Module Code & Module Title:

CS4051NT Fundamentals of Computing

Assessment Weightage & Type:

70% Individual Coursework

Year and Semester:

2025 Spring, 2nd Semester

Student Name: Aaryan Koirala

London Met ID: 24045771

College ID:NP05CP4A240099

Assignment Due Date: 14 May 2025

Assignment Submission Date:

Submitted To: Ajayraj Bhattarai

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.

24045771 Aaryan Koirala

Islington College,Nepal

Document Details

Submission ID
trn:oid:::3618:95807450

Submission Date
May 14, 2025, 12:33 PM GMT+5:45

Download Date
May 14, 2025, 12:43 PM GMT+5:45

File Name
24045771 Aaryan Koirala

File Size
13.1 KB

22 Pages

2,072 Words

11,077 Characters

6% Overall Similarity

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

Match Groups

11 Not Cited or Quoted 5%
Matches with neither in-text citation nor quotation marks

1 Missing Quotations 0%
Matches that are still very similar to source material

1 Missing Citation 1%
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

1% Internet sources

0% Publications

6% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **11** Not Cited or Quoted 5%
  Matches with neither in-text citation nor quotation marks
- **1** Missing Quotations 0%
  Matches that are still very similar to source material
- **1** Missing Citation 1%
  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

1% Internet sources
0% Publications
6% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Submitted works	Asia Pacific University College of Technology and Innovation (UCTI) on 2021-12-13 — 1%
2	Submitted works	islingtoncollege on 2025-05-14 — 1%
3	Submitted works	Botswana Accountancy College on 2021-11-18 — <1%
4	Submitted works	Griffith College Dublin on 2023-03-26 — <1%
5	Internet	harvest.usask.ca — <1%
6	Submitted works	islingtoncollege on 2025-05-13 — <1%
7	Submitted works	Info Myanmar College on 2024-01-15 — <1%
8	Submitted works	London Design Engineering UTC on 2025-05-14 — <1%
9	Submitted works	islingtoncollege on 2025-05-13 — <1%
10	Submitted works	Westcliff University on 2024-07-10 — <1%

Table of Contents

Table of figures

Table of Tables

Introduction

The main purpose of this coursework is to create a sale System for a local vendor known as WeCare that sells the beauty and skincare products. The system created is supposed to store details about products by reading product information from a text file known as Products, Storing all the data needed using proper data structures, and showing it in a user-readable format that includes: “Product Name, Brand Name, Quantity available in stock, cost price per items in rupees and country of origin of items.”

1.1 Aims and Objectives

- To create a system based on python that manages product details.
- To apply a file-based system that reads data of products.
- To store and control data of products using proper Data Structures.
- To make invoices of products purchased by users, restocking and transactions details for further development.

1.2 Tools and Technologies Used

1.2.1 Python – programming language

Python 3 is most used high-level programming language that is used for coding for general purpose that is easily accessible on any platforms. It is versatile and easy-to-learn language. You can mix various types of operators in an one single expression and because of that it has less codes.

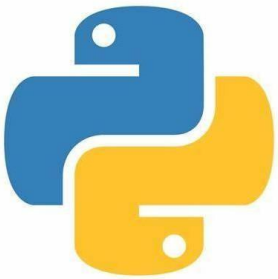

Figure 1. Py

1.2.2 Ms Word

Microsoft Word i.e. Ms word is a most used word processing i.e document making software that is made by Microsoft corporation. Ms word is very easy to use and has so many features so you can make any type of documents. You can create documents with the number of pages you want. Also, Ms word helps in formatting the pages

according to your choice. You can check grammars, spellings along with adding pictures, tables and so on.

Figure 2. word

1.1.1 Integrated Development and Learning Environment

IDLE is an Integrated Development Environment for python. It is a software which allows users / developers to create, design, test and debug & run python programs within one environment in an integrated terminal. It combines developer tools into close graphical user interface.

The features of IDLE's are:

i. Editor: To write codes with highlighting syntax and checking bugs as you write or type.
ii. Compiler: To interpret human-language code into machine-language and execute it on different os.
iii. Debugger: To test and debug codes.
iv. Terminal: To interact with the machine's os.

Data Structures

Data Structures are a way of arranging data so that it can be obtained more efficiently lying upon any situations. Data Structures are the fundamental of programming languages about which programs are built. Python allows users to learn these data structures in a simpler way than any other programming languages.

2.1 Primitive Data Structures:

It is a simple type of data structures used in programming. It is pre-defined data types and has fixed size and format.

2.1.1 Integer

It is a Numeric data type represented by the class called int. It has positive or negative numbers. It doesn't has any limits for its length in value on python.

Example: a = 1

2.1.2 Float

It is a Numeric data types which float class represents its value. It is real number with value after point.

Example : a = 1.1

2.1.3 String

Strings in python are arrays of bytes that represents Unicode characters. A character is a string of length one. It is represented by str class.

Example: a = ' Hi, This is String'

2.1.4 Boolean

Boolean data type has only two values that are: True and False. It is known by class bool.

Example: print(type(True))

print(type(False))

2.2 NON-PRIMITIVE DATA STRUCTURES

2.2.1 Lists

It is a ordered sequence of information which can be accessible by index. It is denoted by [ ]. It contains elements usually of the same data types or can also contain mixed data types. It is mutable because its elements can be changed.

Example:

a = [1,2,3] # list with same data types.

b = [1, ”Hi”, 2.0, True] # list with different data types.

List indices starts at 0 and also supports negative indexing too starting at -1 from the end. **Len() function** returns the length(number of items) of a list.

Operations on List:

i. Add
ii. Remove

2.2.2 Dictionary

It stores the value like (key: pair of value). You can add any data types value and it can be duplicates but you cannot repeat keys and it should not be mutable.

For example:

d = {1: ‘ A’, 2: ‘B’ }

print(d) # {1: ‘ A’, 2: ‘B’}

2.2.3 Sets

It is unordered collection of items. You cannot have any duplicate elements in set but it is mutable that means you can add or remove elements after creating sets but you cannot directly change the individual elements within the set.

S = {1, 2, 3, 4, 5}

print(s) # { 1, 2, 3, 4, 5 }

print(type(s)) # <class ‘set’>

2.2.4 Tuple

It is sequence of ordered elements. You can mix element types in tuples. Tuples are immutable which means you cannot change any tuple created. It is represented with “ ( ) “

Example:

t = (“py”, 2, 2.5)

Indices in tuples:

print(t[0]) # py

Algorithm

Algorithm is the step-by-step process that shows how the code runs.

Step 1: Start: Run the program

1: Call read_products() to load data from products.txt
- If the file doesn't exist, display an error message and return a list that is empty.
- For each line in file:
  - Analyze and break down into name, brand, stock, selling price & origin.
  - Change stock and selling price into integers.
  - Store the data as dictionary and append to product list.

2: Display's menu
I. Display Products
II. Sell Product
III. Restock Product
IV. Exit

3: Ask User to enter a choice.

Step 2: Process's Menu

1. **Display Products (Option 1)**
   - Call display_products(products)
     - Print: Name, Brand, Stock, Price, Origin.
     - For each product:
       - Display in table format.
2. **Sell product (Option 2)**
   - Prompt user to:
     - Product name
     - Quantity
     - Buyer name

i. Call sell_product(products, name, quantity)
- Loop through each product:
  - If name matches (case-insensitive):
    - If stock< quantity: show error and return False.
    - Calculate:
      - Free_items = quantity // 3
      - Total_quantity = quantity + free_items
    - If stock< quantity: show error and return False.
    - Minus total_quantity from stock.
    - Show msg to confirm and ree items.
    - Return True.
- If product not found, return False.

3. If sell_product() returned True:
    - Calll generate_invoice_sell(name,quantity,price,buyer)
        - Calculate:
            - Free_items = quantity//3
            - Total_price = quantity * price
        - Open invoice.txt in append mode.
        - Write:
            - Date/time
            - Buyer name
            - Product name
            - Quantity sold
            - Free items
            - Total items
            - Total price
    - Call write_products(products)
        - Open products.txt
        - For each product, write updated details
4. Else, Show “Product not found” or “Insufficient stock”.

Step 3 : Restock product(option 3)

1. Prompt user for:
    - Product name to restock
    - Quantity
    - Vendor name
2. Call restock_product(products, name, quantity)
    - Call generate_invoice_stock(name, quantity, vendor)
        - Open invoice.txt in append mode
        - Write:
            - Date/Time
            - Vendor name
            - Product name
            - Quantity restocked
3. If restock_product() returned True:
4. Else, show “ product not found”..

Step 4: Exit(option 4)

- Exit the loop and prompt user:
  “ Do you want to restart the system? (yes/no)”
  If yes:
  Call read_products to load data from products.txt
- End Program
    - Display “Goodbye!” when user chooses not to restart

Flowchart

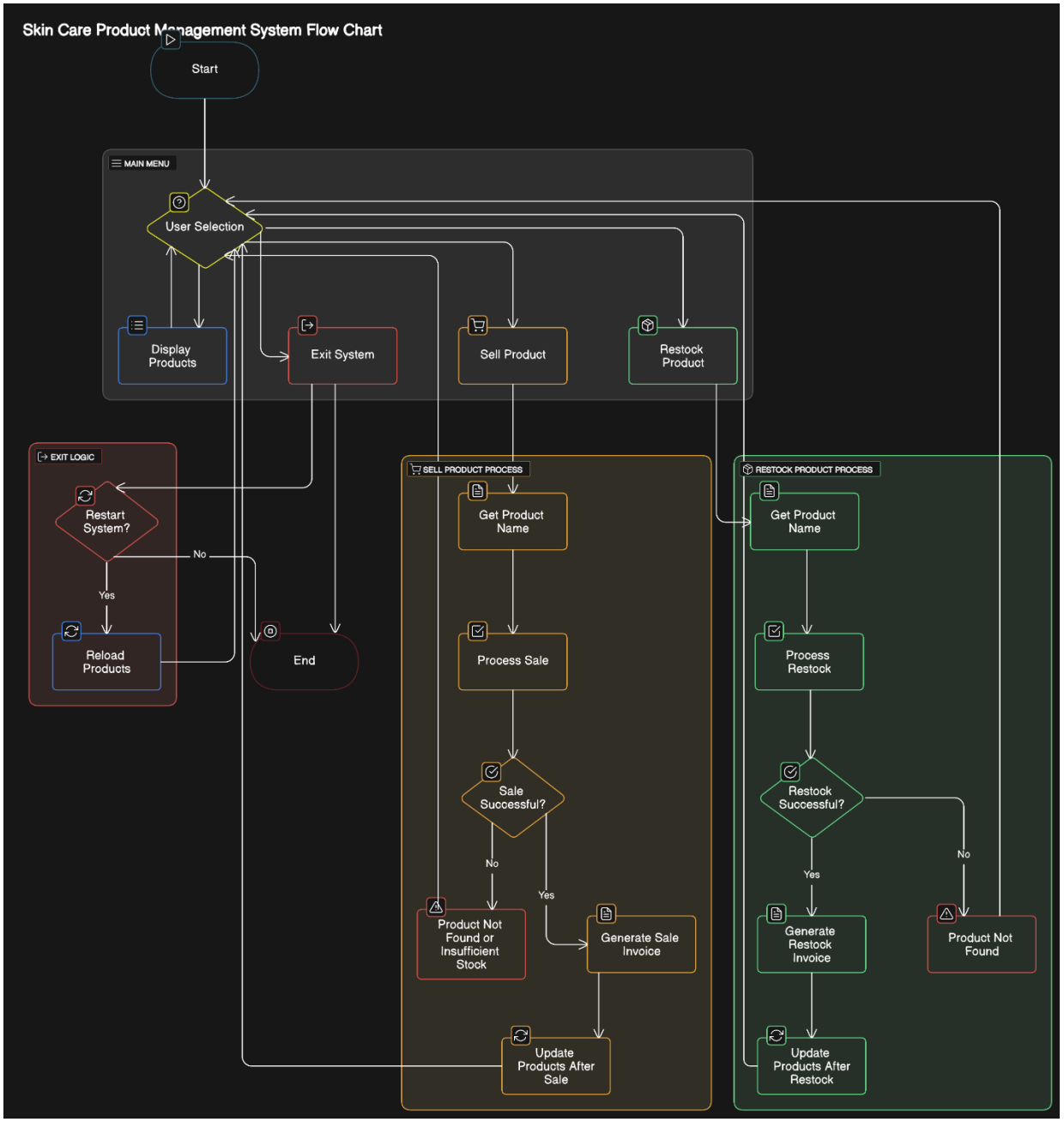

Figure 3. Flowchart

Pseudocode

1. read.py

```
FUNCTION read_products
    INITIALIZE empty list called products


    TRY
        OPEN "products.txt" in read mode AS file
        FOR each line in the file
            IF line is not empty
                SPLIT line by commas INTO name, brand, stock, price, origin
                CONVERT stock and price to integers
                CREATE a dictionary with keys: name, brand, stock, selling_price, origin
                ADD dictionary to products list
    IF file not found
        PRINT "[!] products.txt not found"


    RETURN products
```

2. write.py

```
    FUNCTION write_products(products)
        OPEN "products.txt" in write mode AS file
        FOR each product in products
            FORMAT product info as CSV line
            WRITE line to file


    FUNCTION generate_invoice_sell(product_name, quantity, price, buyer_name)
        CALCULATE free_items = quantity // 3
        CALCULATE total_quantity = quantity + free_items
```

CALCULATE total_price = quantity * price

OPEN "invoice.txt" in append mode AS file
WRITE sale invoice header with current date and time
WRITE buyer name, product name, purchased quantity, free items
WRITE total quantity given, unit price, and total paid
WRITE separator line

FUNCTION generate_invoice_stock(product_name, quantity, vendor_name)
OPEN "invoice.txt" in append mode AS file
WRITE restock invoice header with current date and time
WRITE vendor name, product name, quantity restocked
WRITE separator line

3. Operation.py

DEFINE constant VAT_RATE = 0.13
FUNCTION display_products(products)
PRINT table headers for Product, Brand, Stock, Price (Rs), Origin
PRINT separator line

FOR each product in products
CALCULATE price_with_vat = selling_price * (1 + VAT_RATE)
PRINT product details with VAT-included price
FUNCTION sell_product(products, product_name, quantity)
FOR each product in products
IF product name matches (case-insensitive)
IF stock < quantity
PRINT "Not enough stock"
RETURN False

CALCULATE free_items = quantity // 3
CALCULATE total_quantity = quantity + free_items

IF stock < total_quantity

```
            CALCULATE max_buyable = stock - (stock // 4)
            PRINT "Not enough stock for offer"
            RETURN False

        REDUCE stock by total_quantity
        PRINT sold message
        IF free_items > 0
            PRINT free items message
        RETURN True

    PRINT "Product not found"
    RETURN False

FUNCTION restock_product(products, product_name, quantity)
    FOR each product in products
        IF product name matches (case-insensitive)
            INCREASE stock by quantity
            RETURN True

    PRINT "Product not found"
    RETURN False
```

4. Main.py

```
FUNCTION main
    LOOP indefinitely
        CALL read_products and STORE in products


        LOOP indefinitely
            DISPLAY menu options:
                1. Display Products
                2. Sell Product
```

```
    3. Restock Product
    4. Exit

GET user choice

IF choice == "1"
    CALL display_products(products)

ELSE IF choice == "2"
    GET buyer name
    DISPLAY product list with VAT prices

    TRY
        GET product number
        IF invalid
            PRINT error
            CONTINUE

        SELECT product

        LOOP until valid quantity
            TRY
                GET quantity
                IF invalid
                    PRINT error
```

```
            ELSE
                BREAK

        IF sell_product returns True
            CALL generate_invoice_sell
            CALL write_products
            PRINT invoice success message

    EXCEPT input error
        PRINT error

ELSE IF choice == "3"
    GET vendor name
    DISPLAY product list with stock

    TRY
        GET product number
        IF invalid
            PRINT error
            CONTINUE

        SELECT product

        LOOP until valid quantity
            TRY
```

```
                GET quantity
                IF invalid
                    PRINT error
                ELSE
                    BREAK

            IF restock_product returns True
                CALL generate_invoice_stock
                CALL write_products
                PRINT invoice success message

        EXCEPT input error
            PRINT error

    ELSE IF choice == "4"
        BREAK inner loop

    ELSE
        PRINT "Invalid choice"

ASK user if they want to restart
IF answer is not "yes"
    PRINT "Goodbye"
    BREAK outer loop
```

```
IF __name__ == "__main__"
    CALL main
```

Program

Main.py

This is the entrance point of the application. It deals with the general control flow of the program and integrates all the functional modules. It typically:

i. Shows the user (administrator), the main menu.

ii. Receives the user input to do various actions including viewing products, recording a sale, restocking items, or exiting the program.

iii. Calls operation.py, read.py and write.py appropriate function according to which operation was chosen.

iv. Makes sure that the program is running in a loop for as long as the administrator will not discontinue it.

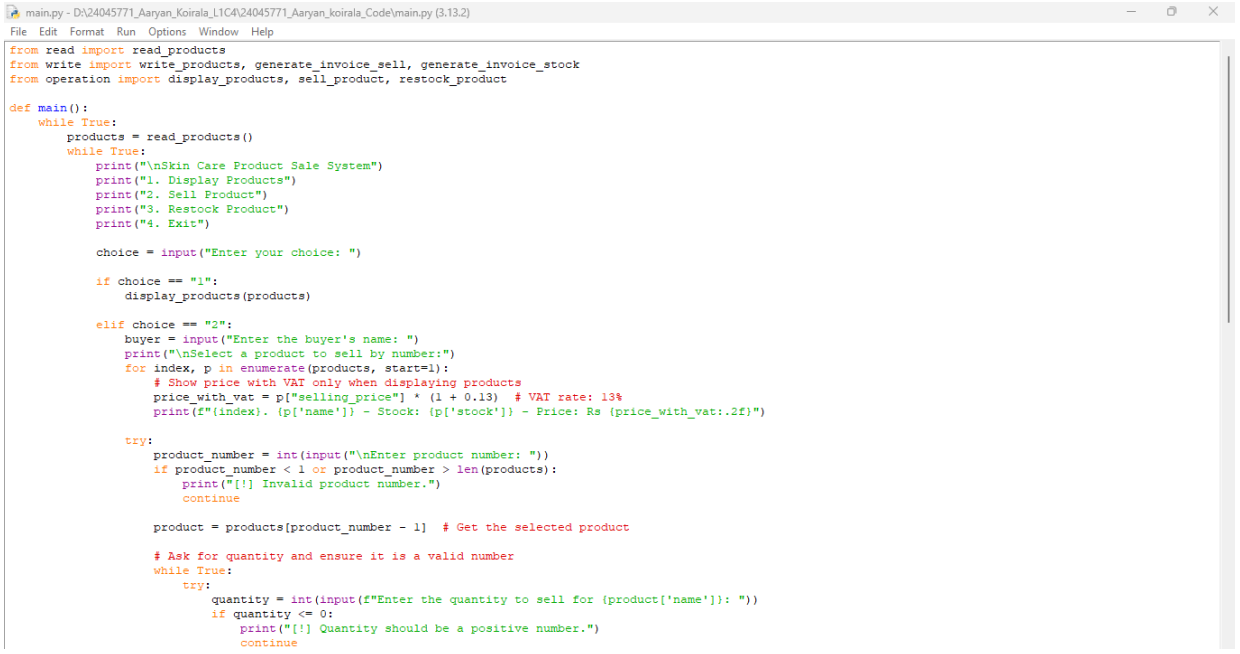

Figure 4 Main

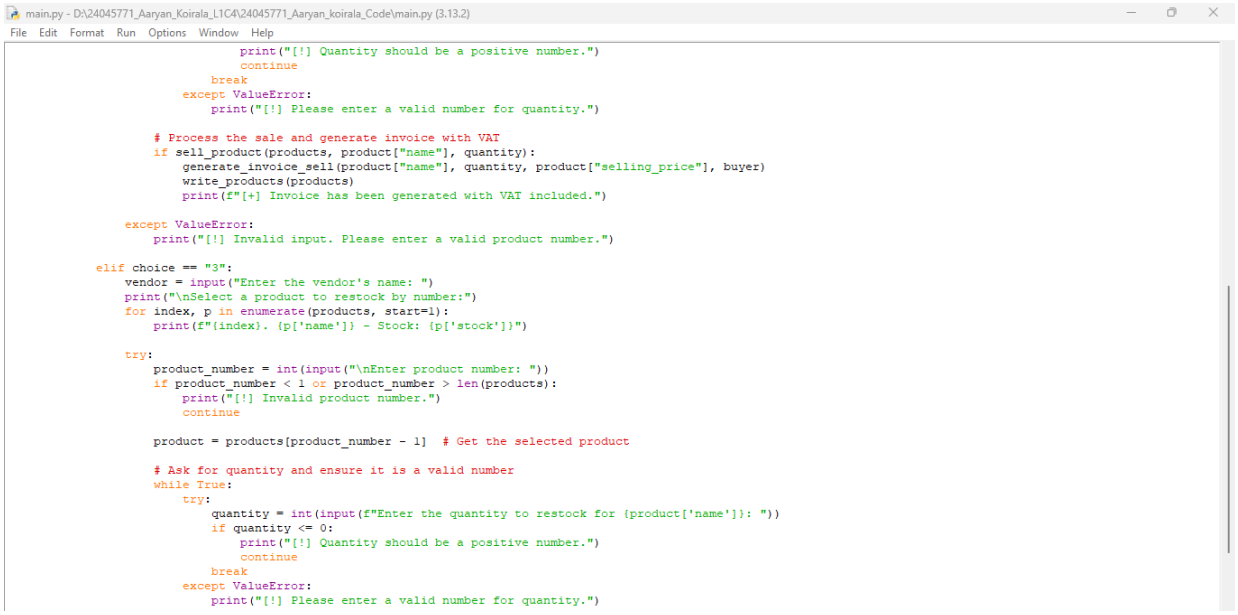

Figure 5 main

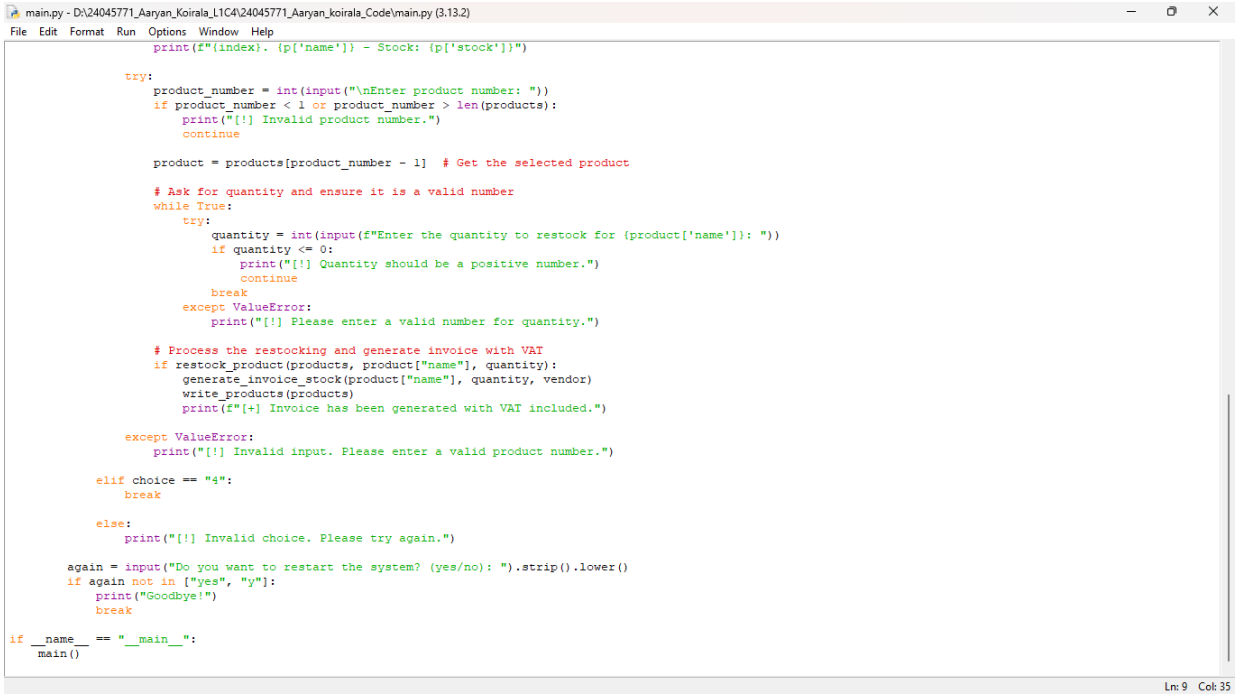

Figure 6. main

Operation.py

operation.py - D:\24045771_Aaryan_Koirala_L1C4\24045771_Aaryan_koirala_Code\operation.py (3.13.2)

File Edit Format Run Options Window Help

```
# operation.py

# VAT rate
VAT_RATE = 0.13  # 13% VAT

def display_products(products):
    print("\n{:<20} {:<15} {:<10} {:<15} {:<10}".format("Product", "Brand", "Stock", "Price (Rs)", "Origin"))
    print("-"*75)
    for p in products:
        # Calculate the price with VAT for display purpose only
        price_with_vat = p["selling_price"] * (1 + VAT_RATE)
        print("{:<20} {:<15} {:<10} {:<15} {:<10}".format(p["name"], p["brand"], p["stock"], f"Rs {price_with_vat:.2f}", p["origin"]))


def sell_product(products, product_name, quantity):
    for p in products:
        if p["name"].lower() == product_name.lower():
            if p["stock"] < quantity:
                print("[!] Not enough stock available.")
                return False

            free_items = quantity // 3  # Buy 3 Get 1 Free
            total_quantity = quantity + free_items

            if p["stock"] < total_quantity:
                print(f"[!] Not enough stock to apply offer. You can only buy up to {p['stock'] - (p['stock'] // 4)} units.")
                return False

            p["stock"] -= total_quantity
            print(f"[+] Sold {quantity} units of {p['name']}.")
            if free_items > 0:
                print(f"[🎁] You received {free_items} unit(s) FREE under the Buy 3 Get 1 offer!")
            return True
    print("[!] Product not found.")
    return False


def restock_product(products, product_name, quantity):
    for p in products:
        if p["name"].lower() == product_name.lower():
            p["stock"] += quantity
            return True
    print("[!] Product not found.")
    return False
```

Ln: 8 Col: 0

Figure 7. operation

Read.py

read.py - D:\24045771_Aaryan_Koirala_L1C4\24045771_Aaryan_koirala_Code\read.py (3.13.2)

File Edit Format Run Options Window Help

```
def read_products():
    products = []
    try:
        with open("products.txt", "r") as f:
            for line in f:
                if line.strip() == "":
                    continue
                name, brand, stock, price, origin = line.strip().split(",")
                products.append({
                    "name": name,
                    "brand": brand,
                    "stock": int(stock),
                    "selling_price": int(price),
                    "origin": origin
                })
    except FileNotFoundError:
        print("[!] products.txt not found.")
    return products
```

Figure 8. read

Write.py

write.py - D:\24045771_Aaryan_Koirala_L1C4\24045771_Aaryan_koirala_Code\write.py (3.13.2)

File Edit Format Run Options Window Help

```
# write.py
import datetime

def write_products(products):
    with open("products.txt", "w") as f:
        for p in products:
            line = f"{p['name']},{p['brand']},{p['stock']},{p['selling_price']},
            f.write(line)

def generate_invoice_sell(product_name, quantity, price, buyer_name):
    free_items = quantity // 3
    total_quantity = quantity + free_items
    total_price = quantity * price  # Use the original price, not VAT-inclusive

    with open("invoice.txt", "a") as f:
        f.write("===== SALE INVOICE =====\n")
        f.write("Date/Time: " + str(datetime.datetime.now()) + "\n")
        f.write(f"Buyer: {buyer_name}\n")
        f.write(f"Product: {product_name}\n")
        f.write(f"Purchased Quantity: {quantity}\n")
        f.write(f"Free Items (Buy 3 Get 1): {free_items}\n")
        f.write(f"Total Items Given: {total_quantity}\n")
        f.write(f"Unit Price: Rs {price}\n")  # Original price, not VAT-inclusiv
        f.write(f"Total Paid: Rs {total_price}\n")
        f.write("-"*40 + "\n")

def generate_invoice_stock(product_name, quantity, vendor_name):
    with open("invoice.txt", "a") as f:
        f.write("===== RESTOCK INVOICE =====\n")
        f.write("Date/Time: " + str(datetime.datetime.now()) + "\n")
        f.write(f"Vendor: {vendor_name}\n")
        f.write(f"Product: {product_name}\n")
        f.write(f"Quantity Restocked: {quantity}\n")
        f.write("-"*40 + "\n")
```

Figure 9. write

Testing

Test case 1: To display products

Table 1. Test 1

Objective	To run the program and display all the products
Action	To display products you should enter choice: “ 1 “
Expected Result	Product list along with its all the details saved in the products.txt file should be displayed.
Actual Result	The products displayed successfully with all the details.
Conclusion	Test was Successful.

Evidence:

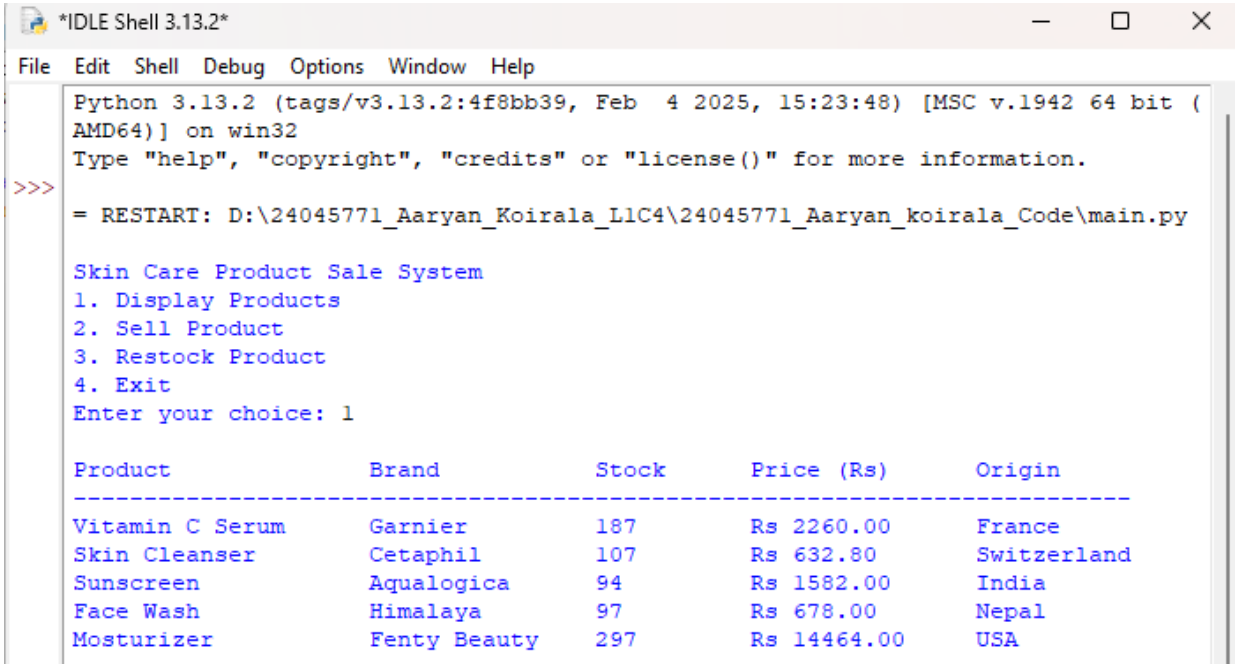

Figure 10 Test 1

Test case 2: To sell products

Table 2. Test 2

Objective	To sell the products to user.
Action	To sell products you should enter : “2”. Then, you should enter buyer’s name, product’s no & quantity of product’s.
Expected Result	After inputting all the data’s, it should print: v. Product name with unit is sold vi. Free items given vii. Alert for invoice generated.
Actual Result	As expected, It run successfully.
Conclusion	Test was Successful.

Evidence:

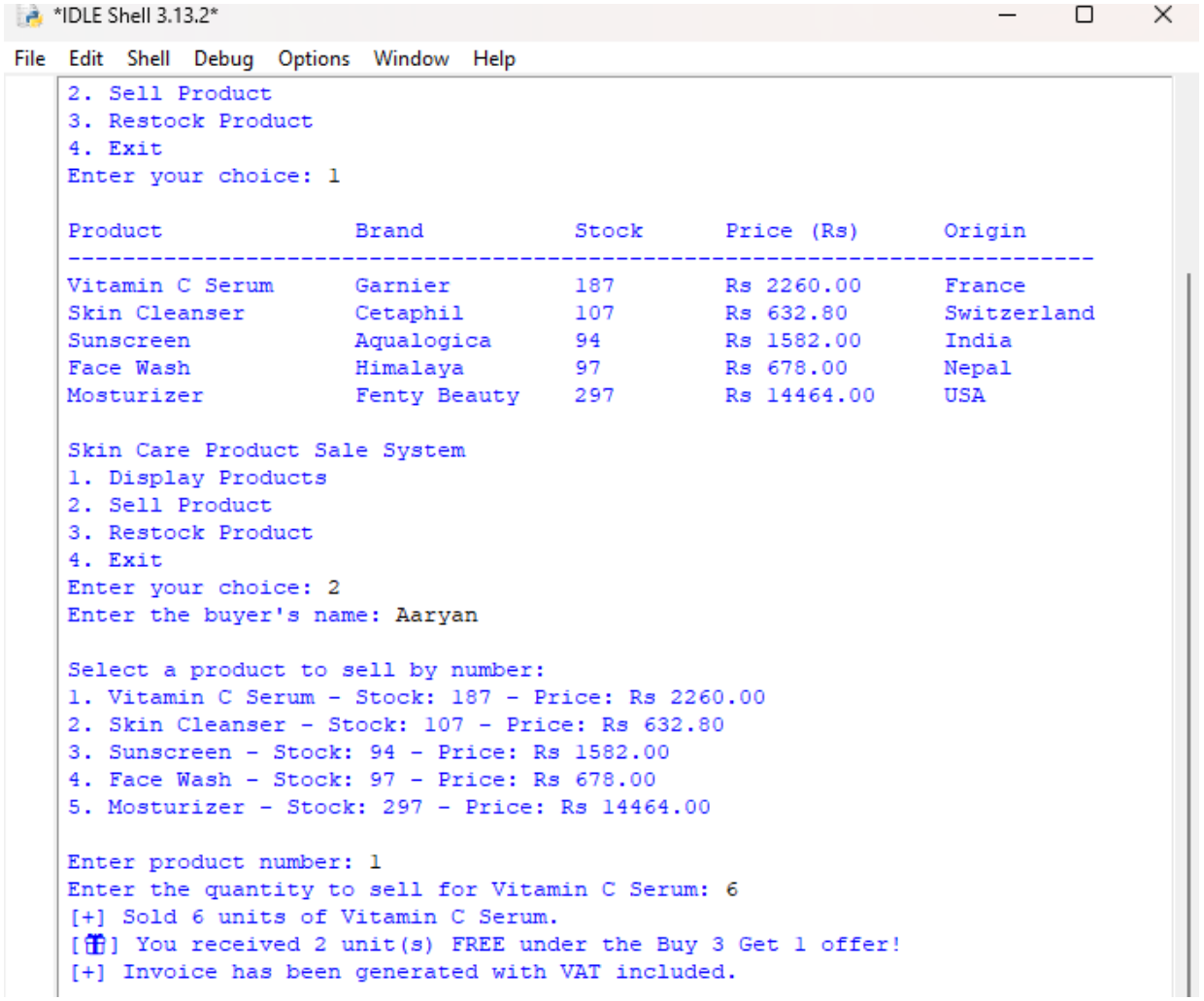

Figure 11. Test 2

```
===== SALE INVOICE =====
Date/Time: 2025-05-14 09:10:00.849873
Buyer: Aaryan
Product: Vitamin C Serum
Purchased Quantity: 6
Free Items (Buy 3 Get 1): 2
Total Items Given: 8
Unit Price: Rs 2000
Total Paid: Rs 12000
----------------------------------------
```

Figure 12. T2

Test case 3 : To Restock Products

Table 3. Test 3

Objective	To sell the products to user.
Action	To sell products you should enter : “2”. Then, you should enter buyer’s name, product’s no & quantity of product’s.
Expected Result	After inputting all the data’s, it should print: i. Product name with unit is sold ii. Free items given iii. Alert for invoice generated.
Actual Result	As expected, It run successfully.
Conclusion	Test was Successful.

Evidence:

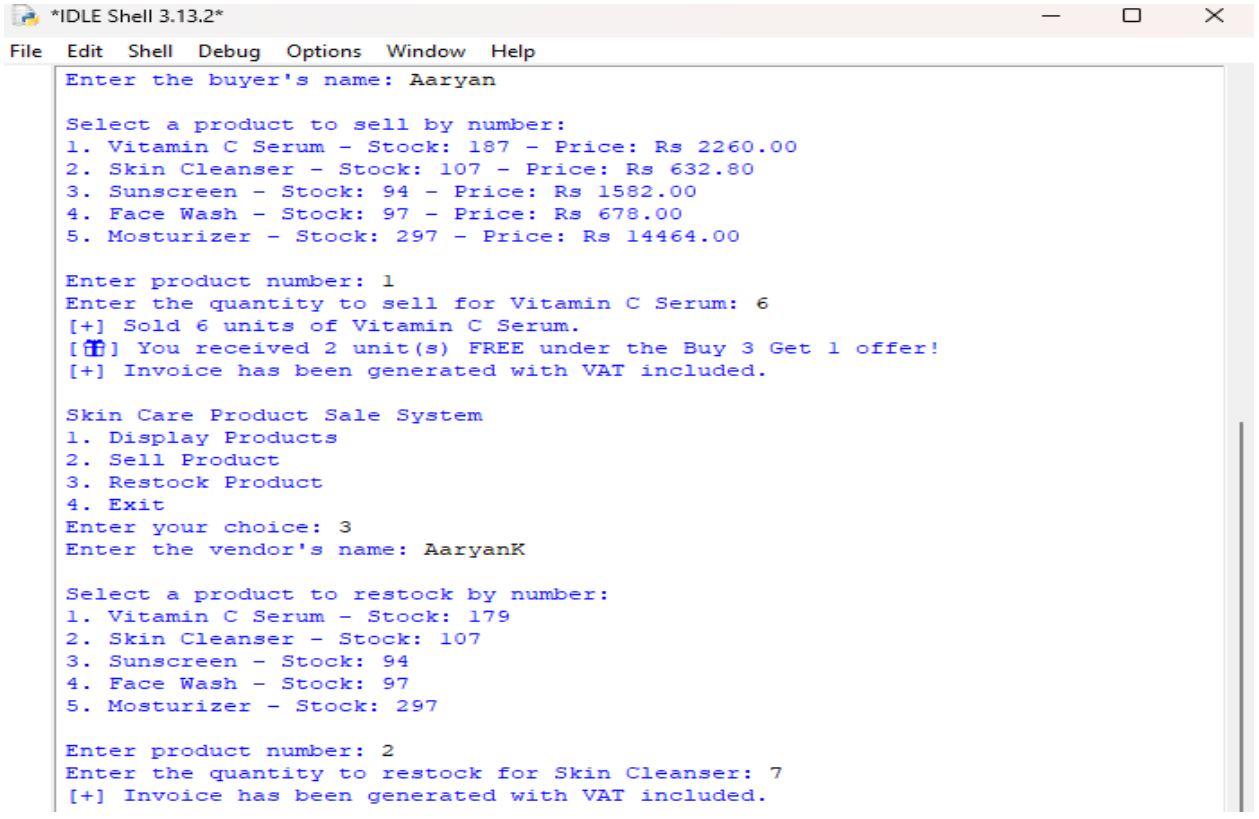

Figure 13. Test 3

```
===== RESTOCK INVOICE =====
Date/Time: 2025-05-14 09:11:18.115850
Vendor: AaryanK
Product: Skin Cleanser
Quantity Restocked: 7
----------------------------------------
```

Test case 4: To validate if you input negative value

Table 4. Test 4

Objective	To validate negative input.
Action	You should input negative value.
Expected Result	It should show " [!] Invalid choice. Please try again." If you input negative value.
Actual Result	As expected, It showed " [!] Invalid choice. Please try again."
Conclusion	Test was Successful.

Evidence:

```
Skin Care Product Sale System
1. Display Products
2. Sell Product
3. Restock Product
4. Exit
Enter your choice: -1
[!] Invalid choice. Please try again.
```

Figure 14 Test 4

Test Case 5: Buy 3 Get 1 Free

Table 5. Test 5

Objective	To give buyers 1 free when they buy 3 products.
Action	Buy 3 products so you get 1 free i.e if you buy 10 products you get 3 products free.
Expected Result	User should get 1 products free everytime they buy 3 products.
Actual Result	As expected,
Conclusion	Test case successful.

Evidence

:

```
Enter product number: 1
Enter the quantity to sell for Vitamin C Serum: 6
[+] Sold 6 units of Vitamin C Serum.
[🎁] You received 2 unit(s) FREE under the Buy 3 Get 1 offer!
[+] Invoice has been generated with VAT included.
```

Figure 15. Test case 5

```
===== SALE INVOICE =====
Date/Time: 2025-05-14 09:10:00.849873
Buyer: Aaryan
Product: Vitamin C Serum
Purchased Quantity: 6
Free Items (Buy 3 Get 1): 2
Total Items Given: 8
Unit Price: Rs 2000
Total Paid: Rs 12000
----------------------------------------
```

Figure 16. Test case 5

Data Analysis

Skin Care Product Sale System is an inventory and sales management application that is file based and can handle product data, customer transactions and restocking vendor in an efficient manner. It works with structured data that is contained in plain text files (products.txt, and invoice.txt), and it uses modular Python scripts to moderate separate functions.

The heart of the system concerns reading product information from a file, processing sales and restocks, updating inventory and producing transaction records. Product information like name, brand, stock level, price, and origin are loaded into memory by use of read product () function which allows manipulation and display in-memory.

In case, a user starts a sale, the system provides an opportunity to choose a product and its quantity applies a "Buy 3 Get 1 Free" promotional logic and defines the total payable amount with 13% VAT. The system also ensures that there is enough stock available before sale is concluded both on the paid and the free items. If successful, then the stock is accordingly updated, the changes are pushed back to the file using write products(), and a detailed invoice is added to invoice.txt using generate-invoice-sell().

In the same way, restocking is taken care of by validating the inputs of product selection and quantity, updating the stock in-memory, saving the changes, and recording restock activity with generate_invoice_stock().

The system makes sure that the data is stable as it ensures that the inputs are valid (e.g., positive amounts, correct product choice), and provides informative feedback in case of mistakes (e.g. with regards to inadequate stock or wrong product numbers). Any transaction is stamped and recorded, which allows traceability and fundamental historical analysis.

As a whole, the system illustrates a well-prescribed algorithm of basic inventory and transaction data processing, which will enable the effortless maintenance, transparency with invoices, and extension to such instruments as data visualization, reporting, and integration with the database.

Conclusion

Finally, the Skin Care Product Sale System has offered an overall opportunity to apply core concepts of computing, such as, file handling, modular programming, data structure implementation and user interface designing through Python. The system effectively automates product inventory management, provides support to transactional operations, and guarantees the creation of accurate and efficient invoices in the case of sales and restocking. Integration of a pricing model that is dynamic and logic for stock adjustment i.e. 'buy three get one free offer ‘, the system effectively simulates how retail operations work while improving interaction with users through persistent input validation and error handling.

During the construction, the project focused on the importance of structured programming and modularity. Functions were precisely designed to encompass processes such as reading the files, validating data, showing products, issuance of invoices, and stock updates. The selection of Python dictionaries and lists as the central data structures worked favorably in structuring information for products and also facilitating effective lookups and modifications.

Testing was of great importance to validate the system’s reliability. The application was tested for several products inputs and edge cases to achieve stability and correctness.

Sample outputs like transaction logs and invoice files depicted the system's functionality as was expected doing this with regard to all specified requirements.

Overall, this project has broadened my knowledge on program design and implementation considerably in practical settings. It also emphasized the need for clear documentation, algorithmic planning, and good robust exception handling. Looking ahead, such hands-on experience sets a great foundation for dealing with other more complex systems in the subsequent computing exercises.

Appendix

```
from read import read_products

from write import write_products, generate_invoice_sell, generate_invoice_stock

from operation import display_products, sell_product, restock_product


def main():
    while True:
        products = read_products()
        while True:
            print("\nSkin Care Product Sale System")
            print("1. Display Products")
            print("2. Sell Product")
            print("3. Restock Product")
            print("4. Exit")
```

```
choice = input("Enter your choice: ")

if choice == "1":
    display_products(products)

elif choice == "2":
    buyer = input("Enter the buyer's name: ")
    print("\nSelect a product to sell by number:")
    for index, p in enumerate(products, start=1):
        # Show price with VAT only when displaying products
        price_with_vat = p["selling_price"] * (1 + 0.13)  # VAT rate: 13%
      print(f"{index}. {p['name']} - Stock: {p['stock']} - Price: Rs {price_with_vat:.2f}")

    try:
        product_number = int(input("\nEnter product number: "))
        if product_number < 1 or product_number > len(products):
            print("[!] Invalid product number.")
            continue

        product = products[product_number - 1]  # Get the selected product

        # Ask for quantity and ensure it is a valid number
        while True:
            try:
```

```
                    quantity = int(input(f"Enter the quantity to sell for {product['name']}: "))
                    if quantity <= 0:
                        print("[!] Quantity should be a positive number.")
                        continue
                    break
                except ValueError:
                    print("[!] Please enter a valid number for quantity.")

                # Process the sale and generate invoice with VAT
                if sell_product(products, product["name"], quantity):
                    generate_invoice_sell(product["name"], quantity, product["selling_price"],
buyer)
                    write_products(products)
                    print(f"[+] Invoice has been generated with VAT included.")

            except ValueError:
                print("[!] Invalid input. Please enter a valid product number.")

        elif choice == "3":
            vendor = input("Enter the vendor's name: ")
            print("\nSelect a product to restock by number:")
            for index, p in enumerate(products, start=1):
                print(f"{index}. {p['name']} - Stock: {p['stock']}")

            try:
```

```
            product_number = int(input("\nEnter product number: "))
            if product_number < 1 or product_number > len(products):
                print("[!] Invalid product number.")
                continue

            product = products[product_number - 1]  # Get the selected product

            # Ask for quantity and ensure it is a valid number
            while True:
                try:
                    quantity = int(input(f"Enter the quantity to restock for {product['name']}:
"))
                    if quantity <= 0:
                        print("[!] Quantity should be a positive number.")
                        continue
                    break
                except ValueError:
                    print("[!] Please enter a valid number for quantity.")

            # Process the restocking and generate invoice with VAT
            if restock_product(products, product["name"], quantity):
                generate_invoice_stock(product["name"], quantity, vendor)
                write_products(products)
                print(f"[+] Invoice has been generated with VAT included.")
```

```
            except ValueError:
                print("[!] Invalid input. Please enter a valid product number.")

        elif choice == "4":
            break

        else:
            print("[!] Invalid choice. Please try again.")

    again = input("Do you want to restart the system? (yes/no): ").strip().lower()
    if again not in ["yes", "y"]:
        print("Goodbye!")
        break

if __name__ == "__main__":
    main()
# operation.py

# VAT rate
VAT_RATE = 0.13  # 13% VAT

def display_products(products):
    print("\n{:<20} {:<15} {:<10} {:<15} {:<10}".format("Product", "Brand", "Stock", "Price (Rs)", "Origin"))
    print("-"*75)
```

```
    for p in products:
        # Calculate the price with VAT for display purpose only
        price_with_vat = p["selling_price"] * (1 + VAT_RATE)
        print("{:<20} {:<15} {:<10} {:<15} {:<10}".format(p["name"], p["brand"], p["stock"], f"Rs {price_with_vat:.2f}", p["origin"]))

def sell_product(products, product_name, quantity):
    for p in products:
        if p["name"].lower() == product_name.lower():
            if p["stock"] < quantity:
                print("[!] Not enough stock available.")
                return False

            free_items = quantity // 3  # Buy 3 Get 1 Free
            total_quantity = quantity + free_items

            if p["stock"] < total_quantity:
                print(f"[!] Not enough stock to apply offer. You can only buy up to {p['stock'] - (p['stock'] // 4)} units.")
                return False

            p["stock"] -= total_quantity
            print(f"[+] Sold {quantity} units of {p['name']}.")
            if free_items > 0:
```

```
            print(f"[🎁] You received {free_items} unit(s) FREE under the Buy 3 Get 1 offer!")
            return True
    print("[!] Product not found.")
    return False

def restock_product(products, product_name, quantity):
    for p in products:
        if p["name"].lower() == product_name.lower():
            p["stock"] += quantity
            return True
    print("[!] Product not found.")
    return False

def read_products():
    products = []
    try:
        with open("products.txt", "r") as f:
            for line in f:
                if line.strip() == "":
                    continue
                name, brand, stock, price, origin = line.strip().split(",")
                products.append({
                    "name": name,
                    "brand": brand,
```

```
                "stock": int(stock),
                "selling_price": int(price),
                "origin": origin
            })
    except FileNotFoundError:
        print("[!] products.txt not found.")
    return products
# write.py
import datetime

def write_products(products):
    with open("products.txt", "w") as f:
        for p in products:
            line = f"{p['name']},{p['brand']},{p['stock']},{p['selling_price']},{p['origin']}\n"
            f.write(line)

def generate_invoice_sell(product_name, quantity, price, buyer_name):
    free_items = quantity // 3
    total_quantity = quantity + free_items
    total_price = quantity * price  # Use the original price, not VAT-inclusive price

    with open("invoice.txt", "a") as f:
        f.write("===== SALE INVOICE =====\n")
        f.write("Date/Time: " + str(datetime.datetime.now()) + "\n")
        f.write(f"Buyer: {buyer_name}\n")
```

```
        f.write(f"Product: {product_name}\n")
        f.write(f"Purchased Quantity: {quantity}\n")
        f.write(f"Free Items (Buy 3 Get 1): {free_items}\n")
        f.write(f"Total Items Given: {total_quantity}\n")
        f.write(f"Unit Price: Rs {price}\n")  # Original price, not VAT-inclusive
        f.write(f"Total Paid: Rs {total_price}\n")
        f.write("-"*40 + "\n")


def generate_invoice_stock(product_name, quantity, vendor_name):
    with open("invoice.txt", "a") as f:
        f.write("===== RESTOCK INVOICE =====\n")
        f.write("Date/Time: " + str(datetime.datetime.now()) + "\n")
        f.write(f"Vendor: {vendor_name}\n")
        f.write(f"Product: {product_name}\n")
        f.write(f"Quantity Restocked: {quantity}\n")
        f.write("-"*40 + "\n")
```