**SBA Report**

**2022-2024**

Chapter 0

**By Tse Chun To 6E24**

About the System

1. **Introduction:**

Online shopping is the latest trend in which provides a variety of goods with **high convenience**. The biggest advantage of **online shopping** comparing to the traditional shopping in the supermarket is that online shopping allows users to purchase goods online which reduce the time cost and beyond the time, geographical barrier as users can check the goods of different shops around the world at any time they want. Notwithstanding, the great variety of goods provided by traditional shops is an unique advantage and the key to remaining competitive towards the trend. If the two advantages combined forming an online shopping platform providing a great variety of goods will be a step in the right direction.

* 1. **Problem identification:**

1. Time and geographical limitations:

People have to spend a lot of time going to the shop and looking for the desired

goods. Moreover, the time needed to go to the shop increases positively related

to the distance to the shop. Of equal importance is that queuing in physical

shops take a lot of time , which may even takes longer than looking for goods in

peak hours. A lot of unnecessary time is spent on travelling.

2. Unable to purchase the goods

For the elderly, going to the market or shop to buy the desired goods and carry

the heavy goods back to their home will be a problem for them due to their

limitation in stamina and strength. There may also be a potential problem where

elderly may get hurt while carrying out the goods.

For the busy white-collar workers, they lack time to go to the market or shop to

choose their desired goods due to their packed schedule.

3. The reducing of competitive for traditional market or shop

Given the convenience of online shopping platforms, traditional shops may

provide some rare goods but not as much as the online shopping platform does

on the general daily necessities.

4. Unable to satisfy customers’ desire, concern

As the online shopping platform has an array of limitations, for instance

difficulties in transporting food that is easily spoiled or raw meat, long delivery

time and the concerns of fake goods, online shopping platform faces a series of

potential problems.

**0.2 Aims and objectives:**

A good online shopping platform should combine the advantages of both

traditional physical shops and the online shopping platform, in which the

customers can enjoy the great variety of goods in online shopping platforms and

allow them to use their familiar way of shopping i.e. buying from the physical

shops. Also, the sales of the online shopping platform can be promoted through

collaboration with different famous chain brand shops or small individual shops.

**0.3 Target users and their needs:**

1. Elderly

They may not be physically able to purchase goods through physical shops

and carry the heavy goods back home. They need a delivery service where

they can shop without going to the physical shop and of the utmost

importance is that they don’t have to carry heavy goods back home by

themselves.

2. Busy people

Busy people, for example white-collar workers, often get off their work late

or even at midnight. They lack time to buy goods or even daily necessities.

They need a delivery service which delivers their desired goods to their home

or a planned pickup of their desired goods in physical shops, reducing the

time of looking for desired goods and queuing to check out.

3. Local shop owner

Local shop owners lack a platform to promote their goods or sell their goods.

By using the online shopping platform, they can put their goods online and

sell their goods through online shopping platform in which the great brand

built allowing more attraction to the customers and reach to more potential

customers.

4. Online shopping platform owner

Online shopping platform owner can promote the online shopping platform by the brandname of the franchises (local shop) shop and can increase their occupational percentage in the highly-competitive online sales market in a short period of time.

The owner can also easily sell mtheir products to the people with demand hence their sales revenue can be increased

* 1. **Functions of the system**

1. Customer

- customer can shop and purchase through an array of goods through online

shopping platforms

- customer can choose to pick-up their purchased goods at the stores owned

by the platform or the franchises (seller) shop

- they can also choose delivery to send their goods to their address

2. Seller

- seller can add goods of their brand to the supermarket (addressing the

problem of insufficient goods variety and promote the sales, fame of the

market)

- seller can also adjust (i.e. managing the stock, price of the goods) their goods

- seller can delete THEIR OWN goods (limited that they can’t delete other

goods – by Permission)

3. Admin

- admin can adjust, add, delete the goods in the shopping markets

- admin can regulate the online shopping platform and adjust accordingly

- admin can check the statistics and analysis the future trending of the

different goods

* 1. **Design Consideration**

1. **Hardware Usage**

A large storage device should be used to store the data of the goods and the users’ information and transaction records.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Desktop Computer | Direct-attached Storage (DAS) | Network Attached  Storage (NAS) |
| Cost | Low | Medium | High |
| Access mode | Direct | Direct | **Centralized** |
| Processing Speed | Low | Highest | High |
| Accessibility | Low | Medium | High |

A **Network Attached Storage (NAS)** is usedbecause of its **centralized access mode** is well-suited for the online shopping system with multiple access given to different users ( More permission access for the admin , less access for seller )

And it has **high processing speed** to allow large amount of data to be processed and show to the user with short response time needed

1. **Backup Device Usage**

A backup device should be used to restore the data after a potential data loss to prevent the breakdown of the transaction and the online shopping platform

|  |  |  |  |
| --- | --- | --- | --- |
|  | External Hard drive ( SSD ) | Network Attached  Storage (NAS) | Cloud Storage |
| Setup Cost | Low | High | Highest |
| Backup/recovery speed | Highest | High | Low |
| Backup Frequency | Low | High | **Highest** |

A **Cloud Storage** is used because of its **high backup frequncy** to minimize the loss of data after a data loss and it allows multiple access of backups at **anywhere anytime** which provide convenience and allow fast restore of backup.

The **storage size** of cloud storage can be adjusted in fast according to the demand due to its high Scalability.

1. **User Interface**

A user interface(UI) is used for the users to manage or buy goods in the online shopping platform .

|  |  |  |  |
| --- | --- | --- | --- |
|  | Command Line Interface | Menu-driven Interface | Graphical User interface (GUI) |
| Accessibility | **Low** | **Medium** | **Highest** |
| Develop Difficulty | **Low** | **Medium** | **Highest** |
| Need to Learn | **Yes** | **No** | **No** |

As a GUI is difficult to develop , in consideration of the accessibility , A **Menu-driven interface** will be used.

# Chatper 1

Data storage

* 1. **Modular Approach**

###### Module import ######

import csv

import re

import time

import datetime

import random

import uuid

Different modules like CSV is used to **faciliate the read and write of the files** used to store the data for goods , customers and shops while the modules like TIME and DATETIME are used to **simulate realtime response and real-life interaction** ( like calculation the age to check if user is available to buy alcohol according to the real-life date)

* 1. **Data storage in program**

###### Global variable ######

permission\_stat = 0 # 1:customer 2:Shopper 3:Admin

p\_name = "" # the name of the user

p\_bday = "" # the birthday of the user

flag\_bit = True

goods\_data = "D:\SBA\SBA\goods\_info.csv"

pickup\_data = "D:\SBA\SBA\pickup\_data.csv"

purchase\_records = "D:\SBA\SBA\purchase\_records.csv"

###### admin information ######

admin\_name = ["admin"]

admin\_pw = ["123"]

###### seller information ######

seller\_name = ["Garden"]

seller\_pw = ["Garden123"]

###### customer information ######

customer\_name = ["Tommy"]

customer\_pw = ["5212023"]

customer\_bday = [datetime.datetime(2023,5,21)]

shopping\_cart = [["NAME","ID","PRICE","Quantity","Total cost"]]

###### goods information ######

age\_required = ["066"]

The data of the customer , seller and admin are stored inside the program itself as they are **not frequently changed** bu**t frequently used** data for every user hence to facilitate the login of users the information is stored inside the program instead of a independent file . **Birthday** is a unique data for the customer as age ( calculated by real-life time to avoid frequent change of age ) is needed for purchasion of alcohol .

* 1. **Data storage in independent files**
     1. **Data for the goods**

NAME,ID,COMPANY,PRICE,STOCK

Chocolate\_Finger\_Biscuits(240g),001,Garden,10,0

Chocolate\_Cake,002,Maxin,129.9,79

Cola(1000mL),003,Coca\_Cola,12,102

White\_Lotus\_Seed\_Paste\_Mooncake,999,Wing\_Wah,99,7

Water(200mL),006,Vita,9.9,10

Protein\_Powder(1000g),007,MyProtein,595,55

Ion\_Water(500mL),059,Pocari,159,8

Fried\_Noodles,018,Indomie,22,5

Sparkling\_Wine(500mL),066,Choco\_Rosso,80.75,18

Pop\_Pan\_Biscuit,099,Garden,35,100

Family\_Assorted\_Biscuits(1340g),128,Garden,128,65

Mini\_Cream\_Wafers,329,Garden,32.9,56

The data of goods is stored in csv file, which can be read in the module CSV , imported in the 1.0 .

It stores the **NAME,ID,COMPANY,PRICE,STOCK** of the goods available in this online platform for the customers to purchase. Only **ADMIN** and the **SELLER of those goods** can adjust the data of thegoods

* + 1. **Data for the pickup locations**

NAME,ADDRESS,PHONE,START,CLOSE

Welcome,34G\_Cambridge Road\_Kowloon,26003555,0800,0000

Market Place,G/F\_99 Caine Road\_Hong Kong,54236520,0830,2300

Parknshop,23F\_Martin Road\_New Territory,36897584,0600,2000

Meaty,23\_Sheung Wan Market\_Hong Kong,25846500,0500,1900

The data of pickup locations is stored in the csv file. **NAME, ADDRESS, PHONE, START, CLOSE** are stored in the file (START is the opening time of the shop, while CLOSE is the closing time for the shop) which are the data of the shops allowed for the customers to pick up what they purchased after the order online.

* + 1. **Data for the records of purchases**

TID,NAME,ID,PRICE,QUANTITY,TOTAL,CUSTOMER,COMANY,**TIME**

c7830e7b,Chocolate\_Finger\_Biscuits(240g),001,10,1,10.0,Victor,Garden,**2024-09-13 23:07**

363787f9,Chocolate\_Finger\_Biscuits(240g),001,10,1,10.0,Tommy,Garden,**2024-09-13 23:08**

363787f9,Water(200mL),006,9.9,1,9.9,Tommy,Vita,**2024-09-13 23:08**

ac8607cb,Mini\_Cream\_Wafers,329,32.9,10,329.0,Tommy,Garden,**2024-09-13 23:15**

ac8607cb,Water(200mL),006,9.9,8,79.2,Tommy,Vita,**2024-09-13 23:15**

cc6ca169,Family\_Assorted\_Biscuits(1340g),128,128,5,640.0,Victor,Garden,**2024-09-13 23:31**

The purchase records of customers stores

**TID,NAME,ID,PRICE,QUANTITY,TOTAL,CUSTOMER,COMANY,TIME**

Those are the data of the customers’ purchases, the TOTAL is the **total sum** of the transaction , calculated by PRICE times QUANTITY . The TID is the **unique** transaction ID to identify different transactions. **ADMIN** can view the statistical data of those transactions and adjust the stock or price of the goods

Chapter 2

Login of Users

**2.0 Login system**

def login() : # Main log in

    global permission\_stat

    global shopping\_cart

    for i in range(50) : print("=" , end="")

    role = str(input("\n You are : A - Admin / S - Seller / C - Customer / NEW - New account registration( Please fill in short form ) : "))

    if role == "A" : # Admin log in

        while admin\_login() == False :

            pass

        permission\_stat = 3

    elif role == "S" : # Seller log in

        while seller\_login() == False :

            pass

        permission\_stat = 2

    elif role == "C" : # Customer log in

        while customer\_login() == False :

            pass

        permission\_stat = 1

    elif role == "NEW" : # Creating new account

        while new\_user() == False :

            pass

        login()

    else : # Invalid input

        print("Please fill in valid short form (A/S/C/NEW) ! ")

        login()

    shopping\_cart = [["NAME","ID","PRICE","Quantity","Total cost"]] # reset the shopping cart

The users can choose if they are **existing** users (admin, customer, seller) or **register** as a new user by inputting the representing short form (A/C/S/NEW)

While their login function is not correct, they must retry until a valid input is inputted

**Different identities have different login functions**

The function of creating new users is used for NEW users

**2.1 Admin Login**

def admin\_login() :

    global p\_name

    login\_name = str(input("Hi Admin , Your username is : "))

    if login\_name in admin\_name :

        location = find(admin\_name,login\_name)

        login\_pw = str(input("Hi Admin , Your password is : "))

        if admin\_name[location] == login\_name and admin\_pw[location] == login\_pw : # check if the account matches with the password

            print("")

            print("Welcome Admin {} !".format(login\_name))

            p\_name = login\_name

            return True

        else :

            print("Your password is NOT correct , please try again")

    else :

        print("Username does NOT exist , please try again")

    return False

The **login name** is inputted by the user. If the input of the users is in the list of admin login name, A **password** is needed for the user to input to log in If both are correct, the login is successfully for the admin. If the password is not correct or the username does not exist, the user must re-enter the username and the password again

def find(matrix,target) : # To find the index of an item in a list

    for pos , item in enumerate(matrix) :

        if item == target :

            return pos

    return False

A find function is used to find the **index** of the input of username in the list of admin names. Matrix (parameter) will be the admin list, as we are **searching** for the name of the admin, and target (parameter) will be the input of user, which we want to find in the name list of admins.

**OUTPUT of admin login :**

**A black screen with white text

Description automatically generated**

**2.2 Seller Login**

def seller\_login() :

    global p\_name

    login\_name = str(input("Hi Seller , Your username is : "))

    if login\_name in seller\_name :

        location = find(seller\_name,login\_name)

        login\_pw = str(input("Hi Seller , Your password is : "))

        if seller\_name[location] == login\_name and seller\_pw[location] == login\_pw : # check if the account matches with the password

            print("")

            print("Welcome Seller {} !".format(login\_name))

            p\_name = login\_name

            return True

        else :

            print("Your password is NOT correct , please try again")

    else :

        print("Username does NOT exist , please try again")

    return False

The **login name** is inputted by the user. If the input of the users is in the list of seller login name, **password** is needed for the user to input to log in. If both are correct, the login is successfully for the seller.

If the password is not correct or the username does not exist, the user has to re-enter the username and the password again

**OUTPUT of seller login:**

A black screen with white text

Description automatically generated

**1.3 Customer login**

def customer\_login() :

    global p\_bday

    global p\_name

    login\_name = str(input("Hi Customer , Your username is : "))

    if login\_name in customer\_name :

        location = find(customer\_name,login\_name)

        login\_pw = str(input("Hi Customer , Your password is : "))

        if customer\_name[location] == login\_name and customer\_pw[location] == login\_pw : # check if the account matches with the password

            print("")

            print("Welcome Customer {} !".format(login\_name))

            p\_name = login\_name

            p\_bday = customer\_bday[location]

            return True

        else :

            print("Your password is NOT correct , please try again")

    else :

        print("Username does NOT exist , please try again")

    return False

The **login name** is inputted by the user. If the input of the users is in the list of customer login names, **password** is needed for the user to input to log in. If both are correct, the login is successful for the customer

If the password is not correct or the username does not exist, the user has to re-enter the username and the password again

**OUTPUT of customer login:**

A black screen with white text

Description automatically generated

* 1. **New user login**

**1.4.1 - registration**

Two different identities / roles, **seller and customer**, can be registered (**admin cannot be registered** as it involves permission of viewing sensitive information)

If the input name is unique (not in the seller or customer name list respectively), the password needs to be created by the users , which certain rules have to be followed . For the **customer**, a **birthday** must be input for a potential discount and the most important thing is to confirm the customer can **legally** purchase alcohol, as the customer under 18 years of age cannot buy alcohol.

###### New user and PW verification ######

def new\_user() :

    new\_role = str(input("Register : S-Seller / C-Customer : "))

    if new\_role == "S" : # Create new seller

        new\_name = str(input("Your name is : "))

        if new\_name not in seller\_name : # check if the new name does not repeat

            seller\_name.append(new\_name)

        else :

            print("Your name has been used , try again !")

            return False

        new\_pw = str(input("Your password is : "))

        while pw\_check(new\_pw) is False : # data validation

            new\_pw = str(input("Your password is : "))

        seller\_pw.append(new\_pw)

        return True

    elif new\_role == "C" :

        new\_name = str(input("Your name is : "))

        if new\_name not in customer\_name :

            customer\_name.append(new\_name)

        else :

            print("Your name has been used , try again !")

            return False

        new\_pw = str(input("Your password is : "))

        while pw\_check(new\_pw) is False :

            new\_pw = str(input("Your password is : "))

        customer\_pw.append(new\_pw)

        print("Please fill in your birthday for more discount and goods available") # only 18+ can buy alcohol

        bday = date\_input()

        while bday >= datetime.datetime.now() :

            print("The birthday must before today")

            bday = date\_input()

        customer\_bday.append(bday)

        return True

    else :

        print("Please fill in S / C !")

        new\_user()

The date of the customer should be input by the function **date input**

Module **datetime** is used in this function for the use of type “datetime”

If the input of the user is not an integer, the customer must re-enter an integer input (Error handling). This function is also used in date input for other use like delivery date entry ( will be mentioned in following sections ).

def date\_input() :

    try :

        x,y,z= input("Enter the date (YYYY/MM/DD) :").split("/")

        new\_date = datetime.datetime(int(x),int(y),int(z))

        return new\_date

    except ValueError: # error handling

        print("RANGE of Date must between 1/1/1 to 9999/12/31 and The inputs must be INTEGER")

        #return False

        return(date\_input())

**\*Output of this function will be showed in the output of new user registration .**

**1.4.2 Password entry**

**Validation:** The password should follow the following rules:

1. Not **shorter** than 8 characters
2. Not **longer** than 20 characters
3. At least have 1 **capital letter**
4. At least have 1 **number**
5. **No space** in the password

If any rules are against, the re-entry of the password by the user is needed until the input is valid.

**Verification**: The user must input the password twice to confirm the password, to prevent mis-input by the user. The user has to retry the password until the password be the same as the previous password .

def pw\_check(pw) :

    if len(pw) <8:

        print("Your password must be at least 8 characters long !")

        return False

    if len(pw) >20 :

        print("Your password must not exceed 20 characters ! ")

        return False

    if any(char.isupper() for char in pw) :

        pass

    else :

        print("There should be at least one capital letter")

        return False

    if any(char.isdigit() for char in pw) :

        pass

    else :

        print("There should be at least one number")

        return False

    if pw.count(' ') > 0 :

        print("No spaces are allowed")

        return False

    pw\_double\_check = str(input("Please input the password again :")) # data verification by entering the data twice

    while pw\_double\_check != pw :

        print("The second password is NOT the same with the first one")

        pw\_double\_check = str(input("Please input the password again : "))

    return True

**OUTPUT of new user registration and password entry:**

**A screenshot of a computer program

Description automatically generated**

Chapter 2

View Goods

**2.0 Main loop:**

###### Main Loop ######

if \_\_name\_\_ == "\_\_main\_\_" :

    login()

    while flag\_bit :

        menu()

        menu\_control(permission\_stat)

This is the **main loop** of the whole program, where users must log in

The menu displays the available command for the user under his permission, while menu control allows users to use commands to show or change the data (further explanation are presented in this report) and the flag bit indicates the **start/end** of the whole program.

**2.1 Main Menu :**

###### Main Menu ######

def menu() : # showing the commands available for different roles

    time.sleep(1.5)

    print("")

    for i in range(50) : print("\*" , end="")

    print("\nWelcome to the Control Menu , {} \nEnjoy your time in our supermarket ! \n"

          #"Here are the Commands for our online market : \n"

          "\n------GENERAL------\n"

          "V - View the available goods in our market \n"

          "SN - Sort the goods by Name \n"

          "SP - Sort the goods by Price \n"

          "SID - Sort the goods by the ID\n"

          "S - Search for specific goods by name\n"

          "SWID - Search for specific goods with the goods ID\n"

          "F - Filter the goods with unwanted brand".format(p\_name))

    if permission\_stat == 3 or permission\_stat == 2 :

        # seller can add , modify , delete their OWN goods while admin can do without limitation

        print("\n------GOODS MANAGEMENT------\n"

              "A - Add goods of YOUR brand \n"

              "M - Modify the status of goods by YOUR brand \n"

              "D - Delete the goods by YOUR brand")

        if permission\_stat == 3 : # admin can view the backstage data of purchases

            print("\n------DATA VISUALIZATION------\n"

                  "T - Show the Total Sales of all the goods\n"

                  "TQ - Show the Top quantity sold Good\n"

                  "TS - Show the Top Sales Good\n"

                  "TC - Show the Company with the most cases sold\n")

    elif permission\_stat == 1 : # customer can use shopping cart to buy goods

        print("\n------SHOPPING CART------\n"

              "VC - View your shopping Cart \n"

              "AC - Add goods to your shopping Cart \n"

              "EC - Edit your shopping Cart\n"

              "DC - Delete the good in your shopping cart\n"

              "CO - Check Out of your shopping cart")

    print("\n------ACCOUNT------\n"

          "LO - Log out of this account\n"

          "QUIT - Quit this application\n")

The goods can be viewed in different ways , which users can show different data of goods in different ways by different commends . **permission\_stat** is used to store the permission of different users ( 1 – Customer , 2 – Seller , 3 – Admin ) ,

**Customer** can view their shopping cart and check out but the other two identities cannot

**Seller** can edit their own goods data while customer cannot.

**Admin** can edit all the goods data without limitation and view the statistical data while the other two cannot.

Only the user with **permission** to use the command can see the command.

By typing the commend short form respectively, different commands can be executed to view different data. **Only** the commands shown can be executed as the commands without permission to execute will NOT be shown .

**OUTPUT of Menu:**

A screenshot of a computer screen

Description automatically generated

**\*Situation: The menu for the customer**

**2.2 Data visualization**

**2.2.0 View**

def menu\_control(access) :

    global permission\_stat

    global flag\_bit

    global p\_name

    control = str(input("Your command is : "))

    if control in ["V","SN","SP","SID","S","SWID","F"] :

        sorting\_show(control)

The command is typed in the menu hence the different ways of showing the data can be used. Command “V” is used to show the raw form of the data of goods

###### Viewing data in format ######

def view(data) : # output the formatted table-form of data of goods

    for i in range(91) : print("-" , end="")

    print("")

    for row in data :

        print('| {:>40} | {:>3} | {:>12} | {:>10} | {:>10} | '.format(row[0],row[1],row[2],row[3],row[4]))

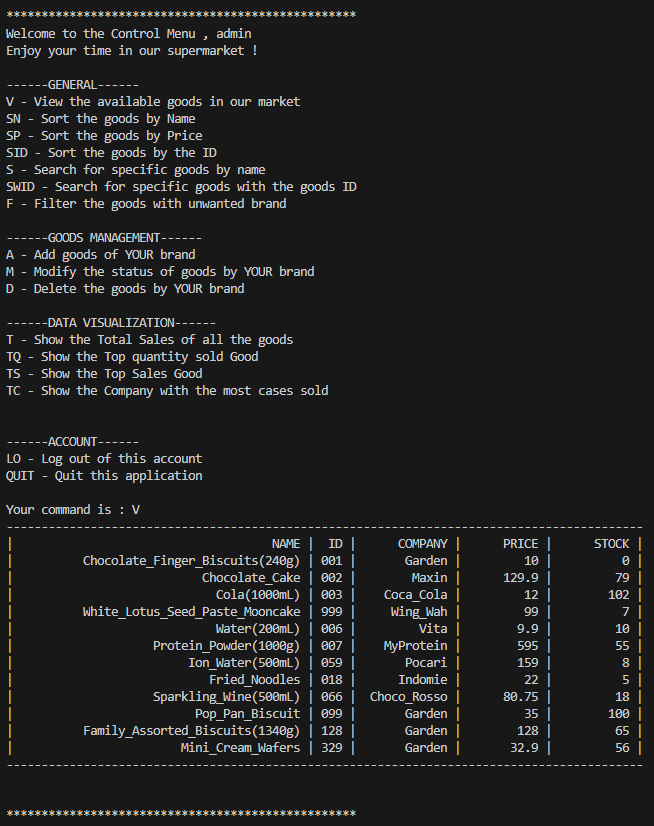
    for i in range(91) : print("-" , end="")

    for i in range(2) : print("")

The 2D array of the goods data can be presented formatted by the view function

The **data** (parameter), a 2D array , should be input and a formatted table will be outputted .   
The view function will be frequently used in the display of 2D array like goods data, shopping cart.

The output of **VIEW**:

****

**2.2.1 Sort by Name**

###### Sorting with 2d array ######

def selection\_sort(raw\_list,index,start) : #find the smallest item and switch

    for i in range(start,len(raw\_list)-1) :

        pos = i

        for j in range(i+1,len(raw\_list)) :

            if raw\_list[j][index] < raw\_list[pos][index] :

                pos = j

        raw\_list[i],raw\_list[pos] = raw\_list[pos],raw\_list[i]

The data of goods is input to the selection sort function as the raw\_list (parameter), and the index marks the NAME column in the file of goods\_data , in this case is **1**. The start parameter is 1, stating the first row of the file of goods\_data should **not** be sorted, which is the row of **heading**

The Selection sort algorithm finds the **smallest element and swaps** it with the first element, getting the smallest element at its correct position. Then the smallest among remaining elements is to move it to its correct position by swapping. This process will be repeated until all the elements in this list in sorted.

**OUTPUT of SN:**

**A screen shot of a computer

Description automatically generated**

**2.2.2 Sort by Price**

def bubble\_sort(raw\_list,index,start) : #switch the nearby items

    for i in range(start,len(raw\_list)) :

        current\_pos = i

        while current\_pos >= start and float(raw\_list[current\_pos][index]) < float(raw\_list[current\_pos-1][index]) :

            raw\_list[current\_pos] , raw\_list[current\_pos-1] = raw\_list[current\_pos-1] ,  raw\_list[current\_pos]

            current\_pos -=1

The data of goods is input to the bubble sort function as the raw\_list parameter, and the index marks the **PRICE** column in the file of goods\_data , in this case is **3**. The start parameter is 2, stating the first row of the file of goods\_data should not be sorted , which is the row of **heading**

The bubble sort works by **comparing the item with the next item**, if the next item is smaller than the item, they swap their position. Hence the largest item will sort to the last position. The process repeats until all the items are sorted

**OUTPUT of SP:**

**A screen shot of a computer

Description automatically generated**

**2.1.3 Sort by SID**

def insertion\_sort(raw\_list,index,start) : #pull the item back until find a correct place

    for i in range(start,len(raw\_list)) :

        store2d = raw\_list[i]

        store = raw\_list[i][index]

        next = i-1

        while next >= 0 and raw\_list[next][index] > store :

            raw\_list[next+1] = raw\_list[next]

            next -=1

        raw\_list[next+1] = store2d

The data of goods is input to the insertion sort function as the raw\_list parameter, and the index marks the **SID** column in the file of goods\_data, in this case is 1. The start parameter is 2, stating the first row of the file of goods\_data should not be sorted, which is the row of **heading**

The insertion sort works by **comparing the second element with the first element** and checking if the second element is smaller than the first element, if so, swap them. Repeat the process for the second element and the third element, and so on. The sorting will be done.

**OUTPUT of SID:**

**A screen shot of a computer

Description automatically generated**

**2.1.4 Searching**

def searching(raw\_list,index,item) :

    new\_list = []

    new\_list.append(raw\_list[0])

    for row in raw\_list :

        if item in row[index] and row != raw\_list[0]: # if the searching keyword is in the searching item

            new\_list.append(row) # add the result to the new list

    return new\_list

The function **“S”** and **“SID”** using the **search** function

It works by first creating an empty list. Looping through the raw\_list (parameter), a 2D array, and **finding the item in the specific position** is the same with the desired item(parameter), if so, add the whole item into the new\_list and return to show. In the case of S and SID, the index will be 0 and 1, stand for the NAME and the ID in this case respectively.

**A screen shot of a computer

Description automatically generated**

**A black screen with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

**Chapter 3**

**Filtering**

            elif way == "F" :

                new\_list = goods

                print("Lets find out your desired good !")

                # price checking

                print("Price :")

                direction1 = str(input("ABOVE or BELOW than or No need(A/B/N):"))

                if direction1 not in ["A","B","N"] : # check if the input is valid for price checking

                    print("ERROR : Fill in A or B or N only ")

                    return False

                if direction1 == "N" :

                    pass

                else :

                    try :

                        num = float(input("The price desired :"))

                        new\_list = compare(new\_list,(filtering(goods,3,num,direction1))) # filter out the items not in the desired price range

                    except ValueError :

                        print("ERROR : Number and decimals only") # verification

                        return False

                # stock checking

                print("Stock :")

                direction2 = str(input("ABOVE or BELOW than or No need(A/B/N):")) # check if the input is valid for stock checking

                if direction2 not in ["A","B","N"] : # check if the input is valid

                    print("ERROR : Fill in A or B or N only ")

                    return False

                if direction2 == "N" :

                    pass

                else :

                    try :

                        num = float(input("The stock desired is :"))

                        new\_list = compare(new\_list,(filtering(goods,4,num,direction2)))

                    except ValueError :

                        print("ERROR : Number and decimals only")

                        return False

                avoid = str(input("The company you want to display is (N - No need):")) # find the desired company if needed

                if avoid == "N" :

                    view(new\_list)

                    return None

                new\_list = compare(new\_list,filtering(goods,2,avoid))

                view(new\_list) # view the new list if no company is required

                return None

view(goods)

def compare(raw\_list1,raw\_list2) : #find the repeated items in the first list

    new\_list = []

    for pos1,item1 in enumerate(raw\_list1) :

        for pos2,item2 in enumerate(raw\_list2) :

            if item1[1] == item2[1] : # check if the item in the first list is in the second list

                new\_list.append(item1)

    return new\_list

def filtering(raw\_list,index,item,d="A") : # default is finding the value above the given item

    new\_list = []

    new\_list.append(raw\_list[0])

    if index in [3,4] :

        for row in raw\_list :

            if row != raw\_list[0] :

                if d == "A" and item <= float(row[index]) and row != raw\_list[0]: # finding the result and add it to the list

                    new\_list.append(row)

                elif d == "B" and item >= float(row[index]) and row != raw\_list[0]:

                    new\_list.append(row)

    else :

        for row in raw\_list :

            if item == row[index] and row != raw\_list[0]: # if the company is in the searching item

                new\_list.append(row)

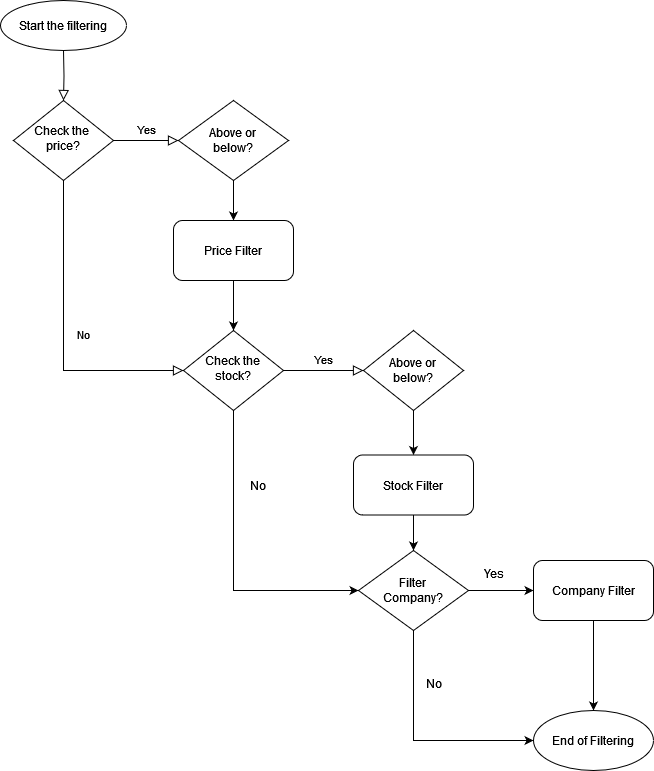
    return new\_list

The filtering will have 3 criteria : Filter of **Price, Stock, Company**

Each of them can choose filter or not to filter it. i.e. it is a independent process , you can choose either 1 or 2 or all 3 criteria to filter.

**Error handling** is used in this case when the user does not input a float or an integer (for example string), Error message will pop, and the user must redo the whole filtering process. The **filtering** function is used to find the desired output of each filtering criteria (i.e. will do three times at most) and the **compare** function is used to compare the result to the main list and only left the items that satisfy the filtering

**Illustration of the filtering process**



**OUTPUT of filtering:**

**A computer screen with white text

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**Chapter 4**

**Manipulation of Goods Data**

**4.1 Add goods**

Only **seller and admin** can add the goods to the goods data, but not customer, by permission check function. The seller can only add goods **with their company’s name** while the admin can add goods with any company name without limitation.

**NAME, ID, PRICE, STOCK** of the goods are needed to fill in to add a good to the file of goods data (COMPANY is needed only by admin) If the inputs are less than 4, meaning the missing of at least 1 item, will have quit to the main menu and re-enter the whole inputs of 4**(Error handling)**. Verifications are also required as the ID must be unique. Also, the number of stock and price should not be less than 0. A age check will be needed for the alcoholic goods, which ID of that goods will be added to the list of age checking.

def menu\_control(access) :

    global permission\_stat

    global flag\_bit

    global p\_name

    control = str(input("Your command is : "))

    if control in ["V","SN","SP","SID","S","SWID","F"] :

        sorting\_show(control)

    elif control == "A" :

        if permission\_check(permission\_stat,2) : # if the user is seller

            while add\_goods(permission\_stat,p\_name) == False : # use the log in name as the company name to add goods

                pass

        elif permission\_check(permission\_stat,3) :

            add\_name = str(input("The name of company of the adding good :")) # admin can choose the name of company by itself

            while add\_goods(permission\_stat,add\_name) == False : # use the log in name as the company name to add goods

                pass

        else :

            print("Access Denied : Admin and Seller ONLY")

###### Change in Goods ######

def add\_goods(p,c) : # write the new data to the csv file with the company name filled

    try :

        n,id,p,s = str(input("Please input the NAME , ID , PRICE , STOCK of the goods\n" # input new data

                             "\*Separate by SPACE\* e.g.Banana 001 10 1 :")).split(" ")

    except ValueError : # error handling

        print("ERROR : Please fill in all 4 inputs with space separated")

        print("//////////////////////////")

        return False

    with open(goods\_data,"r+", newline='', encoding='utf-8') as goods\_info :

        goods = list(csv.reader(goods\_info))

        for x in range(1,len(goods)) :

            if id == goods[x][1] :

                print("ERROR : The ID of the good should be UNIQUE ") # search if any repeat of ID

                print("//////////////////////////")

                return False

    if float(p) <=0 or float(s) <=0 :

        print("The price or stock must NOT less than 0")

        return False

    while age\_limit(id) == False :

        pass

    add([n,id,c,p,s])

    print("Command ADD has successfully executed")

    return True

def age\_limit(id) : # check if the products requires 18 yrs old or above to purchase

    age\_limit = str(input("Does your product has AGE LIMITATION\n"

                          "(Under the law of Hong Kong, intoxicating liquor must not be sold or supplied to a minor in the course of business.) (Y/N) :"))

    if age\_limit == "Y" or age\_limit == "N" :

        if age\_limit == "Y" :

            age\_required.append(id) # add in the list of goods id which requires age verification while purchasing

        return True

    else :

        print("Please fill in either Y/N ")

        return False

**OUTPUT of add\_goods:**

**A screenshot of a computer program

Description automatically generated**

**Before adding goods in file of goods information:**

NAME,ID,COMPANY,PRICE,STOCK

Chocolate\_Finger\_Biscuits(240g),001,Garden,10,0

Chocolate\_Cake,002,Maxin,129.9,79

Cola(1000mL),003,Coca\_Cola,12,102

White\_Lotus\_Seed\_Paste\_Mooncake,999,Wing\_Wah,99,7

Water(200mL),006,Vita,9.9,10

Protein\_Powder(1000g),007,MyProtein,595,55

Ion\_Water(500mL),059,Pocari,159,8

Fried\_Noodles,018,Indomie,22,5

Sparkling\_Wine(500mL),066,Choco\_Rosso,80.75,18

Pop\_Pan\_Biscuit,099,Garden,35,100

Family\_Assorted\_Biscuits(1340g),128,Garden,128,65

Mini\_Cream\_Wafers,329,Garden,32.9,56

**After adding goods in file of goods information:**

NAME,ID,COMPANY,PRICE,STOCK

Chocolate\_Finger\_Biscuits(240g),001,Garden,10,0

Chocolate\_Cake,002,Maxin,129.9,79

Cola(1000mL),003,Coca\_Cola,12,102

White\_Lotus\_Seed\_Paste\_Mooncake,999,Wing\_Wah,99,7

Water(200mL),006,Vita,9.9,10

Protein\_Powder(1000g),007,MyProtein,595,55

Ion\_Water(500mL),059,Pocari,159,8

Fried\_Noodles,018,Indomie,22,5

Sparkling\_Wine(500mL),066,Choco\_Rosso,80.75,18

Pop\_Pan\_Biscuit,099,Garden,35,100

Family\_Assorted\_Biscuits(1340g),128,Garden,128,65

Mini\_Cream\_Wafers,329,Garden,32.9,56

white\_bread,136,Garden,1,1

A new record has been added in the goods information and the table of viewing the goods will be updated and will be shown by the view function

**4.2 Modify goods**

    elif control == "M" :

        if permission\_check(permission\_stat,2) or permission\_check(permission\_stat,3): # only admin and seller can modify goods

            change = str(input("The change of the good is (NAME/COMPANY/PRICE/STOCK) :"))

            if change == "COMPANY" and permission\_check(permission\_stat,2) :

                print("Seller can NOT change the company of goods")

                return False

            input\_dicts = {"NAME":0,"COMPANY":2,"PRICE":3,"STOCK":4}

            if change in input\_dicts :

                id = str(input("The ID of the good is :"))

                new\_value = str(input("The new value of the good is :"))

                modify(p\_name,id,input\_dicts[change],new\_value)

                print("Command MODIFY has successfully executed")

            # The ID should NOT be change by anyone , even for admin

            #elif change == 1 :

            #    print("The ID of the good is NOT allowed to Change")

            else :

                print("INVALID input , please try again")

        else :

            print("Access Denied : Admin and Seller ONLY")

**\*This is a part in the menu function for modification of goods**

def modify(company,id,new\_pos,new\_variable) : # changing the data of goods by overwriting the original data of the goods

    finding = False

    with open(goods\_data, newline='', encoding='utf-8') as f:

        r = csv.reader(f) #read the original data

        lines = list(r) #change the raw data into lists for better indexation

        for pos,row in enumerate(lines) :

                if pos != 0 and (row[2] == company or permission\_check(permission\_stat,3)) and row[1] == id :

                # if the ID belongs to the company that logged in # Admin can edit without limitation

                    new\_pos\_dict = {4:0,3:1}

                    try :

                        if (new\_pos == 4 or new\_pos == 3) :

                            if float(new\_variable) >= new\_pos\_dict[new\_pos] : # the price must be larger than 1 and stock must be larger than 0

                                pass

                            else :

                                print("ERROR : the Price must be larger than 1 and Stock must be larger than 0")

                                return False

                    except ValueError :

                        print("ERROR : Your input must be a integer or decimals")

                        return False

                    lines[pos][new\_pos] = new\_variable # Change the data

                    finding = True # end the finding process

                    return True

        if not finding :

            print("ERROR : The ID does NOT BELONG to YOUR Company or the ID does NOT EXIST")

            return False

    writeData(lines)

def writeData(lines) :

    with open(goods\_data,"w", newline='', encoding='utf-8') as goods\_info :

        writer = csv.writer(goods\_info) # overwrite the data into the file by replacing old data and writing new data

        writer.writerows(lines)

The modify function is only available to **seller and admin**, which sellers can only change the goods that belong to their company while admin has no limitation. After the opening of the goods\_data file and store data into the lines(variable), if the ID input can be found in the goods\_data , the **PRICE , NAME ,COMPANY the STOCK** can be changed ( **ID** is not allowed to change in order to keep the consistency of data) , the input for **PRICE and STOCK** must be larger than **1**. The data of that good in lines is changed, and then writeData function rewrite the whole goods\_data file with the data stored in lines for modification of the data.

**OUTPUT of modify:**

**A screen shot of a computer

Description automatically generated**

**Before the modification :**

**A screen shot of a computer program

Description automatically generated**

**After the modification:**

**A screen shot of a computer program

Description automatically generated**

**4.3 Delete of goods :**

    elif control == "D" :

        if permission\_check(permission\_stat,2) or permission\_check(permission\_stat,3): # only admin and seller can delete goods

            delete\_id = str(input("The ID of the good you want to delete is :"))

            delete(p\_name,delete\_id)

        else :

            print("Access Denied : Admin and Seller ONLY")

**\*This is a part in the menu function for delete of goods**

def delete(company,id) :

    finding = False

    with open(goods\_data,"r", newline='', encoding='utf-8') as f:

        r = csv.reader(f) #read the original data

        lines = list(r) #change the raw data into lists for better indexation

        for pos,row in enumerate(lines) :

                if pos != 0 and (row[2] == company or permission\_check(permission\_stat,3)) and row[1] == id :

                    # seller can only delete their OWN goods while admin can delete ALL goods

                    lines.remove(row)

                    finding = True

        if not finding :

            print("The ID does NOT BELONG to YOUR Company or the ID does NOT EXIST")

            return False

    writeData(lines)

    print("Command DELETE has successfully executed")

The delete function is only available to **seller and admin**, which sellers can only delete the goods that **belong to their company** while admin has **no limitation**. After the opening of the goods\_data file and store data into the lines(variable), if the ID input can be found in the goods\_data , the good of that ID is delete in the data of lines.ASthe data of that good in lines is changed, writeData function **rewrite** the whole goods\_data file with the data stored in lines for deleting of the data.

**Output of delete :**

**A black screen with white text

Description automatically generated**

\* Situation: The company cannot delete the goods belonging to other companies

**A black screen with white text

Description automatically generated**

**\*** Situation: The goods cannot be deleted if the good does not exist

**A screen shot of a computer screen

Description automatically generated**

\*Successful delete

**Before delete :**

**A screen shot of a computer program

Description automatically generated**

**After delete :**

**A screen shot of a computer program

Description automatically generated**

The line of data has been deleted in the file storing goods data and will not be shown by the view function anymore .

Chapter 5

Shopping cart

Shopping cart can only be used by the customer , but not seller or admin .

Shopping cart shows the **NAME,ID,PRICE,Quantity,Total cost** of the goods , the total cost is calculated by the price multiplied by the quantity, customer can go check out after adding item into the shopping cart

**5.1 – Add cart**

    elif control == "AC" :

        if permission\_check(permission\_stat,1) : # only customer can use the shopping cart function

            try :

                c\_id = input("The ID of the goods :")

                c\_q = int(input("The quantity of the goods :"))

            except ValueError :

                print("Your input must be a integer") # error handling

                return False

            add\_cart(c\_id,c\_q)

        else :

            print("Access Denied : Customer ONLY")

            return False

\*This is a part in the menu function for add cart

Customer needs to input the ID and the desired quantity. Error handling for invalid input here.

###### Change in Shopping Cart ######

def add\_cart(id,quantity) :

    global shopping\_cart

    finding = False

    for item in shopping\_cart :

        if item[1] == id :

            print("ERROR : Good already exists in the shopping cart\n"

                  "Please use the Edit Cart function to edit")

            return False

    with open(goods\_data,"r+", newline='', encoding='utf-8') as goods\_info :

        goods = list(csv.reader(goods\_info))

        for x in range(1,len(goods)) :

            if id == goods[x][1] :

                if id in age\_required :

                    if age\_check(p\_bday) : # check if the goods has age restriction

                        pass

                    else :

                        return False

                if int(goods[x][4]) >= quantity > 0: # check if the stock is sufficient for purchase

                    shopping\_cart.append([goods[x][0],goods[x][1],goods[x][3],quantity,(float(goods[x][3])\*float(quantity))])

                    finding = True

                    print("Command ADD CART has successfully executed")

                    return True

                else :

                    print("ERROR : Insufficient goods for purchase or Invalid amount of purchase")

                    return False

        if finding == False :

            print("ERROR : ID do NOT exists")

            return False

def age\_check(date) :

    today = datetime.datetime.today()

    age = today-date

    if age.days >= 6570 : # Check if the user is above 18 yrs old

        return True

    print("Your age is under 18 \n"

          "Under the law of Hong Kong, intoxicating liquor must not be sold or supplied to a minor in the course of business.\n"

          "We can NOT sell this goods to you")

    return False

Shopping cart is a **2D array** which stores the heading at first. Customer has to input a **ID** and a **quantity** to purchase. The add\_cart function can only be used for the goods that are **not in the shopping cart yet**, if the customer wants to add quantity of the goods that are **already** in the shopping cart, modify\_cart function should be used. After opening of the goods\_data file, the data of the desired good with given ID will be found (if it exists) and the age\_check function checks the customer is above 18 years old if the good has age limitation, an ERROR message will be given if the customer does not satisfy that. If the stock of that good is more than the quantity needed **( Error handling )**

**OUTPUT of add\_cart:**

A screen shot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

\* Situation: Customer under the age of 18 cannot buy alcoholic goods

A screen shot of a computer

Description automatically generated

\* Situation: The ID does not exist in the goods\_data

A black screen with white text

Description automatically generated

\*Situation: The stock of the good “001” is not enough for the customer to purchase

**5.2 View cart**

    elif control == "VC" :

        if permission\_check(permission\_stat,1) : # only customer can use the shopping cart function

            view(shopping\_cart)

        else :

            print("Access Denied : Customer ONLY")

            return False

**\*This is a part in the menu function for view cart**

The shopping cart is viewed by the function view which is shown at the previous part of this report

Only **customers** can access the shopping cart but not the admin and seller

**OUTPUT of view cart:**

A black screen with white text

Description automatically generated

**\*** Situation: The goods 001,002,003 are added to the shopping cart with quantity 1,2,3 respectively

And the shopping cart is viewed.

**5.3 Delete cart**

    elif control == "DC" :

        if permission\_check(permission\_stat,1) :

            c\_id = input("The ID of the goods :")

            for pos in range(len(shopping\_cart)) :

                if shopping\_cart[pos][1] == c\_id :

                    shopping\_cart.pop(pos)

                    print("Command DC has successfully executed")

                    return True

            print("ERROR : ID not found in the shopping cart")

            return False

        else :

            print("Access Denied : Customer ONLY")

            return False

\*This is a part in the menu function for delete cart

Only customers can access the shopping cart but not the admin and seller.

The ID of the good which is desired to be removed from the shopping cart must be input.

If that ID is found in the shopping cart, the good is removed from the shopping cart by pop()

If that ID is NOT found in the shopping cart, an Error message will be shown

**OUTPUT of Delete cart :**

Before delete :

A black screen with white text

Description automatically generated

\*Situation: The goods 001,002,003 are added to the shopping cart with quantity 1,2,3 respectively

And the shopping cart is viewed.

**A screen shot of a computer

Description automatically generated**

\*The delete cart commend

A black screen with white text

Description automatically generated

\*Situation: The viewing of shopping cart after the delete cart commend is executed

Chapter 6

Check-out and collection of goods

Customer can check out after their satisficing shopping and plenty of goods are added to the shopping cart . The check out can choose two different methods, delivery or pickup, for collection of purchased goods.

**6.0 Check out**

    elif control == "CO" :

        global flag\_bit

        total = check\_out()

        if not total :

            return False

        print("The total amount will be ${} ".format(total))

        CO\_choice = str(input("Delivery and Pickup are available (D/P) :"))

        if CO\_choice == "P" :

            view\_pickup()

            while not pickup\_check() : # to confirm the location and time

                pass

            flag\_bit = False

        elif CO\_choice == "D" :

            address = str(input("Your address is :"))

            delivery(address)

        else :

            print("Unknown input , please retry")

            return False

###### Check out , pick up , payment ######

def check\_out() :

    total = 0

    for pos,row in enumerate(shopping\_cart) :

        if pos != 0 :

            total += float(shopping\_cart[pos][4]) # find the sum of all the goods

    if total == 0 :

        print("ERROR : There should be at least 1 good to check out for") # there must be something for check out

        return False

    return float("{:.2f}".format(total))

The flag\_bit indicated the end of the whole program when it changes to False.

The sum of all the goods is calculated in the check\_out function (excluding the heading in the shopping list to avoid error) by adding all the amount of the total\_cost of every goods in the shopping cart. Customers are not allowed to check out if there is nothing in the shopping cart.

**OUTPUT of CO :**

**A black background with white text

Description automatically generated**

**6.1 Delivery**

def delivery(address) :

    global flag\_bit

    delivery\_date = date\_input()

    if delivery\_date < datetime.datetime.now() : # the delivery date is before now

        print("The date of delivery must not before the date of the ordering date")

        return False

    if delivery\_date < datetime.datetime.now()+ datetime.timedelta(days=7) : # the delivery date is shorter than 7 days from now

        print("The date of delivery must be 7 days after the ordering date\n")

        return False

    if delivery\_date  > datetime.datetime.now()+ datetime.timedelta(days=30) : # the delivery date is longer than 30 days from now

        print("The delivery date should be within 30 days from the date of ordering")

        return False

    print("The delivery service available from 0900-1200 (AM) and 1400-1800(PM) ")

    delivery\_time = str(input("The time for delivery is (AM/PM) :"))

    if delivery\_time not in ["AM","PM"] :

        print("Please fill in AM/PM")

        return False

    transaction()

    print(f"The address for this delivery is {address} \n"

          f"Your delivery will be arrived on {delivery\_date.strftime('%Y-%m-%d')} at {delivery\_time}")

    flag\_bit = False

In the delivery function, the customer has to first input their location. The date\_input function (presented at the previous part of the report) is called for the user to input a date for the delivery. **Datetime** module is used to calculate the available range for the delivery (in between 7 days after the checkout and 30 days after the checkout , based on the real time of the checkout has been done) And verify the customer's input of date is in the range or not, if not, then an **error message** will be showed. If the input of date is valid, the customer can choose the time of delivery **(AM or PM)** . If the input of time of delivery is valid , the **payment** function will be called.

def payment(t\_id) :

    animation = ["■□□□□□□□□□","■■□□□□□□□□", "■■■□□□□□□□", "■■■■□□□□□□", "■■■■■□□□□□", "■■■■■■□□□□", "■■■■■■■□□□", "■■■■■■■■□□", "■■■■■■■■■□", "■■■■■■■■■■"]

    # The simulation of payment

    payment = str(input("Please enter ur credit card number :"))

    v\_code = str(input("Please enter ur safety code :"))

    v\_date = str(input("Please enter ur expire date :"))

    for x in animation :

        print(x)

        time.sleep(random.randint(0,2)) # simulation real payment

    print("Payment in process ... ")

    time.sleep(3)

    print("Payment succeed\n"

          f"Your CODE for this purchase is {t\_id} \n"

          "Welcome for your next purchase !")

def transaction() :

    t\_id = str(uuid.uuid4())[:8] # an unique purchase code

    for i in shopping\_cart:

        if i != shopping\_cart[0] :

            add\_record(t\_id,i[1],i[3],i[4])

    payment(t\_id)

Several information of the credit of the customer are needed to be input. An **animation** of the payment will be displayed for a demonstration of the payment process. Finally, a transaction ID will be created by the transaction function , in which **uuid** module is used for generating a 8- characters purchase code for the customer to act as a receipt ( it will be used in pickup , which will be explained in the later part) and the **confirmation message** of the delivery will be displayed to the customer for reference.

**OUTPUT of the Delivery:**

A screenshot of a computer

Description automatically generated

**6.2 Pickup**

def pickup\_check() : # check the pickup location and time are available

    with open(pickup\_data,"r") as data :

        times = list(csv.reader(data))

        name = str(input("The name of the pickup store :"))

        pickup\_date = date\_input()

        if pickup\_date < datetime.datetime.now() : # avoid the pickup date is before or in today

            print("The date allocated must be the at least 1 day after the date of ordering")

            return False

        if pickup\_date  > datetime.datetime.now()+ datetime.timedelta(days=30) : # avoid the pickup date is after 30 days from now on

            print("The pickup date should be within 30 days from the date of ordering")

            return False

        try :

            h,min= input("Please input the time for the pickup\n"

                         "with hours and minutes in form (13:30):").split(":") # input the desire time

        except ValueError :

            print("Please fill in the data with format")

            return False

        for x in range(1,len(times)) :

            if times[x][0] == name :

                start = datetime.time(int(times[x][3][:2]), int(times[x][3][2:]), 0) # extracting the hour

                end = datetime.time(int(times[x][4][:2]),int(times[x][4][2:]),0) # extracting the minute

                need\_time =datetime.time(int(h),int(min),0)

                if start <= end or (start <= need\_time or need\_time <= end): # return if possible for pickup

                        transaction() # pay the money through online method

                        print(f"The pickup will be allocated at {need\_time.strftime('%H:%M')} on {pickup\_date.date()}\n"

                        "Please pickup your goods with code given on the booked time at the pickup store\n")

                        return True

                else:

                    print("The store is NOT at service at that time , please try another time within the service hours")

                    return False

        print("Store NOT found , please check the name of the available stores") # if the loop does not return , the store name is not found in the data

        return False

def view\_pickup() :

    with open(pickup\_data,"r", encoding='utf') as pickup\_L : # read the location for pickup

        p\_L = list(csv.reader(pickup\_L))

        for i in range(84) : print("-" , end="")

        print("")

        for row in p\_L :

            print('| {:>20} | {:>30} | {:>8} | {:>5} | {:>5} | '.format(row[0],row[1],row[2],row[3],row[4]))

        for i in range(84) : print("-" , end="")

        for i in range(2) : print("")

The **pickup locations / shops** available for the customer to collect their goods will be shown by the view\_pickup function, which opens the **pickup\_data** file and present the **2D array in a formatted way**. The **pickup date** will be input by the customer by the date\_input function. The date should in the range of the day after the **checking out and the 30 days after the day of checking out**. The customer has to choose the location that they want to pickup at. The time of the pickup needs to be input and it should be within the opening hours of the shop, which will be shown by the view\_pickup function. Transaction will be called afterwards to complete a time of purchase

**OUTPUT of Pickup:**

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

Chapter 7

Data visualization

**7.0 Purchase records\**

###### Data visualization ######

def add\_record(TID,ID,Q,T) : # add a record in the purchase record for each good

    with open(purchase\_records,"a", newline='',encoding='utf-8') as p\_records , open(goods\_data,"r+", newline='',encoding='utf-8') as goods\_info :

            goods = list(csv.reader(goods\_info))

            for good in goods :

                if good[1] == ID and good != goods[0]:

                    join\_list = [TID,good[0],ID,good[3],Q,T,p\_name,good[2],str(datetime.datetime.now().strftime("%Y-%m-%d %H:%M"))] # join the goods\_info with the goods in shopping cart

                    writer = csv.writer(p\_records)

                    writer.writerow(join\_list)

                    modify(good[2],good[1],4,int(good[4])-int(Q)) # minus the quantity of the purchased goods in the goods\_info

The function add\_record will add the transaction record of each good and its information of each good in the shopping cart. The total sum and the quantity will be calculated and written in the file of purchase records. Also, the quantity available of the purchased goods will be reduced by the quantity sold, which will be changed in the file of goods data.

TID,NAME,ID,PRICE,QUANTITY,TOTAL,CUSTOMER,COMANY,**TIME**

c7830e7b,Chocolate\_Finger\_Biscuits(240g),001,10,1,10.0,Victor,Garden,**2024-09-13 23:07**

363787f9,Chocolate\_Finger\_Biscuits(240g),001,10,1,10.0,Tommy,Garden,**2024-09-13 23:08**

363787f9,Water(200mL),006,9.9,1,9.9,Tommy,Vita,**2024-09-13 23:08**

ac8607cb,Mini\_Cream\_Wafers,329,32.9,10,329.0,Tommy,Garden,**2024-09-13 23:15**

ac8607cb,Water(200mL),006,9.9,8,79.2,Tommy,Vita,**2024-09-13 23:15**

**Each good** in the shopping cart will be appended to the file of purchase record. The TID is a **unique** ID for each transaction. However, a TID may appear **more than once** when there are multiple goods in the shopping cart. TOTAL will be calculated based on the Price times Quantity of each goods, but not the total for each transaction.

**7.1 Total sales**

def total\_sales() :

    with open(purchase\_records,"r", newline='',encoding='utf-8') as p\_records :

        records = list(csv.reader(p\_records))

        total\_s = 0

        for x in records :

            if x != records[0] :

                total\_s += float(x[5])

        return total\_s

    elif control == "T" :

        if permission\_check(permission\_stat,3) :

            print("The total sales of all the goods is {}".format(total\_sales()))

        else :

            print("Access Denied : Admin ONLY")

\*This is a part in menu() for showing total sales

The total sales will be calculated, summing up all the **TOTAL** of each transaction in the file of purchase records. Only admin can view the statistical data but not customer and seller .

OUTPUT of total sales:

A black background with white text

Description automatically generated

**7.2 Total Quantity , Total Sales and Total case**

    elif control in ["TQ","TS","TC"]:

        if permission\_check(permission\_stat,3) :

            sales(control)

        else :

            print("Access Denied : Admin ONLY")

\*This is a part in menu() for showing Total Quantity , Total Sales and Total case

def sales(t) :

    with open(purchase\_records,"r", newline='',encoding='utf-8') as p\_records :

        records = list(csv.reader(p\_records))

        sales\_dict = {}

        quantity\_dict = {}

        company\_dict = {}

        for row in records :

            if row != records[0] :

                sales\_dict[row[1]] = 0

                quantity\_dict[row[1]] = 0

                company\_dict[row[7]] = 0

        for row in records :

            if row != records[0] :

                sales\_dict[row[1]] += float(row[5]) # sum up all the sales of each item

                quantity\_dict[row[1]] += int(row[4]) # sum up all the quantity sold of each item

                company\_dict[row[7]] += 1 # sum up all the transaction for each company

        if t == "TS" :

            for name , sale in sales\_dict.items() :

                print("The sales of {} is {:.2f}".format(name,sale)) # showing the sales data

            time.sleep(2)

            print("The top sales among all the goods is {} with {} dollars".format(max(sales\_dict, key=sales\_dict.get),sales\_dict[max(sales\_dict, key=sales\_dict.get)]))

            time.sleep(2)

        if t == "TQ" :

            for name , q in quantity\_dict.items() :

                print("The quantity sold of {} is {}".format(name,q)) # showing the quantity sold data

            time.sleep(2)

            print("The most sold good among all the goods is {} with {} pieces".format(max(quantity\_dict, key=quantity\_dict.get),quantity\_dict[max(quantity\_dict, key=quantity\_dict.get)]))

            time.sleep(2)

        if t == "TC" :

            for name , q in company\_dict.items() :

                print("The cases sold of {} is {}".format(name,q)) # showing the quantity sold data

            time.sleep(2)

            print("The most sale Company is {} with {} cases".format(max(company\_dict, key=company\_dict.get),company\_dict[max(company\_dict, key=company\_dict.get)]))

            time.sleep(2)

The **TS** function provides the **sales of different goods** according to the file of transaction records and sums up all the TOTAL according to the goods as the **sales**.

**Top Sales** will be **calculated** by comparison between all the **sales** of different goods and find the largest one.

The **TC** function provides the **cases of different companies** according to the file of transaction records and sums up all the number of transactions according to the company as the **cases**.

**Most Sale** will be **calculated** by comparison between all the **cases** of different companies and find the largest one.

The **TQ** function provides **quantities of different goods** according to the file of transaction records and sums up all the number of transactions according to the goods as the **quantity**.

**Most Sold Good** will be **calculated** by comparison between all the **quantities** of different goods and find the largest one.

A dictionary will be created to count the occurrence or required data (like total)of each record to **calculate the statistical data**

**OUTPUT of TQ ,TC ,TS :**

A computer screen with white text

Description automatically generated

A screen shot of a computer

Description automatically generated

A computer screen with white text

Description automatically generated

Chapter 8

Test Input

**Test input 1: The new price of the goods**

**A screenshot of a computer screen

Description automatically generated**

**\*The Output from the main program**

**\***def modify(company,id,new\_pos,new\_variable) : # changing the data of goods by overwriting the original data of the goods

    finding = False

    with open(goods\_data, newline='', encoding='utf-8') as f:

        r = csv.reader(f) #read the original data

        lines = list(r) #change the raw data into lists for better indexation

        for pos,row in enumerate(lines) :

                if pos != 0 and (row[2] == company or permission\_check(permission\_stat,3)) and row[1] == id :

                # if the ID belongs to the company that logged in # Admin can edit without limitation

                    if (new\_pos == 4 or new\_pos == 3) :

                        if float(new\_variable) >= 0 : # the price and stock must be larger than 0

                            pass

                        else :

                            print("ERROR : The price or stock must NOT less than 0")

                            return False

                    lines[pos][new\_pos] = new\_variable # Change the data

                    finding = True # end the finding process

        if not finding :

            print("ERROR : The ID does NOT BELONG to YOUR Company or the ID does NOT EXIST")

            return False

    writeData(lines)

**\*The code from the main program**

NAME,ID,COMPANY,PRICE,STOCK

Chocolate\_Finger\_Biscuits(240g),001,Garden,10,10

Chocolate\_Cake,002,Maxin,129.9,79

Cola(1000mL),003,Coca\_Cola,12,102

White\_Lotus\_Seed\_Paste\_Mooncake,999,Wing\_Wah,99,7

Water(200mL),006,Vita,9.9,10

Protein\_Powder(1000g),007,MyProtein,595,55

Ion\_Water(500mL),059,Pocari,159,8

Fried\_Noodles,018,Indomie,22,5

Sparkling\_Wine(500mL),066,Choco\_Rosso,80.75,18

Pop\_Pan\_Biscuit,099,Garden,35,100

Family\_Assorted\_Biscuits(1340g),128,Garden,128,65

Mini\_Cream\_Wafers,329,Garden,32.9,56

White\_bread,136,Garden,1,1

Papa\_Pepper,009,Trash,0,0

**\*The updated goods data after the change of price**

**Details of testing:**

The **PRICE** of the goods can be set to **0**, which is not a good design as the users may wrongly input the price as 0, which might cause unwanted errors.

There may also be **system errors** caused by the price as zero under calculations of the statistical data (divided by 0 may happen, causing math errors)

A new dictionary containing the position of PRICE, STOCK (3,4) is created

The PRICE must be **larger or equal to 1** while the STOCK must be larger or equal to zero

def modify(company,id,new\_pos,new\_variable) : # changing the data of goods by overwriting the original data of the goods

    finding = False

    with open(goods\_data, newline='', encoding='utf-8') as f:

        r = csv.reader(f) #read the original data

        lines = list(r) #change the raw data into lists for better indexation

        for pos,row in enumerate(lines) :

                if pos != 0 and (row[2] == company or permission\_check(permission\_stat,3)) and row[1] == id :

                # if the ID belongs to the company that logged in # Admin can edit without limitation

                    new\_pos\_dict = {4:0,3:1}

                    try :

                        if (new\_pos == 4 or new\_pos == 3) :

                            if float(new\_variable) >= new\_pos\_dict[new\_pos] : # the price must be larger than 1 and stock must be larger than 0

                                pass

                            else :

                                print("ERROR : the Price must be larger than 1 and Stock must be larger than 0")

                                return False

                    except ValueError :

                        print("ERROR : Your input must be a integer or decimals")

                        return False

                    lines[pos][new\_pos] = new\_variable # Change the data

                    finding = True # end the finding process

                    return True

        if not finding :

            print("ERROR : The ID does NOT BELONG to YOUR Company or the ID does NOT EXIST")

            return False

    writeData(lines)

\*New main program after debugging

A screen shot of a computer screen

Description automatically generated

\*New output after changing

|  |  |
| --- | --- |
| Input Data (Price) | Result |
| 0 | Invalid (re-enter is required) |
| 1 | Valid |
| -1 | Invalid (re-enter is required) |
| 100.06 | Valid |
| A | Invalid (re-enter is required) |

**The Output results of each test data:**

A screen shot of a black screen

Description automatically generated

Input Data: 0

A screen shot of a computer

Description automatically generated

Input Data: 1

A screen shot of a black screen

Description automatically generated

Input Data: -1

A screen shot of a computer

Description automatically generated

Input Data: 100.06

A screen shot of a computer

Description automatically generated

Input Data: A

**Test input 2: The password input verification**

**Validation:** The password should follow the following rules:

1. Not **shorter** than 8 characters
2. Not **longer** than 20 characters
3. At least have 1 **capital letter**
4. At least have 1 **number**
5. **No space** in the password

**Verification:**

1. The password input should be the **same** as the previous one
2. The birthday should be in a **correct format** within the limit`

|  |  |
| --- | --- |
| Input Data (Validation) | Results |
| abcd | Invalid |
| Abcdefgh123456789123456789 | Invalid |
| abcdefgh | Invalid |
| Abcdefgh | Invalid |
| Abcdefgh 1 | Invalid |
| Abcdefgh1 | **Valid** |

|  |  |
| --- | --- |
| Input Data (Verification) | Results |
| A | Invalid |
| Abcdefgh1 | **Valid** |
| 2024/12/32 | Invalid |
| 2024/12/1 | **Valid** |

A screenshot of a computer

Description automatically generated

\*Output of the program for the password and account registration

Chapter 9

Debugging

**Error: ValueError due to String input**

def add\_goods(p,c) : # write the new data to the csv file with the company name filled

    try :

        n,id,p,s = str(input("Please input the NAME , ID , PRICE , STOCK of the goods\n" # input new data

                             "\*Separate by SPACE\* e.g.Banana 001 10 1 :")).split(" ")

    except ValueError : # error handling

        print("ERROR : Please fill in all 4 inputs with space separated")

        print("//////////////////////////")

        return False

    with open(goods\_data,"r+", newline='', encoding='utf-8') as goods\_info :

        goods = list(csv.reader(goods\_info))

        for x in range(1,len(goods)) :

            if id == goods[x][1] :

                print("ERROR : The ID of the good should be UNIQUE ") # search if any repeat of ID

                print("//////////////////////////")

                return False

    if float(p) <=0 or float(s) <=0 :

        print("The price or stock must NOT less than 0")

        return False

    while age\_limit(id) == False :

        pass

    add([n,id,c,p,s])

    print("Command ADD has successfully executed")

    return True

\*Original main program

A computer screen shot of white text

Description automatically generated

\*Output of the code when error occurs

**Details of error:**

The program was designed to convert the user’s input integers or decimals for the price and stock of the desired goods to add and the **float() function** converts the users’ input into **Float** (a data type) for further calculation and data management.

However, if the users enter string (another data type), **the float() function** could not convert the string user input into the **Float** .

An Error Handling condition should be added to prevent the ValueError interrupting the running of the whole program. If the user inputs the string , an error message should be displayed and asks user to re-input the value for the good

def add\_goods(p,c) : # write the new data to the csv file with the company name filled

    try :

        n,id,p,s = str(input("Please input the NAME , ID , PRICE , STOCK of the goods\n" # input new data

                             "\*Separate by SPACE\* e.g.Banana 001 10 1 :")).split(" ")

    except ValueError : # error handling

        print("ERROR : Please fill in all 4 inputs with space separated")

        print("//////////////////////////")

        return False

    with open(goods\_data,"r+", newline='', encoding='utf-8') as goods\_info :

        goods = list(csv.reader(goods\_info))

        for x in range(1,len(goods)) :

            if id == goods[x][1] :

                print("ERROR : The ID of the good should be UNIQUE ") # search if any repeat of ID

                print("//////////////////////////")

                return False

    try :

        if float(p) <=0 or float(s) <=0 :

            print("The price or stock must NOT less than 0")

            return False

    except ValueError:

        print("ERROR : Your input for the Price and Stock must be in Integers or Decimals")

        print("//////////////////////////")

        time.sleep(3)

        return False

    while age\_limit(id) == False :

        pass

    add([n,id,c,p,s])

    print("Command ADD has successfully executed")

    return True

\*The updated main program

A screen shot of a computer

Description automatically generated

\*The updated output

A black background with white rectangles and white rectangles

Description automatically generated

\*The new flow of the Add command and its mechanism

Chapter 10

Improvement

Future development Direction

A membership system can be introduced, and the membership points can be acquired through shopping by the customers. Members can enjoy various benefits from being a member, for example they can enjoy 5% off per shopping trip, enjoy birthday discounts and use their membership points to exchange coupons.

The membership system can increase the willingness of the members to purchase more of our goods to increase our revenue. Also, the fame of this online system will be boosted to attract more sellers to join our online shopping platform and further expand our business. The Seasonal discounts and the birthday discounts keep the sticky of the customers i.e. the customers will choose our platform rather than other platforms and increase our competitiveness.

Chapter 11

Investigation and Improvement

**11.1 Efficiency investigation**

###### Sorting with 2d array ######

def selection\_sort(raw\_list,index,start) : #find the smallest item and switch

    for i in range(start,len(raw\_list)-1) :

        pos = i

        for j in range(i+1,len(raw\_list)) :

            if raw\_list[j][index] < raw\_list[pos][index] :

                pos = j

        raw\_list[i],raw\_list[pos] = raw\_list[pos],raw\_list[i]

def bubble\_sort(raw\_list,index,start) : #switch the nearby items

    for i in range(start,len(raw\_list)) :

        current\_pos = i

        while current\_pos >= start and float(raw\_list[current\_pos][index]) < float(raw\_list[current\_pos-1][index]) :

            raw\_list[current\_pos] , raw\_list[current\_pos-1] = raw\_list[current\_pos-1] ,  raw\_list[current\_pos]

            current\_pos -=1

def insertion\_sort(raw\_list,index,start) : #pull the item back until find a correct place

    for i in range(start,len(raw\_list)) :

        store2d = raw\_list[i]

        store = raw\_list[i][index]

        next = i-1

        while next >= 0 and raw\_list[next][index] > store :

            raw\_list[next+1] = raw\_list[next]

            next -=1

        raw\_list[next+1] = store2d

goods\_data = "D:\SBA\SBA\goods\_info.csv"

def test(way) :

    with open(goods\_data,"r", newline='', encoding='utf-8-sig') as goods\_info :

                goods = list(csv.reader(goods\_info))

                if way == "SID" :

                    insertion\_sort(goods,1,2)

                elif way == "SP" :

                    bubble\_sort(goods,3,2)

                elif way == "SN" :

                    selection\_sort(goods,0,1)

import time

from datetime import datetime

way\_list = ["SID","SP","SN"]

for way in way\_list :

    start\_time = datetime.now()

    test(way)

    print("--- %s seconds ---" % (datetime.now() - start\_time))

From this testing program, the efficiency of the 3 different sorting method ( selection sort , bubble sort , insertion sort ) will be investigated by the runtime of different methods . In details, each sorting method will sort the goods data from the goods\_data file and investigate the time required for each sorting method to finish the sorting.

The runtime will be outputted by the program by reducing the end time from the start time of the sorting .

|  |  |  |  |
| --- | --- | --- | --- |
| Sorting Method | Runtime (in seconds) | Time Complexity (max.) | Time Complexity  (min.) |
| Selection sort | 0:00:00.001006 | O(n^2) | O(n^2) |
| Insertion sort | 0:00:00.000995 | O(n^2) | O(n) |
| Bubble sort | 0:00:00.000999 | O(n^2) | O(n) |

From the perspective of efficiency in terms of running time, **insertion sort** is the most efficient one as it spends the **least time** completing the sorting.

However, **selection sort** provides an advantage of which performs at most(n−1)

swaps. Selection sort may become more efficiency when the total number of goods increases and the number of swaps increases dramatically for the other two sorting methods , selection sort can perform the **most stable performance** out of the three methods .

Moreover , **bubble sort** has high **simplicity** which lower the maintainence cost of different sorting methods . Bubble sort can be used for an alternatives when errors occur with the other two relatively more complex sorting methods , bubble sort comes in clutch with its high readability to provide a simple , stable alternative choice for the programmer .

In consideration of efficiency , long-term development and back-up alternatives , all three sorting methods are used in the program.

**11.2 Algorithm and program improvement**

###### admin information ######

admin\_name = ["admin"]

admin\_pw = ["123"]

###### seller information ######

seller\_name = ["Garden"]

seller\_pw = ["Garden123"]

###### customer information ######

customer\_name = ["Tommy"]

customer\_pw = ["5212023"]

customer\_bday = [datetime.datetime(2023,5,21)]

shopping\_cart = [["NAME","ID","PRICE","Quantity","Total cost"]]

###### goods information ######

age\_required = ["066"]

\*Code from main program

**Problem 1 : data lost after termination of program**

**a)**The users information is not stored in a separate file or files in consideratin of privacy and security . However , the change of users information or adding new users may not be able to be stored after the execution and termination of the whole program. **b)**Age requirement for the goods should be a colomn for the data of the goods in the goods\_data file for storing the requirement after the termination of the program without security concerns .

Solution :

**a)3 seperate** files for **admin , seller and customer** should be used to store the data of them storage thus the data of them can be refered and used for login in future and retain the data privacy and security.

**b)**The age requirement should be set as a column of the goods as ‘0’/’1’ which stands for required or not to faciliate the efficiency of checking whether the goods is age-required or not and allow the data be kept after program termination .

**Problem 2 : Rebundancy of the codes**

def admin\_login() :

    global p\_name

    login\_name = str(input("Hi Admin , Your username is : "))

    if login\_name in admin\_name :

        location = find(admin\_name,login\_name)

        login\_pw = str(input("Hi Admin , Your password is : "))

        if admin\_name[location] == login\_name and admin\_pw[location] == login\_pw : # check if the account matches with the password

            print("")

            print("Welcome Admin {} !".format(login\_name))

            p\_name = login\_name

            return True

        else :

            print("Your password is NOT correct , please try again")

    else :

        print("Username does NOT exist , please try again")

    return False

def seller\_login() :

    global p\_name

    login\_name = str(input("Hi Seller , Your username is : "))

    if login\_name in seller\_name :

        location = find(seller\_name,login\_name)

        login\_pw = str(input("Hi Seller , Your password is : "))

        if seller\_name[location] == login\_name and seller\_pw[location] == login\_pw : # check if the account matches with the password

            print("")

            print("Welcome Seller {} !".format(login\_name))

            p\_name = login\_name

            return True

        else :

            print("Your password is NOT correct , please try again")

    else :

        print("Username does NOT exist , please try again")

    return False

def customer\_login() :

    global p\_bday

    global p\_name

    login\_name = str(input("Hi Customer , Your username is : "))

    if login\_name in customer\_name :

        location = find(customer\_name,login\_name)

        login\_pw = str(input("Hi Customer , Your password is : "))

        if customer\_name[location] == login\_name and customer\_pw[location] == login\_pw : # check if the account matches with the password

            print("")

            print("Welcome Customer {} !".format(login\_name))

            p\_name = login\_name

            p\_bday = customer\_bday[location]

            return True

        else :

            print("Your password is NOT correct , please try again")

    else :

        print("Username does NOT exist , please try again")

    return False

\*Code from main program

As shown above , the codes for the login of customer , admin and seller are more or less the same or have high similarity . The initial design for different login method of different uses have a high variety and difference

( **Customers** are required to enter the **birthday** to login toverify the age requirement to ensure the identity of the user , **Admin** is required to enter the **One-time passcode** from their phones for security concerns as Admin are required to manage sensitive data )

However , this design is abandoned due to its high complexity and unnecesserity .

Solution :

a)The above functions for identity verification are added into the code for high level of data security and integrity

b)The three functions of logins will be combined into one simple login program for simplification of the code and to allow easier maintainence given the high simplicity.

The runtime for the program can be reduced and less resource are required to store the code hence enhance the efficiency of the codes .