CS4051NI/CC4059NI Fundamentals of Computing

70% Individual Coursework

Final Submission

2024/25 Spring

Student Name: Anjelina Dong

London Met ID: 24046710

College ID: NP01CP4A240298

Assignment Due Date: Wednesday, May 14, 2025

Assignment Submission Date: Wednesday, May 14, 2025

Word Count:

I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a marks of zero will be awarded.

turnitin Page 2 of 55 - Integrity Overview Submission ID trn:oid:::3618:95802402

14% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- 62 Not Cited or Quoted 10%
  Matches with neither in-text citation nor quotation marks
- 2 Missing Quotations 0%
  Matches that are still very similar to source material
- 23 Missing Citation 3%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

2% Internet sources
0% Publications
13% Submitted works (Student Papers)

Table of content

1.Introduction

WeCare is a simple Python based inventory system for skincare distribution. The system's main goal is to process product sales, monitor inventory amounts, and apply promotional offer. The system runs by obtaining the product data from a text file that has information about the product names, quantities, and cost. The selling cost of each product is calculated by using a 200% markup on the cost price. One of the most important parts of the system is its application of a "buy three, get one free" promotional policy. Under this policy, the customers' bulk purchases will be incentivized as they can receive a free product with every three products they buy.

WeCare is an efficient and user-friendly system that deals with stock management and billing activities, and can therefore be expanded in the long term. With simplification of the inventory and sales processes and automated invoice generation it is a handy tool for the skincare product distributors. The Ability to maintain proper inventory level is one of the main strengths of the system. The system forwards the changes to in-program data and to the external text file each time a sale or restocking operation has been performed. Additionally, it generates an invoice that contains information about the items sold or added, their prices and free items if any and their totals, hence being transparent and storing the data properly. The system automatically adjusts the inventory level in the text file and generates an invoice in program itself which helps in maintaining the stock levels accurate after each transaction updating in both the program itself and the text file.

Therefore, the design followed in WeCare uses the concepts of functions, lists, dictionaries, handling of files and more of python to organize the program in a format readable manner. Error handling is also implemented to make it more convenient to the user. The system is not only a liveable option addressing the current inventory and sales issue but also a ground for further enhancement, for example, GUI building, or incorporating the system into the database making it productive and efficient at the same time.

Goal

The main goal of this coursework is to develop a Skin Care Product Sale system that manages product inventory, conducts transaction with promotional logic, generates invoice, and updates stock with suitable data structure and module programming approach.

Objectives

The main objective of this project is to design a Skin Care Product Sale System using Python, which will effectively track the stock as well as transactions of the customers for a local vendor. The system helps to extract product information from a text file and save it using relevant data structures including lists and dictionaries for convenient retrieval and manipulation. It must be capable of showing product information clearly, process the sales at a “Buy Three Get One Free” promotional policy and automatically renew stock quantity after each sale. The system is also to produce detailed invoices for both sales and restocking and vital information such as name of product, brand, quantity, customers/vendors details, date of transaction, and amount spent. And in order to guarantee usability and maintainability, the program will be implemented in a modular manner disposed with adequate input validation and exception handling mechanism. In addition, the project is intended to show clear understanding of programming ideas and data handling and logical algorithms through well-organized development and testing.

2. Tools used

IDLE-Python

Users get IDLE through their download of Python from its official website because it comes bundled by default with Python. The platform functions as a lightweight interface with friendly design elements to support new programmers in Python program development. The built in Python shell in IDLE supports users to execute code lines while its built-in text editor assists in full Python program development. The editor features a programming efficiency package that comprises automated indenting with highlighting functions and issue detection

features to enhance readability. Through its debugging feature IDLE allows programmers to execute their code lines sequentially for locating logical mistakes. Educational institutions commonly use IDLE because students can install and operate it efficiently to deliver Python programming basics to newcomers. An IDE is an application that provides a comprehensive environment for writing, debugging, and testing code. (Jaishree, n.d.)

MS word

Microsoft Word, word-processor software launched in 1983 by the Microsoft Corporation. Microsoft Word represents a text processing program from Microsoft which enables users to both make and modify and style their written text documents. The software receives extensive use because users employ Word to compose both reports and academic papers and business letters. The application Word provides users with features which include spell checking tools in addition to text styling capabilities and table creation functions and image insertion options. The software enables document files to be saved in two different formats which include .docx and .pdf so users can easily share documents and print them. (University of Arkansas at Little Rock, n.d.)

Draw.io

Through its free online interface Draw.io lets users create various graphical diagrams alongside mind maps and flowcharts as well as organizational structures and other types of representations. Users can access Draw.io as a web-based application which simultaneously connects to Google Drive. The application enables you to save work automatically when you log into your Google Workspace and Gmail account. With its open-source capabilities and self-hosting functionality this tool facilitates team members to develop visual charts through pre-made templates that can be easily moved by drag-and-drop methods as well as diagram import and export capabilities for various file formats. (Paraschiv, 2023)

Notepad

Windows Notepad is simple text editor, which is installed with Microsoft Windows operating system by default. It enables you to create and edit plain text files that have extension of .txt files. (Computer Hope , 2023)

2.Algorithm

Step 1. Start the program.

Step 2. Load products from file:

- Read each line from products.txt.
- Splitting names of the product, brand, quantity, cost price, and origin.
- Put them in dictionaries within a list products.

Step 3. Display the main menu:

- 1: Display/Purchase Products
- 2: Restock Products
- 3: Exit

Stem 4. If Purchase Products:

- Input customer name.
- Show list of available products.
- Repeatedly**:**
    - Request for product number and quantity.
    - Apply “Buy 3 Get 1 Free”:
        - Free items = quantity // 3
        - Total items = quantity and free.
    - Ensure that there is sufficient stock.
    - Calculate:
        - Selling price = 2 x cost price

        - Subtotal = quantity x selling price type (free not charged).
    - Update stock (subtract total items).
    - Save the transaction details on a cart
- After pressing done:
    - Generate invoice using write_purchase_invoice().
    - Update file using write_products().

Step 5.If Restock Products:

- Input vendor name.
- Repeatedly:
    - Type product name, brand, quantity, cost, origin.
    - If product already exists:
        - Update quantity and cost.
    - Else:
        - Add new product.
    - Add to cart.
- After pressing done:
    - Generate restock invoice using write_restock_invoice().
    - Update file using write_products().

Step 6.If Exit:

- Print thank you message and terminate the program.

3.Pseudocode

```
BEGIN

products ← read_products()


LOOP

DISPLAY main menu

INPUT choice


IF choice = 1 THEN

INPUT customer_name

cart ← []

total ← 0


LOOP

display_products(products)

INPUT product_number

IF product_number = 0 THEN BREAK


INPUT quantity

free_items ← quantity // 3

total_items ← quantity + free_items
```

IF stock < total_items THEN

DISPLAY "Not enough stock"

CONTINUE

selling_price ← cost_price × 2

subtotal ← quantity × selling_price

stock ← stock - total_items

ADD details to cart

total ← total + subtotal

IF cart not empty THEN

write_purchase_invoice()

write_products()

DISPLAY invoice

ELSE IF choice = 2 THEN

INPUT vendor_name

cart ← []

total ← 0

LOOP

INPUT product_name (or “end” to quit)

IF name == “end” THEN BREAK

INPUT brand, quantity, cost, origin

IF product exists THEN

UPDATE quantity and cost

ELSE

ADD new product to products

subtotal ← cost × quantity

ADD details to cart

total ← total + subtotal

IF cart not empty THEN

write_restock_invoice()

write_products()

DISPLAY invoice

ELSE IF choice = 3 THEN

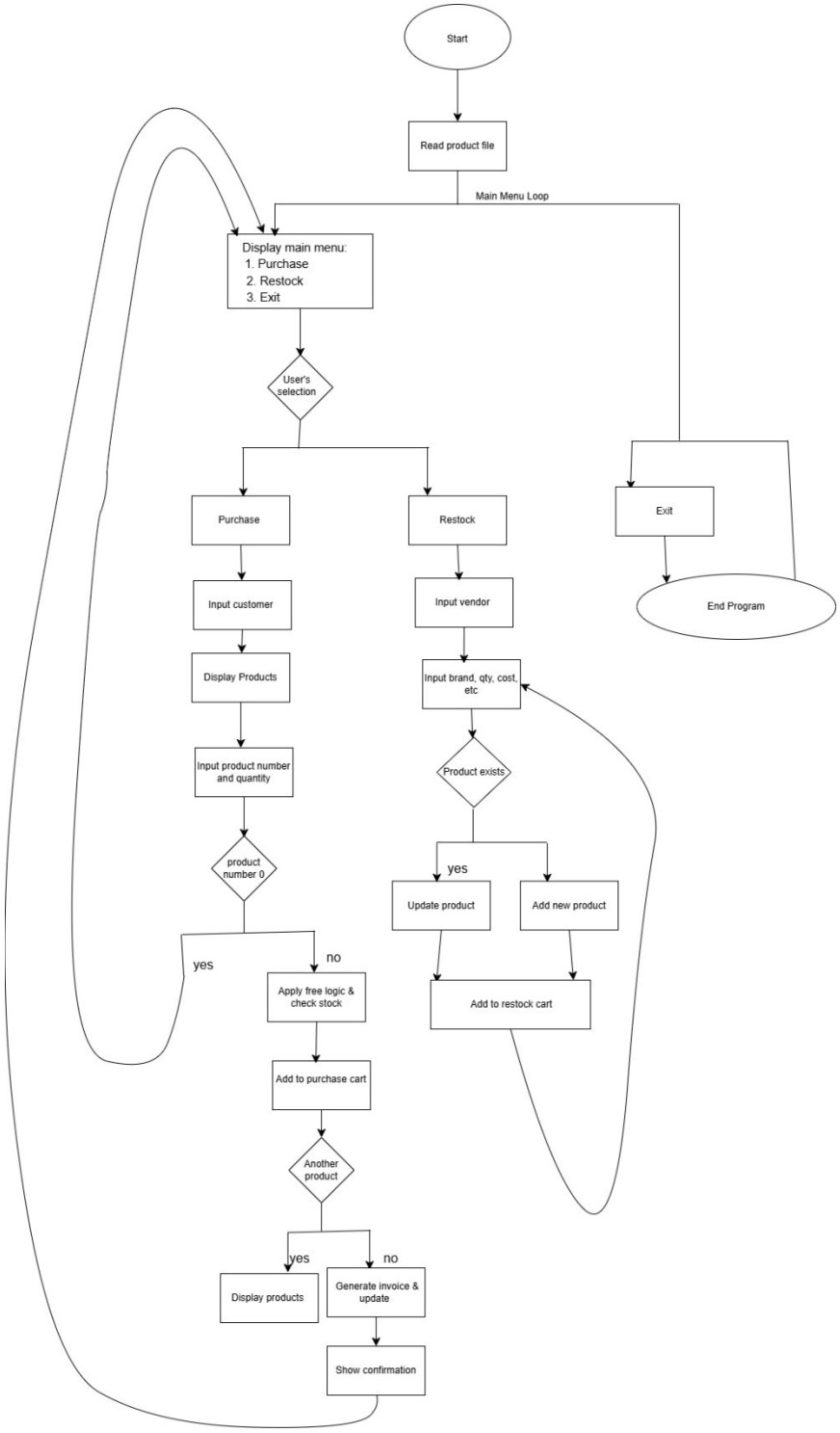
EXIT LOOP

ELSE

DISPLAY "Invalid input

END

4.Flow chart


Start
Read product file
Main Menu Loop
Display main menu:
1. Purchase
2. Restock
3. Exit
User's selection
Purchase
Restock
Exit
End Program
Input customer
Input vendor
Display Products
Input brand, qty, cost, etc
Input product number and quantity
Product exists
product number 0
yes
Update product
Add new product
yes
no
Apply free logic & check stock
Add to restock cart
Add to purchase cart
Another product
yes
no
Display products
Generate invoice & update
Show confirmation

Data Structures

List

Python includes built-in list data structure which lets you store various items inside a single variable. Lists feature both ordering and capacity to store dissimilar items within their boundaries regardless of numerical, text-based or recursive data types. A list takes its shape from placing data items within square brackets while separating the items through commas. Mutable Python lists enable you to modify their content because you can both add and remove elements and change individual items once they have been created. The Python programming language includes several methods that enable lists management through append(), remove(), sort() and len().

read.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\read.py (3.12.6)

File Edit Format Run Options Window Help

```python
Reads product information from the text file and stores it in a list of dictionaries.
 Each product line must have name, brand, quantity, cost_price, and origin separated by commas.

 Parameters:
   filename (str): The name of a file to read from. Default is "products.txt"

 Returns:
   list: A list of dictionaries. Each dictionary contains:
         'name', 'brand', 'quantity' (int), 'cost_price' (float), and 'origin'.
'''
def read_products(filename="products.txt"):

    products = []
    try:
        file = open(filename, "r")
        lines = file.readlines()
        file.close()

        for line in lines:
            if line != "\n":    #Empty line is skipped
                parts = line.split(",")
                if len(parts) == 5:
                    origin_value = parts[4]
                    if origin_value.endswith("\n"):
                        origin_value = origin_value[:-1]  # helps in removing newline manually

                    product = {
                        "name": parts[0],
                        "brand": parts[1],
                        "quantity": int(parts[2]),
                        "cost_price": float(parts[3]),
                        "origin": origin_value
                    }
                    products.append(product)
                else:
                    print("Skipping malformed line:", line)
    except IOError:
        print("Error: Could not open the file", filename)
    except Exception as e:
        print("An error occurred while reading the file:", e)
```

Dictionaries

Python includes dictionary as its built-in data structure which stores its data through key-value pairs. When compared to lists dictionaries need unique keys to retrieve values because they do not use numbered indexes. The definition of dictionaries in Python depends on curly braces {} along with key: value pair orders. Dictionaries serve as a valuable data structure for connecting various pieces of associated information such as product names together with prices. Python dictionaries maintain mutation eligibility so you can both append or alter or delete key-value pairs from the dictionary after its original creation. Standard dictionary operations involve the get() method in conjunction with update() together with keys() and values().

operations.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\operations.py (3.12.6)

File Edit Format Run Options Window Help

```python
import read
import write
import datetime

'''Function for the purchase of a product.
This function receives a list of products as an input, gives the user an option to select products and quantities.
and sends a purchase invoice adding all the amount. The purpose also guarantees availability of stock
and uses free-item policy (after every third item purchased free item is provided).

Parameters in operations.py:
products (list): A list of product dictionaries and in each there will be the details of the product.
(e.g., name, brand, cost_price, quantity).

 Returns:
None. The function creates a purchase invoice and updates stock of products.'''

def purchase_product(products):

    customer = input("Enter customer name: ")
    cart = []  #Declaring an empty cart which will store the purchased items
    total = 0  #Seting the initial price of all purchased items

    while True:
        read.display_products(products) #Displaying the products that can be purchased.
        try:
            choice = int(input("Enter product number to purchase (press 0 to end): "))
            if choice == 0:
                break       #If 0 is entered then exit the loop
            if choice < 1 or choice > len(products):
                print("Invalid choice.") #ensuring correct product selection
                continue

            quantity = int(input("Enter quantity to purchase: "))
            product = products[choice - 1]  #Get chosen product details
            free_items = quantity // 3      #Calculating free products from the purchase quantity
            total_items = quantity + free_items  #Total items including the free ones

            if product["quantity"] < total_items: #checking if sufficient stock is available
                print("Not enough stock. Free item policy requires more stock.")
                continue
```

Function

A Python function comprises a section of reusable programming code which carries out distinct operations. Program functions enable programmers to split complex applications into multiple smaller sections which simplifies both code readability as well as maintenance tasks. Reusability of code becomes possible when using functions because these elements reduce both repetition and enhance system performance.

read.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\read.py (3.12.6)

File Edit Format Run Options Window Help

```python
#read.py
#stores by keeping the list inside the dictionary

'''
The main Function is display_products

Summary:
 Displays a neatly formatted table of all products from the products list.
 Each product is shown with its number, name, brand, quantity, price (double the cost price), and origin.

Parameters:
products (list): A list of dictionaries. Each dictionary describes a product with keys:
                 'name', 'brand', 'quantity', 'cost_price', and 'origin'.
Returns:
  None (This function only prints the product information on the screen.)
'''

def display_products(products):

    print("Available Products:\n")

    print("-----|---------------------|------------------|---------|-------|-----------------------|")
    print("No.  | Name                | Brand            | Quantity| Price | Origin                |")
    print("-----|---------------------|------------------|---------|-------|-----------------------|")

    index = 1
    for product in products:
        name = product["name"]
        brand = product["brand"]
        quantity = str(product["quantity"])
        price = str(int(product["cost_price"] * 2))
        origin = product["origin"]

        index_str = str(index)                  # Padding to align table columns properly
        name_space = " " * (20 - len(name))
        brand_space = " " * (17 - len(brand))
```

File handling

Python file handling allows programs to access computer files through which they can both read saved data and write new information. The procedure serves essential requirements for data storage as well as configuration management and enables applications to exchange data at operation level.

```
'''
def read_products(filename="products.txt"):

    products = []
    try:
        file = open(filename, "r")
        lines = file.readlines()
        file.close()

        for line in lines:
            if line != "\n":    #Empty line is skipped
                parts = line.split(",")
                if len(parts) == 5:
                    origin_value = parts[4]
                    if origin_value.endswith("\n"):
                        origin_value = origin_value[:-1]  # helps in removing newline manually

                    product = {
                        "name": parts[0],
                        "brand": parts[1],
                        "quantity": int(parts[2]),
                        "cost_price": float(parts[3]),
                        "origin": origin_value
                    }
                    products.append(product)
                else:
                    print("Skipping malformed line:", line)
    except IOError:
        print("Error: Could not open the file", filename)
    except Exception as e:
        print("An error occurred while reading the file:", e)
```

Exception Handling

Python implements exception handling as a system to control program errors which happen at run time. The system enables programs to persist running operations when errors trigger by stopping unexpected crashes.

Try and Except Blocks

A try block contains the code sequences which potentially create exceptions. The program automatically transitions from the try block to its matching except block whenever an exception happens inside the try section. Within the except block developers define the procedures to handle exceptions

```
Reads product information from the text file and stores it in a list of dictionaries.
 Each product line must have name, brand, quantity, cost_price, and origin separated by commas.

 Parameters:
   filename (str): The name of a file to read from. Default is "products.txt"

 Returns:
   list: A list of dictionaries. Each dictionary contains:
         'name', 'brand', 'quantity' (int), 'cost_price' (float), and 'origin'.
'''
def read_products(filename="products.txt"):

    products = []
    try:
        file = open(filename, "r")
        lines = file.readlines()
        file.close()

        for line in lines:
            if line != "\n":    #Empty line is skipped
                parts = line.split(",")
                if len(parts) == 5:
                    origin_value = parts[4]
                    if origin_value.endswith("\n"):
                        origin_value = origin_value[:-1]  # helps in removing newline manually

                    product = {
                        "name": parts[0],
                        "brand": parts[1],
                        "quantity": int(parts[2]),
                        "cost_price": float(parts[3]),
                        "origin": origin_value
                    }
                    products.append(product)
                else:
                    print("Skipping malformed line:", line)
    except IOError:
        print("Error: Could not open the file", filename)
    except Exception as e:
        print("An error occurred while reading the file:", e)
```


String Manipulation

Working with Python strings requires diverse operations which allow users to analyse and change string content. Python strings remain immutable because they become unalterable once they are created. The inactivity of Python strings forces programmers to generate fresh strings after applying alterations to their existing values

Date and Time Handling

Time management in Python presents a difficult challenge to programmers. Programmers have access to a built-in solution through the Python datetime module. DateTime operates as an identification and processing system for time-specific components including dates, hours, minutes, seconds along with days of the week, months and years. Users can handle time zone operations and daylight savings rules through this service. The module functions with timestamp data records. The module enables string parsing to recover the day of the week as well as day of month information and multiple time and date formatting patterns

operations.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\operations.py (3.12.6)

File Edit Format Run Options Window Help

```python
import read
import write
import datetime

'''Function for the purchase of a product.
This function receives a list of products as an input, gives the user an option to select products and quantities.
and sends a purchase invoice adding all the amount. The purpose also guarantees availability of stock
and uses free-item policy (after every third item purchased free item is provided).

Parameters in operations.py:
products (list): A list of product dictionaries and in each there will be the details of the product.
(e.g., name, brand, cost_price, quantity).

 Returns:
None. The function creates a purchase invoice and updates stock of products.'''

def purchase_product(products):

    customer = input("Enter customer name: ")
    cart = []  #Declaring an empty cart which will store the purchased items
    total = 0  #Seting the initial price of all purchased items

    while True:
        read.display_products(products) #Displaying the products that can be purchased.
        try:
            choice = int(input("Enter product number to purchase (press 0 to end): "))
            if choice == 0:
                break       #If 0 is entered then exit the loop
            if choice < 1 or choice > len(products):
                print("Invalid choice.") #ensuring correct product selection
                continue

            quantity = int(input("Enter quantity to purchase: "))
            product = products[choice - 1]   #Get chosen product details
            free_items = quantity // 3       #Calculating free products from the purchase quantity
            total_items = quantity + free_items  #Total items including the free ones

            if product["quantity"] < total_items: #checking if sufficient stock is available
                print("Not enough stock. Free item policy requires more stock.")
                continue
```

```python
              "cost_price": cost,
              "subtotal": subtotal
          })

      except ValueError:
          print("Invalid input. Please enter valid numbers for quantity and cost.")   #Handling invalid input

  if len(cart) > 0:          #If the products were restocked teh generating and displaying the restock invoice
      write.write_restock_invoice(vendor, cart, total)  #Write the restock invoice to a file
      write.write_products(products)                    #Write updated product list to a file

      print("\n*-------- Restock Invoice --------*")
      print("Vendor Name:", vendor)
      date=str(datetime.datetime.now().year)+'-'+str(datetime.datetime.now().month)+'-'+str(datetime.datetime.now().day)

      print("Date:", date)

      print("----------------------------------")
```

6.1 Implementation of main.py

main.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\main.py (3.12.6)

File Edit Format Run Options Window Help

```python
# main.py

import read
import operations

def main():

    '''
        This is the starting point of WeCare Skin Care Product System.

        This function displays welcome message, provides menu to the users to either restock or pruchase the given products from the inventory or exit the system.

        The program  continuously loops until the user chooses to exit'''


    #Displays a welcome message

    print("                        ***Welcome to WeCare Skin Care Product System***")
    print("                                    Since 2025")

    #Reads product data from file

    products = read.read_products()

    #Loop has been created for main menu

    while True:
        print("\nPlease choose an option to purchase or restock products:")
        print("1. Display/Purchase Products")
        print("2. Restock Products")
        print("3. Exit")

        #Gets user's input

        choice = input("Choose options from(1-3): ")

        #Handles the user's choice

        if choice == "1":
            operations.purchase_product(products) #Calls the function to handle product purchase
        elif choice == "2":
```



```python
            operations.restock_product(products) #Calls function to handle restocking products
        elif choice == "3":
            print("Thank you for using WeCare System.")
            break
        else:
            print("Invalid option! Please choose again.") #Handles the invalid input

#calls the main function to run the program

main()
```


The main.py file is the entry point of WeCare Skin Care Product System. It starts out by showing a welcome message and then loads product data off a file by calling on the read.read_products(), which would probably read and return product information stored in another file (e.g., text file). Then the program goes into an infinite while True loop to keep on providing to the user a menu with three options. acquire/ exhibit products, replenish products or commercialize out of the system. Depending on the user-inputed input, it invokes either the operations.purchase_product(products) or operations.restock_product(products) operation to process purchasing and restocking using the products. If the user picks “3”, the program outputs a farewell message and leaves the loop through the break. If the user puts in an erroneous option, the system provides an error message and prompts again. The program terminates when the user decides to exit from the system. Finally, at the end of the file, we call the main() function, which starts the execution of the program.

6.2Implementation of operations.py

operations.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\operations.py (3.12.6)

File Edit Format Run Options Window Help

```python
import read
import write
import datetime

'''Function for the purchase of a product.
This function receives a list of products as an input, gives the user an option to select products and quantities.
and sends a purchase invoice adding all the amount. The purpose also guarantees availability of stock
and uses free-item policy (after every third item purchased free item is provided).

Parameters in operations.py:
products (list): A list of product dictionaries and in each there will be the details of the product.
(e.g., name, brand, cost_price, quantity).

 Returns:
None. The function creates a purchase invoice and updates stock of products.'''

def purchase_product(products):

    customer = input("Enter customer name: ")
    cart = []  #Declaring an empty cart which will store the purchased items
    total = 0  #Seting the initial price of all purchased items

    while True:
        read.display_products(products) #Displaying the products that can be purchased.
        try:
            choice = int(input("Enter product number to purchase (press 0 to end): "))
            if choice == 0:
                break       #If 0 is entered then exit the loop
            if choice < 1 or choice > len(products):
                print("Invalid choice.") #ensuring correct product selection
                continue

            quantity = int(input("Enter quantity to purchase: "))
            product = products[choice - 1]   #Get chosen product details
            free_items = quantity // 3       #Calculating free products from the purchase quantity
            total_items = quantity + free_items  #Total items including the free ones

            if product["quantity"] < total_items: #checking if sufficient stock is available
                print("Not enough stock. Free item policy requires more stock.")
                continue
```

operations.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\operations.py (3.12.6)

File Edit Format Run Options Window Help

```python
            total_items = quantity + free_items  #Total items including the free ones

            if product["quantity"] < total_items: #checking if sufficient stock is available
                print("Not enough stock. Free item policy requires more stock.")
                continue

            selling_price = product["cost_price"] * 2  #setting selling price as twice the cost price
            subtotal = quantity * selling_price       #calculating subtotal for this product
            total += subtotal                         #Adding into the final purchase amount
            product["quantity"] -= total_items        #Stock is updated after buying

            cart.append({                             #Adding product purchase details to the cart
                "name": product["name"],
                "brand": product["brand"],
                "final_quantity": total_items,
                "selling_price": selling_price,
                "subtotal": subtotal
            })

        except ValueError:                            #handles invalid input
            print("Invalid input. Please try again.")

    if len(cart) > 0:                                 #If items are placed on the cart, then the invoice is  produced and presented
        write.write_purchase_invoice(customer, cart, total)
        write.write_products(products)

        print("\n----------* Purchase Invoice *----------")
        print("Customer Name:", customer)
        date=str(datetime.datetime.now().year)+'-'+str(datetime.datetime.now().month)+'-'+str(datetime.datetime.now().day)
        print("Date:", date)                            #Displaying the current date
        print("----------------------------------")

        for item in cart:                              #Displaying details of each item in the cart
            print("Product:", item["name"])
            print("Brand:", item["brand"])
            print("Quantity (including free):", item["final_quantity"])
            print("Unit Price:", item["selling_price"])
            print("Subtotal:", item["subtotal"])
            print("----------------------------------")

        print("Total Amount:", total)                   #Displaying the total amount of the pruchase
```

operations.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\operations.py (3.12.6)

File Edit Format Run Options Window Help

```
        print("Total Amount:", total)                        #Displaying the total amount of the pruchase
        print("Purchase has been completed. Invoice is generated.")


'''Function for restocking of products
This function enables the user to enter or update products in the inventory and gives a restock invoice.
It adds new goods if the product is not already within the list and updates the stocks of the existing products.

Parameters: products (list): A list of product dictionaries, with each dictionary having product details.
returns: None. The function creates a restock invoice and updates products' stock.
'''

def restock_product(products):

    vendor = input("Enter vendor's name: ")
    cart = []
    total = 0

    while True:
        try:
            name = input("Enter product name (or type 'end' to finish): ")
            if name.lower() == "end":
                break

            brand = input("Enter brand: ")
            quantity = int(input("Enter quantity to restock: "))
            cost = float(input("Enter cost price per unit: ")) #Cost price per unit
            origin = input("Enter country of origin: ")

            found = False  #Uisng Boolean to check if the product already exists or not
            for each in products:
                if each["name"].lower()==name.lower():  #Checking if product already exists
                    each["quantity"]=int(each["quantity"])+quantity  #updating the quantity
                    each["cost_price"]=cost                          #updating the cost price
                    store=each["brand"]
                    found=True                                       #if the product is dound



            subtotal = cost * quantity                              #Calculating subtotal for the restock
```

operations.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\operations.py (3.12.6)

File Edit Format Run Options Window Help

```
            if found==False:                                        #If the product is new then adding it to the product list
                new_product = {
                    "name": name,
                    "brand": brand,
                    "quantity": quantity,
                    "cost_price": cost,
                    "origin": origin
                }
                products.append(new_product)
                print(f"Added new product: {name} ({brand})")

            total += subtotal                                        #Adding to the total restock cost
            cart.append({
                "name": name,
                "brand": brand,
                "quantity": quantity,
                "cost_price": cost,
                "subtotal": subtotal
            })

        except ValueError:
            print("Invalid input. Please enter valid numbers for quantity and cost.")   #Handling invalid input

    if len(cart) > 0:          #If the products were restocked teh generating and displaying the restock invoice
        write.write_restock_invoice(vendor, cart, total)  #Write the restock invoice to a file
        write.write_products(products)                  #Write updated product list to a file

        print("\n*-------- Restock Invoice --------*")
        print("Vendor Name:", vendor)
        date=str(datetime.datetime.now().year)+'-'+str(datetime.datetime.now().month)+'-'+str(datetime.datetime.now().day)

        print("Date:", date)

        print("---------------------------------")

        for item in cart:
            print("Product:", item["name"])
            if found==False:
                print("Brand:", new_product["brand"])
            else:
                print("Brand:", store)
```

```
                print("Brand:", store)
            print("Quantity Restocked:", item["quantity"])
            print("Cost Price per Unit:", item["cost_price"])
            print("Subtotal:", item["subtotal"])
            print("---------------------------------")

        print("Total Amount:", total)
        print("Restock complete. Invoice has been generated!")
```

The operations.py file provided has two main functions. purchase_product() and restock_product(). These manage the two major operations of the WeCare Skin Care Product System, i.e., purchase and restocking commodities. In purchase_product(), the system first requests for the name of the customer and then proceeds to allow them choose what to buy and quantities to buy. It calculates free items according to the policy that third purchased one adds up a free one. It determines if the stock is adequate (including free items); calculates the selling price (double of the cost price); adds a purchase details to the cart. When the user ends, a purchase invoice is then created and the product list is updated through functions from write.py. Likewise, the system enquires about the name of the vendor and the description of products to add or edit in restock_product(). It sees if there is an existing product – updating its quantity and cost if it does, or adding it as new if not. Restocked items are added into a cart and as soon as done a restock invoice is created and product data is saved. Both have domain for handling errors in case of invalid inputs and contain explicit step-wise procedures to handle the product inventory properly.

Implementation of read.py

read.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\read.py (3.12.6)

File Edit Format Run Options Window Help

```python
#read.py
#stores by keeping the list inside the dictionary

'''
The main Function is display_products

Summary:
 Displays a neatly formatted table of all products from the products list.
 Each product is shown with its number, name, brand, quantity, price (double the cost price), and origin.

Parameters:
products (list): A list of dictionaries. Each dictionary describes a product with keys:
                 'name', 'brand', 'quantity', 'cost_price', and 'origin'.
Returns:
  None (This function only prints the product information on the screen.)
'''

def display_products(products):

    print("Available Products:\n")

    print("-----|--------------------|------------------|---------|-------|-----------------------|")
    print("No.  | Name               | Brand            | Quantity| Price | Origin                |")
    print("-----|--------------------|------------------|---------|-------|-----------------------|")

    index = 1
    for product in products:
        name = product["name"]
        brand = product["brand"]
        quantity = str(product["quantity"])
        price = str(int(product["cost_price"] * 2))
        origin = product["origin"]

        index_str = str(index)                # Padding to align table columns properly
        name_space = " " * (20 - len(name))
        brand_space = " " * (17 - len(brand))
        quantity_space = " " * (8 - len(quantity))
        price_space = " " * (6 - len(price))
        origin_space = " " * (22 - len(origin))

        row = index_str + "    | " + name + name_space + "| " + brand + brand_space + "| " + quantity + quantity_space + "| " + price + price_space + "| " + origin + o
```

```python
        print(row)

        index = index + 1


'''
The Function is read_products

Summary:
Reads product information from the text file and stores it in a list of dictionaries.
 Each product line must have name, brand, quantity, cost_price, and origin separated by commas.

 Parameters:
   filename (str): The name of a file to read from. Default is "products.txt"

 Returns:
   list: A list of dictionaries. Each dictionary contains:
         'name', 'brand', 'quantity' (int), 'cost_price' (float), and 'origin'.
'''
def read_products(filename="products.txt"):

    products = []
    try:
        file = open(filename, "r")
        lines = file.readlines()
        file.close()

        for line in lines:
            if line != "\n":    #Empty line is skipped
                parts = line.split(",")
                if len(parts) == 5:
                    origin_value = parts[4]
                    if origin_value.endswith("\n"):
                        origin_value = origin_value[:-1]  # helps in removing newline manually

                    product = {
                        "name": parts[0],
                        "brand": parts[1],
                        "quantity": int(parts[2]),
                        "cost_price": float(parts[3]),
```

```python
                        "cost_price": float(parts[3]),
                        "origin": origin_value
                    }
                    products.append(product)
                else:
                    print("Skipping malformed line:", line)
    except IOError:
        print("Error: Could not open the file", filename)
    except Exception as e:
        print("An error occurred while reading the file:", e)


    return products
```

The read.py file is very important in managing and viewing product information for WeCare Skin Care Product System. It contains two main functions: display_products() and read_products(). The display_products() function receives a list of product dictionaries and displays them in a tidy form of a table that includes product number, name, brand, quantity, selling price (the cost price doubled), and country of origin. It employs string formatting techniques to make the columns line up and make them readable. Conversely, the read_products() function in turn reads data from a file known as “products.txt”. It goes through every line of the file, breaks it into components

according to commas and converts quantity and cost price to int and float type respectively. It also deals with minor problems regarding reading files like skipping malformed lines, removing new-line characters in files. The function yields a list of dictionaries, which are the products with such keys such as name, brand, quantity, cost_price and origin. These functions will ensure that the system will always have current and presented in clear way product information availed for purchasing or restocking exercises.

6.4Implementation of write.py

write.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\write.py (3.12.6)

File Edit Format Run Options Window Help

```python
import time
'''
Updates the new list of product dictionaries into a text file. Every product goes on a new line in the following format:
name,brand,quantity,cost_price,origin

Parameters:
products (list): A list of dictionaries. Each dictionary represents a product with keys:
                 'name', 'brand', 'quantity', 'cost_price', and 'origin'.
filename (str): The file name where product data will be written to. Default is "products.txt".
 Returns:
 None (The function writes to a file but does not return any value.)
'''

def write_products(products, filename="products.txt"):

    try:
        file = open(filename, "w")
        for product in products:
            line = product["name"] + "," + product["brand"] + "," + str(product["quantity"]) + "," + str(product["cost_price"]) + "," + product["origin"] + "\n"
            file.write(line)
        file.close()
    except IOError:
        print("Error: Unable to write to file", filename)

'''
 Generates a purchase invoice file containing customer details, purchased items,
  quantities (including free items), prices, and the total amount.

 Parameters:
   customer_name (str): The name of the customer making the purchase.
   items       (list): A list of dictionaries, each representing a purchased item with keys:
                       'name', 'brand', 'final_quantity', 'selling_price', and 'subtotal'.
   total       (float): The total bill amount for the entire purchase.

 Returns:
  None (The function writes an invoice to a file but does not return any value.)

'''
def write_purchase_invoice(customer_name, items, total):

    filename = "invoice_purchase_" + customer_name + ".txt"
```



```python
    try:
        file = open(filename, "w")
        file.write("Purchase Invoice\n")
        file.write("Customer Name: " + customer_name + "\n")
        file.write("Date: [Generated at time of transaction]\n")
        file.write("----------------------------------\n")
        for item in items:
            file.write("Product: " + item["name"] + "\n")
            file.write("Brand: " + item["brand"] + "\n")
            file.write("Quantity (including free): " + str(item["final_quantity"]) + "\n")
            file.write("Unit Price: " + str(item["selling_price"]) + "\n")
            file.write("Subtotal: " + str(item["subtotal"]) + "\n")
            file.write("----------------------------------\n")
        file.write("Total Amount: " + str(total) + "\n")
        file.close()
    except IOError:
        print("Error: Could not write invoice file")

'''
  Generates a restock invoice file that includes vendor details, restocked items,
    quantities, cost prices, and the total restocking cost.

 Parameters:
   vendor_name (str): The name of the vendor who supplied the products.
   items       (list): A list of dictionaries, each representing a restocked item with keys:
                       'name', 'brand', 'quantity', 'cost_price', and 'subtotal'.
   total       (float): The total cost for all restocked items.

 Returns:
   None (The function writes an invoice to a file but does not return any value.)
'''
def write_restock_invoice(vendor_name, items, total):

    filename = "invoice_restock_" + vendor_name + ".txt"
    try:
        file = open(filename, "w")
        file.write("Restock Invoice\n")
        file.write("Vendor Name: " + vendor_name + "\n")
        file.write("Date: [Generated at time of transaction]\n")
        file.write("----------------------------------\n")
```

write.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\write.py (3.12.6)

File Edit Format Run Options Window Help

```
            file.write("Subtotal: " + str(item["subtotal"]) + "\n")
            file.write("----------------------------------\n")
        file.write("Total Amount: " + str(total) + "\n")
        file.close()
    except IOError:
        print("Error: Could not write invoice file")

'''
  Generates a restock invoice file that includes vendor details, restocked items,
    quantities, cost prices, and the total restocking cost.

 Parameters:
   vendor_name (str): The name of the vendor who supplied the products.
   items       (list): A list of dictionaries, each representing a restocked item with keys:
                       'name', 'brand', 'quantity', 'cost_price', and 'subtotal'.
   total       (float): The total cost for all restocked items.

 Returns:
   None (The function writes an invoice to a file but does not return any value.)
'''
def write_restock_invoice(vendor_name, items, total):

    filename = "invoice_restock_" + vendor_name + ".txt"
    try:
        file = open(filename, "w")
        file.write("Restock Invoice\n")
        file.write("Vendor Name: " + vendor_name + "\n")
        file.write("Date: [Generated at time of transaction]\n")
        file.write("----------------------------------\n")
        for item in items:
            file.write("Product: " + item["name"] + "\n")
            file.write("Brand: " + item["brand"] + "\n")
            file.write("Quantity Restocked: " + str(item["quantity"]) + "\n")
            file.write("Cost Price per Unit: " + str(item["cost_price"]) + "\n")
            file.write("Subtotal: " + str(item["subtotal"]) + "\n")
            file.write("----------------------------------\n")
        file.write("Total Amount: " + str(total) + "\n")
        file.close()
    except IOError:    #deals with a case in which the file cannot be opened or written. This could occur if the file is locked or if the disk is full or there is a pr
        print("Error: Could not write restock invoice file")
```


The write.py is the file that takes care of all the file-writing operations in the WeCare Skin Care Product System. It contains three main functions: write_products(), write_purchase_invoice(), and write_restock_invoice(). The write_products() function is used to write the current list of the products to a text file (products.txt) based on the product name, brand, quantity, cost price, and origin of each product in a comma separated format. This means that the inventory is maintained up to date following any purchase or restock. The write_purchase_invoice() function creates a file which represents a purchase invoice with a customer name e.g. invoice_purchase_John.txt and it provides detailed information like products names, brands, purchased quantities(including free items), selling prices, and subtotals, and the total amount. In the same way, write_restock_invoice() generates restock invoice file for a vendor (e.g., invoice_restock_Amazon.txt) and records each item that has been restocked by their name, brand, quantity, cost price, subtotal, and total. Each of the three functions involves simple file handling with exception management to rule out crashes when dealing with errors related to getting access to the file, such as permission errors and the like or disk errors. These functions make sure that all transactions will be recorded accordingly and will be traceable.

7.1 Working Mechanism of Main.py

```
*IDLE Shell 3.12.6*
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\main.py
                        ***Welcome to WeCare Skin Care Product System***
                                      Since 2025

Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3):
```

The Program Starts, then prints the message of welcome. It uses read.read_products() in order to retrieve the existing product information stored in products.txt file into a list of dictionaries (products). It presents a menu loop with 3 choices. Purchase products Restock products Exit

Working Mechanism of operations.py

```
Choose options from(1-3): 1
Enter customer name: Anjelina Dong
Available Products:

-----|---------------------|------------------|---------|-------|-----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin                |
-----|---------------------|------------------|---------|-------|-----------------------|
1    | Vitamin C Serum     |  Garnier         | 200     | 2000  |  France               |
2    | Skin Cleanser       |  Cetaphil        | 100     | 560   |  Switzerland          |
3    | Sunscreen           |  Aqualogica      | 200     | 912   |  India                |
Enter product number to purchase (press 0 to end): 1
Enter quantity to purchase: 10
Available Products:

-----|---------------------|------------------|---------|-------|-----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin                |
-----|---------------------|------------------|---------|-------|-----------------------|
1    | Vitamin C Serum     |  Garnier         | 187     | 2000  |  France               |
2    | Skin Cleanser       |  Cetaphil        | 100     | 560   |  Switzerland          |
3    | Sunscreen           |  Aqualogica      | 200     | 912   |  India                |
Enter product number to purchase (press 0 to end): 0

----------* Purchase Invoice *----------
Customer Name: Anjelina Dong
Date: 2025-5-14
--------------------------------
Product: Vitamin C Serum
Brand:  Garnier
Quantity (including free): 13
Unit Price: 2000.0
Subtotal: 20000.0
--------------------------------
Total Amount: 20000.0
Purchase has been completed. Invoice is generated.

Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3):
```

If the user selects "Purchase": They enter their name. Products are displayed using read.display_products(). The user chooses a product and the quantity. The system checks: Valid product selection. Adequate stock (including free item policy: 1 free item for every 3). Selling price is 2 cost price. The purchase details (that include free items) are recorded in a cart. The stock of the product is updated locally on the products list. After the user finishes: A write.write_purchase_invoice() is used to write a purchase invoice. The modified list of products is saved by using write.write_products().

```
invoice_purchase_Anjelina Dong .t
File   Edit   View

Purchase Invoice
Customer Name: Anjelina Dong
Date: [Generated at time of transaction]
---------------------------------
Product: Vitamin C Serum
Brand:  Garnier
Quantity (including free): 13
Unit Price: 2000.0
Subtotal: 20000.0
---------------------------------
Total Amount: 20000.0
```

The invoice of purchase has been generated in the text file.

```
*IDLE Shell 3.12.6*
File  Edit  Shell  Debug  Options  Window  Help

-----|---------------------|------------------|---------|-------|-----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin                |
-----|---------------------|------------------|---------|-------|-----------------------|
1    | Vitamin C Serum     |  Garnier         | 187     | 2000  |  France               |
2    | Skin Cleanser       |  Cetaphil        | 100     | 560   |  Switzerland          |
3    | Sunscreen           |  Aqualogica      | 200     | 912   |  India                |
Enter product number to purchase (press 0 to end): 0

Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3): 2
Enter vendor's name: ANie
Enter product name (or type 'end' to finish): Vitamin C Serum
Enter brand: abc
Enter quantity to restock: 13
Enter cost price per unit: 123
Enter country of origin: abc
Added new product: Vitamin C Serum  (abc)
Enter product name (or type 'end' to finish): end

*-------- Restock Invoice --------*
Vendor Name: ANie
Date: 2025-5-14
---------------------------------
Product: Vitamin C Serum
Brand: abc
Quantity Restocked: 13
Cost Price per Unit: 123.0
Subtotal: 1599.0
---------------------------------
Total Amount: 1599.0
Restock complete. Invoice has been generated!

Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
```

If the user selects "Restock": They enter the vendor's name. For each product to restock: Product name, brand, quantity, cost price, and origin are the input. If the products (by name) are there, they are updated as per quantity and cost. If it is a new product then it is put in the list. Each restock item is added into a cart. After finishing: To write a restock invoice one uses write.write_restock_invoice(). The system will save the updated product list using write.write_products().read_products(): Reads from products.txt and extract each line, it builds the products list. display_products(): Arranges and prints details of product neatly in a table. write.py: write_products(): Saves the full products list into the products.txt. write_purchase_invoice(): Writes the purchase invoice in a file that has the customer's name. write_restock_invoice(): Writes the remaining restock invoice in a file whose name is that of the vendor.

```
invoice_restock_ANie.txt
File   Edit   View

9Restock Invoice
Vendor Name: ANie
Date: [Generated at time of transaction]
---------------------------------
Product: Vitamin C Serum
Brand: abc
Quantity Restocked: 13
Cost Price per Unit: 123.0
Subtotal: 1599.0
---------------------------------
Total Amount: 1599.0
```

The invoice of purchase has been generated in the text file.

```
Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3): 3
Thank you for using WeCare System.
```

If the user clicks "exit" the loop ceases and the program writes a goodbye message.

Testing

Test 1

- To show implementation of try, except and Provide invalid input and show the message

Objective	To ensure that the try-except block helps to avoid the program from crashing as a result of incorrect (non-numeric) input when purchasing a product.
Action	In the Purchase Products option: – When asked to enter the product number type in: "abc" instead of a number
Expected Result	The program should catch the ValueError and display: "Invalid input. Please try again."
Actual Result	The program printed "Invalid input. Please try again." It did not crash but it returned to the input of the product selection.
Conclusion	Successful. The system was able to manage the invalid input with the help of try-except; and program flow was sustained.

```
                    ***Welcome to WeCare Skin Care Product System***
                                  Since 2025

Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3): 1
Enter customer name: Anjelinaa
Available Products:

-----|---------------------|------------------|---------|-------|-----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin                |
-----|---------------------|------------------|---------|-------|-----------------------|
1    | Vitamin C Serum     |  Garnier         | 187     | 2000  |  France               |
2    | Skin Cleanser       |  Cetaphil        | 100     | 560   |  Switzerland          |
3    | Sunscreen           |  Aqualogica      | 200     | 912   |  India                |
4    | Vitamin C Serum     | abc              | 13      | 246   | abc                   |
Enter product number to purchase (press 0 to end): abc
Invalid input. Please try again.
Available Products:

-----|---------------------|------------------|---------|-------|-----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin                |
-----|---------------------|------------------|---------|-------|-----------------------|
1    | Vitamin C Serum     |  Garnier         | 187     | 2000  |  France               |
2    | Skin Cleanser       |  Cetaphil        | 100     | 560   |  Switzerland          |
3    | Sunscreen           |  Aqualogica      | 200     | 912   |  India                |
4    | Vitamin C Serum     | abc              | 13      | 246   | abc                   |
Enter product number to purchase (press 0 to end):
```

operations.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\operations.py (3.12.6)

File Edit Format Run Options Window Help

```python
def purchase_product(products):

    customer = input("Enter customer name: ")
    cart = []  #Declaring an empty cart which will store the purchased items
    total = 0  #Seting the initial price of all purchased items

    while True:
        read.display_products(products) #Displaying the products that can be purchased.
        try:
            choice = int(input("Enter product number to purchase (press 0 to end): "))
            if choice == 0:
                break       #If 0 is entered then exit the loop
            if choice < 1 or choice > len(products):
                print("Invalid choice.") #ensuring correct product selection
                continue

            quantity = int(input("Enter quantity to purchase: "))
            product = products[choice - 1]   #Get chosen product details
            free_items = quantity // 3       #Calculating free products from the purchase quantity
            total_items = quantity + free_items  #Total items including the free ones

            if product["quantity"] < total_items: #checking if sufficient stock is available
                print("Not enough stock. Free item policy requires more stock.")
                continue

            selling_price = product["cost_price"] * 2  #setting selling price as twice the cost price
            subtotal = quantity * selling_price       #calculating subtotal for this product
            total += subtotal                         #Adding into the final purchase amount
            product["quantity"] -= total_items        #Stock is updated after buying

            cart.append({                             #Adding product purchase details to the cart
                "name": product["name"],
                "brand": product["brand"],
                "final_quantity": total_items,
                "selling_price": selling_price,
                "subtotal": subtotal
            })

        except ValueError:                            #handles invalid input
            print("Invalid input. Please try again.")
```

Test 2

- To show Selection purchase and sale of products by providing the negative value as input and providing the non existed value as input

Objective	To test the way system handles invalid product choices such as negative numbers and non-existing product numbers.
Action	In the Purchase Products option: keeping the Key in – 1 as the product number. and Entering 999 (non-existent product index).
Expected Result	The program should test the invalid selections and report: "Invalid choice." The loop should be stable and not crash.
Actual Result	When -1 or 999 was entered, the message when printed was: "Invalid choice." The system carried out the expected result and landed back on the product selection prompt.
Conclusion	Successful. The code makes sure that the range is correct and does not allow any non-existent or negative product access. It maintains the good flow of the program.

```
Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3): 1
Enter customer name: Anjee
Available Products:

-----|---------------------|------------------|---------|-------|-----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin                |
-----|---------------------|------------------|---------|-------|-----------------------|
1    | Vitamin C Serum     |  Garnier         | 187     | 2000  |  France               |
2    | Skin Cleanser       |  Cetaphil        | 100     | 560   |  Switzerland          |
3    | Sunscreen           |  Aqualogica      | 200     | 912   |  India                |
4    | Vitamin C Serum     | abc              | 13      | 246   | abc                   |
Enter product number to purchase (press 0 to end): -1
Invalid choice.
Available Products:

-----|---------------------|------------------|---------|-------|-----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin                |
-----|---------------------|------------------|---------|-------|-----------------------|
1    | Vitamin C Serum     |  Garnier         | 187     | 2000  |  France               |
2    | Skin Cleanser       |  Cetaphil        | 100     | 560   |  Switzerland          |
3    | Sunscreen           |  Aqualogica      | 200     | 912   |  India                |
4    | Vitamin C Serum     | abc              | 13      | 246   | abc                   |
Enter product number to purchase (press 0 to end): 999
Invalid choice.
Available Products:

-----|---------------------|------------------|---------|-------|-----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin                |
-----|---------------------|------------------|---------|-------|-----------------------|
1    | Vitamin C Serum     |  Garnier         | 187     | 2000  |  France               |
2    | Skin Cleanser       |  Cetaphil        | 100     | 560   |  Switzerland          |
3    | Sunscreen           |  Aqualogica      | 200     | 912   |  India                |
4    | Vitamin C Serum     | abc              | 13      | 246   | abc                   |
Enter product number to purchase (press 0 to end):
```

operations.py - C:\Users\dell\OneDrive\Desktop\FINAL PYTHON COURSEWORK ANJELINA DONG 24046720\operations.py (3.12.6)

File Edit Format Run Options Window Help

```python
def purchase_product(products):

    customer = input("Enter customer name: ")
    cart = []  #Declaring an empty cart which will store the purchased items
    total = 0  #Seting the initial price of all purchased items

    while True:
        read.display_products(products) #Displaying the products that can be purchased.
        try:
            choice = int(input("Enter product number to purchase (press 0 to end): "))
            if choice == 0:
                break        #If 0 is entered then exit the loop
            if choice < 1 or choice > len(products):
                print("Invalid choice.") #ensuring correct product selection
                continue
```

Test 3

- File generation of purchase of product(s) (Purchasing multiple product(s)). To show complete purchase process, output in the shell as well the purchased products details in a text file.

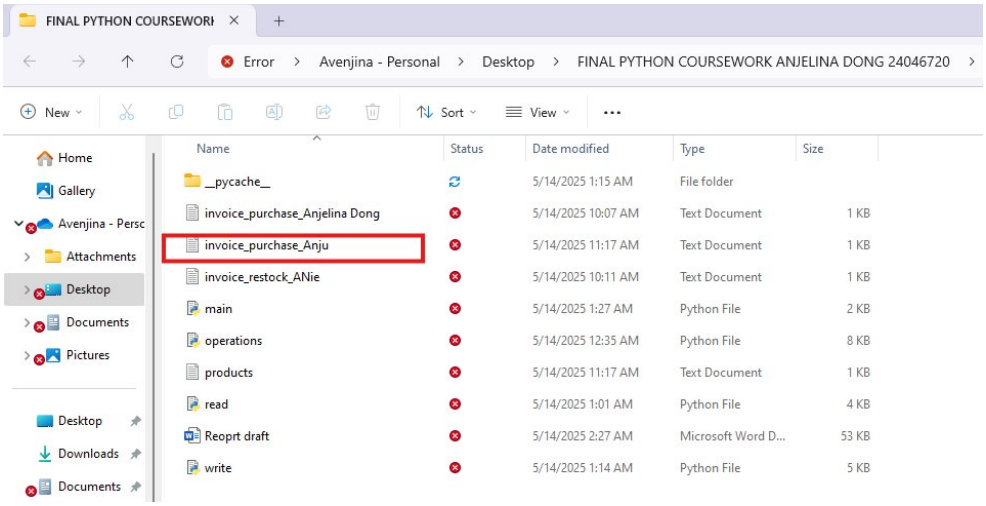
Objective	To test the whole process of purchasing several products and to ensure the creation of invoice file.
Action	1. Choose option 1 to purchase products. 2. Enter customer name as (e.g. Anju). 3. Select product 1 with quantity 3. 4. Select product 2 with quantity 2. 5. Enter 0 to finish.
Expected Result	System should implement the free item policy for the product 1 (3+1 free). The subtotals and total amount should be done aright. On the shell, invoice should be shown. There should be a file created with the name: invoice_purchase_Alice.txt that includes all the details in the invoice..
Actual Result	Invoice displayed in shell correctly. Invoice file invoice_purchase_Anju.txt generated that is written with correct item, price, quantity, and total figures. reduced stock in products.txt.
Conclusion	Successful. The system does multi-item purchase correctly, does free item logic and produces invoice both shell and file.

```
File  Edit  Shell  Debug  Options  Window  Help
Enter quantity to purchase: 3
Available Products:

-----|---------------------|------------------|---------|-------|----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin               |
-----|---------------------|------------------|---------|-------|----------------------|
1    | Vitamin C Serum     |  Garnier         | 183     | 2000  |  France              |
2    | Skin Cleanser       |  Cetaphil        | 100     | 560   |  Switzerland         |
3    | Sunscreen           |  Aqualogica      | 200     | 912   |  India               |
4    | Vitamin C Serum     | abc              | 13      | 246   | abc                  |
Enter product number to purchase (press 0 to end): 2
Enter quantity to purchase: 2
Available Products:

-----|---------------------|------------------|---------|-------|----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin               |
-----|---------------------|------------------|---------|-------|----------------------|
1    | Vitamin C Serum     |  Garnier         | 183     | 2000  |  France              |
2    | Skin Cleanser       |  Cetaphil        | 98      | 560   |  Switzerland         |
3    | Sunscreen           |  Aqualogica      | 200     | 912   |  India               |
4    | Vitamin C Serum     | abc              | 13      | 246   | abc                  |
Enter product number to purchase (press 0 to end): 0

----------* Purchase Invoice *----------
Customer Name: Anju
Date: 2025-5-14
--------------------------------
Product: Vitamin C Serum
Brand:  Garnier
Quantity (including free): 4
Unit Price: 2000.0
Subtotal: 6000.0
--------------------------------
Product: Skin Cleanser
Brand:  Cetaphil
Quantity (including free): 2
Unit Price: 560.0
Subtotal: 1120.0
--------------------------------
Total Amount: 7120.0
Purchase has been completed. Invoice is generated.
```



invoice_rest products.txt invoice_pu

File Edit View

```
Purchase Invoice
Customer Name: Anju
Date: [Generated at time of transaction]
--------------------------------
Product: Vitamin C Serum
Brand:  Garnier
Quantity (including free): 4
Unit Price: 2000.0
Subtotal: 6000.0
--------------------------------
Product: Skin Cleanser
Brand:  Cetaphil
Quantity (including free): 2
Unit Price: 560.0
Subtotal: 1120.0
--------------------------------
Total Amount: 7120.0
```

invoice_rest products.tx invoice_purchase_

File Edit View

```
Vitamin C Serum, Garnier,183,1000.0, France
Skin Cleanser, Cetaphil,98,280.0, Switzerland
Sunscreen, Aqualogica,200,456.0, India
Vitamin C Serum ,abc,13,123.0,abc
```

Test 4

-File generation of sales process of product(s) (Selling multiple product(s)) to show the complete sales process of the product(s)) output in the shell as well finally showing the sold product(s) details in text file

Objective	In order to ensure that the system enables purchasing several products and applies business logic (such as free item policy) correctly and creates an invoice file.
Action	Run main.py.Choose Option 1 (Display/Purchase Products).Enter customer name: Anju.Select Product 1, quantity 3.Select Product 2, quantity 2. Enter 0 to finish the purchase.
Expected Result	Product list displayed. Free item logic applied (e.g., buy 3, get 1 free).Total cost calculated and shown.Products removed from stock.Invoice shown in the shell. - File invoice_purchase_Anju.txt created with correct contents.
Actual Result	Products were selected and free items applied correctly.Total shown in shell matched calculations. Invoice_purchase_Anju.txt created with all expected content.Updated products.txt reflects reduced stock..
Conclusion	Successful. Sales process works as intended, with invoice generated in the shell and as a file. Stock updates are also correct.

```
*IDLE Shell 3.12.6*
File  Edit  Shell  Debug  Options  Window  Help
Choose options from(1-3):
...     3. Exit
Invalid option! Please choose again.

Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3): Invalid option! Please choose again.

Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3): 2
Enter vendor's name: Anju
Enter product name (or type 'end' to finish): Sunscreen
Enter brand: aasd
Enter quantity to restock: 12
Enter cost price per unit: 122
Enter country of origin: asdf
Enter product name (or type 'end' to finish): end

*-------- Restock Invoice --------*
Vendor Name: Anju
Date: 2025-5-14
--------------------------------
Product: Sunscreen
Brand:  Aqualogica
Quantity Restocked: 12
Cost Price per Unit: 122.0
Subtotal: 1464.0
--------------------------------
Total Amount: 1464.0
Restock complete. Invoice has been generated!

Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3):
```

invoice • products.txt invoice_purcl invoic

File Edit View

```
Restock Invoice
Vendor Name: Anju
Date: [Generated at time of transaction]
---------------------------------
Product: Sunscreen
Brand: aasd
Quantity Restocked: 12
Cost Price per Unit: 122.0
Subtotal: 1464.0
---------------------------------
Total Amount: 1464.0
```

invoice • produ invoice_purcl invoice_resto

File Edit View

```
Vitamin C Serum, Garnier,183,1000.0, France
Skin Cleanser, Cetaphil,98,280.0, Switzerland
Sunscreen, Aqualogica,212,122.0, India
Vitamin C Serum ,abc,13,123.0,abc
```

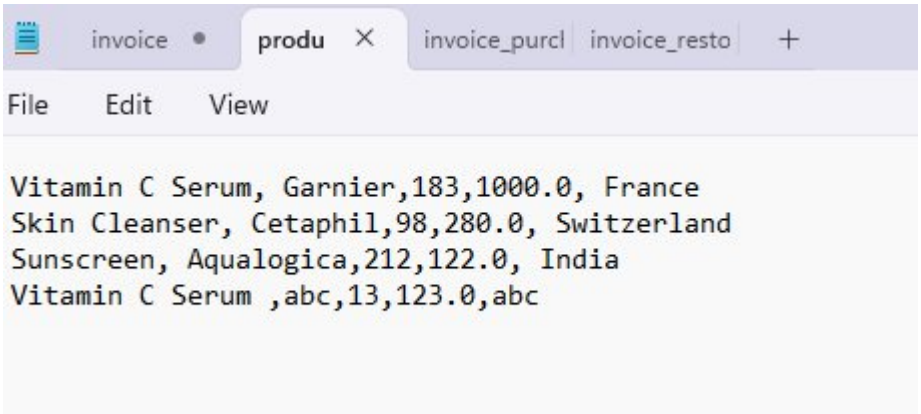
Test 5

- To Show the update in stock of product(s), the quantity being deducted while purchasing the product (Update should be reflected in a .txt file as well), being added while selling the product (Update should be reflected in a .txt file as well)

Objective	To verify that the value of stocked products is reduced whenever a purchase is made and increased after restoration and these updates are stored in products.txt file.
Action	1. Open products.txt and note current quantities of products. 2. Run main.py. 3. Choose **Option 1** (Purchase) and buy **3 units** of a product. 4. Reopen products.txt to verify decreased quantity. 5. Run again and choose **Option 2** (Restock) to add **5 units** of the same product. 6. Reopen products.txt again to verify the updated quantity.
Expected Result	On purchase: Quantity should reduce by purchased + free. On restock: Quantity should increase by the restocked value. These changes must reflect in products.txt.
Actual Result	Quantity deducted correctly after purchase. Quantity increased correctly after restock. File products.txt updated properly both

	times.
Conclusion	Successful. Stock is updated properly after both sale and restock operations and the changes are permanently written to the text file.shell and as a file. Stock updates are also correct.

Before the purchase:




```
-----|---------------------|------------------|---------|-------|-----------------------|
No.  | Name                | Brand            | Quantity| Price | Origin                |
-----|---------------------|------------------|---------|-------|-----------------------|
1    | Vitamin C Serum     |  Garnier         | 167     | 2000  |  France               |
2    | Skin Cleanser       |  Cetaphil        | 98      | 560   |  Switzerland          |
3    | Sunscreen           |  Aqualogica      | 212     | 244   |  India                |
4    | Vitamin C Serum     | abc              | 13      | 246   | abc                   |
Enter product number to purchase (press 0 to end): 0

----------* Purchase Invoice *----------
Customer Name: ANJJ
Date: 2025-5-14
---------------------------------
Product: Vitamin C Serum
Brand:  Garnier
Quantity (including free): 16
Unit Price: 2000.0
Subtotal: 24000.0
---------------------------------
Total Amount: 24000.0
Purchase has been completed. Invoice is generated.
```

After the purchase

```
Vitamin C Serum, Garnier 167 1000.0, France
Skin Cleanser, Cetaphil,98,280.0, Switzerland
Sunscreen, Aqualogica,212,122.0, India
Vitamin C Serum ,abc,13,123.0,abc
```

Before the

```
Vitamin C Serum, Garnier,167,1000.0, France
Skin Cleanser, Cetaphil,98,280.0, Switzerland
Sunscreen, Aqualogica,212,122.0, India
Vitamin C Serum ,abc,13,123.0,abc
```

```
                                    Since 2025

Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3): 2
Enter vendor's name: anjee
Enter product name (or type 'end' to finish): Sunscreen
Enter brand: asd
Enter quantity to restock: 11
Enter cost price per unit: 122
Enter country of origin: ssdf
Enter product name (or type 'end' to finish): end

*-------- Restock Invoice --------*
Vendor Name: anjee
Date: 2025-5-14
--------------------------------
Product: Sunscreen
Brand:  Aqualogica
Quantity Restocked: 11
Cost Price per Unit: 122.0
Subtotal: 1342.0
--------------------------------
Total Amount: 1342.0
Restock complete. Invoice has been generated!

Please choose an option to purchase or restock products:
1. Display/Purchase Products
2. Restock Products
3. Exit
Choose options from(1-3): |
```

After the restock:

File Edit View

```
Vitamin C Serum, Garnier,167,1000.0, France
Skin Cleanser, Cetaphil,98,280.0, Switzerland
Sunscreen, Aqualogica,223,122.0, India
Vitamin C Serum ,abc,13,123.0,abc
```

Discussion and Analysis

The WeCare skincare product sale system was intended to automate major functions of a business like inventory tracking, product sale and handling offers. In this part, the extent to which the system achieved the goals, the difficulties faced during development and the means through which improvements are marked are reflected.

To find a lasting solution to the problem; the project was broken into different functional self-contained parts in different Python files. processing, reading and writing data, dealing with such operations as a purchase and restore, and controlling the primary program loop. This modular architecture aided to keep the code organized and sustainable. Each module was having a particular responsibility meaning the debugging process and the logic implementation process was being efficient.

The system worked well on a number of points. An accurate loading of the inventory from the text file and correct selling prices calculation based on the 200% markup formula took place. The “Buy 3 Get 1 Free” promotional policy successfully applied even in the case when several items were bought during a single transaction. The program created invoices clearly, item names, quantities, free products, and so on. After each transaction, the new inventory was saved back to the file correctly on the proper basis of ensuring data consistency.

Some challenges emerged during development. One of the problems was to guarantee free products to be provided only when the eligible quantities were purchased, and this reasoning had to be consistent for different products. Another problem was the check of user input and blocking the actions that were invalid like selling more items than there are or entering letters instead of numbers. Such problems were resolved when input checks and try-except blocks were added.

However, the system has limitations. It relies on a plain text file for data storage, so it will be less appropriate in larger scale applications. Better scalability and security could be presented by a database. Also, the interface is text-based that might not be user-friendly to all users. Some of the improvements that could be witnessed in the future include the implementation of graphical user interface (GUI) and exporting invoices as files that can be printed out.

However, the project contributed to achieving a deeper understanding of file management, program structure, and data validation. It was an excellent opportunity to apply central programming concepts while building a completely functional Python application.

Conclusion

The design of the WeCare Skincare Product Sale system allowed applying fundamental principles of programming in Python to a real-world business issue. The system is quite capable of achieving its principal goals: inventory management, automatic sales under the promotional offers, invoice creation, all while keeping the stock levels accurate.

By the use of file handling, functions, loops and conditional logic, the system works well and manages both sales and restocking in an expeditious manner. The promotional rationale of “Buy 3 Get 1 Free” provides the project with a practical commercial exaggeration, whilst the automated invoice creation increases professionalism and ease of use. Although the program is fully functional, one can still make subsequent

improvements, for example switching to a database in order to enhance data management and development of a graphical user interface within the program to improve user experience. These improvements would increase the scope and the flexibility of the system for wider use.

The project improved knowledge on handling files, structures of programs, and data validation. Building this system with a set of fully functional Python application based on basic programming notions, focusing on modular design and reasonable implementation.

Bibliography

Computer Hope , 2023. *Computer Hope.* [Online]
Available at: https://www.computerhope.com/jargon/n/notepad.htm
[Accessed 13 May 2025].
Jaishree, n.d. *GUVI.* [Online]
Available at: https://www.guvi.in/hub/python/what-is-idle/
[Accessed 13 May 2025].
Paraschiv, L., 2023. *FOTC.* [Online]
Available at: https://fotc.com/blog/draw-io-online-guide/
[Accessed 13 May 2025].
University of Arkansas at Little Rock, n.d. [Online]
Available at: https://ualr.edu/itservices/services/microsoft-word/
[Accessed 13 May 2025].

Appendix

```
# main.py

import read
import operations

def main():

    '''
        This is the starting point of WeCare Skin Care Product System.
```

This function displays welcome message, provides menu to the users to either restock or pruchase the given products from the inventory or exit the system.

The program continuously loops until the user chooses to exit'''

```
#Displays a welcome message

print("                    ***Welcome to WeCare Skin Care Product System***")
print("                                Since 2025")

#Reads product data from file

products = read.read_products()

#Loop has been created for main menu

while True:
    print("\nPlease choose an option to purchase or restock products:")
    print("1. Display/Purchase Products")
    print("2. Restock Products")
    print("3. Exit")

    #Gets user's input
```

```
    choice = input("Choose options from(1-3): ")

    #Handles the user's choice

    if choice == "1":
        operations.purchase_product(products) #Calls the function to
handle product purchase
    elif choice == "2":
        operations.restock_product(products) #Calls function to handle
restocking products
    elif choice == "3":
        print("Thank you for using WeCare System.")
        break
    else:
        print("Invalid option! Please choose again.") #Handles the invalid
input

#calls the main function to run the program

main()

import read
import write
import datetime

'''Function for the purchase of a product.
```

This function receives a list of products as an input, gives the user an option to select products and quantities.
and sends a purchase invoice adding all the amount. The purpose also guarantees availability of stock
and uses free-item policy (after every third item purchased free item is provided).

Parameters in operations.py:
products (list): A list of product dictionaries and in each there will be the details of the product.
(e.g., name, brand, cost_price, quantity).

Returns:
None. The function creates a purchase invoice and updates stock of products.'''

```
def purchase_product(products):

    customer = input("Enter customer name: ")
    cart = []  #Declaring an empty cart which will store the purchased items
    total = 0  #Seting the initial price of all purchased items

    while True:
        read.display_products(products) #Displaying the products that can be purchased.
        try:
```

```
        choice = int(input("Enter product number to purchase (press 0 to
end): "))
        if choice == 0:
            break       #If 0 is entered then exit the loop
        if choice < 1 or choice > len(products):
            print("Invalid choice.") #ensuring correct product selection
            continue

        quantity = int(input("Enter quantity to purchase: "))
        product = products[choice - 1]   #Get chosen product details
        free_items = quantity // 3       #Calculating free products from the
purchase quantity
        total_items = quantity + free_items  #Total items including the free
ones

        if product["quantity"] < total_items: #checking if sufficient stock is
available
            print("Not enough stock. Free item policy requires more stock.")
            continue

        selling_price = product["cost_price"] * 2  #setting selling price as
twice the cost price
        subtotal = quantity * selling_price       #calculating subtotal for this
product
        total += subtotal                         #Adding into the final purchase
amount
        product["quantity"] -= total_items        #Stock is updated after buying
```

```
            cart.append({                           #Adding product purchase details to
the cart
                "name": product["name"],
                "brand": product["brand"],
                "final_quantity": total_items,
                "selling_price": selling_price,
                "subtotal": subtotal
            })

        except ValueError:                          #handles invalid input
            print("Invalid input. Please try again.")

    if len(cart) > 0:                               #If items are placed on the cart, then
the invoice is  produced and presented
        write.write_purchase_invoice(customer, cart, total)
        write.write_products(products)

        print("\n----------* Purchase Invoice *----------")
        print("Customer Name:", customer)
        date=str(datetime.datetime.now().year)+'-
'+str(datetime.datetime.now().month)+'-'+str(datetime.datetime.now().day)
        print("Date:", date)                        #Displaying the current date
        print("--------------------------------")

        for item in cart:                           #Displaying details of each item in
the cart
```

```
        print("Product:", item["name"])
        print("Brand:", item["brand"])
        print("Quantity (including free):", item["final_quantity"])
        print("Unit Price:", item["selling_price"])
        print("Subtotal:", item["subtotal"])
        print("--------------------------------")

    print("Total Amount:", total)                    #Displaying the total amount
of the pruchase
    print("Purchase has been completed. Invoice is generated.")


'''Function for restocking of products
This function enables the user to enter or update products in the inventory
and gives a restock invoice.
It adds new goods if the product is not already within the list and updates
the stocks of the existing products.

Parameters: products (list): A list of product dictionaries, with each
dictionary having product details.
returns: None. The function creates a restock invoice and updates
products' stock.
'''

def restock_product(products):

   vendor = input("Enter vendor's name: ")
```

```
cart = []
total = 0

while True:
    try:
        name = input("Enter product name (or type 'end' to finish): ")
        if name.lower() == "end":
            break

        brand = input("Enter brand: ")
        quantity = int(input("Enter quantity to restock: "))
        cost = float(input("Enter cost price per unit: ")) #Cost price per unit
        origin = input("Enter country of origin: ")

        found = False  #Uisng Boolean to check if the product already
exists or not
        for each in products:
            if each["name"].lower()==name.lower():  #Checking if product
already exists
                each["quantity"]=int(each["quantity"])+quantity  #updating the
quantity
                each["cost_price"]=cost                          #updating the cost
price
                store=each["brand"]
                found=True                                  #if the product is dound
```

```
        subtotal = cost * quantity                          #Calculating subtotal for
the restock

        if found==False:                                    #If the product is new
then adding it to the product list
            new_product = {
                "name": name,
                "brand": brand,
                "quantity": quantity,
                "cost_price": cost,
                "origin": origin
            }
            products.append(new_product)
            print(f"Added new product: {name} ({brand})")

        total += subtotal                                   #Adding to the total
restock cost
        cart.append({
            "name": name,
            "brand": brand,
            "quantity": quantity,
            "cost_price": cost,
            "subtotal": subtotal
        })
```

```
        except ValueError:
            print("Invalid input. Please enter valid numbers for quantity and
cost.")   #Handling invalid input

    if len(cart) > 0:          #If the products were restocked teh generating and
displaying the restock invoice
        write.write_restock_invoice(vendor, cart, total)  #Write the restock
invoice to a file
        write.write_products(products)                    #Write updated product list
to a file

        print("\n*-------- Restock Invoice --------*")
        print("Vendor Name:", vendor)
        date=str(datetime.datetime.now().year)+'-
'+str(datetime.datetime.now().month)+'-'+str(datetime.datetime.now().day)

        print("Date:", date)

        print("--------------------------------")

        for item in cart:
            print("Product:", item["name"])
            if found==False:
                print("Brand:", new_product["brand"])
            else:
                print("Brand:", store)
            print("Quantity Restocked:", item["quantity"])
```

```
        print("Cost Price per Unit:", item["cost_price"])
        print("Subtotal:", item["subtotal"])
        print("--------------------------------")

    print("Total Amount:", total)
    print("Restock complete. Invoice has been generated!")
#read.py
#stores by keeping the list inside the dictionary

'''
The main Function is display_products

Summary:
 Displays a neatly formatted table of all products from the products list.
 Each product is shown with its number, name, brand, quantity, price
(double the cost price), and origin.

Parameters:
products (list): A list of dictionaries. Each dictionary describes a product
with keys:
                'name', 'brand', 'quantity', 'cost_price', and 'origin'.
Returns:
  None (This function only prints the product information on the screen.)
'''

def display_products(products):
```

```
    print("Available Products:\n")

    print("-----|--------------------|-----------------|---------|-------|----------------------
|")
    print("No.  | Name               | Brand           | Quantity| Price | Origin
|")
    print("-----|--------------------|-----------------|---------|-------|----------------------
|")

    index = 1
    for product in products:
        name = product["name"]
        brand = product["brand"]
        quantity = str(product["quantity"])
        price = str(int(product["cost_price"] * 2))
        origin = product["origin"]

        index_str = str(index)              # Padding to align table columns
properly
        name_space = " " * (20 - len(name))
        brand_space = " " * (17 - len(brand))
        quantity_space = " " * (8 - len(quantity))
        price_space = " " * (6 - len(price))
        origin_space = " " * (22 - len(origin))
```

```
        row = index_str + "    | " + name + name_space + "| " + brand +
brand_space + "| " + quantity + quantity_space + "| " + price + price_space
+ "| " + origin + origin_space + "|"
        print(row)

        index = index + 1



'''
The Function is read_products

Summary:
Reads product information from the text file and stores it in a list of
dictionaries.
 Each product line must have name, brand, quantity, cost_price, and origin
separated by commas.

 Parameters:
   filename (str): The name of a file to read from. Default is "products.txt"

 Returns:
   list: A list of dictionaries. Each dictionary contains:
         'name', 'brand', 'quantity' (int), 'cost_price' (float), and 'origin'.
'''
def read_products(filename="products.txt"):

    products = []
```

```
    try:
        file = open(filename, "r")
        lines = file.readlines()
        file.close()

        for line in lines:
            if line != "\n":    #Empty line is skipped
                parts = line.split(",")
                if len(parts) == 5:
                    origin_value = parts[4]
                    if origin_value.endswith("\n"):
                        origin_value = origin_value[:-1]  # helps in removing newline
manually

                    product = {
                        "name": parts[0],
                        "brand": parts[1],
                        "quantity": int(parts[2]),
                        "cost_price": float(parts[3]),
                        "origin": origin_value
                    }
                    products.append(product)
                else:
                    print("Skipping malformed line:", line)
    except IOError:
        print("Error: Could not open the file", filename)
    except Exception as e:
```

```
        print("An error occurred while reading the file:", e)



    return products
import time
'''
Updates the new list of product dictionaries into a text file. Every product
goes on a new line in the following format:
name,brand,quantity,cost_price,origin

Parameters:
products (list): A list of dictionaries. Each dictionary represents a product
with keys:
                    'name', 'brand', 'quantity', 'cost_price', and 'origin'.
filename (str): The file name where product data will be written to. Default is
"products.txt".
 Returns:
 None (The function writes to a file but does not return any value.)
'''

def write_products(products, filename="products.txt"):

    try:
        file = open(filename, "w")
        for product in products:
```

```
            line = product["name"] + "," + product["brand"] + "," +
str(product["quantity"]) + "," + str(product["cost_price"]) + "," +
product["origin"] + "\n"
            file.write(line)
        file.close()
    except IOError:
        print("Error: Unable to write to file", filename)

'''
 Generates a purchase invoice file containing customer details, purchased
items,
  quantities (including free items), prices, and the total amount.

 Parameters:
   customer_name (str): The name of the customer making the purchase.
   items       (list): A list of dictionaries, each representing a purchased item
with keys:
                        'name', 'brand', 'final_quantity', 'selling_price', and 'subtotal'.
   total       (float): The total bill amount for the entire purchase.

 Returns:
  None (The function writes an invoice to a file but does not return any
value.)

'''
def write_purchase_invoice(customer_name, items, total):
```

```
    filename = "invoice_purchase_" + customer_name + ".txt"
    try:
        file = open(filename, "w")
        file.write("Purchase Invoice\n")
        file.write("Customer Name: " + customer_name + "\n")
        file.write("Date: [Generated at time of transaction]\n")
        file.write("--------------------------------\n")
        for item in items:
            file.write("Product: " + item["name"] + "\n")
            file.write("Brand: " + item["brand"] + "\n")
            file.write("Quantity (including free): " + str(item["final_quantity"]) +
"\n")
            file.write("Unit Price: " + str(item["selling_price"]) + "\n")
            file.write("Subtotal: " + str(item["subtotal"]) + "\n")
            file.write("--------------------------------\n")
        file.write("Total Amount: " + str(total) + "\n")
        file.close()
    except IOError:
        print("Error: Could not write invoice file")

'''
  Generates a restock invoice file that includes vendor details, restocked
items,
    quantities, cost prices, and the total restocking cost.

 Parameters:
   vendor_name (str): The name of the vendor who supplied the products.
```

```
    items       (list): A list of dictionaries, each representing a restocked item
with keys:
                        'name', 'brand', 'quantity', 'cost_price', and 'subtotal'.
    total       (float): The total cost for all restocked items.

  Returns:
    None (The function writes an invoice to a file but does not return any
value.)
'''
def write_restock_invoice(vendor_name, items, total):

    filename = "invoice_restock_" + vendor_name + ".txt"
    try:
        file = open(filename, "w")
        file.write("Restock Invoice\n")
        file.write("Vendor Name: " + vendor_name + "\n")
        file.write("Date: [Generated at time of transaction]\n")
        file.write("--------------------------------\n")
        for item in items:
            file.write("Product: " + item["name"] + "\n")
            file.write("Brand: " + item["brand"] + "\n")
            file.write("Quantity Restocked: " + str(item["quantity"]) + "\n")
            file.write("Cost Price per Unit: " + str(item["cost_price"]) + "\n")
            file.write("Subtotal: " + str(item["subtotal"]) + "\n")
            file.write("--------------------------------\n")
        file.write("Total Amount: " + str(total) + "\n")
        file.close()
```

```
    except IOError:    #deals with a case in which the file cannot be opened
or written. This could occur if the file is locked or if the disk is full or there is
a problem of permission.
        print("Error: Could not write restock invoice file")
```