Table of Contents

1. Key Information

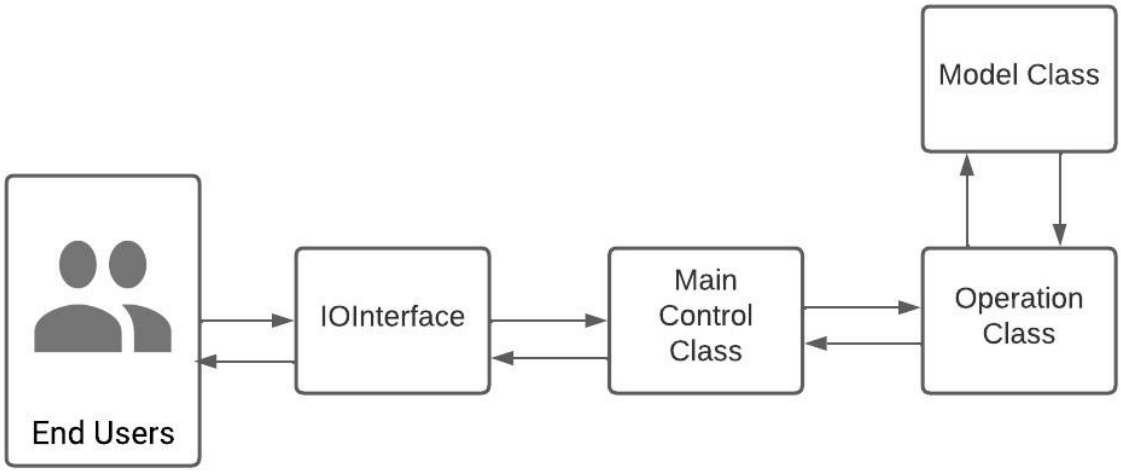
Purpose	This assignment will develop your skills in designing, constructing, testing, and documenting a small Python program according to specific programming standards. This assessment is related to the following learning outcome (LO): • **LO4:** Investigate useful Python packages for scientific computing and data analysis; • **LO5:** Experiment with data manipulation, analysis, and visualisation techniques to formulate business insight.
Your task	This assignment is an **Individual task** where you will write Python code for a simple application whereby you will be developing a simple **e-commerce** information management system as per the specification.
Value	**35**% of your total marks for the unit.
Due Date	**Friday, 9 June 2023, 4:30 PM (AEST)**
Submission	• Via Moodle Assignment Submission. • FIT GitLab check-ins will be used to assess the history of development • JPlag will be used for similarity checking of all submissions.
Assessment Criteria	The following aspects will be assessed: 1. Program functionality in accordance with the requirements 2. Code Architecture and Adherence to Python coding standards 3. The comprehensiveness of documented code
Late Penalties	• 10% deduction per calendar day or part thereof for up to one week • Submissions more than 7 calendar days after the due date will receive a mark of zero (0) and no assessment feedback will be provided.
Support Resources	See Moodle Assessment page and Section 7 in this document
Feedback	Feedback will be provided on student work via • specific student feedback ten working days post submission

2. Instruction

This assignment requires you to create an e-commerce system which allows customers to login to the system, perform some shopping operations like purchasing products, viewing

order history and showing user consumption reports. Besides, admin users need to be created to manage the whole system, who are able to create/delete/view customers, products and all the orders. Except for the management part, admin users can view the statistical figures about this system. Since the whole system is executed in the command line system, it is better to design a well-formatted interface and always show proper messages to guide users to use your system. In this assignment, we use the open source data from data.world, which contains 9 files of products. All the product's data should be retrieved from these files.

In this assignment, we are going to decouple the relationship between various classes. As you can see from the image below, we have four main parts and when using the system, end users only need to interact with the IOInterface class. The Main Control class handles the main business logic. The operation classes use the model classes as templates to manipulate the data reading/writing. With this design pattern, the input() and print() functions only exist in the I/O interface class. **No other classes have these functions**. The file reading/writing operations happen in the operation classes, which simulate the database activities.



All the Operation classes should not contain __init__() and __str__() methods. All the methods declared in the operation class can be static. However, for simplicity, we still implement them as normal class methods. When you use these methods, simply declare an empty operation class object and use the object to invoke methods.

2.1. User Class

User is the base class for Customer and Admin classes	
Required Class Variables	● *N/A (You can add if you need)*

<table>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.1.1. __init__()
Constructs a user object.</td></tr>
<tr><td>Positional Arguments</td><td>● user_id(must be unique, format: u_10 digits, such as u_1234567890, u_0000000001)
● user_name
● user_password
● user_register_time(default value: “00-00-0000_00:00:00”, format: “DD-MM-YYYY_HH:MM:SS””)
● user_role(default value: “customer”)</td></tr>
<tr><td>Returns</td><td>● N/A</td></tr>
</table>
*All positional arguments of the constructor must have a default value.
<table>
<tr><td colspan="2">2.1.2. __str__()
Return the user Information as a formatted string.</td></tr>
<tr><td>Positional Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● String returned in the format of (xxx is demo value, not the real value):
“{‘user_id’:’u_1234567890’, ‘user_name’:’xxx’, ‘user_password’:’xxx’, ‘user_register_time’:’xxx’, ‘user_role’:’customer’}”</td></tr>
</table>
<u>Note:</u>
● All the users are saved in the file “data/users.txt”.
</td></tr>
</table>

2.2. Customer Class

<table>
<tr><td colspan="2">Customer class inherits from the User class</td></tr>
<tr><td>Required Class Variables</td><td>● N/A (You can add if you need)</td></tr>
</table>

<table>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.2.1. __init__()
Constructs a customer object.</td></tr>
<tr><td>Positional Arguments</td><td>

user_id(must be unique, format: u_10 digits, such as u_1234567890)
user_name
user_password
user_register_time(default value: “00-00-0000_00:00:00”, format: “DD-MM-YYYY_HH:MM:SS””)
user_role(default value: “customer”)
user_email
user_mobile

</td></tr>
<tr><td>Returns</td><td>N/A</td></tr>
</table>
*All positional arguments of the constructor must have a default value.
<table>
<tr><td colspan="2">2.2.2. __str__()
Return the customer information as a formatted string.</td></tr>
<tr><td>Positional Arguments</td><td>N/A</td></tr>
<tr><td>Returns</td><td>String returned in the format of (xxx is demo value, not the real value):
“{‘user_id’:’u_1234567890’, ‘user_name’:’xxx’, ‘user_password’:’xxx’, ‘user_register_time’:’xxx’, ‘user_role’:’customer’, ‘user_email’:’xxxxxxxx@gmail.com’, ‘user_mobbile’:’0412345689’}”</td></tr>
</table>
<u>Note:</u>
All the customers are saved in the file “data/users.txt”.
</td></tr>
</table>

2.3. Admin Class

Admin class inherits from the User class

<table>
<tr><td>Required Class Variables</td><td>● N/A (You can add if you need)</td></tr>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.3.1. __init__()
Constructs an admin object.</td></tr>
<tr><td>Positional Arguments</td><td>● user_id(must be unique, format: u_10 digits, such as u_1234567890)
● user_name
● user_password
● user_register_time(default value: “00-00-0000_00:00:00”, format: “DD-MM-YYYY_HH:MM:SS””)
● user_role(default value: “admin”)</td></tr>
<tr><td>Returns</td><td>● N/A</td></tr>
</table>
*All positional arguments of the constructor must have a default value.
<table>
<tr><td colspan="2">2.3.2. __str__()
Return all the admin’s attributes as a formatted string.</td></tr>
<tr><td>Positional Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● String returned in the format of (xxx is demo value, not the real value):
“{‘user_id’:’u_1234567890’, ‘user_name’:’xxx’, ‘user_password’:’xxx’, ‘user_register_time’:’xxx’, ‘user_role’:’admin’}”</td></tr>
</table>
<u>Note:</u>
● All the admins are saved in the file “data/users.txt”.
</td></tr>
</table>

2.4. Product Class

Model class of product.

<table>
<tr><td>Required Class Variables</td><td>● N/A (You can add if you need)</td></tr>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.4.1. __init__()
Constructs a product object.</td></tr>
<tr><td>Positional Arguments</td><td>● pro_id (must be unique)
● pro_model
● pro_category
● pro_name
● pro_current_price
● pro_raw_price
● pro_discount
● pro_likes_count</td></tr>
<tr><td>Returns</td><td>● N/A</td></tr>
</table>
*All positional arguments of the constructor must have a default value.
<table>
<tr><td colspan="2">2.4.2. __str__()
Return the product information as a formatted string.</td></tr>
<tr><td>Positional Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● String returned in the format of (xxx is demo value, not the real value):
“{‘pro_id’:’xxx’, ‘pro_model’:’xxx’, ‘pro_category’:’xxx’, ‘pro_name’:’xxx’, ‘pro_current_price’:’xxx’, ‘pro_raw_price’:’xxx’, ‘pro_discount’:’xxx’, ‘pro_likes_count’:’xxx’}”</td></tr>
</table>
<u>Note:</u>
● All the products are saved in the files “data/products.txt”.
</td></tr>
</table>

2.5. Order Class

Model class of order.

<table>
<tr><td>Required Class Variables</td><td>● N/A (You can add if you need)</td></tr>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.5.1. __init__()
Constructs a unit object.</td></tr>
<tr><td>Positional Arguments</td><td>● order_id (must be a unique integer, the format is o_5 digits such as u_12345)
● user_id
● pro_id
● order_time (default value: “00-00-0000_00:00:00”, format: “DD-MM-YYYY_HH:MM:SS”)</td></tr>
<tr><td>Returns</td><td>● N/A</td></tr>
</table>
*All positional arguments of the constructor must have a default value.
<table>
<tr><td colspan="2">2.5.2. __str__()
Return the order information as a formatted string.</td></tr>
<tr><td>Positional Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● String returned in the format of (xxx is demo value, not the real value):
“{‘order_id’:’xxx’, ‘user_id’:’xxx’, ‘pro_id’:’xxx’, ‘order_time’:’xxx’}”</td></tr>
</table>
<u>Note:</u>
● All the orders are saved in the file “data/orders.txt”.
● To reduce the program difficulty, we assume each order only has one product.
</td></tr>
</table>

2.6. UserOperation Class

Contains all the operations related to a user.

<table>
<tr><td>Required Class Variables</td><td>● N/A (You can add if you need)</td></tr>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.6.1. generate_unique_user_id()
This method is used to generate and return a 10-digit unique user id starting with ‘u_’ every time when a new user is registered.</td></tr>
<tr><td>Positional Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● a string value in the format 'u_10digits', where 'u' is a prefix and it is followed by a underscore and a 10-digit numerical value. For example, the returned string will follow the pattern 'u_1234567890'."</td></tr>
<tr><td colspan="2">2.6.2. encrypt_password() Encode a user-provided password.
Encryption steps:
1. Generate a random string with a length equal to two times the length of the user-provided password. The random string should consist of characters chosen from a set of 26 lowercase letters, 26 uppercase letters, and 10 digits (i.e., a-zA-Z0-9).
2. Combine the random string and the input password text to create an encrypted password, following the rule of selecting two letters sequentially from the random string and appending one letter from the input password. This process is repeated until all the characters in the user-provided password are encrypted. Finally, add "^^" at the beginning and "$$" at the end of the encrypted password to indicate its beginning and ending respectively.</td></tr>
<tr><td>Positional Arguments</td><td>● user_password</td></tr>
<tr><td>Returns</td><td>● Encrypted password</td></tr>
</table>
<u>*Examples:</u>
● User provided password: “admin1”
</td></tr>
</table>

Generated random string: “qwyroioadfbh”

Encrypted password: “^^qw**a**yr**d**oi**m**oa**i**df**n**bh**1**$$”

- User provided password: “FIT9136”

  Generated random string: “q0FuYI67Tf395n1fi3PA6”

  Encrypted password: “^^q0**F**uY**I**67**T**f3**9**5n**1**fi**3**PA**6**$$”

2.6.3. decrypt_password()

Decode the encrypted password with a similar rule as the encryption method.

Positional Arguments	● *encrypted_password*
Returns	● user-provided password

2.6.4. check_username_exist()

Verify whether a user is already registered or exists in the system.

Positional Argument	● *user_name*
Returns	● True (exist) / False (not exist)

2.6.5. validate_username()

Validate the user's name. The name should only contain letters or underscores, and its length should be at least 5 characters.

Positional Arguments	● *user_name*
Returns	● True/False

2.6.6. validate_password()

Validate the user's password. The password should contain at least one letter (this letter can be either uppercase or lowercase) and one number. The length of the password must be greater than or equal to 5 characters.

Positional Argument	● *user_password*

Returns	● True/False

2.6.7. login()

	Verify the provided user's name and password combination against stored user data to determine the authorization status for accessing the system.
Positional Argument	● *user_name* ● *user_password*
Returns	● A Customer/Admin object

2.7. Customer Operation Class

Contains all the operations related to the customer.	
Required Class Variables	• *N/A*

<table>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.7.1. validate_email()
Validate the provided email address format. An email address consists of four parts:
Username: The part of the email address before the @ symbol.
@ symbol: Separates the username and domain name.
Domain name: Refers to the mail server that stores or routes the email.
Dot (.): Separates a portion of the address from the domain name.</td></tr>
<tr><td>Positional Arguments</td><td>● user_email</td></tr>
<tr><td>Returns</td><td>● True/False</td></tr>
<tr><td colspan="2">2.7.2. validate_mobile()
Validate the provided mobile number format. The mobile number should be exactly 10 digits long, consisting only of numbers, and starting with either '04' or '03'.</td></tr>
<tr><td>Positional Arguments</td><td>● user_mobile</td></tr>
<tr><td>Returns</td><td>● True/False</td></tr>
<tr><td colspan="2">2.7.3. register_customer()
Save the information of the new customer into the data/users.txt file.</td></tr>
<tr><td>Positional Arguments</td><td>● user_name
● user_password
● user_email
● user_mobile</td></tr>
<tr><td>Returns</td><td>● True (success) / False (failure)</td></tr>
</table>
</td></tr>
</table>

Notes:

- Need to apply validations in this method to make sure all the values are valid. If not, return false.
- If the user_name exists in the database, return False.
- A unique user id is required when registering a new user.
- Register time can be obtained by using the time library.
- If the user registers successfully, return true and write the customer info into the database (the data/users.txt file) in the same format as the str() method of the customer class.

2.7.4. update_profile()

Update the given customer object's attribute value. According to different attributes, it is necessary to perform the validations to control the input value. If the input value is invalid, return false. If it is a valid input, the changes should be written into the data/users.txt file immediately.

Positional Arguments	● *attribute_name* ● *value* ● *customer_object*
Returns	● True(updated)/False(failed)

2.7.5. delete_customer()

Delete the customer from the data/users.txt file based on the provided customer_id.

Positional Arguments	● *customer_id*
Returns	● True(deleted)/False(failed)

2.7.6. get_customer_list()

Retrieve one page of customers from the data/users.txt. One page contains a maximum of 10 customers.

Positional Argument	● *page_number*

Returns	● a tuple including a list of customers objects and the total number of pages. For example, ([Customer1, Customer2,...., Customer10], page_number, total_page).

	***Example:** Assuming there are 35 customers listed in data/users.txt, calling the get_customer_list(2) method will return customers 11 to 20. The total number of pages is 4.*

2.7.7. delete_all_customers() Removes all the customers from the data/users.txt file.	
Positional Argument	● *N/A*
Returns	● N/A

2.8. AdminOperation Class

<table>
<tr><td colspan="2">Contains all the operations related to the admin.</td></tr>
<tr><td>Required Class Variables</td><td>● N/A</td></tr>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.8.1. register_admin()
Commonly in a system, the admin account should not allow users to register by themselves. We add this function to manually create an admin account. This function should be called every time you run the system. The same admin account should not be registered multiple times. In this method, you need to write the admin account info into the database.</td></tr>
<tr><td>Positional Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● N/A</td></tr>
</table>
</td></tr>
</table>

2.9. ProductOperation Class

Contains all the operations related to the product.	
Required Class Variables	● *N/A*

<table>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.9.1. extract_products_from_files()
Extracts product information from the given product data files. The data files are csv files (in source/*.csv) which contain many attributes. We only retrieve the necessary data based on the Product class design. The data format is “{‘pro_id’:’xxx’, ‘pro_model’:’xxx’, ‘pro_category’:’xxx’, ‘pro_name’:’xxx’, ‘pro_current_price’:’xxx’, ‘pro_raw_price’:’xxx’, ‘pro_discount’:’xxx’, ‘pro_likes_count’:’xxx’}”.
The data is saved into the data/products.txt file. The data/products.txt file will be used as the file storing all the product information.</td></tr>
<tr><td>Positional Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● N/A</td></tr>
<tr><td colspan="2">2.9.2. get_product_list()
This method retrieves one page of products from the database. One page contains a maximum of 10 items from data/products.txt file.</td></tr>
<tr><td>Positional Arguments</td><td>● page_number</td></tr>
<tr><td>Returns</td><td>● A tuple including a list of products objects and the total number of pages. For example, ([Product1,Product2,Product3,...Product10],page_number, total_page).</td></tr>
<tr><td colspan="2">2.9.3. delete_product()
This method can delete the product info from the system (i.e., data/products.txt) based on the provided product_id.</td></tr>
<tr><td>Positional Arguments</td><td>● product_id</td></tr>
<tr><td>Returns</td><td>● True/False</td></tr>
</table>
</td></tr>
</table>

2.9.4. get_product_list_by_keyword()

This method retrieves all the products whose name contains the keyword (case insensitive).

Positional Arguments	● *keyword*
Returns	● The return result will be a list of product objects. No page limitation.

2.9.5. get_product_by_id()

This method returns one product object based on the given product_id.

Positional Arguments	● *product_id*
Returns	● A product object or None if cannot be found.

2.9.6. generate_category_figure()

This method generates a bar chart that shows the total number of products for each category in descending order. The figure is saved into the data/figure folder.

Positional Argument	● *N/A*
Returns	● N/A

2.9.7. generate_discount_figure()

This method generates a pie chart that shows the proportion of products that have a discount value less than 30, between 30 and 60 inclusive, and greater than 60. The figure is saved into the data/figure folder.

Positional Argument	● *N/A*

Returns	• N/A

2.9.8. generate_likes_count_figure()

This method generates a chart (you think is the most suitable) displaying the sum of products' likes_count for each category in ascending order. The figure is saved into the data/figure folder.

Positional Argument	• *N/A*
Returns	• N/A

2.9.9. generate_discount_likes_count_figure()

This method generates a scatter chart showing the relationship between likes_count and discount for all products. The figure is saved into the data/figure folder.

Positional Arguments	• *N/A*
Returns	• N/A

2.9.10. delete_all_products()

This method removes all the product data in the data/products.txt file.

Positional Argument	• *N/A*
Returns	• N/A

Notes:

- All the figure names can be {the method name}.png/jpg.

2.10. OrderOperation Class

Contains all the operations related to the order.

<table>
<tr><td>Required Class Variables</td><td>● N/A</td></tr>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.10.1. generate_unique_order_id()
This method is used to generate and return a 5 digit unique order id starting with “o_” every time when a new order is created. All the order information is saved inside the database. It is required to check this file when generating a new order id to make sure there is no duplicate.</td></tr>
<tr><td>Positional Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● This method returns a string result such as o_12345.</td></tr>
<tr><td colspan="2">2.10.2. create_an_order()
Every time creating a new order, a unique order id needs to be generated. Use the time library to get the current time. The order data is saved into the data/orders.txt file.</td></tr>
<tr><td>Positional Arguments</td><td>● customer_id
● product_id
● create_time (use the current time if not provided)</td></tr>
<tr><td>Returns</td><td>● True/False</td></tr>
<tr><td colspan="2">2.10.3. delete_order()
This method deletes the order info from the data/orders.txt file based on the provided order_id.</td></tr>
<tr><td>Positional Arguments</td><td>● order_id</td></tr>
<tr><td>Returns</td><td>● True/False</td></tr>
</table>
</td></tr>
</table>

2.10.4. get_order_list()

This method retrieves one page of orders from the database which belongs to the given customer. One page contains a maximum of 10 items.

Positional Arguments	● *customer_id* ● *page_number*
Returns	● This function returns a tuple including a list of order objects and the total number of pages. For example, ([Order(), Order(), Order()...], page_number, total_page).

2.10.5. generate_test_order_data()

Since manually inputting multiple order data is time-consuming, we use this method to automatically generate some test data. In this method, you need to create 10 customers and randomly generate 50 to 200 orders for each customer. Try to control the order time for each order and let the time be scattered into different 12 months of the year. The product of each order is obtained randomly from the database. Use some functions defined in previous tasks.

Positional Arguments	● *N/A*
Returns	● N/A

2.10.6. generate_single_customer_consumption_figure()

Generate a graph(any type of chart) to show the consumption(sum of order price) of 12 different months (only consider month value, ignore year) for the given customer.

Positional Argument	● *customer_id*
Returns	● N/A

2.10.7. generate_all_customers_consumption_figure()

Generate a graph(any type of chart) to show the consumption(sum of order price) of 12 different months (only consider month value, ignore year) for all customers.

Positional Argument	● *N/A*

Returns	● N/A

2.10.8. generate_all_top_10_best_sellers_figure()
Generate a graph to show the top 10 best-selling products and sort the result in descending order.

Positional Argument	● *N/A*
Returns	● N/A

2.10.9. delete_all_orders()
This method removes all the data in the data/orders.txt file.

Positional Arguments	● *N/A*
Returns	● N/A

Notes:

- All the figure names can be {the method name}.png/jpg.

2.11. Interface Class

This Class handles all the I/O operations. All the input(get data from users) / output(print out info) should be defined in this class. No constructor and __str__ methods are needed for this class.	
Required Class Variables	● *N/A*

<table>
<tr><td>Required Methods</td><td>
<table>
<tr><td colspan="2">2.11.1. get_user_input() Accept user input.</td></tr>
<tr><td>Positional Arguments</td><td>● message
● num_of_args</td></tr>
<tr><td>Returns</td><td>● The return result is [“arg1”, “arg2”, “arg3”]. If the number of user’s input arguments is less than the num_of_args, return the rest as empty str “”. For example, the num_of_args=3, but user input is “arg1 arg2”.
The return result will be [“arg1”, “arg2”, “”].</td></tr>
</table>
<u>Notes:</u>

The message is used for the input() function.
The user inputs have only one format with all the arguments connected by a whitespace “ ”. For example, the input could be “arg1 arg2 arg3…”.
The num_of_args determines how many arguments can be accepted and used. If users input more than num_of_args arguments into the system, ignore the others and only use the num_of_args arguments. For instance, the num_of_args=3, but user input is “arg1 arg2 arg3 arg4”. Only use the first 3 args and ignore the last one.

<table>
<tr><td colspan="2">2.11.2. main_menu()
Display the login menu, which includes three options: (1) Login, (2) Register, and (3) Quit. The admin account cannot be registered.</td></tr>
<tr><td>Positional Arguments</td><td>● N/A</td></tr>
<tr><td>Returns</td><td>● N/A</td></tr>
<tr><td colspan="2">2.11.3. admin_menu()
Display the admin menu, which includes seven options:</td></tr>
</table>
</td></tr>
</table>

(1). Show products
(2). Add customers
(3). Show customers
(4). Show orders
(5). Generate test data
(6). Generate all statistical figures
(7). Delete all data
(8). Logout

Positional Arguments	• *N/A*
Returns	• N/A

2.11.4. customer_menu()

Display the customer menu, which includes six options:
(1). Show profile
(2). Update profile
(3). Show products (user input could be “3 keyword” or “3”)
(4). Show history orders
(5). Generate all consumption figures (6).
Logout

Positional Arguments	• *N/A*
Returns	• N/A

2.11.5. show_list()

Prints out the different types of list. In this system, there are three types of lists - “Customer”, “Product” and “Order”. If user_role is “customer”, only product list and order list can be displayed. If user_role is “admin”, all types of list can be displayed. The output list should also show the row number, the page number and the total page number.

Positional Arguments	● *user_role* ● *list_type* ● *object_list (the format is [[Customer1, Customer2, …Customer10], page_number, total_page]. For product and order, the format is similar)*
Returns	● N/A

2.11.6. print_error_message()
Prints out an error message and shows where the error occurred. For example, when the login has an error, you can call this function print_error_message("UserOperation.login", "username or password incorrect").

Positional Argument	● *error_source* ● *error_message*
Returns	● N/A

2.11.7. print_message()
Print out the given message.

Positional Argument	● *message*
Returns	● N/A

2.11.8. print_object()
Print out the object using the str() function.

Positional Argument	● *target_object*
Returns	● N/A

2.12. Main File

In this file, you will construct the main control logic for the application(e.g. main() function). The design and implementation is up to you but must include the menu items outlined in section **2.11.2,** section **2.11.3, and** section **2.11.4** using the classes and methods implemented.

You must ensure that your menu and control logic handles exceptions appropriately.

You can break down your code to several functions if you wish but you need to call the extra-defined functions in the main function. In the if __name__=="__main__" part, only call main()

function. **Your tutor will only run your main.py file.**

For each operation that the user performs, try to give enough instructional messages.

For all the tasks above, you can change the class/function/variables name to follow your own naming conventions. It is allowed to add more class variables, methods in classes and functions in the main file. However, you need to make sure all the required methods and functions are implemented. Any unused code in your application will receive mark penalties. Besides, since there could be many file reading/writing operations, you can decide to read/write files in each operation or save the reading/writing content into temporary variables. Only need to make sure all the data is persisted into files and no data loss. Do not add extra classes/files.

2.13. User Manual

It is required to provide user instructions saved into a file named **userManual_{studentid}.pdf** which describes how to use your application. In your pdf, list all the commands used to reach the tasks listed in section **2.11.2,** section **2.11.3, and** section **2.11.4**. Your marker will follow your manual to test all the functions. Make sure you have also demoed the special cases like user enrollment failure if the unit capacity is reached.

Please do not show too much content in this document. No more than 5 pages.

3. Do and Do NOT

Do	DoNOT
• Maintain appropriate citing and referencing[1], • Get support early from this unit and other services within the university, • Apply for special consideration or for extensions[2] early if needed.	• Leave your assignment in draft mode (assignments in draft mode will not be marked), • Submit late (10% daily penalty applies)[3], • Attempt to submit after 7 days of the due date (they will not be accepted), unless you have special consideration.

3.1. Important Notes:

- DO NOT use absolute paths and follow the exact structure for the path as provided in the examples in each section. All the path issues that cause your program crash or exception will lead to no mark for functionality.

- You must implement all required methods but you may add additional methods if you require.

- Please make sure your file reading/writing operations include encoding type **utf8**. Changing the application running environment could cause issues if you do not have the encoding type. Any character encoding issues/exceptions will cause serious mark deduction.
- If one method is not working and it hinders the program from continuing running to show other functionalities, the following functionalities will get no mark. For example, if your system cannot login, the functionality that needs to be marked after login will get no mark and you will only get marks for the menu part. Therefore, it is important to finish the methods one by one and make sure they can work properly.

- If any exception/errors happen when running your program, you will lose 50% of allocated function logic marks. For example, if the main menu function returns any error, then the maximum mark you can get is 5% instead of 10% in the function logic.

- Add correct validation and output messages to make your code more user-friendly to users.

- The assignment must be done using the **Pycharm**, **Python Version 3.9**.

- The Python code for this assignment must be implemented according to the PEP 8-Style Guide for Python Code.

- The allowed libraries are **random**, **math**, **os**, **string**, **time**, **numpy**, **pandas**, **matplotlib**, **re**. You will not receive marks for the components if you use any other libraries apart from the mentioned library.

- In this assignment, it is required to add as many validation codes as you can to make sure your system always works. Any exception that happens in your system will lead to a mark deduction.

- All the Model classes and Operation classes should not contain any input/output operations, such as print()/input().

- Commenting on your code is an essential part of the assessment criteria. In addition to inline and function commenting on your code, you should include comments at the beginning of your program file which specify your name, Student ID, the creation

date, and the last modified date of the program, as well as a high-level description of the program.

- This assignment cannot be completed in a few days and requires students to apply what we learn each week as we move closer to the submission date. Please remember to show your progress weekly to your tutor.

- You must keep up to date with the Moodle Ed Assignment 3 forum where further clarifications may be posted (this forum is to be treated as your client).

- Please be careful to ensure you do not publicly post anything which includes your reasoning, logic or any part of your work to this forum, doing so violates Monash plagiarism/ collusion rules and has significant academic penalties. Use private posts or email your allocated tutor to raise questions that may reveal part of your reasoning or solution.

- **In this Assessments, you must NOT use generative artificial intelligence (AI) to generate any materials or content in relation to the assessment task.**

4. Submission Requirements

The following files are to be submitted and must exist in your FITGITLab server repo:

- 12 **.py** files (i.e., {model}.py, {operation}.py, and main.py)**.**
    - A template, A3_student_template.zip, is available on the Moodle Assessments page. You have to use this template.
- A **userManual_{studentid}.pdf** file**.**

The above files must be compressed to a .zip file named **ass3_{studentid}.zip** and submitted via Moodle.
The **.py** files must also have been pushed to your FIT GitLab server repo with an appropriate history as you developed your solutions. Please ensure your committed comments are meaningful. **Do NOT** need to push the history of userManual_{studentid}.pdf file to the FIT GitLab server. Only push the final version of PDF file. **DO NOT** push the .zip file.

- No submissions will be accepted via email,

- Please note we **cannot mark any work on the GitLab Server**, you need to ensure that you submit correctly via Moodle since it is only in this process that you complete the required student declaration without which work cannot be assessed.

- It is your responsibility to **ENSURE** that the submitted files are the correct files. We strongly recommend after uploading a submission, and prior to actually submitting in Moodle, that you download the submission and double-check its contents.

- Please **carefully** read the documentation under the "**Special Consideration**" and "**Assignment Task Submission**" on the Moodle Assessments page which covers things such as extensions, correct submission, and resubmission.

- Please note, if you need to resubmit, you cannot depend on your tutors' availability, for this reason, please be VERY CAREFUL with your submission. **It is strongly recommended that you submit several hours before due to avoid such issues.**

- **Marks will be deducted for any of these requirements that are not strictly complied with.**

5. Academic Integrity

Section 1.9:

Students are responsible for their own good academic practice and must:

- undertake their studies and research responsibly and with honesty and integrity;
- credit the work of others and seek permission to use that work where required;
- not plagiarise, cheat or falsify their work;
- ensure that their work is not falsified;
- not resubmit any assessment they have previously submitted, without the permission of the chief examiner; appropriately acknowledge the work of others;
- take reasonable steps to ensure that other students are unable to copy or misuse their work; and
- be aware of and comply with University regulations, policies and procedures relating to academic integrity.

and **Section 2.9:**

any course-related material produced by students themselves or other students (such as class notes, past assignments), nor to receive such material, without the permission of the chief examiner. The penalties for breaches of academic misconduct include

- a zero mark for the assessment task
- a zero mark for the unit
- suspension from the course
- exclusion from the University.

Where a penalty or disciplinary action is applied, the outcome is recorded and kept for seven

years, or for 15 years if the penalty was excluded.

6. Marking Guide

Your work will be marked as per the following:

- Application Functionality - 45 Marks
    - Your tutor will run your main.py to check all the basic operations listed in menu items outlined in section **2.11.2,** section **2.11.3, and** section **2.11.4.**
    - If your program crashes/ any operation won't be implemented correctly, then 0 mark for this criteria.
- Classes Implementation (methods and attributes) - 20 Marks
    - User Class - 1 Mark
    - Customer Class - 2 Marks
    - Admin Class - 2 Marks
    - Product Class - 2 Marks
    - Order Class - 2 Marks

    - UserOperation Class - 2 Marks
    - CustomerOperation Class - 2 Marks
    - AdminOperation Class - 2 Marks
    - ProductOperation Class - 2 Marks
    - OrderOperation Class - 2 Marks

    - IOInterface Class - 1 Marks
- Main File Design - 15 Marks
    - Program Logic and validations - 15 Marks
- User Manual - 5 Marks
- Code Architecture and Style, Documentation - 15 Marks
- Penalty - up to 20 marks
    - Missing submission requirements