# 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:

1. Follow coding, convention and behavioural requirements provided in this document and in the course lessons;
2. Independently solve a problem by using programming concepts taught over the duration of the course;
3. Write and debug Python code independently;
4. Document code;
5. Provide references where due;
6. Meet deadlines;
7. 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 restaurant ordering system as in Assignment 1 using the object-oriented programming (OOP) paradigm. Same as in Assignment 1, the cashiers or the restaurant managers of the restaurant are the ones that use this system to process customers' orders. 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 (12 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

All customers have a unique **id**, unique **name** (a name will not include any digit nor white space). You are required to write the class named **Customer** to support the following:

1. Attributes **id** and **name**
2. Constructor takes the values of **id, name** as arguments
3. Appropriate getter methods for the attributes of this class iv. A method **get\_service\_fee(self, cost)** which serves as a super method and will have a more complex implementation in the sub classes.
4. A method **get\_discount(self, cost)** which serves as a super method and will have a more complex implementation in the sub classes.
5. A method **display\_info(self)** that prints the values of the **Customer** attributes and the customer type (i.e., Customer or C).

## 2. Class BronzeCustomer

A bronze customer is a normal customer that is not in the rewards program. When ordering a meal, a bronze customer will be charged for the service fee. All bronze customers have a flat service fee rate of 10% (i.e., the service fee rate is the same for all bronze customers). The class BronzeCustomer should have the following components:

1. An attribute for the **service fee rate**, by default it is 10%.
2. Constructor takes the appropriate parameters/arguments (be careful)
3. Appropriate getter methods for the attributes of this class
4. A method **get\_service\_fee(self, cost)** that takes the cost of the meal and returns the service fee. For example, this method returns 50 when the service fee rate is 10% and the cost of the meal is 500$.
5. A method **get\_discount(self, cost)** that returns 0

vii. A method **display\_info(self)** that prints the values of the **BronzeCustomer** attributes and the customer type (i.e., Bronze Customer or B).

vi. A method **set\_service\_fee\_rate** to adjust the service fee rate. This affects all bronze customers.

## 3. Class SilverCustomer

A silver customer is a customer that is in the rewards program, and with status *Silver*. When ordering a meal, a silver customer will not be charged for the service fee. The class **SilverCustomer** should have the following components:

1. Constructor takes the appropriate parameters/arguments (be careful)
2. Appropriate getter methods for the attributes of this class
3. A method **get\_service\_fee(self, cost)** that returns 0
4. A method **get\_discount(self, cost)** that returns 0
5. A method **display\_info(self)** that prints the values of the **SilverCustomer** attributes and the customer type (i.e., Silver Customer or S).

## 4. Class GoldCustomer

A gold customer is a customer that is in the rewards program, and with status *Gold*. A gold customer not just receives free service fee but also gets discount for the meal cost. The discount rate is different among the gold customers. If not specified, it is set as 8%.

The class **GoldCustomer** should support the following components:

i. Attributes to support the **discount rate** ii. Necessary constructors iii. Appropriate getter methods for the attributes of this class iv. A method **get\_service\_fee(self, cost)** that returns 0

1. A method **get\_discount(self, cost)** that takes the cost of the meal and returns the discount. For example, this method returns 6.4 when the discount rate is 8% and the cost of the meal is 80$.
2. A method **display\_info** that prints the values of the **GoldCustomer** attributes and the customer type (i.e., Gold Customer or G).
3. A method **set\_discount** (or a setter method for the attribute **discount rate**) to adjust the discount value of an individual gold customer.

**Items:**

## 5. Class Item

This class is to keep track of information on different items (food dishes and drinks) that the restaurant offers. This class supports the following information:

* **id**: a unique identifier of the item (e.g., 1, 2, 3)
* **name**: the name of the item (you can assume the item names are unique and they do not include any digit)
* **price**: the unit price of the item
* A method **display\_info** that prints the values of the **Item** attributes. • Extra attributes and methods if you want to define.

## 6. Class FoodDish

A food dish is an item, specifically a food dish. You can define extra attributes and methods to support this class if you want.

## 7. Class Drink

A drink is an item, specifically a drink. You can define extra attributes and methods to support this class if you want.

# Orders

## 8. Class Order

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

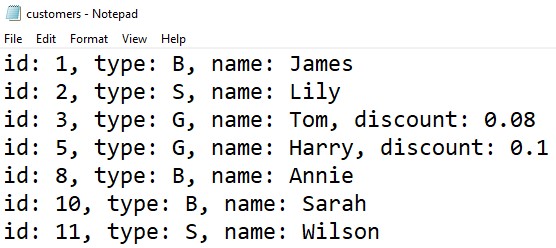
* **customer**: the one who places the order (can be a bronze, silver, or gold customer). You need to think/analyse carefully if this should be an ID, name, or something else.
* **item**: the item in the order (can be a food dish or a drink). You need to think/analyse carefully if this should be an ID, name, or something else.
* **quantity**: the quantity of the item ordered by customers.
* A method **compute\_cost** that returns the original cost (the total cost without service fee and discount), the service fee, and the discount. For example, if the order is of the customer Tom (Gold customer with discount rate 8%), and the item ordered is a pizza (unit price: 12.5) and quantity 2, then this method will return 25, 0, 2.
* Extra attributes and methods if you want to define.

# Records

## 9. Class Records

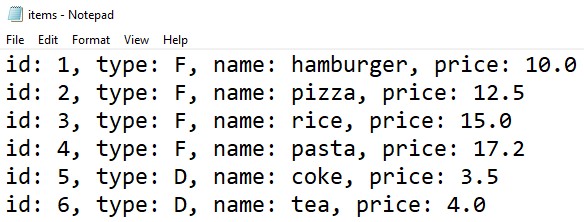
This class is the central data repository of your program. It supports the following information:

* **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 items (food dishes or drinks) the restaurant offers** – you need to think about what you should store in this list (item ID, item name, or something else?)
* This class has a method named **read\_customers**.This method takes in a file name and can 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.



In this file, the customers are always in this format: *id: customer\_ID, type: customer\_type, name: customer\_name, discount: discount\_value* (if that is a gold customer). For example, in the 1st line, the *id* is 1, the *type* is B(i.e., Bronze), the *name* is James. In the 2nd line, the *id* is 2, the *name* is Lily, the *type* is S(i.e., Silver). In the 3rd line, the *id* is 3, the *name* is Tom, the *type* is G(i.e., Gold), and the *discount* is 0.08 (i.e., 8%). The IDs and the names are all unique. 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 discount values are always valid).

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



In this file, the items are always in this format: *id*: *item\_ID, type: item\_type, name: item\_name, price: item\_unit\_price*. The IDs and the names are all unique. There are two item types: Food (F) or Drink (D). 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\_item**. These two methods take in a search value (can be either a name or an ID of a customer or item), search through the list of customers/items and then return the corresponding customer or item if found or return None if not found.
* This class also has two methods **display\_customers**, and **display\_items**.These two methods can display the information of existing customers and items on screen. Specifically, the method **display\_customers** will display at least the customer ID, type, name, service fee rate, and the discount rate (only for Gold customers). The method **display\_items** will display item ID, type, name, and price. Note the two methods can be used to validate the reading from these two .txt files.

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

# Operations

## 10. Class Operations

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

1. *Order a meal*: this option allows users to order a meal for a customer. Detailed requirements for this option are below (Requirements vi-viii).
2. *Display existing customers information*: this option can display all the information of all existing customers: ID, type, name, service fee rate, discount rate (only for gold customers).
3. *Display existing items information:* this option can display all the information of all existing items: ID, type, name, and unit price.
4. *Exit the program:* this option allows users to exit the program.

Other requirements of the menu program are as follows:

1. When the program starts, it looks for the files *customers.txt* (the customer file) and *items.txt* (the item 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.
2. Your menu program will allow the user to order a meal as specified in PART 1 of Assignment 1. Note that in this assignment, the customer can choose to register for the rewards program as a silver or a gold customer. More detailed information regarding the registration for the rewards program is in section vii below. In this part, like PART 1 of Assignment 1, you can assume users always enter valid items, valid item quantity, and valid *"y"* or *"n"* answers. You can also assume users always enter the status for the rewards program accurately, for example, *"S"* for a silver customer registration, and *"G"* for a gold customer registration.
3. When a customer finish ordering a meal,
   1. 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). If the customer answers *"n"* for the question of registering for the rewards program, then the customer is just a bronze customer. If the customer answers *"y"*, then the program will ask what type of rewards they want. If the answer is *"S"*, then the customer will become a silver customer. If the answer is *"G"*, then the customer will become a gold customer. Note the customer will need to pay an extra 100$ to register to be a silver customer and 300$ to be a gold customer. The free service fee and the discount are applied immediately after the rewards program registration. Note the discount of gold customers is NOT applied to this 300$ fee. Again, you can assume the users enter the rewards type correctly ("*S*" or "*G*").
   2. If the customer is an existing customer, the program will print out a message showing the customer type (e.g., bronze/silver/gold customer), then proceed the order and display the receipt. Also, for existing customers, you DO NOT need to ask if they want to register for the rewards program. This is slightly different compared to the requirements in Assignment 1, so please be careful.
4. The total cost of an order can be displayed as a formatted message as below (for existing bronze/silver customers or new bronze customers):

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*Receipt of Customer <customer\_name>*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*<item\_name>: <unit\_price> (AUD) x <quantity>*

*Service fee: <service\_fee> (AUD)*

*Total cost: <total\_cost> (AUD)*

The formatted message is as below for new silver customers:

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*Receipt of Customer <customer\_name>*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*<item\_name>: <unit\_price> (AUD) x <quantity>*

*Service fee: <service\_fee> (AUD)*

*Registration fee: <the registration fee> (AUD)*

*Total cost: <total\_cost> (AUD)*

The formatted message is as below for gold customers (note that the line about the registration fee only appears for new gold customers):

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*Receipt of Customer <customer\_name>*

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*<item\_name>: <unit\_price> (AUD) x <quantity> Service fee: <service\_fee> (AUD)*

*Discount: <discount> (AUD)*

*Registration fee: <the registration fee> (AUD)*

*Total cost: <total\_cost> (AUD)*

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 (3 marks, please do not attempt this level before completing the PASS level) ------------ Operations

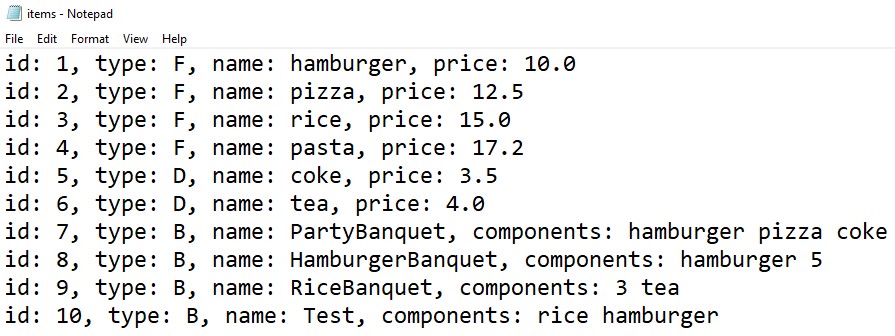
1. In this level, in the "*Order a meal*" option, your program will allow ordering a Banquet, which is a special item that contains multiple food dishes and drinks. For example, a banquet can consist of a hamburger, a pizza, and a coke. You can assume all components of a banquet are existing food dishes and drinks in the program. You can also assume the quantity of each item in a banquet is only 1. And note each banquet needs to have at least 1 food dish and 1 drink.

The price of a banquet is the total price of the food dishes (the drinks are free in a banquet). For example, if a banquet consists of a hamburger, a pizza and a coke, and if a hamburger costs 10.0$, a pizza costs 12.5$, and a coke costs 3.5$, then the price of this banquet is 10.0 + 12.5 = 22.5$.

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

**11. Class Banquet:** Each banquet has a unique **ID** and **name** (as with **Item**). You need to define the appropriate attributes and methods to support the class **Banquet**.

With this modification, the item file at this level may look like this:



The type of a Banquet is indicated with the letter *"***B***"*. Note that the data format of a banquet is different compared to a normal item. Its format is as follows: *id*: *banquet\_ID, type: B, name: banquet\_name, components: item\_1 item\_2 item\_3 …* The IDs/names of all items (including banquets) are all unique. The components in a banquet can be shown as names or IDs. In the above example, the RiceBanquet consists of one rice and one tea. You can assume all the components in a banquet are existing items and unique (no duplicates). You can assume banquets are always stored at the end of a file, after all normal items. You can assume the items' names never have white spaces nor commas nor colons.

Note that as each banquet needs to have at least one food dish and one drink, so when your program reads a banquet from the items file, and if it does not satisfy this requirement, it will display a message saying something wrong with this banquet and ignore it. Note your program still process other items/banquets in the file as normal.

1. At this level, for the option "*Display existing items information*", when displaying banquets, your program will display the ID, type, name of the banquet, the names of the component items, and the price. On the other hand, the information of the items is the same as in the PASS level. iii. At this level, your program should support both customers' IDs and names when ordering a meal. That is, your program will ask users to choose to enter their IDs or their names. Your program should also support the items' IDs and names when ordering a meal.

Finally, 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:

1. Display an error message if the item entered by the user does not exist in the list of items. When this error occurs, the user will be given another chance, until a valid item is entered.
2. Display an error message if the item quantity is 0, negative, not an integer. When this error occurs, the user will be given another chance, until a valid quantity is entered.
3. Display an error message if the answer by the user is not *y* or *n* when asking if the customer wants to register for the rewards program. When this error occurs, the user will be given another chance, until a valid answer (i.e., *y*, *n*) is entered.
4. Display an error message if the answer by the user is not *S* or *G* when asking the type of the rewards program. When this error occurs, the user will be given another chance, until a valid answer (i.e., *S, G*) is entered.

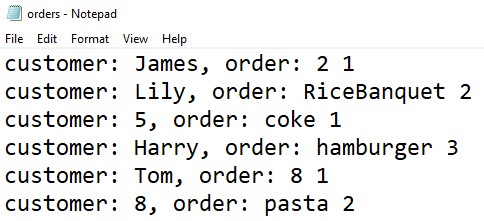
# ---------- DI Level (3 marks, please do not attempt this part before completing the CREDIT level) -----------

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 should:

1. Have an option *"Order meals via a file"* that asks users to specify a filename, and then place orders in that file. In the sequel, we will call this the *order file*. For example, if the user enters the filename *orders.txt* and the *orders.txt* file contains the following orders:

 then your program will place these orders automatically and sequentially.

Each line in the order file is an order. The format is: *customer:* *customer\_name/ID, order: item\_name/ID item\_quantity*. This option only supports existing customers; hence you can assume all customers in the order file are existing customers (they are in the customer file). You can assume all items in the order file are existing items (they are in the item file). The items can be normal items or banquets. Both customers and items can be referred by IDs or names in this order file. You can assume all other information (quantity) in this order file is always valid, and the file entered by the user is always in the same directory with the .py 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. Go back to the main menu.*".

1. Have an option *"Adjust the service fee rate of all bronze customers"* to adjust the service fee rate of all Bronze customers. This adjustment will affect all Bronze 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.
2. Have an option *"Adjust the discount rate of a gold customer"*. The option will ask for the name or ID of the gold customer, then ask for a new discount rate (e.g., 0.1 which corresponds to

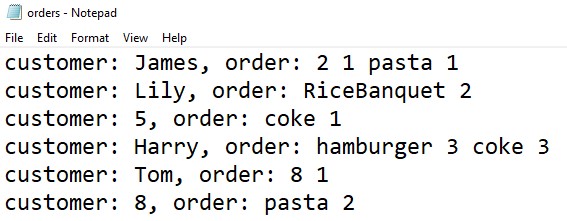
10% discount rate). Invalid customers (non-existent or non-gold customers) needs to be handled, i.e., your program will give the user another chance until valid gold customer is entered. Invalid inputs (non-number or negative values) should also be handled via exceptions, and the user will be given another chance until a valid input is entered. Also, 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.

1. Have an option "*Display all orders*" to display all orders information including the customer's name, the item's name, and the quantity. The printed message is flexible.
2. 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 can:

1. Your program now can use command line arguments to accept the two file names (the first being the customer file name, and the second being the item file name). If no file names are provided, your program will look for *customers.txt,* and *items.txt* in the local directory. If a wrong number of arguments are provided, the program will display a message indicating the correct usage of arguments and exit.
2. In this level, in the "*Order a meal*" option, your program will allow customers to order multiple items in one order. The requirements are as in Assignment 1 for this option (requirements 1 and 3 of Part 3). You can design extra classes or modify existing classes to support this requirement.
3. In this level, in the *"Order meals via a file"* option, the order file format will be slightly different compared to previous levels to support orders with multiple items. An example of the order file is as below.



The format of each line in the order file is *customer:* *customer\_name/ID, order: item1\_name/ID item1\_quantity item2\_name/ID item2\_quantity* ... The items in each order can be repetitive, e.g., an order can have 2 pastas, 3 cokes and 1 pasta.

1. In this level, your menu program will have an option named *"Add sold-out items"* with the description and input format as in Assignment 1 (requirement 2 of Part 3). When an item is sold out, it cannot be sold, and a message will be shown if customers attempt to order such dishes. When this error occurs, your program will skip this item, and continue to process the next item. If the customer decides to not order any item at all, then your program will directly

go back to the main menu and cancel the order. Note the user can specify either the names or the IDs of the items.

1. The menu now has an option "*Display the most frequent customer*" to display the customer with the maximum number of orders to date and the number of orders they made. If there are multiple customers with the same maximum number of orders, you can display only one customer or all customers, it's your choice.
2. Your program will now have an option "*Display a customer order history*". The option will display a message asking the user to enter the name or ID of the customer, and the program will display all the previous orders of that customer, including the items and the quantities they ordered and the total cost (including the service fee and discount). For example, if a customer named Kate ordered 2 times previously, the first time with 2 hamburgers, 1 coke and the total cost of 28.0, and the second time with 1 pizza, 1 coke and the total cost of 17.0, then the program will display the formatted message as follows.

*This is the order history of Kate.*

*Items Total Cost*

*Order 1 2 x hamburger, 1 x coke 28.0*

*Order 2 1 x pizza, 1 x coke 17.0*

Note that for this option, if the user enters a non-existing customer, then your program needs to give the user another chance, until an existing customer is entered.

1. When your program terminates, it will update the two files: customers and items 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 (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 libraries allowed in this assignment are sys, copy, 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.

1. **- Documentation Requirements:**

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

**Your comments (documentation) should be in the same Python file, before the code blocks (e.g., functions/methods, loops, if, etc.) and important variable declarations that the comments refer to**. Please DO NOT write a separate file for comments (documentation).

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 you have not met.** For example, scenarios that might cause the program to crash or behave abnormally. Note, you do not need to handle or address errors that are not covered in the course.

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

* + Explain your code in a precise but succinct manner. It should include a brief analysis of your approaches instead of simply translating the Python code to English. For example, you can comment on why you introduce a particular function/method, why you choose to use a while loop instead of other loops, why you choose a particular data type to store the data information.
  + Document any problems of your code and requirements that you have not met, e.g., the situations that might cause the program to crash or behave abnormally, the requirements your program do not satisfy. Note that you do not need to handle or address errors that are not covered in the course material yet.
  + Document some analysis/discussion/reflection as a part of your code, e.g., how your code could be improved if you have more time, which part you find most challenging, etc.

1. **- Rubric:**

Overall:

|  |  |
| --- | --- |
| **Level** | **Points** |
| PASS level | 12 |
| CREDIT level | 3 |
| DI level | 3 |
| HD level | 6 |
| Others (code quality, modularity, comments) | 3 |
| Others (weekly submission) | 3 |

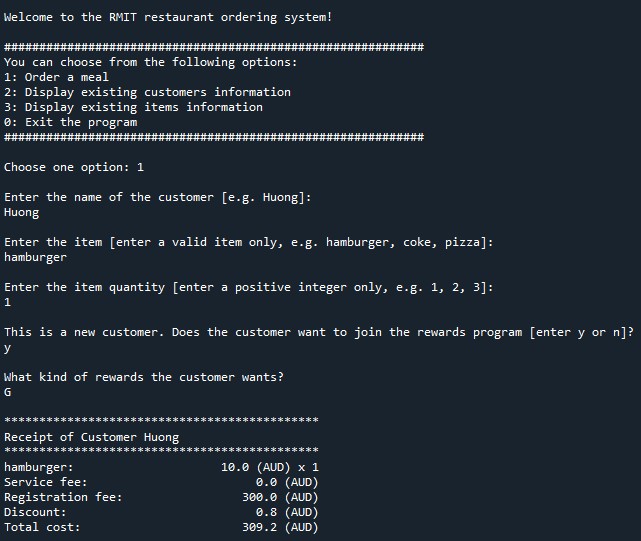
More details of the rubric of this assignment can be found on Canvas [(here)](https://rmit.instructure.com/courses/90658/assignments/716972). Students are required to look at the rubric to understand how the assignment will be graded.

## 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. PASS Level

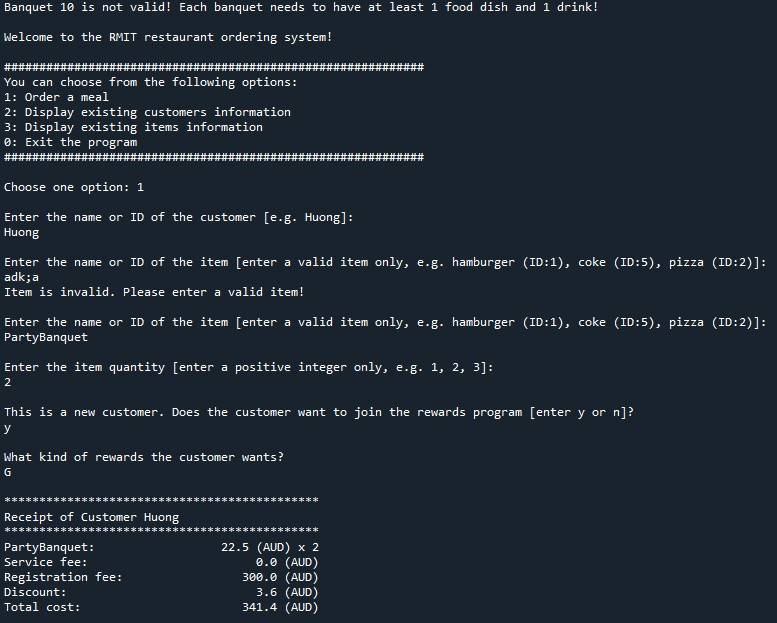
As an example, this is how the output screen of our sample program looks like for the PASS level, when we choose option 1, which is to order a meal with a customer named *Huong*, ordering 1 hamburger and registering for the Gold rewards program. You should test your program with different test cases, e.g., customers choose *n* (no) for the rewards program registration options or customers choose the Silver rewards option, to make sure your program satisfy the requirements of this level. Note that here, the program is implemented with the object-oriented paradigm and the classes described in the PASS level of Section 4.



### 5.2. CREDIT Level

As an example, this is what the output screen of our sample program looks like for the CREDIT level, when we choose option 1, which is to order a meal with a customer named *Huong*, ordering 2 *PartyBanquet* and registering for a Gold rewards. Here, we also test if the program can handle some types of invalid inputs. You should test your program with different test cases to make sure your program satisfies all the requirements of this level.

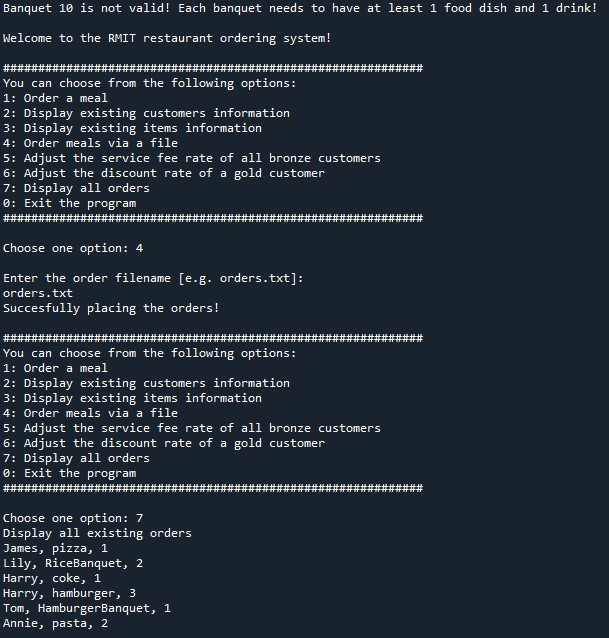
Besides, you should note that at the beginning, there is a message displaying *"Banquet 10 is not valid! Each banquet needs to have at least 1 food dish and 1 drink!"* as the Banquet 10 (Test) in the item file has both components as food dishes, thus, this banquet is invalid.



### 5.3. DI Level

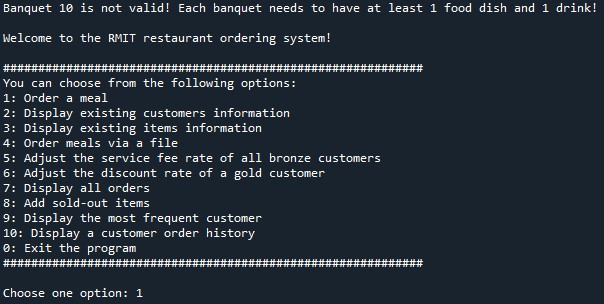
As an example, this is what the output screen of our sample program looks like for the DI level, when we choose option 4 to order via a file. In this case, we enter the filename *orders.txt*. After that, we choose option 7 to display all orders to check if the program successfully places the orders in the *orders.txt* file. It shows that the program indeed placed the orders successfully.

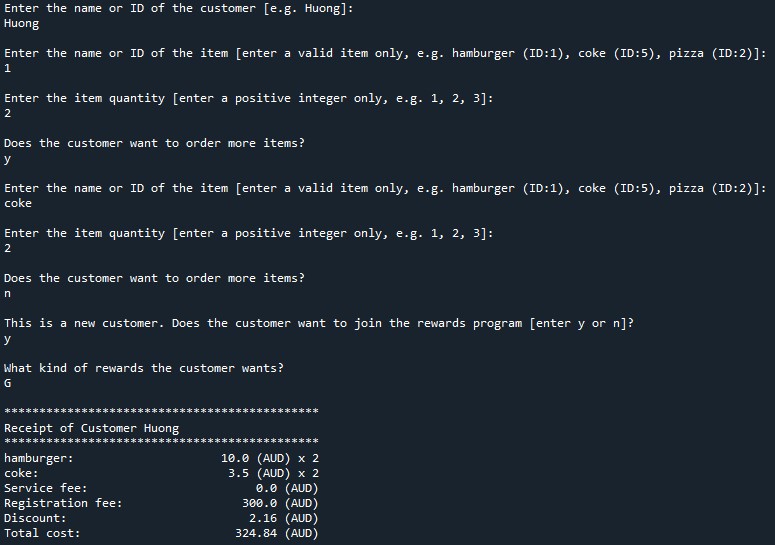
Note that you should test your program with different test cases to make sure your program satisfies all the requirements of this level.



### 5.3. HD Level

As an example, this is what the output screen of our sample program looks like for the HD level, when we choose option 1 to place an order with multiple items. You should test your program with different test cases to make sure your program satisfies all the requirements of this level.





## 6. 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.