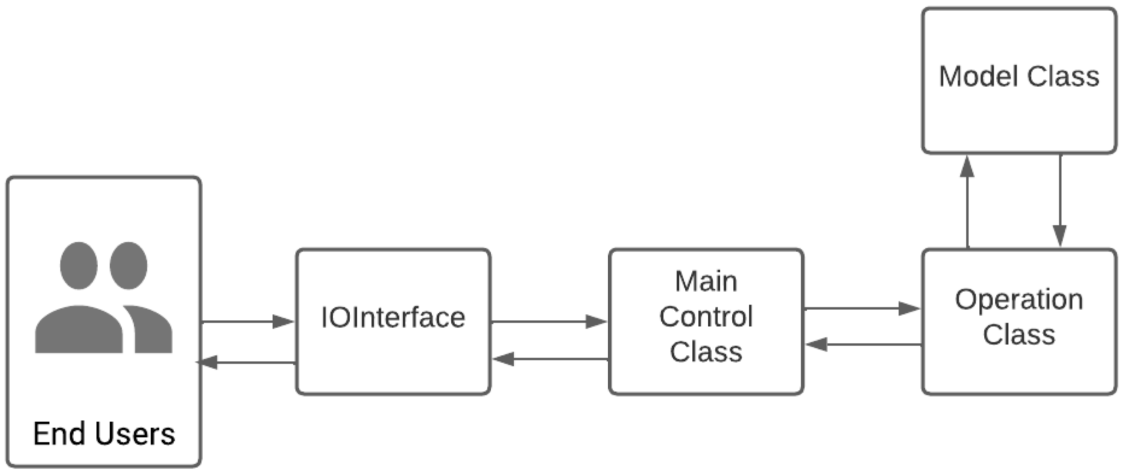
2. Instruction

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 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)*
Required Methods	

2.1.1. __init__() Constructs a user object.	
Positional Arguments	• *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”)
Returns	• N/A

**All positional arguments of the constructor must have a default value.*

2.1.2. __str__() Return the user Information as a formatted string.	
Positional Arguments	• N/A
Returns	• 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’}”

Note:

- All the users are saved in the file “data/users.txt”.

2.2. Customer Class

Customer class inherits from the User class	
Required Class Variables	● *N/A (You can add if you need)*
Required Methods	See 2.2.1 and 2.2.2 below

2.2.1. __init__() Constructs a customer object.	
Positional Arguments	● *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*
Returns	● N/A

**All positional arguments of the constructor must have a default value.*

2.2.2. __str__() Return the customer information as a formatted string.	
Positional Arguments	● N/A
Returns	● 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’}”

Note:
- All the customers are saved in the file “data/users.txt”.

2.3. Admin Class

Admin class inherits from the User class	
Required Class Variables	● *N/A (You can add if you need)*

Required Methods

2.3.1. __init__() Constructs an admin object.	
Positional Arguments	● *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”)
Returns	● N/A

**All positional arguments of the constructor must have a default value.*

2.3.2. __str__() Return all the admin’s attributes as a formatted string.	
Positional Arguments	● N/A
Returns	● 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’}”

Note:
- All the admins are saved in the file “data/users.txt”.

2.4. Product Class

Model class of product.	
Required Class Variables	● *N/A (You can add if you need)*
Required Methods	

2.4.1. __init__() Constructs a product object.	
Positional Arguments	● *pro_id* (must be unique) ● *pro_model* ● *pro_category* ● *pro_name* ● *pro_current_price* ● *pro_raw_price* ● *pro_discount* ● *pro_likes_count*
Returns	● N/A

**All positional arguments of the constructor must have a default value.*

2.4.2. __str__() Return the product information as a formatted string.	
Positional Arguments	● N/A
Returns	● 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’}”

Note:

- All the products are saved in the files “data/products.txt”.

2.5. Order Class

Model class of order.	
Required Class Variables	● *N/A (You can add if you need)*
Required Methods	

2.5.1. __init__() Constructs a unit object.	
Positional Arguments	● *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”)
Returns	● N/A

**All positional arguments of the constructor must have a default value.*

2.5.2. __str__() Return the order information as a formatted string.	
Positional Arguments	● N/A
Returns	● 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’}”

Note:

- All the orders are saved in the file “data/orders.txt”.
- To reduce the program difficulty, we assume each order only has one product.

2.6. UserOperation Class

Contains all the operations related to a user.	
Required Class Variables	● *N/A (You can add if you need)*
Required Methods	

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.

Positional Arguments	● *N/A*
Returns	● 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'."

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.

Positional Arguments	● user_password
Returns	● Encrypted password

Examples:

- User provided password: “admin1”

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*
Required Methods	

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.	
Positional Arguments	• *user_email*
Returns	• True/False

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'.	
Positional Arguments	• user_mobile
Returns	• True/False

2.7.3. register_customer() Save the information of the new customer into the data/users.txt file.	
Positional Arguments	• *user_name* • *user_password* • *user_email* • *user_mobile*
Returns	• True (success) / False (failure)

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

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

2.8.1. register_admin()	
Positional Arguments	• *N/A*
Returns	• N/A

2.9. ProductOperation Class

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

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

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.	
Positional Arguments	● *page_number*
Returns	● A tuple including a list of products objects and the total number of pages. For example, ([Product1,Product2,Product3,...Product10],page_number, total_page).

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.	
Positional Arguments	● *product_id*
Returns	● True/False

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

<u>Notes:</u>

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

2.10. OrderOperation Class

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

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.	
Positional Arguments	● *N/A*
Returns	● This method returns a string result such as o_12345.

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.	
Positional Arguments	● *customer_id* ● *product_id* ● *create_time (use the current time if not provided)*
Returns	● True/False

2.10.3. delete_order() This method deletes the order info from the data/orders.txt file based on the provided order_id.	
Positional Arguments	● *order_id*
Returns	● True/False

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*
Required Methods	

2.11.1. get_user_input()
Accept user input.

Positional Arguments	• *message* • *num_of_args*
Returns	• 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”, “”].

Notes:
- 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.

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.

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

2.11.3. admin_menu()
Display the admin menu, which includes seven options:

(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.