# Programming Fundamentals (COSC2531)
# Assignment 2

| | |
|---|---|
| **Assessment Type** | **Individual assignment** (no group work).<br>Submit online via Canvas/Assignments/Assignment 2.<br><br>Marks are awarded per rubric (please see the rubric on Canvas).<br>Clarifications/updates may be made via announcements.<br>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.<br><br>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 Assignment 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 further developing "Pythonia" serviced apartment booking system as in Assignment 1 using the Object-Oriented Programming (OOP) paradigm. Same as in Assignment 1, the booking manager uses this system to make reservations on behalf of apartment guests, process and print out receipts of the guests' bookings and purchases. You should implement the program following the requirements below. Note the requirements in this assignment are sometimes slightly different and more complex compared to those in Assignment 1. Sample input data in .csv files can be downloaded on A2 page on Canvas as a starting point. You should change the data in these files to test your program. Your markers will use various input .csv files when they grade 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** of requirements. You should only proceed to the next level after completing the previous one, as each level builds upon the previous one to create a single, running program.

## ---------- 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 "**7. Class Operations**".

# Guest:

### 1. Class Guest

Each guest has a unique **ID**, unique **name** (a name will only contain alphabet characters). Each guest also has some **reward** points. You are required to write the class named Guest to support the following:

i. Attributes **ID**, **name**, **reward_rate, reward, redeem_rate.**
ii. The constructor should accept three arguments: **ID, name, and reward**, and set the default values for **reward_rate** to 100% and **redeem_rate** to 1%.
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 to the nearest integer. 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 x 100%) = 15.
i. A method **update_reward** which takes a value and increase the attribute **reward** with that value.
ii. A method **display_info** that prints the values of the Guest **attributes**.
iii. A method **set_reward_rate** to adjust the reward rate.
iv. A method **set_redeem_rate** to adjust the redeem rate.

# Products:

### 2. Class Product

This class is to keep track of information on different products and services that the serviced apartment company offers, broadly classified as 'Product'. This class supports the following information:

- **ID**: a unique identifier of the product.
- **name**: the name of the product.
- **price**: the unit price of the product.
- Constructor takes values of **ID**, **name**, and **price** as arguments.
- Appropriate getter methods for the attributes of this class.
- A method **display_info** which should be an empty super method.

### 3. Class ApartmentUnit

ApartmentUnit is a subclass of Product, to keep track of information on different apartment units to rent. Due to inheritance, all super class's attributes and methods are available in this class. This class supports the following information:

- Capacity: the apartment unit's maximum number of guests allowed to stay in the apartment unit at any single night.
- Constructor takes the appropriate parameters/arguments.
- Appropriate getter methods for the attributes of this class.
- A method **display_info** that prints the values of the ApartmentUnit attributes.

### 4. Class SupplementaryItem

Supplementary item is another subclass of Product, to keep track of information on different supplementary items that the serviced apartment company offers. Due to inheritance, all super class's attributes and methods are available in this class. This class supports the following information:

- Constructor takes the appropriate parameters/arguments.
- Appropriate getter methods for the attributes of this class.
- A method **display_info** that prints the values of the SupplementaryItem attributes.
- Extra attributes and methods if you want to define

## Orders

### 5. Class Order

This class is to store a guest's purchase information. This class supports the following information of an order:

- **guest**: the one who makes the purchase. 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 guest.
- 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.
- 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 guests – you need to think what you should store** in this list (guestID, guest 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_guests**. This method takes in a file name and then read and add the guests in this file to the guest list of the class. In the sequel, we call this the *guest file*. See an example of the guest file below.

```
1, Alyssa, 100, 200, 1
2, Luigi, 100, 300, 1
3, James, 100, 150,1
```

In this file, the guests are always in this format: *guest_ID, guestname, reward rate, reward, and redeem rate*. In this level, you can assume there will be no error in this

guestfile (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.

U12swan, Unit 12 Swan Building 200.00, 3
U13swan, Unit 13 Swan Building 190.70, 2
SI1, Car Park, 25.00
U20goose Unit 20 Goose Building 165.00, 1
U21goose Unit 21 Goose Building 175.00, 2
U20goose Unit 20 Goose Building 185.00, 3
SI2, Breakfast, 25.30
SI3, Tooth brush, 10.00
SI4, Tooth paste, 5.00
SI5, Shampoo, 20.50
U63duck Unit 63 Duck Building 134.50, 2
U63duck Unit 63 Duck Building 148.00, 2
SI6, Double extra bed (2 people), 50

In this file, all products will have the first 3 columns: *product_ID*, *product_name*, *unit_price*. In the case of apartment units to rent, there is an additional column for *capacity*. Apartment units always start with the letter "U", whereas Supplementary Items start with 2 letters “SI”. 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_guest** and **find_product**. These methods take in a search value (can be either a name or an ID of a guest or product), search through the list of guests/products and then return the corresponding guest/product if found or return None if not found.
- This class also has two methods **list_guest** and **list_products**. These methods can display the information of existing guests and guests on screen.
    - The method **list_guests** will display the guestID, name, the reward rate, the reward, and the redeem rate.
    - The method **list_products** will take in *product_type* as argument display the product ID, product name, and the unit price accordingly. If the product_type equals “apartment” then it will display product ID, product name, the unit price labelled as “Rate per Night”, and capacity.
    - Note the two methods can be used to validate the reading from the two .csv files associated with the guest 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

**Menu options**

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

i. *Make a booking*: this option allows users to make a purchase for a guest detailed requirements for this option are below (Detailed Requirements i-v).
ii. *Display existing guests*: this option can display all the information of all existing guests: ID, name, reward rate, reward, and redeem rate.
iii. *Display existing apartment units:* this option can display all the information of existing apartment units' ID, name, unit price (rate per night), and capacity.
iv. *Display existing supplementary items:* this option can display all the information of all existing supplementary items: ID, name, and unit price.
v. *Exit the program:* this option allows users to exit the program.

**Detaild Requirements**

Detailed requirements of the menu program are as follows:

i. When the program starts, it looks for the files *guests.csv* (the guest file) and *products.csv* (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 5 options described above. If any file is missing, the program will quit gracefully with an error message indicating the corresponding file is missing.
ii. Your menu program will allow the user to make an order as specified in PART 1 of Assignment 1. In this level, the users will need to enter the following fields:
    a. Guest name
    b. Number of guests
    c. Apartment id
    d. Check-in date
    e. Check-out date
    f. Length of stay
    g. Booking date
    h. Supplementary item id
    i. Supplementary item quantity

    The rest of fields should be calculated or automatically filled as much as possible by getting a calculation result or value from functions or referring to data structure content.

iii. When a guest finishes making an order,
    a. If the guest is a new guest, the program will add the information of that guest into the data collection (think/analyse carefully which information needs to be added).
    b. If the guest is an existing guest, the program will print out a message showing the reward points, then proceed with the purchase and display the receipt.
    c. The user is given the opportunity to use their reward points to claim for discounts. By default, every 100 points is equivalent to 1 redeemable dollar. That means having 200 points is the same as being able to redeem 2 dollars and so on. The final total cost = the original cost – discount, and the **earned reward points are computed by rounding the final total cost**. The user may select not to claim any discount. That will result in the final total cost to be the same as the original cost.
iv. The improved receipt of a guest order can be displayed as a formatted message as below.

```
==========================================================
Guest name:          <guest_name>
Number of guests:    <number_of_guests>
```

```
Apartment name:      <name> (auto-complete based on id)
Apartment rate: $    <unit_price> (AUD) (auto-complete based on id)
Check-in date:       <checkin_date>
Check-out date:      <checkout_date>
Length of stay:      <quantity> (nights)
Booking date:        <booking_date>

Sub-total: $         <apartment_sub_total> (AUD)
---------------------------------------------------------------------------
Supplementary items
ID      Name         Quantity       Unit Price $   Cost $
<id>    <name>       <quantity>     <unit_price>   <cost>
<id>    <name>       <quantity>     <unit_price>   <cost>

Sub-total: $ <supplementary_items_sub_total>
---------------------------------------------------------------------------
Total cost: $ (AUD)
Reward points to redeem: (points)
Discount based on points: $ (AUD)
Final total cost: $ (AUD)
Earned rewards: (points)

Thank you for your booking!
We hope you will have an enjoyable stay.
===========================================================
```

v. When a task is accomplished, the menu will appear again for the next task. The program always exits gracefully from the menu.

## ---------- CREDIT level (5 marks, please do not attempt this level before completing the PASS level) ----------

### Operations

i. At this level, your program will no longer require user to input the "Length of stay". Instead, it needs to calculate the length of stay (that is the quantity, in the number of nights) based on the difference between requested check-in and check-out dates.

ii. At this level, your program will no longer require user to input for "Booking date and time". Instead, it will auto-complete the field based on timestamp of when the reservation order was made. The expected format should show D/M/YYYY h:m such as 8/8/2024 17:45.

iii. 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 guest'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. Product means apartment unit or supplementary item.

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 any of the following discrepancies with the dates occur:
    i. If the check-in date is earlier than the booking date.
    ii. If the check-out date is earlier than the booking date.
    iii. If the check-out date is earlier than check-in date.
    iv. If the check-in date is the same as the check-out date.

iv. Furthermore, in this level, in the "*Make a booking*" 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 U12swan, breakfast for 2, and a carpark. You can assume all parts of a bundle are existing products in the system.

The price of a bundle is 80% of the total price of all individual products. For example, if U12swan costs $200.0, breakfast costs $25.30, another breakfast costs $25.30, and carpark costs $25 then the price of this bundle is 80% x (200.0 + 25.30 + 25.30 + 25) = $220.48.

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**. The ID of a Bundle always starts with the letter "*B*" as indicated in the lines at the end of the products.csv file.

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

U12swan, Unit 12 Swan Building 200.00, 3
U13swan, Unit 13 Swan Building 190.70, 2
SI1, Car Park, 25.00
U20goose Unit 20 Goose Building 165.00, 1
U21goose Unit 21 Goose Building 175.00, 2
U22goose Unit 22 Goose Building 185.00, 3
SI2, Breakfast, 25.30
SI3, Tooth brush, 10.00
SI4, Tooth paste, 5.00
SI5, Shampoo, 20.50
U63duck Unit 63 Duck Building 134.50, 2
U63duck Unit 63 Duck Building 148.00, 2
SI6, Double extra bed (2 people), 50
**B1, Bed and breakfast for two, U12swan, SI2, SI2, SI1, 220.48**
**B2, Bed and breakfast family deluxe, U22goose, SI2, SI2, SI2, SI2, SI1, SI6, 712.96**

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. It must contain an apartment unit and it allows for duplicates IDs for supplementary items, for example if there are 4 breakfasts in the bundle then there are 4 occurrence of SI2 in the line. Your program should process this as 4 quantities of the same item. You can assume bundles are always stored at the end of a file, after all normal products.

v. 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. When there is more than one occurrence of the same component, it will just list the component in a line with the correct quantity displayed, for example:

| ID | Name | Components | Price |
|---|---|---|---|
| B1 | Bed and breakfast for two and a car park bundle | U12swan, 2 x SI2, SI1 | 220.48 |

*vi.* Finally, your program should support both IDs and names of the guest, products/bundles for any functions that use this information. For example, instead of entering the guest names, product/bundle names, now, the user can enter the guest IDs and product/bundle IDs, respectively.

## ---------- DI Level (5 marks, please do not attempt this level before completing the CREDIT level) ----------

In this level, there are some additional main features for some classes in your program. Some features might be challenging. Details of these features are described as follows.

### Operations

Your program now has the following new features.

i. In this level, in the "*Make a booking*" option, your program will allow guest to enter multiple products and quantities in one order. The requirements are as in Assignment 1 for this option (requirement 1 of PART 3). The following business rules should be enforced and applied in the program:
   i. Every order must contain an apartment reservation.
   ii. When the number of guests exceed, then extra bed must be offered, and the quantity ordered must be according to the number of nights that the number of apartment unit is required. For example, if the guest needs 2 nights, then the number of extra beds should be minimum 2. Other rules regarding the extra bed are the same as those in Assignment 1.
   iii. Car park should also be validated according to the number of nights the apartment unit is required. Some people may need to fit more than one car, but when car park is ordered it must be at least the same as the number of nights required.

   You can modify the existing classes or design extra classes to support this requirement.

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

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

iv. Your program now has an option "*Add/update information of bundles*" to add special bundle products. This is a new requirement, however you can re-use and extend functions that you have built for the other product types as per the above ii and iii features.

v. Your program now has an option *"Adjust the reward rate of all guests"* to adjust the reward rate of all guests. This adjustment will affect all guests 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. By default, reward rate is 100%, that means, each dollar will convert to a reward point on 1:1 ratio.

vi. Your program now has an option *"Adjust the redeem rate of all guests"* to adjust the redeem rate of all guests. This adjustment will affect all guests in all future orders. Invalid inputs (non-number or 0 or negative rate) should be handled via exceptions; Any input number below 1% is also considered invalid, i.e. "too low to be worth claiming". The user will be given another chance until a valid input is entered. By default, the redeem rate is 1%, which means, every 100 points can be converted to 1 dollar discount. If the user enters 2, this means the dollar discount will be equal to 2% of the redeemed points. The guest is only eligible to use any reward point for discounts when their current reward point balance has reached 100 points.

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 (5 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. Your program now has an option "*Display all orders*" to display all orders' information as in the order file (i.e., the guest's name, products, quantities, total cost, earned rewards, and the order date time). The print format for this output is flexible, however for clarity you should consider having tabular format.

ii. The program can now save these orders into a comma separated value (CSV) file named *orders.csv*, that is located in the same directory with the .py file. This will the record in the following format: *guest_name/ID*, *quantity* x *apartment_unit_id*, *quantity* x *supplementary_item1*, *quantity* x *suplementary_item2*, ..., *total_cost*, *earned_rewads*, *order_date_time*. For example:

Alyssa, 2 x U12swan, 4 x SI2, 2x SI1, 551.2, 551, 12/9/2024 19:55

iii. Your program has another new option called "Generate key statistics" to display a short report that shows key business statistics based on the orders made so far. The report needs to contain the following:

- "Top 3 most valuable guests" report that shows the guest names and their total order amount in descending order, that is to show the highest amount at the top, followed by others.
- "Top 3 most popular products" report that shows the name of the product its sold quantity in descending order, that is to show the highest quantity order at the top, followed by others.
- The above 2 key statistics should be saved in a text file called *stats.txt*.

iv. The program now can automatically load previous orders that are stored in a comma separated file named *orders.csv* that is in the same directory with the .py file. This file will be loaded when the program starts (as with the guest and product files). In the sequel, we will call this the *order file*.
Each line in the order file is the information of an order. You can assume all the guests in the order file are existing guests (they are in the guest file). You can assume all product/bundles in the order file are existing products/bundles (they are in the product file) and are valid. Guests and products/bundles can be referred to 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 guests 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*".

v. The program now can use command line arguments to accept three file names (the first being the guest 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 *guests.csv*, *products.csv*, and *orders.csv* 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.

vi. Your program will now have an option *"Display a guest order history"*. The option will show a table displaying all the previous orders of a particular guest. The specification is as in Assignment 1. Note that here, the total cost is the final total cost (after the discount).

*This is the booking and order history for Alyssa.*

| *Order ID* | *Products Ordered* | *Total Cost* | *Earned Rewards* |
|---|---|---|---|
| *Order1* | *1 x U12swan* | *95.0* | *95* |
| *Order2* | *1 x U209duck, 2 x breakfast* | *148.7* | *149* |
| *Order3* | *2 x U49goose, 4 x breakfast, 1 carpark* | *424.4* | *424* |

vii. When your program terminates, it will update the three files: guest, 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 dyou came up with the design of the program, how you started writing the code after the design process, the challenges you met during the code development.
- Document the references, i.e., any sources of information (e.g., websites, tools) you used other than the course contents directly under Canvas/Modules, **you must give acknowledgement of the sources**, **explaining how you use the sources in this assignment**. More detailed information regarding the references can be found in Section 7.

## D - Rubric:

Overall:

| Level | Points |
|---|---|
| PASS level | 10 |

| | |
|---|---|
| CREDIT level | 5 |
| DI level | 5 |
| HD level | 5 |
| Others (code quality, modularity, comments) | 3 |
| Others (weekly submission) | 2 |

More details of the rubric of this assignment can be found on Canvas. 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. You should also refer to Easy Cite.

## 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