablet, 6.4, n
P5, vaccine, 32.6, y
P6, fragrance, 25.0, n
P7, covidTablet, 20.5, y
B8, multivitamin, P1, P2, P3
B9, tablet, P4, P7
B10, beaty, P1, P2, P3, P6
```

The ID of a Bundle always starts with the letter "*B*". Note that the data format of a bundle is different compared to a normal product; it includes the IDs of the product components. The IDs/names of the products and bundles are all unique. You can assume all the products in a bundle are existing products and unique (no duplicates). You can assume bundles are always stored at the end of a file, after all normal products.

iv. At this level, for the option "*Display existing products*", when displaying bundles, your program will display the bundle's ID, name, the IDs of the components, the unit price, and the doctor's prescription requirement. On the other hand, the products information is the same as in the PASS level + the doctor's prescription requirement.

v. Finally, your program should support both IDs and names of the customers, products/bundles for any functions that use this information. For example, instead of entering the customer names,

product/bundle names, now, the user can enter the customer IDs and product/bundle IDs, respectively.

---------- DI Level (4 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.

ii.   In this level, in the "*Make a purchase*" option, your program will allow customers to enter multiple products and quantities in one order. 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. Note that in this level, in the option "*Make a purchase*", your program can deduct the reward points of a customer before finishing a purchase. The specification is as in Requirement 2 of PART 3 in Assignment 1. Note that the deduction of the reward points is from the final total cost.

i.   Your program now has an option "*Add/update information of products*" to add products. The specification is same as Assignment 1 for this option (requirement 2 in PART 2 and requirement 3 in PART 3).

ii.   Your program now has an option *"Adjust the reward rate of all Basic customers"* to adjust the reward 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. Note that in this option, if the user enters 1, this means the reward rate is 100%.

iii. Your program now has an option *"Adjust the discount rate of a VIP customer"*. The option will ask for the name or ID of a VIP customer, then ask for a new discount rate. Invalid customers (non-existent or non-VIP customers) needs to be handled, i.e., your program will give the user another chance until a valid VIP 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. Besides, note that in this option, if the user enters 0.2, this means the discount rate is 20%.

iv.   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 automatically load previous orders that are stored in a comma separated file named *orders.txt* that is located in the same directory with the .py file. This file will be loaded when the program starts (as with the customer and product files). In the sequel, we will call this the *order file*. Below is an example of the order file:

Each line in the order file is the information of an order. The format is: *customer_name/ID, product1_name/ID, quantity1, product2_name/ID, quantity2, ..., total_cost, earned_rewards, date.* You can assume all the customers in the order file are existing customers (they are in the customer file). You can assume all product/bundles in the order file are existing products/bundles (they are in the product file) and are valid. Customers and products/bundles can be referred by IDs or names in this order file. You can assume all other information (quantity, total cost, earned reward points) in this order file is always valid. Note that after loading this order file, the reward points of the customers will also be updated based on the reward points from this order file.

Note that errors when loading the order file should also be handled. When there are any errors loading the file, your program will print a message saying: "*Cannot load the order file*".

iii. Your program now has an option "*Display all orders*" to display all orders' information as in the order file (i.e., the customer's name, products, quantities, total cost, earned rewards, and the order time). The printed message is flexible.

iv. The program now can use command line arguments to accept three file names (the first being the customer file name, the second being the product file name, the third being the order file). The first two files are mandatory whilst the third file (order) is optional. If no file names are provided, your program will look for *customers.txt, products.txt*, and *orders.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. Your program will now have an option *"Display a customer order history"*. The option will show a table displaying all the previous orders of a particular customer. The specification is as in Assignment 1. Note that here, the total cost is the final total cost (after the discount).

This is the order history of James.

	Products	Total Cost	Earned Rewards
Order 1	1 x coldTablet, 1 x vaccine	35.88	36
Order 2	1 x tablet	19.80	20

vi. When your program terminates, it will update the three files: customer, product, and order, 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, datetime 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	4
DI level	4
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 of 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