LONDON METROPOLITAN UNIVERSITY

Islington college
(इस्लिङ्टन कलेज)

CS4051NI/CC4059NI Fundamentals of Computing

70% Individual Coursework

2024/25 Spring

Student Name: Preeti Phuyal
London Met ID: 24046619
College ID: NP01AI4A240072
Assignment Due Date: Wednesday, May 14, 2025
Assignment Submission Date: Wednesday, May 14, 2025

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

Turnitin:

Preeti Phuyal.docx

Islington College,Nepal

Document Details

Submission ID
trn:oid:::3618:95789930

Submission Date
May 14, 2025, 10:46 AM GMT+5:45

Download Date
May 14, 2025, 10:47 AM GMT+5:45

File Name
Preeti Phuyal.docx

File Size
31.1 KB

45 Pages

4,290 Words

23,276 Characters

9% Overall Similarity

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

Match Groups

- **38** Not Cited or Quoted 9%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  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

- 4% Internet sources
- 0% Publications
- 8% 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

- **38** Not Cited or Quoted 9%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  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

4% Internet sources
0% Publications
8% 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	
islingtoncollege on 2025-05-13		1%
2	Submitted works	
islingtoncollege on 2025-05-13		<1%
3	Internet	
ojg39.gch2020.eu		<1%
4	Internet	
skillapp.co		<1%
5	Submitted works	
Johns Hopkins Unversity on 2023-09-23		<1%
6	Submitted works	

Table of Contents

Table of Tables

1 Introduction

This project focuses on building a Skin Care product sale system using Python. The system is intended for a local beauty and skincare product shop called WeCare, which needs help in keeping track of its products. The store currently maintains a list of items for sale in its .text file. But keeping track of stock, price changes, and product information by hand is time-consuming and prone to mistakes. This project aims to automate a part of that process using Python programming. This work will make it easier to see and handle information about products.

The program gets its information from a text file that has information about a product's name, brand name, quantity available, price, and the country where it was made. Once the data has been read, it will be put into the right data structures, such as lists and dictionaries. This milestone focuses on the basic stage: reading product information from the file, storing it properly, and displaying it clearly. This phase is a crucial step because it sets the stage for the whole application. Later, this system will be improved so that customers can buy things, bills can be printed, stock levels can be updated, and the store's "buy three, get one free" deal can be used.

Through this task, I also learned about file handling, working with strings, and using loops and data structures in Python. It gave me a clear idea of how basic inventory systems work in real life and how programming can help solve those problems.

1.1 Aims

To develop a python wholesale system that efficiently reads, stores, and displays product data from a structured text file in an organized and user-friendly manner**.**

1.2 Objectives

- To develop a program that reads product information from a .txt file.
- To store the extracted data using appropriate Python data structures such as lists and dictionaries.
- The goal is to use fundamental programming concepts such as file handling, loops, and functions.
- The goal is to develop the foundations for future features like selling products, giving away free items, creating invoices, and keeping track of stock.
- To apply simple programming ideas, like reading files, loops, conditionals, and working with strings.

1.3 Technologies/tools.

- **Python programming language**

  Python is a high-level, interpreted programming language that is popular due to its simplicity and readability**.** Like English, Python's easy-to-read syntax enhances code readability, making it ideal for both new and experienced developers. Python can be a very popular and powerful programming language in the modern programmatic world, as it is open-source and cross-platform and accepts any operating system (Windows, macOS, or Linux).

Figure 1 Python

- ***IDLE (Integrated Development and Learning Environment)***
  IDLE was used to write and run the Python code. It is the default editor for Python, and it was very handy for this project. It also allowed me to test my program very quickly and fix any small problems right away.

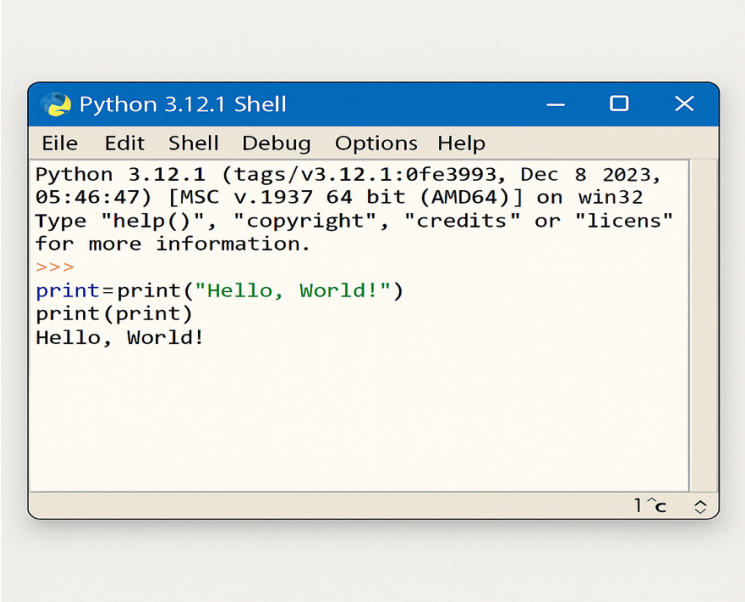


Figure 2: Idle code

- ***Terminal***

  The terminal (also called the command line, shell, or console) is a text-based interface that lets you interact with your computer by typing commands instead of clicking button.

Figure 3 Terminal text

- **Draw.io**

  Draw.io (also referred to as diagram.net) is an online free drawing tool for diagrams to create flowcharts, organizational charts, and network diagrams. Drag-and-drop features make it easy to use. Your diagrams can be saved in cloud storage facilities like Google Drive or Dropbox and even edited offline. It has many templates and shapes to choose from and can export diagrams to multiple formats like PNG or PDF. You can even collaborate with other people in real time, which is ideal for group projects. It is a multi-tool, ideal for both every day and technical diagramming.

Figure 4 Draw.io

1.4 Verdict

This project has enabled me to develop a basic inventory system for We Care, a local skincare store. I successfully read product data from a text file, organized that information appropriately using Python, and presented it in an appealing format. This approach saves time and reduces errors by eliminating reliance on handwritten records. Additionally, I learned fundamental programming concepts, including file handling, loops, and functions. While this prototype represents a basic system, it is intended to be enhanced in the future with features such as selling products and generating bills. Overall, this project demonstrated how simple Python skills can effectively address everyday challenges in a retail environment.

2 Data Structures

Python data structures are used to store and organize data in a format that allows for effective data manipulation and retrieval. Lists, tuples, sets, and dictionaries are the most common in-built data structures in Python.

The Data Structures which are mostly used in python are:

2.1 Primitive data types

Primitive data types are the most basic data types that are built into programming languages like Python. They represent single, indivisible values, such as numbers, characters, or true/false conditions. Primitive types are not made up of other types, and they are universal forming the foundation for most common programming tasks.

Primitive Data Types in Python are as follows:

2.1.1 Integer (Int)

The int data type in Python is used to store whole numbers. These are numbers without decimal points, and they can be positive, negative, or zero. Python can handle very large integers because it manages memory automatically. Integers are often used in counting, loops, indexing, and math operations. Common operations with integers include addition (+), subtraction (-), multiplication (*), and integer division (//).

For example:

My_list = [5, 10, 15, 20, 25]

This list contains only integer values.

2.1.2 Floating Point (float)

Python uses the float data type to store numbers with decimal points. It is useful for measurements, scientific data, money calculations, and anything that needs precision. You can also express floats in scientific notation, such as 1.5e2, which represents the value 150.0. Float numbers work with arithmetic just like integers, but occasionally you may see small rounding errors. This phenomenon happens because floats are stored in binary, which can't always perfectly represent decimal values.

For example:

```
my_list = [3.14, 0.5, 100.0, 1.5e2, -2.75]
```

This list contains floating-point numbers, including one in scientific notation (1.5e2 = 150.0) and a negative float.

2.1.3 Boolean (bool)

The bool type represents truth values and can hold only two possible states: true or false. These values are particularly useful in conditional logic and decision-making processes. Booleans typically arise from comparison operations. Additionally, if statements, loops, and logical expressions also utilize Booleans. In memory, True is stored as the integer 1, and False is stored as the integer 0. Although it's not their primary purpose, this storage method enables certain arithmetic operations on these integers.

For example:

]my_list = [True, False, 5>3, 2==2, 10 ! =5] This list contains Boolean values, including results from comparison operations (5 > 3 is True, 2 == 2 is True, 10 != 5 is True).

2.2 Non- Primitive Data types

In Python, non-primitive data types can store multiple values or structured data. Unlike primitive types, such as int, float, or bool, which hold a single value, non-primitive data types allow you to group related data together. Often, their functions are more robust.

Non-Primitive Data Types in Python are as follows:

2.2.1 List

A list in Python is a data structure that can hold more than one item in a sequence. It is like a container that can hold different pieces of information one after another. Commas separate the items within a list and place them inside square brackets. Lists can hold numbers, strings, or even other data structures like dictionaries.

For example:

my List = [1, 2, 3, "Two", 3.0]

This list contains integers, a string, and a floating-point number.

2.2.2 Dictionary

A dictionary stores data as key-value pairs. I used a dictionary to keep each product assigned to a unique ID. The ID is the key, and the value is the list of details. A dictionary is a collection which is ordered, changeable and do not allow duplicates.

For example:

myDict = {"name": "Preeti", "age": 20, "city": "Kathmandu"}

The dictionaries contain the key-value pairs where the keys are string, and the values can be any data type.

2.2.3 Set

Sets are used to store multiple items in a single variable. Set is one of 4 built-in data types in Python used to store collections of data. Python sets can store heterogeneous elements in them, i.e., a set can store a mixture

of string, integer, Boolean, etc. datatypes. Sets are enclosed in curly braces {} or can be created using the set () constructor.

For example:

my Set = {1,2,3,4,5,6}

This set contains integers and does not allow duplicate elements.

2.2.4 Tuple

A tuple is a built-in data type in Python used to hold an ordered collection of items. However, tuples differ from lists in that their items remain immutable once they are assigned. Their immutability allows them to function as dictionary keys and provides enhanced memory efficiency. Iteration, slicing, and indexing are all common operations performed on tuples. They are beneficial for situations requiring a fixed set of values due to their immutability, which ensures data integrity.

For example:

my Tuple = [1, 2, 3, "Two", 4.0]

This tuple contains integers, a string, and a floating-point number.

2.2.5 String

A string is a sequence of characters enclosed in either single quotes (') or double quotes (“”). Strings are immutable, which means that their contents cannot be changed after creation. Strings are useful when you need to store and manipulate text or character data.

For example:

my String = "Hello, World!"

This line stores the text "Hello, World!" in the variable my String.

2.3 DATA STRUCTURES I USED IN MY PROJECT

In my project, I mainly used dictionaries, lists, and strings as the main data structures and as means of managing the inventory and processing transactions. The primary data structure is the dictionary, where all the product information is stored, keyed by the product ID. The dictionary, called product dict, has each product ID mapped to a list, which contains the product name, cost price, and available quantity associated with that product ID. A dictionary is an orderly data structure (in this case) and quickly stores the product information for future sales and purchases.

Since the system will have transactions that could include multiple products, it must then use a list of lists data structure. This means the total transactions can contain multiple product transactions; each selling or purchasing transaction has every product represented by a list that logs the product's ID, name, unit selling price, quantity, and total price. The system then stores the product lists in a larger list, either sale items or restock items, which includes the entire transaction. This feature is useful because you can now loop through all the items in a transaction instead of just looping through each of the products. This process results in the creation of itemized invoices for both sales and restocking transactions.

In addition to utilizing the dictionary and list of lists, the program primarily employs string and integer data types for the product's ID, name, prices, quantities, timestamps (which may change in the event of a sale), and filenames. These fundamental data types are crucial for accurate calculations, formatting invoice details, and performing input and output operations for the user.

Screenshot:

```
# Function to save updated product data to the file
def save_products(product):
    with open("product.txt", "w") as file:  # Open product file in write mode
        for pid in product:  # Iterate over the product dictionary
            line = product[pid][0] + "," + product[pid][1] + "," + str(product[pid][2]) + "," + str(product[pid][3]) + "," + product[pid][4] + "\n"
            # Prepare each product's data in comma-separated format
            file.write(line)  # Write each product's data to the file
    print("Stock updated and saved!")  # Notify the user that data was saved
```

Figure 5 product is dictionary

Here, product is a dictionary where each key-value pair stores details about a product.

```
        # Add sold item to sold items list for invoice generation
        sold_items.append([name, brand, country, quantity, free_items, total_items, price_for_product])

    if len(sold_items) == 0:  # If no items were sold, notify the user
        print("No valid items were purchased.")
        return

    print("\nTransaction Successful!")
    print("Total Price: Rs. " + str(total_price))

    # Calculate VAT and total with VAT
    vat_amount = (total_price * vat_percentage) / 100
    total_with_vat = total_price + vat_amount

    # Ask if the user wants to add shipping charge
    shipping_charge = input("Do you want to add shipping charge? (yes/no): ").lower()
    if shipping_charge == "yes":
        shipping_fee = float(input("Enter shipping charge: "))
    else:
        shipping_fee = 0.0

    final_total = total_with_vat + shipping_fee  # Calculate the final total

    # Generate the invoice timestamp and filename
    time = datetime.datetime.now()


    # Call function to generate the invoice
    generate_invoice(customer_name, timestamp, sold_items, total_price, vat_percentage, shipping_fee, final_total)

except ValueError:
    print("Invalid input! Please enter valid numbers.")  # Handle invalid inputs
```

Figure 6 Using List

Using List for Sold Items in handle_sale()

```python
# Function to handle sale operations (customer buying products)
def handle_sale(product):
    try:
        customer_name = input("Enter customer name: ")
        if not isinstance(customer_name, str) or customer_name.strip().isdigit():
            print("Invalid customer name. A name cannot be a number. Please enter a valid name.")
            return "error"
        timestamp = datetime.datetime.now()
        total_price = 0  # Initialize total price of sale
        sold_items = []  # List to store sold items for the invoice
        vat_percentage = 13  # Example VAT percentage
```

Figure 7 Using String

String type validation to ensure the customer name is entered as a valid non-numeric string, with any extra spaces removed using strip().

3 Algorithm

A well-defined series of steps given to solve a problem or carry out a particular task defines an algorithm. An algorithm provides a logical and clear sequence of actions to achieve a desired outcome. Though there are numerous fields where algorithms are useful, they are most often used on computers, where they help manage data and execute programs. A good algorithm is one that is correct, runs rapidly, and performs its task in a defined number of steps. It ensures quick, accurate, and consistent problem-solving or workflow automation.

The Algorithm of the program to make skin care sale system is given below:

Step 1: START

Step 2: Load product data from "product.txt" and store it in a dictionary.

Step 3: Display the main menu with options:

- 0 to Display Products
- 1 to Sale
- 2 to Restock
- 3 to Exit

Step 4: Ask the user for their choice.

- **If user choice == 0 (Display Products):**

Step 5: Display all products: ID, Name, Brand, Quantity, Selling Price, and Country.

Step 6: Return to Step 3.

- **If user choice == 1 (Sale):**

Step 7: Ask for customer name and initialize total price to 0.

Step 8: Ask for product ID and quantity.

Step 9: Apply "buy 3, get 1 free" offer and check stock.

- If stock is sufficient, continue to Step 10.
- If stock is insufficient, display "Item unavailable" and return to Step 3.

Step 10: Deduct sold items from stock, calculate total price considering the offer.

Step 11: Generate and save sale invoice.

Step 12: Return to Step 3.

- **If user choice == 2 (Restock):**

Step 13: Ask for supplier name and restock details (product ID, quantity, and cost price).

Step 14: Update stock and cost price in the dictionary.

Step 15: Save restock invoice.

Step 16: Return to Step 3.

- **If user choice == 3 (Exit):**

Step 17: Write updated data to "product.txt".

Step 18: End the program.

In summary, the algorithm starts with a file reader that loads product data and presents a main menu that consists of four options: 0 to display products, 1 to make a sale, 2 to restock products, and 3, which allows the user to exit the program. Upon selection of Display Option (0), the model displays a table containing the skincare items on offer with all the required pricing details. When the user chooses to process a sale (1), the user is prompted for a customer name, and multiple product sales can be completed. The algorithm applies the buy three, get one promotion while decreasing the stock of the product, calculates the total amount, and generates a detailed sale invoice containing details of time and date. If option 2 is chosen, the program asks for supplier details, adjusts stock based on the specified quantity and invoice cost price, and creates a stock invoice. When option 3 is selected, the loop ends, and any updated product data is saved back to the original file. This system efficiently manages skincare product inventory and customer sales while tracking stock restocking.

4 Pseudocode

```
BEGIN
  product ← read_products()
  main_loop ← TRUE

  WHILE main_loop DO
    DISPLAY "CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->"
    DISPLAY "0 to Display Products"
    DISPLAY "1 to Sale"
    DISPLAY "2 to Restock"
    DISPLAY "3 to Exit"

    INPUT choice

    IF choice = 0 THEN
      DISPLAY "THESE ARE THE AVAILABLE PRODUCTS IN THE STOCK -->"
      CALL display_products(product)
    ELSE IF choice = 1 THEN
      INPUT customer_name
      IF customer_name is invalid THEN
        DISPLAY "Invalid customer name"
      ELSE
        total_price ← 0
        sold_items ← []

        REPEAT
          INPUT product_id
          IF product_id ≠ 0 THEN
            IF product_id exists in product THEN
              INPUT quantity

              free_items ← quantity ÷ 3 (integer division)
              total_items ← quantity + free_items

              IF stock ≥ total_items THEN
                selling_price ← cost_price × 2
                price_for_product ← quantity × selling_price
                total_price ← total_price + price_for_product
                stock ← stock - total_items

                ADD item details to sold_items
              ELSE
                DISPLAY "Not enough stock!"
              END IF
            ELSE
              DISPLAY "Invalid Product ID!"
```

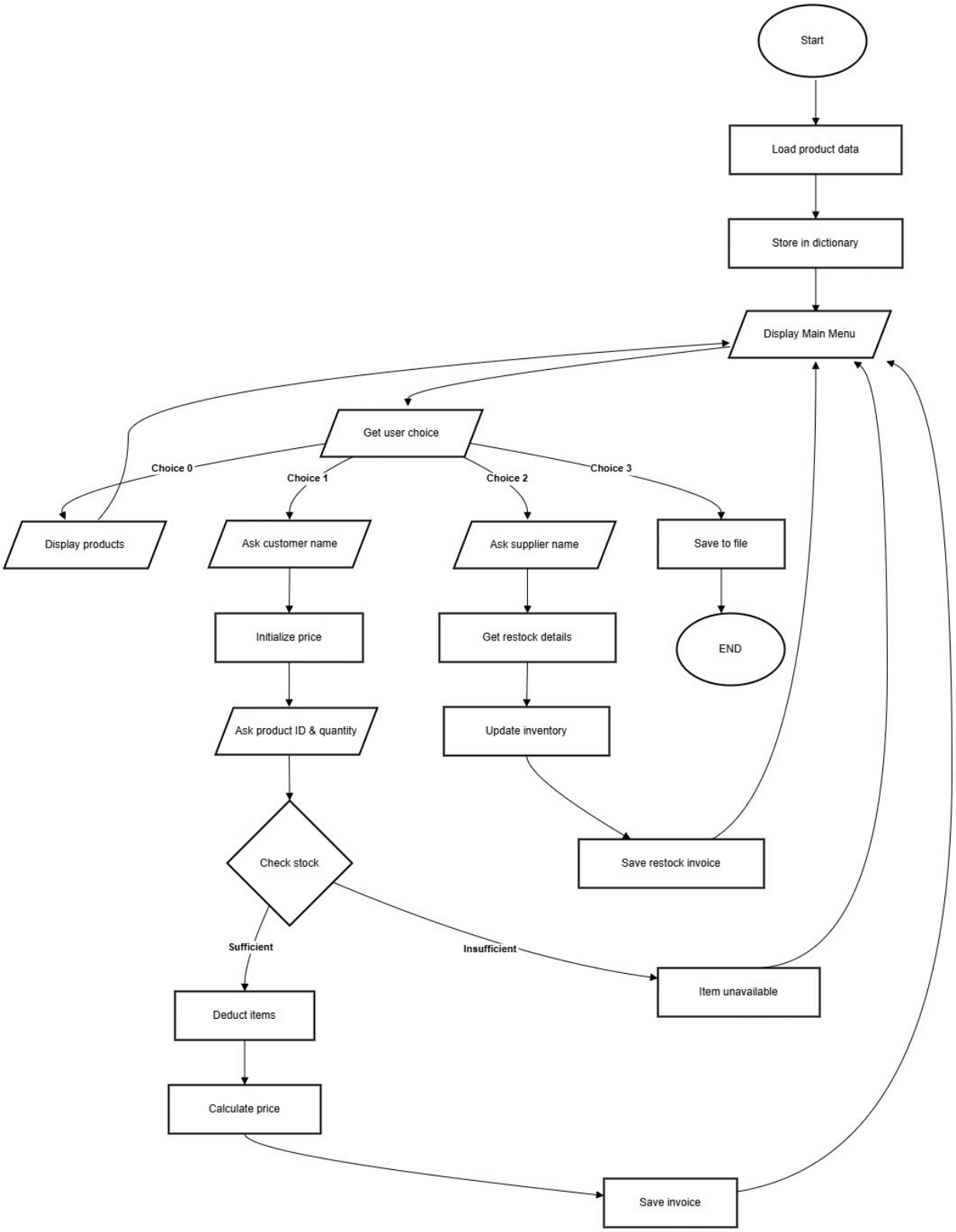
```
                END IF
            END IF
        UNTIL product_id = 0

        IF sold_items is not empty THEN
            CALCULATE VAT and shipping
            CALL generate_invoice(customer_name, timestamp, sold_items, total_price,
vat_percentage, shipping_fee, total_with_vat)
            CALL save_products(product)
        END IF
    END IF
  ELSE IF choice = 2 THEN
    INPUT supplier_name
    INPUT vat_percentage
    total_cost ← 0
    restock_items ← []

    REPEAT
      INPUT product_id
      IF product_id ≠ 0 THEN
        IF product_id exists in product THEN
          INPUT restock_quantity
          INPUT new_cost_price

          stock ← stock + restock_quantity
          cost_price ← new_cost_price

          item_cost ← restock_quantity × new_cost_price
          total_cost ← total_cost + item_cost

          ADD item details to restock_items
        ELSE
          DISPLAY "Invalid Product ID!"
        END IF
      END IF
    UNTIL product_id = 0

    IF restock_items is not empty THEN
      CALCULATE VAT
      CREATE and SAVE restock invoice
      CALL save_products(product)
    END IF
  ELSE IF choice = 3 THEN
    DISPLAY "Thank you for using our system!"
    CALL save_products(product)
    main_loop ← FALSE
```

```
        ELSE
            DISPLAY "Invalid input. Please select from 0 to 3."
        END IF
    END WHILE
END

FUNCTION read_products()
    product ← {}
    OPEN "product.txt" in read mode
    data ← all lines from file

    pid ← 1
    FOR each line in data DO
        line ← line.replace("\n", "").split(",")
        line[2] ← INTEGER(line[2])
        line[3] ← INTEGER(line[3])
        product[pid] ← line
        pid ← pid + 1
    END FOR

    RETURN product
END FUNCTION

FUNCTION save_products(product)
    OPEN "product.txt" in write mode
    FOR each pid in product DO
        line ← format product data with commas
        WRITE line to file
    END FOR
    DISPLAY "Stock updated and saved!"
END FUNCTION

FUNCTION display_products(product)
    DISPLAY table header
    FOR each item in product DO
        DISPLAY formatted product details
    END FOR
    DISPLAY table footer
END FUNCTION

FUNCTION generate_invoice(customer_name, timestamp, sold_items, total_price,
vat_percentage, shipping_fee, total_with_vat)
    invoice_name ← "Sales_Invoices/sale_invoice_" + customer_name + "_" + timestamp + ".txt"

    CREATE invoice content with headers, item details, and totals
```

```
    DISPLAY invoice preview
    WRITE invoice to file

    DISPLAY "Sale invoice created successfully!"
END FUNCTION
```

5 FlowChart

A flowchart is a graphical representation of a process, system, or algorithm with specific symbols to represent different operations or conclusions. It simplifies complex tasks by breaking them down into small, understandable steps and illustrating the order of task execution along with their relationships. Flowcharts typically use standardized symbols such as ovals for the beginning and end points, rectangles for processing points, diamonds for decision points, and arrows to represent process flow. Flowcharts are widely used by programmers, system architects, business process managers, and problem solvers to strategize, document, and analyze workflows. Flowcharts help people understand problems, increase efficiency, and communicate ideas clearly to others by visualizing information.


Start
Load product data
Store in dictionary
Display Main Menu
Get user choice
Choice 0
Choice 1
Choice 2
Choice 3
Display products
Ask customer name
Ask supplier name
Save to file
Initialize price
Get restock details
END
Ask product ID & quantity
Update inventory
Check stock
Save restock invoice
Sufficient
Insufficient
Item unavailable
Deduct items
Calculate price
Save invoice

6 Program:

6.1 Implementation of the program (Overall program with short explanation)

The purpose of the program is to keep track of product inventory and the sales process, and it includes restocking as well. We break down the program into four modules, each with a specific purpose to enforce separation of concerns.

The main.py module is the entry point of the application, presents a user interface, displays a menu of all of the functional options (viewing products, selling items, restocking) and then makes calls to functions from the other three modules based on user input. It also does input validation and has a looping function contained within it to permit ongoing interaction until the user decides to terminate it. As such, these four modules work together as an efficient and easy to use inventory and billing management program.

The read.py is used to read data on products from a file (product.txt, for example) and return it as a dictionary. It reads a line, splits it by commas, and then converts certain values (quantity and price) to integers, It then collects that data and a unique product ID to allow the program to manage products efficiently and retrieve and update product information.

The file write.py saves the updated product data back into the file and processes invoices for sales transactions. It has a function that saves the product information that was added to the product.txt file after a sale or restock, and it also has a function that creates an invoice detailing each sale. The invoice is printed to the terminal as well as saved in a text file with a unique name based on the customer's name and timestamp.

Operation.py contains the main business logic for sales and restocking activities. The program allows for "buy 3 and receive 1 free," calculates the total price, applies

VAT, and trains support for optional shipping fees. The program can also update the quantity of products after transactions, create invoices as per transactions, ensure the transactions are recorded, and account for stock levels accordingly.

Together, these modules successfully provide a functional system for maintaining an effective and organized inventory system that manages customer transactions, produces restocking orders, creates invoices for each sale, and reports current product data.

Showing the complete process for the purchase and sale of the product:

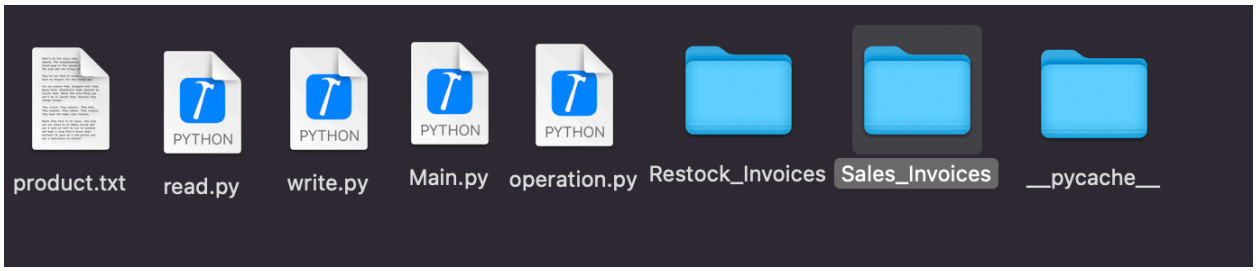
FOR SELLING OF Products:

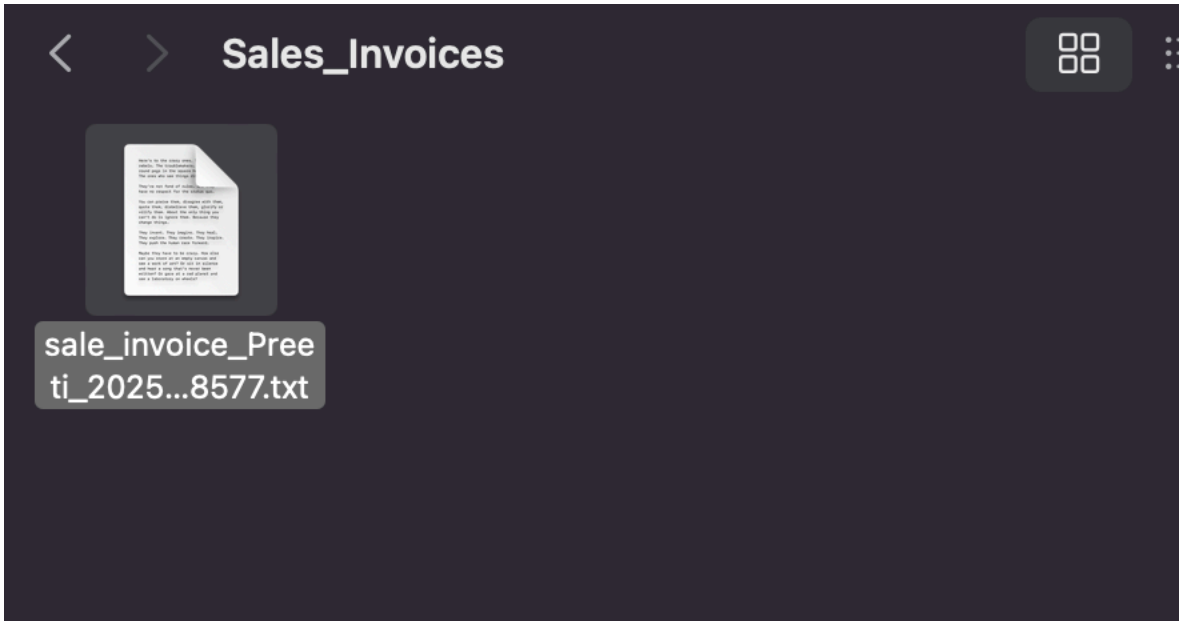
```
---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action                |
|--------|-----------------------|
|   0    | Display product       |
|   1    | Sale (Customer Buy)   |
|   2    | Purchase (Restock)    |
|   3    | Exit                  |
---------------------------------
>> Enter your choice: 0

# THESE ARE THE AVAILABLE PRODUCTS IN THE STOCK -->
-------------------------------------------------------------------------------------
ID     |Name               |Brand           |Qty        |Price       |Country
-------------------------------------------------------------------------------------
|1     |Vitamin C Serum    |Garnier         |73         |300         |France
-------------------------------------------------------------------------------------
|2     |Skin Cleanser      |Cetaphil        |20         |200         |Switzerland
-------------------------------------------------------------------------------------
|3     |Sunscreen          |Aqualogica      |152        |444         |India
-------------------------------------------------------------------------------------

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action                |
|--------|-----------------------|
|   0    | Display product       |
|   1    | Sale (Customer Buy)   |
|   2    | Purchase (Restock)    |
|   3    | Exit                  |
---------------------------------
>> Enter your choice: 1
Enter customer name: Preeti
Enter product ID to buy (0 to finish): 2
Enter quantity to buy for Skin Cleanser: 5
Enter product ID to buy (0 to finish): 3
Enter quantity to buy for Sunscreen: 2
Enter product ID to buy (0 to finish): 0

Transaction Successful!
Total Price: Rs. 3776
Do you want to add shipping charge? (yes/no): yes
Enter shipping charge: 100
```

```
=== INVOICE PREVIEW ===
+------------------------------------------------------------+
|                      SALES INVOICE                         |
+------------------------------------------------------------+
| Customer Name: Preeti
| Date: 2025-05-14 07:28:49.298577
+------------------------------------------------------------+
+-----------------+---------------+-----+------+------------+------------+
| Product         | Brand         | Qty | Free | Unit Price | Amount     |
+-----------------+---------------+-----+------+------------+------------+
| Skin Cleanser   | Cetaphil      |   5 |    1 |        400 |       2000 |
| Sunscreen       | Aqualogica    |   2 |    0 |        888 |       1776 |
+-----------------+---------------+-----+------+------------+------------+
| Subtotal      : Rs.       3776 |
| VAT (13%)     : Rs.     490.88 |
| Shipping      : Rs.      100.0 |
+------------------------------------------------------------+
| GRAND TOTAL   : Rs.    4466.88 |
+------------------------------------------------------------+

Sale invoice created successfully!
Invoice saved as: Sales_Invoices/sale_invoice_Preeti_2025-05-14 07:28:49.298577.txt
Total amount: Rs. 4466.88
Stock updated and saved!
```

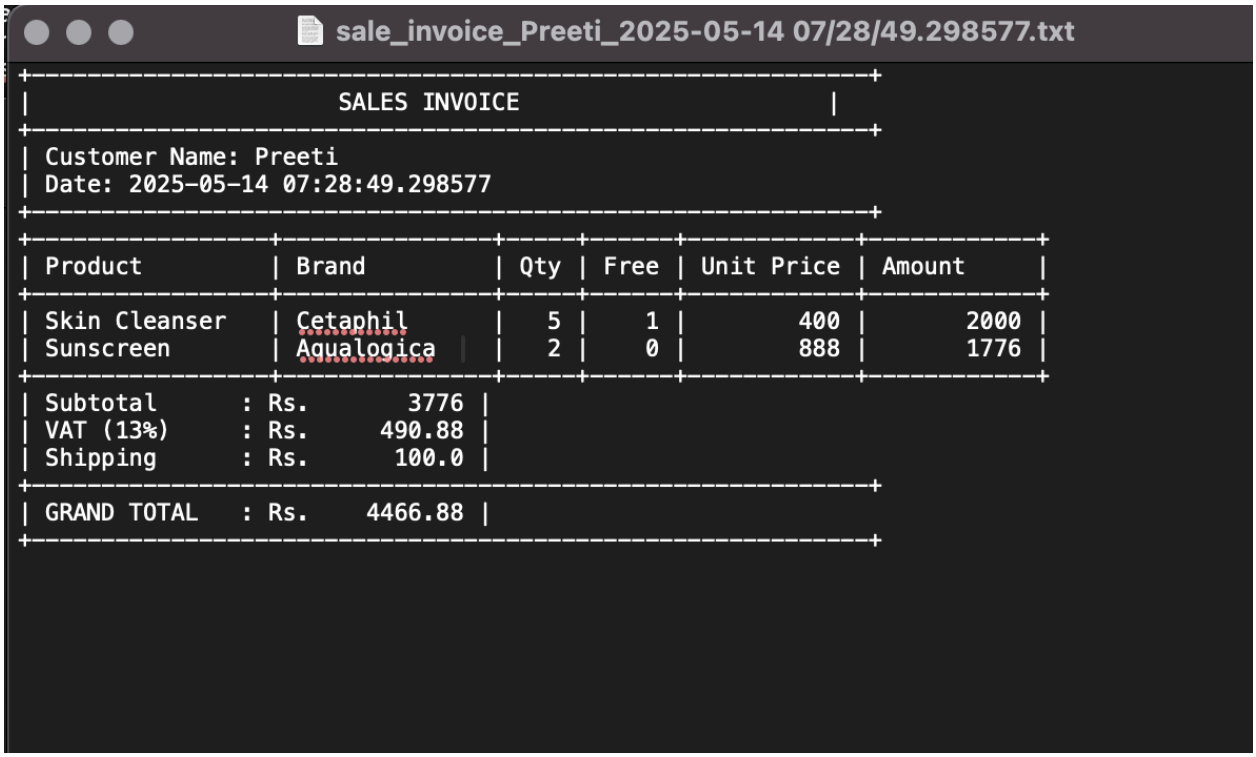
```
----------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
----------------------------------
| Option | Action                |
|--------|-----------------------|
|   0    | Display product       |
|   1    | Sale (Customer Buy)   |
|   2    | Purchase (Restock)    |
|   3    | Exit                  |
----------------------------------
>> Enter your choice: 3
Thank you for using our system!
Stock updated and saved!
```

SHOWING CREATION OF TXT FILE AFTER SELL OF Products:





OPENING THE TEXT FILE AND SHOWING THE BILL

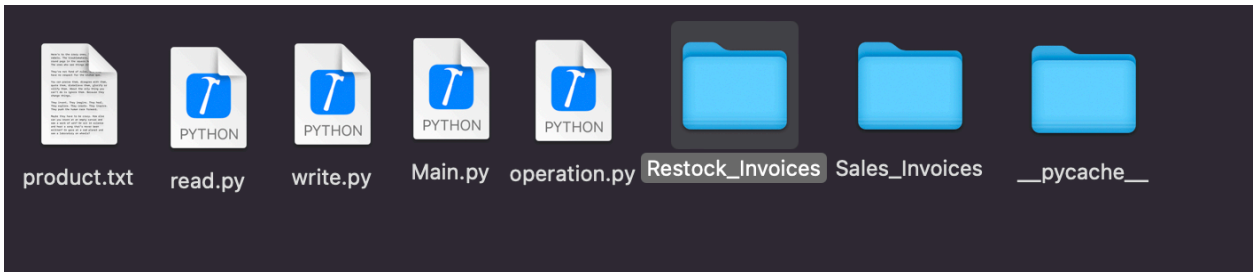


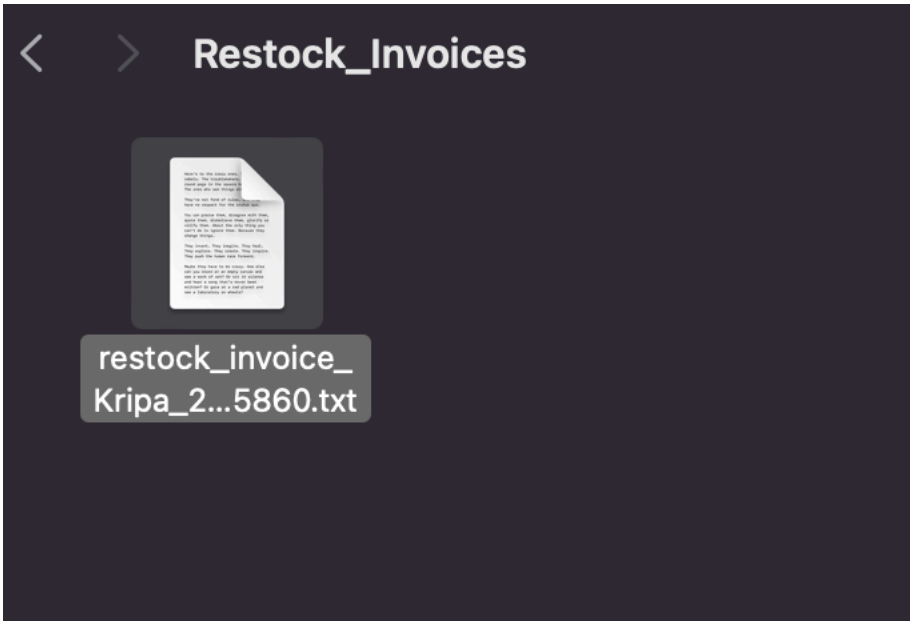
FOR Purchase/Restock OF Products:

```
---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
>> Enter your choice: 2
Enter supplier name: Kripa
Enter VAT percentage: 13
Enter product ID to restock (0 to finish): 2
Enter restock quantity for Skin Cleanser: 4
Enter new cost price for Skin Cleanser: 400
Enter product ID to restock (0 to finish): 3
Enter restock quantity for Sunscreen: 2
Enter new cost price for Sunscreen: 333
Enter product ID to restock (0 to finish): 0

Restock Successful!
Total Restock Cost: Rs. 2266
VAT Amount (13.0%): Rs. 294.58
Total Cost with VAT: Rs. 2560.58

==================================================
RESTOCK INVOICE
==================================================
Supplier Name: Kripa
Date: 2025-05-14 07:40:58.385860
--------------------------------------------------
Product: Skin Cleanser
Brand: Cetaphil
Country: Switzerland
Quantity Restocked: 4
New Cost Price: Rs. 400
Item Total Cost: Rs. 1600
--------------------------------------------------
Product: Sunscreen
Brand: Aqualogica
Country: India
Quantity Restocked: 2
New Cost Price: Rs. 333
Item Total Cost: Rs. 666
--------------------------------------------------
Total Cost: Rs. 2266
VAT Amount (13.0%): Rs. 294.58
Total Cost with VAT: Rs. 2560.58
==================================================
Restock invoice saved as: Restock_Invoices/restock_invoice_Kripa_2025-05-14_07-40-58.385860.txt
Stock updated and saved!
```

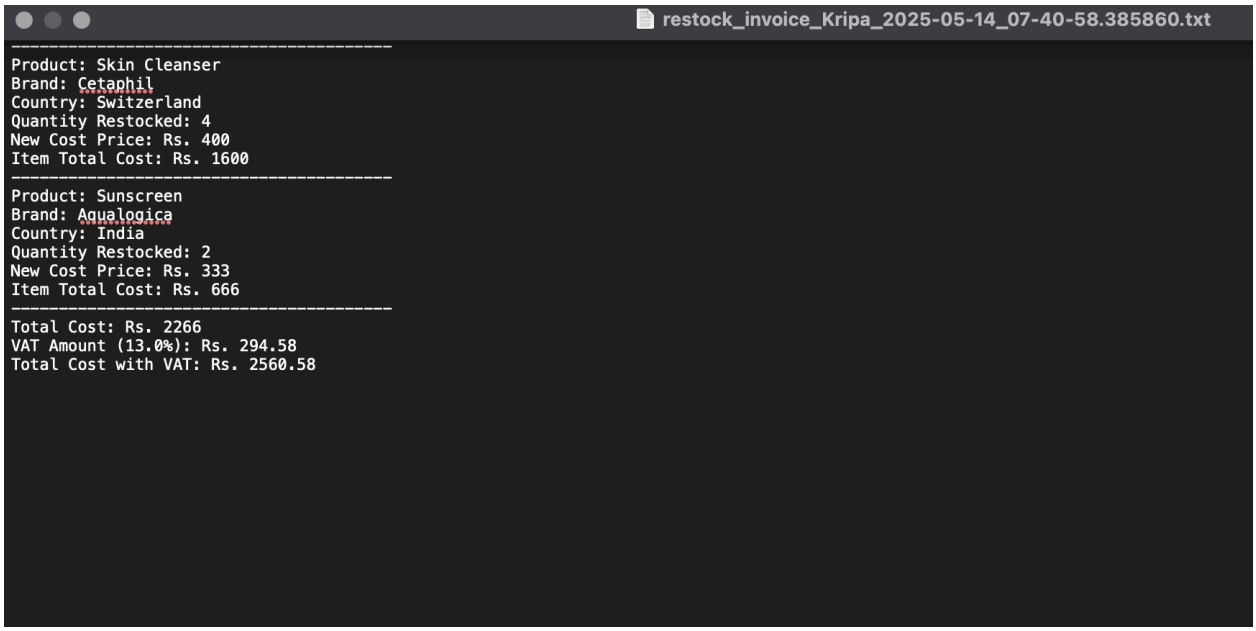
```
---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action                 |
|--------|------------------------|
|   0    | Display product        |
|   1    | Sale (Customer Buy)    |
|   2    | Purchase (Restock)     |
|   3    | Exit                   |
---------------------------------
>> Enter your choice: 3
Thank you for using our system!
Stock updated and saved!
```

SHOWING CREATION OF TXT FILE AFTER Purchase/Restock OF Products:





OPENING THE TEXT FILE AND SHOWING THE BILL



7 Testing

Test 1 – To Show Implementation and Working of try, except

Test 1	
Objective	To Show Implementation and Working of try, except.
Action	At the menu or quantity prompt, input a non-integer value (e.g., abc).
Expected Result	Program should catch the error and display: Invalid input! Please enter a valid number.
Actual Result	Program displayed: Invalid input! Please enter a valid number.
Conclusion	The test was successful.

Table 1:To Show Implementation and Working of try, except

```python
try:
    choice = int(input(">> Enter your choice: "))  # Get user input for the action

    if choice == 0:
        print("\n# THESE ARE THE AVAILABLE PRODUCTS IN THE STOCK -->")
        display_products(product)  # Display the available products
    elif choice == 1:
        flag = handle_sale(product)
        if flag == "error":
            continue
        else:
            save_products(product)  # Save updated stock
    elif choice == 2:
        handle_restock(product)  # Handle product restocking
        save_products(product)  # Save updated stock


    elif choice == 3:
        print("Thank you for using our system!")  # Exit message
        main_loop = False  # End the main loop
    else:
        print("Invalid input. Please select from 0 to 3.")  # Handle invalid input
except ValueError:
    print("Invalid input! Please enter a number.")  # Handle invalid input
```

Figure 8 Screenshot of Implementation of try, except in code

```
>> Enter your choice: 0

# THESE ARE THE AVAILABLE PRODUCTS IN THE STOCK -->
-------------------------------------------------------------------------------
ID     |Name               |Brand           |Qty        |Price       |Country
-------------------------------------------------------------------------------
|1     |Vitamin C Serum    |Garnier         |49         |34          |France
-------------------------------------------------------------------------------
|2     |Skin Cleanser      |Cetaphil        |29         |200         |Switzerland
-------------------------------------------------------------------------------
|3     |Sunscreen          |Aqualogica      |147        |700         |India
-------------------------------------------------------------------------------

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
>> Enter your choice: 1
Enter customer name: preeti
Enter product ID to buy (0 to finish): abc
Invalid input! Please enter valid numbers.
Stock updated and saved!

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
>> Enter your choice:
```

Figure 9 Screenshot of Working of try, catch

Test 2-Selection purchase and sale of products

Test 2	
Objective	To test error handling when selecting a product with negative or non-existent product ID.
Action	1. Enter a negative product ID (e.g., -1) during purchase/sale. 2. Enter a non-existent product ID (e.g., 8).
Expected Result	Program should catch the error and display: Invalid Product ID! or a similar message.
Actual Result	Program displayed: Invalid Product ID!
Conclusion	The test was successful.

Table 2: Selection purchase and sale of products

```
while True:
    product_id = int(input("Enter product ID to buy (0 to finish): "))  # Get product ID
    if product_id == 0:  # If user enters 0, end the sale
        break

    if product_id not in product:  # If product ID is invalid
        print("Invalid Product ID!")
        continue
```

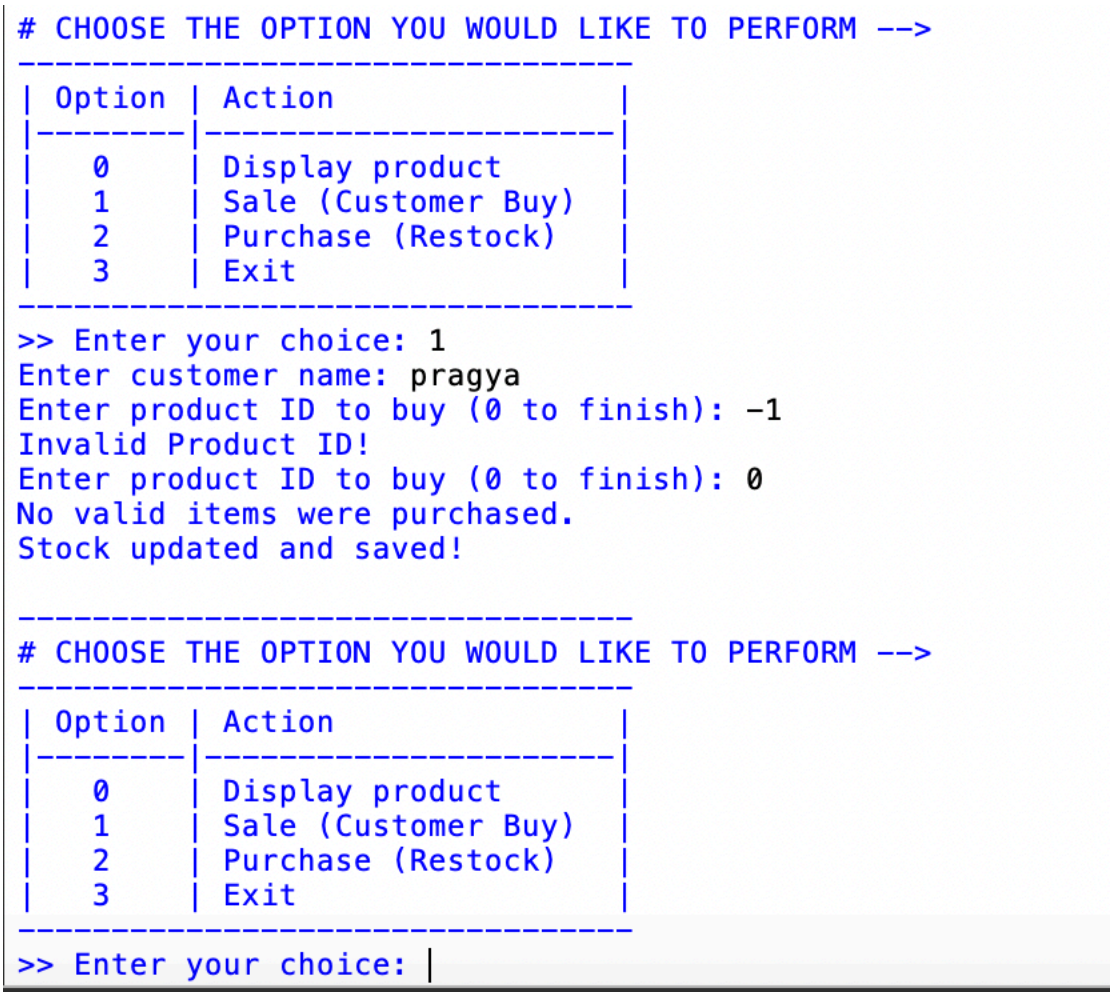


Figure 10 Screenshot of negative product ID as Input

```
---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action                |
|--------|-----------------------|
|   0    | Display product       |
|   1    | Sale (Customer Buy)   |
|   2    | Purchase (Restock)    |
|   3    | Exit                  |
---------------------------------
>> Enter your choice: 0

# THESE ARE THE AVAILABLE PRODUCTS IN THE STOCK -->
-------------------------------------------------------------------------------
ID     |Name               |Brand           |Qty        |Price       |Country
-------------------------------------------------------------------------------
|1     |Vitamin C Serum    |Garnier         |74         |400         |France
-------------------------------------------------------------------------------
|2     |Skin Cleanser      |Cetaphil        |24         |200         |Switzerland
-------------------------------------------------------------------------------
|3     |Sunscreen          |Aqualogica      |150        |300         |India
-------------------------------------------------------------------------------


---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action                |
|--------|-----------------------|
|   0    | Display product       |
|   1    | Sale (Customer Buy)   |
|   2    | Purchase (Restock)    |
|   3    | Exit                  |
---------------------------------
>> Enter your choice: 1
Enter customer name: Manash
Enter product ID to buy (0 to finish): 8
Invalid Product ID!
Enter product ID to buy (0 to finish):
```

Figure 11 screenshot of giving non existing productid as Input

Test 3 – To Show File generation of purchase of products (Purchasing multiple products)

Test 3	
Objective	To Show File generation of purchase of products (Purchasing multiple products)
Action	• Select purchase option • Enter valid customer name • Buy Product ID 1 (Qty: 33), Product ID 3 (Qty: 2) • Finish with 0 • Add shipping (optional) • Check shell output and invoice file
Expected Result	• Products purchased • Summary shown in shell • Invoice file created with all details
Actual Result	• Products purchased successfully • Output shown correctly • Invoice file generated with complete data
Conclusion	The test was successful.

Table 3: To Show File generation of purchase of products (Purchasing multiple product)

```
---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
>> Enter your choice: 0

# THESE ARE THE AVAILABLE PRODUCTS IN THE STOCK -->
-------------------------------------------------------------------------------------
ID     |Name               |Brand           |Qty         |Price       |Country
-------------------------------------------------------------------------------------
|1     |Vitamin C Serum    |Garnier         |49          |34          |France
-------------------------------------------------------------------------------------
|2     |Skin Cleanser      |Cetaphil        |29          |200         |Switzerland
-------------------------------------------------------------------------------------
|3     |Sunscreen          |Aqualogica      |147         |700         |India
-------------------------------------------------------------------------------------

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
>> Enter your choice: 2
Enter supplier name: sita
Enter VAT percentage: 13
Enter product ID to restock (0 to finish): 1
Enter restock quantity for Vitamin C Serum: 33
Enter new cost price for Vitamin C Serum: 400
Enter product ID to restock (0 to finish): 3
Enter restock quantity for Sunscreen: 2
Enter new cost price for Sunscreen: 200
Enter product ID to restock (0 to finish): 0

Restock Successful!
Total Restock Cost: Rs. 13600
VAT Amount (13.0%): Rs. 1768.0
Total Cost with VAT: Rs. 15368.0
```

```
==================================================
RESTOCK INVOICE
==================================================
Supplier Name: sita
Date: 2025-05-14 03:26:09.037840
--------------------------------------------------
Product: Vitamin C Serum
Brand: Garnier
Country: France
Quantity Restocked: 33
New Cost Price: Rs. 400
Item Total Cost: Rs. 13200
--------------------------------------------------
Product: Sunscreen
Brand: Aqualogica
Country: India
Quantity Restocked: 2
New Cost Price: Rs. 200
Item Total Cost: Rs. 400
--------------------------------------------------
Total Cost: Rs. 13600
VAT Amount (13.0%): Rs. 1768.0
Total Cost with VAT: Rs. 15368.0
==================================================
Restock invoice saved as: Restock_Invoices/restock_invoice_sita_2025-05-14_03-26-09.037840.txt
Stock updated and saved!

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
>> Enter your choice:
```

Figure 12 Screenshot of Complete process of Purchase of Multiple product with Purchase Invoice output in Shell

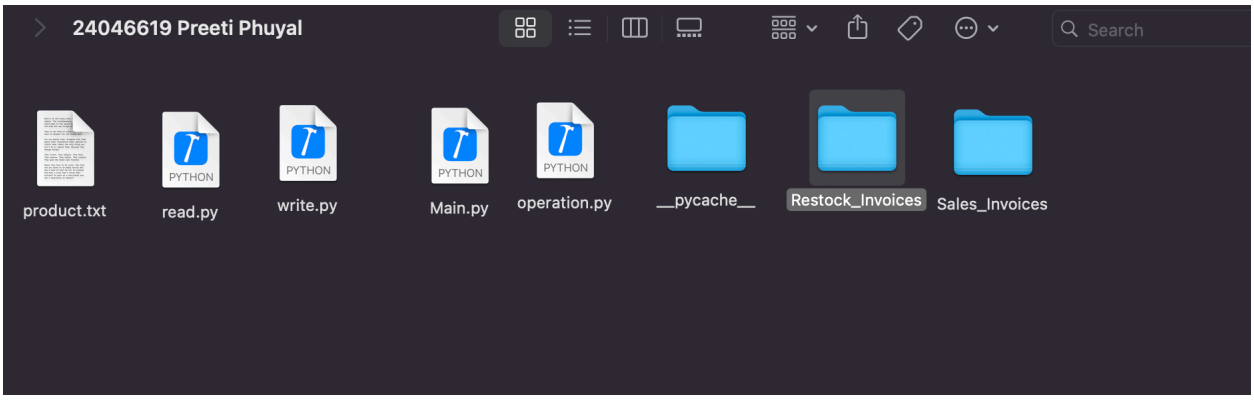


Figure 13 Screenshot of opening Purchase Invoice in txt File

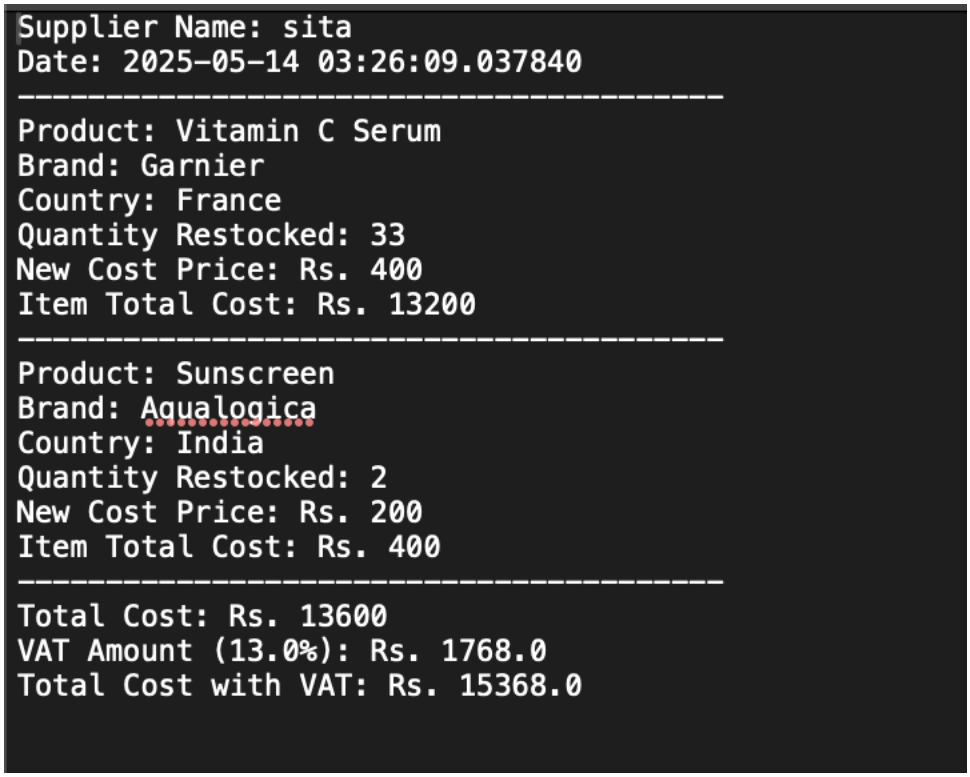


Figure 14 Screenshot of Purchase of multiple products Invoice in txt file

Test 4: File Generation of Sales Process (Selling Multiple Products)

Test 4	
Objective	To test the generation of an invoice file when multiple products are sold
Action	• Enter valid customer name • Select multiple existing product IDs • Provide valid positive quantities for each • Complete sale and choose to add shipping
Expected Result	• Products sold successfully • Shell displays total, VAT, and shipping • A .txt invoice is created in Sales_Invoices with product details
Actual Result	• All steps completed correctly • Output shown in terminal • Sales invoice file is generated with correct details
Conclusion	The test was successful

:

Table 4:File Generation of Sales Process (Selling Multiple Products)

```
---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action                |
|--------|-----------------------|
|   0    | Display product       |
|   1    | Sale (Customer Buy)   |
|   2    | Purchase (Restock)    |
|   3    | Exit                  |
---------------------------------
>> Enter your choice: 0

# THESE ARE THE AVAILABLE PRODUCTS IN THE STOCK -->
-------------------------------------------------------------------------------------
ID     |Name               |Brand            |Qty         |Price       |Country
-------------------------------------------------------------------------------------
|1     |Vitamin C Serum    |Garnier          |82          |400         |France
-------------------------------------------------------------------------------------
|2     |Skin Cleanser      |Cetaphil         |29          |200         |Switzerland
-------------------------------------------------------------------------------------
|3     |Sunscreen          |Aqualogica       |149         |200         |India
-------------------------------------------------------------------------------------

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action                |
|--------|-----------------------|
|   0    | Display product       |
|   1    | Sale (Customer Buy)   |
|   2    | Purchase (Restock)    |
|   3    | Exit                  |
---------------------------------
>> Enter your choice: 1
Enter customer name: sheela
Enter product ID to buy (0 to finish): 2
Enter quantity to buy for Skin Cleanser: 4
Enter product ID to buy (0 to finish): 3
Enter quantity to buy for Sunscreen: 2
Enter product ID to buy (0 to finish): 1
Enter quantity to buy for Vitamin C Serum: 6
Enter product ID to buy (0 to finish): 0

Transaction Successful!
Total Price: Rs. 7200
Do you want to add shipping charge? (yes/no): yes
Enter shipping charge: 100
```

```
=== INVOICE PREVIEW ===
+------------------------------------------------------------+
|                      SALES INVOICE                         |
+------------------------------------------------------------+
| Customer Name: sheela
| Date: 2025-05-14 03:52:31.095366
+------------------------------------------------------------+
+-----------------+---------------+-----+------+------------+------------+
| Product         | Brand         | Qty | Free | Unit Price | Amount     |
+-----------------+---------------+-----+------+------------+------------+
| Skin Cleanser   | Cetaphil      |   4 |    1 |        400 |       1600 |
| Sunscreen       | Aqualogica    |   2 |    0 |        400 |        800 |
| Vitamin C Serum | Garnier       |   6 |    2 |        800 |       4800 |
+-----------------+---------------+-----+------+------------+------------+
| Subtotal      : Rs.       7200 |
| VAT (13%)     : Rs.      936.0 |
| Shipping      : Rs.      100.0 |
+------------------------------------------------------------+
| GRAND TOTAL   : Rs.     8336.0 |
+------------------------------------------------------------+

Sale invoice created successfully!
Invoice saved as: Sales_Invoices/sale_invoice_sheela_2025-05-14 03:52:31.095366.txt
Total amount: Rs. 8336.0
Stock updated and saved!

---------------------------------
```

Figure 15 Screenshot of Complete process of Sales of Multiple product with Sales Invoice output in Shell

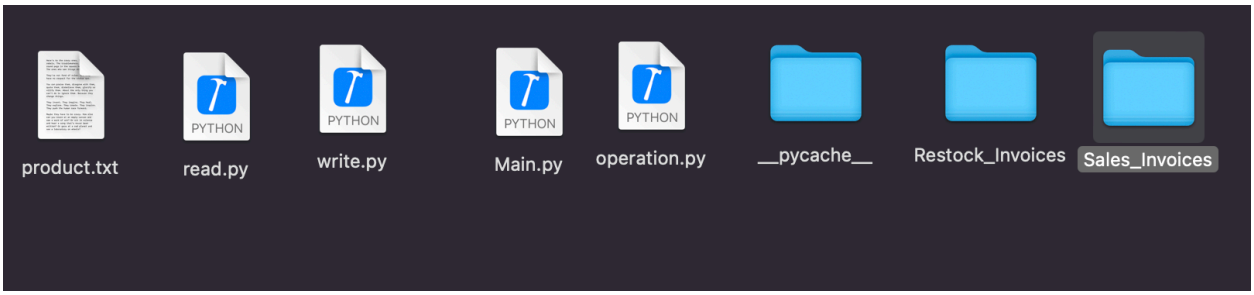
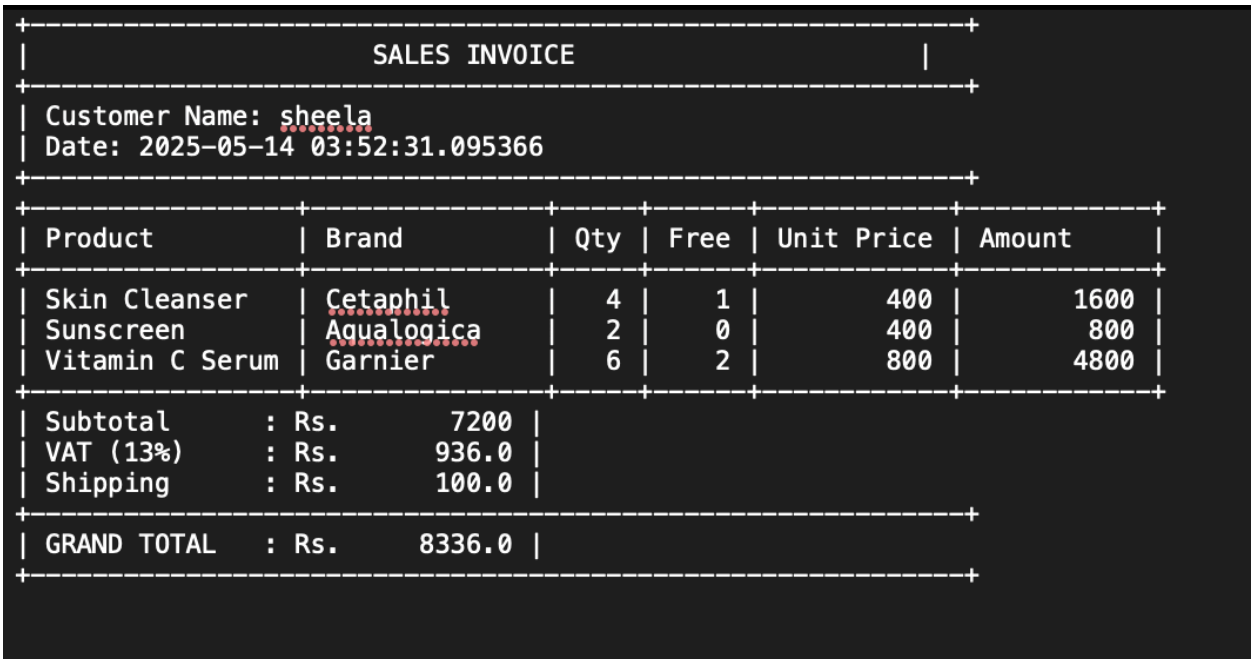


Figure 16 Screenshot of opening Sales Invoice in txt File



```
+-------------------------------------------------------------+
|                       SALES INVOICE                         |
+-------------------------------------------------------------+
| Customer Name: sheela
| Date: 2025-05-14 03:52:31.095366
+-------------------------------------------------------------+
+-----------------+---------------+-----+------+------------+------------+
| Product         | Brand         | Qty | Free | Unit Price | Amount     |
+-----------------+---------------+-----+------+------------+------------+
| Skin Cleanser   | Cetaphil      |   4 |    1 |        400 |       1600 |
| Sunscreen       | Aqualogica    |   2 |    0 |        400 |        800 |
| Vitamin C Serum | Garnier       |   6 |    2 |        800 |       4800 |
+-----------------+---------------+-----+------+------------+------------+
| Subtotal      : Rs.       7200 |
| VAT (13%)     : Rs.      936.0 |
| Shipping      : Rs.      100.0 |
+-------------------------------------------------------------+
| GRAND TOTAL   : Rs.     8336.0 |
+-------------------------------------------------------------+
```

Figure 17 Screenshot of Sales of multiple products Invoice in txt file

Test 5: To Show the quantity being added while purchasing the products.

Test 5	
Objective	To verify that the stock is updated correctly after purchasing products, and that the update is reflected in the .txt file.
Action	• Select a product to purchase. • Enter the quantity to purchase. • Confirm the purchase is processed and stock is reduced. • Check the products.txt file to confirm the updated stock is saved correctly.
Expected Result	• After purchasing the product, the stock quantity for Product ID 2 should decrease by 3 units. • The new stock level should be written and saved in the products.txt file.
Actual Result	• The stock quantity for Product ID P101 decreased correctly in the program. • The updated stock was reflected accurately in the products.txt file.

Conclusion	The test was successful.

Table 5: To Show the quantity being added while purchasing the products.

.

```
---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
>> Enter your choice: 0

# THESE ARE THE AVAILABLE PRODUCTS IN THE STOCK -->
-------------------------------------------------------------------------------------
ID     |Name               |Brand           |Qty        |Price       |Country
-------------------------------------------------------------------------------------
|1     |Vitamin C Serum    |Garnier         |74         |400         |France
-------------------------------------------------------------------------------------
|2     |Skin Cleanser      |Cetaphil        |27         |300         |Switzerland
-------------------------------------------------------------------------------------
|3     |Sunscreen          |Aqualogica      |150        |300         |India
-------------------------------------------------------------------------------------

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
>> Enter your choice: 2
Enter supplier name: riya
Enter VAT percentage: 13
Enter product ID to restock (0 to finish): 2
Enter restock quantity for Skin Cleanser: 3
Enter new cost price for Skin Cleanser: 200
Enter product ID to restock (0 to finish): 0

Restock Successful!
Total Restock Cost: Rs. 600
VAT Amount (13.0%): Rs. 78.0
Total Cost with VAT: Rs. 678.0

==================================================
RESTOCK INVOICE
==================================================
Supplier Name: riya
Date: 2025-05-14 04:30:42.390645
--------------------------------------------------
Product: Skin Cleanser
Brand: Cetaphil
Country: Switzerland
Quantity Restocked: 3
New Cost Price: Rs. 200
Item Total Cost: Rs. 600
--------------------------------------------------
Total Cost: Rs. 600
```

```
Total Cost: Rs. 600
VAT Amount (13.0%): Rs. 78.0
Total Cost with VAT: Rs. 678.0
================================================
Restock invoice saved as: Restock_Invoices/restock_invoice_riya_2025-05-14_04-30-42.390645.txt
Stock updated and saved!

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action                |
|--------|-----------------------|
|   0    | Display product       |
|   1    | Sale (Customer Buy)   |
|   2    | Purchase (Restock)    |
|   3    | Exit                  |
---------------------------------
>> Enter your choice:
```

Figure 18: Screenshot of giving details to Purchase skin cleanser

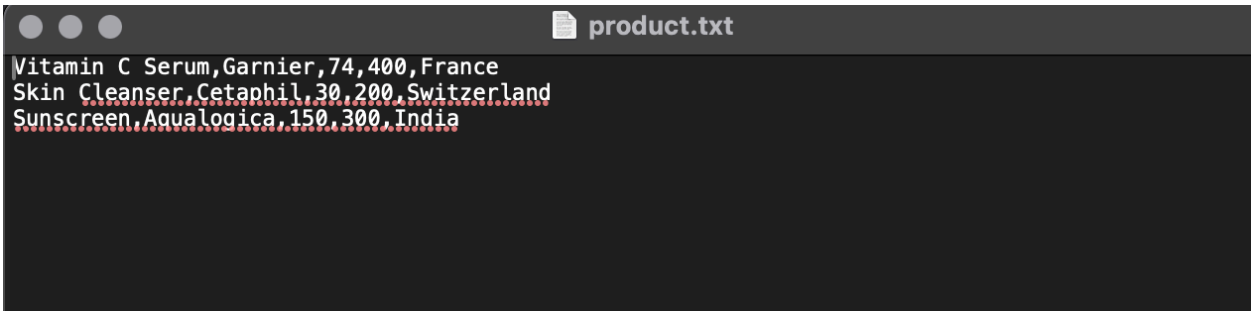


Figure 19: Screenshot of decrement in Quantity of skin cleanser after Purchase in the txt file.

Test 6: Show the quantity being added while selling the product (Update should be reflected in a .txt file as well)

Objective	To verify that the stock is correctly increased after restocking a product and the update is reflected in the products.txt file.
Action	Select a product to restock. • Enter the quantity to add. • Confirm the restocking process is completed. • Check the products.txt file to ensure the updated stock is saved.
Excepted Result	• After restocking the product, the stock quantity for Product ID 2 should increase by 5 units. • The new stock level should be written and saved in the products.txt file.
Actual Result	The terminal output confirmed that the stock of Product ID 2 was increased by 5 units. • The products.txt file reflects the updated stock correctly.
Conclusion	The test was successful.

Table 6: Show the quantity being added while selling the product (Update should be reflected in a .txt file as well)

```
- RESTART: /Users/alfm2/Desktop/24040019 Preeti Phuyal 2/24040019 Preeti Phuyal/main.py

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
>> Enter your choice: 0

# THESE ARE THE AVAILABLE PRODUCTS IN THE STOCK -->
-------------------------------------------------------------------------------
ID     |Name               |Brand           |Qty        |Price       |Country
-------------------------------------------------------------------------------
|1     |Vitamin C Serum    |Garnier         |68         |400         |France
-------------------------------------------------------------------------------
|2     |Skin Cleanser      |Cetaphil        |24         |200         |Switzerland
-------------------------------------------------------------------------------
|3     |Sunscreen          |Aqualogica      |150        |300         |India
-------------------------------------------------------------------------------

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
>> Enter your choice: 2
Enter supplier name: heena
Enter VAT percentage: 13
Enter product ID to restock (0 to finish): 1
Enter restock quantity for Vitamin C Serum: 5
Enter new cost price for Vitamin C Serum: 300
Enter product ID to restock (0 to finish): 0

Restock Successful!
Total Restock Cost: Rs. 1500
VAT Amount (13.0%): Rs. 195.0
Total Cost with VAT: Rs. 1695.0
```

```
==================================================
RESTOCK INVOICE
==================================================
Supplier Name: heena
Date: 2025-05-14 06:09:20.333860
--------------------------------------------------
Product: Vitamin C Serum
Brand: Garnier
Country: France
Quantity Restocked: 5
New Cost Price: Rs. 300
Item Total Cost: Rs. 1500
--------------------------------------------------
Total Cost: Rs. 1500
VAT Amount (13.0%): Rs. 195.0
Total Cost with VAT: Rs. 1695.0
==================================================
Restock invoice saved as: Restock_Invoices/restock_invoice_heena_2025-05-14_06-09-20.333860.txt
Stock updated and saved!

---------------------------------
# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->
---------------------------------
| Option | Action               |
|--------|----------------------|
|   0    | Display product      |
|   1    | Sale (Customer Buy)  |
|   2    | Purchase (Restock)   |
|   3    | Exit                 |
---------------------------------
```

Figure 20:Screenshot of giving details to sale skin cleanser

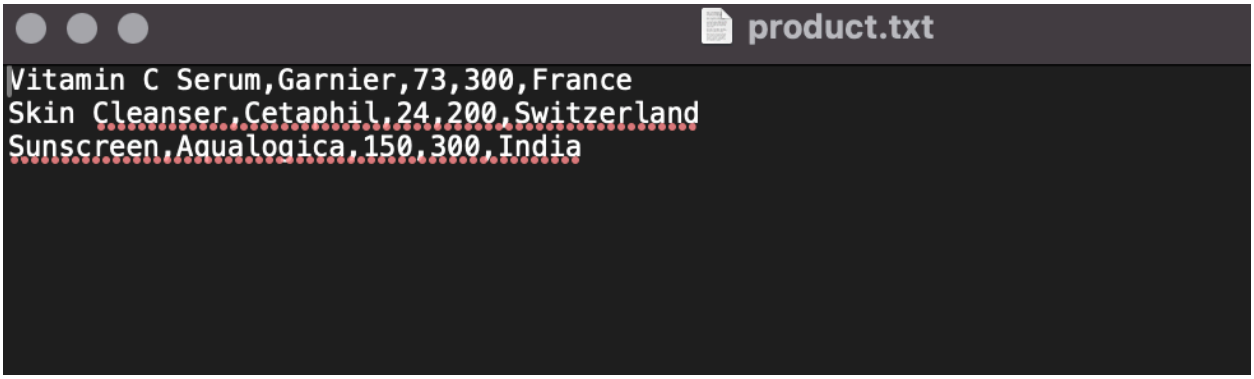


Figure 21: Screenshot of increment in Quantity of skin cleanser after sale in the

8 Conclusion

The WeCare system for inventory management met all functional requirements of the project. It allows the user to see the products available with the correct selling prices set as a 200% markup on the cost price. It functions well for sales and restocks with multiple products in one transaction. The sales process handles quantity checks, the "buy 3, get 1 free" offer, VAT, and optional shipping correctly. Each transaction has a lovely invoice file created when the user saves the transaction, with an invoice file created based on customer/supplier name and a given timestamp, which satisfies the documentation project requirement. Each transaction's processing and saving to the products.txt file accurately updates the stock levels. The system also performs input validations to prevent errors like entering invalid product IDs or a negative quantity. I have modularized the entire code into four different Python files: read.py, write.py, operation.py, and main.py. The formatting and coding conventions strictly followed the coursework constraints (for example, I never used f-strings, format(), or strip() anywhere on any of my files). Overall, I am satisfied with this project, and its functionality has been thoroughly tested and works successfully both through the terminal outputs and along with the files created and updated through them, which would achieve what was required by the brief.

9 Appendix

```
# main.py
from read import read_products  # Import function to read product data
from write import save_products  # Import function to save updated product data
from write import generate_invoice  # Import the generate_invoice function from write.py

from operation import display_products, handle_sale, handle_restock  # Import
operation functions
import datetime  # Import datetime for timestamping invoices

# Load product data from the file
product = read_products()

main_loop = True  # Main loop for program operation

# Program loop to continuously ask for user input
while main_loop:
    print("\n" + "-" * 33)
    print("# CHOOSE THE OPTION YOU WOULD LIKE TO PERFORM -->")
    print("-" * 33)
    print("| Option | Action              |")
    print("|--------|---------------------|")
    print("|   0    | Display product     |")
    print("|   1    | Sale (Customer Buy) |")
    print("|   2    | Purchase (Restock)  |")
    print("|   3    | Exit                |")
    print("-" * 33)

    try:
        choice = int(input(">> Enter your choice: "))  # Get user input for the action
```

```
        if choice == 0:
            print("\n# THESE ARE THE AVAILABLE PRODUCTS IN THE STOCK -->")
            display_products(product)  # Display the available products
        elif choice == 1:
            flag = handle_sale(product)
            if flag == "error":
                continue
            else:
                save_products(product)  # Save updated stock
        elif choice == 2:
            handle_restock(product)  # Handle product restocking
            save_products(product)  # Save updated stock



        elif choice == 3:
            print("Thank you for using our system!")  # Exit message
            main_loop = False  # End the main loop
        else:
            print("Invalid input. Please select from 0 to 3.")  # Handle invalid input
    except ValueError:
        print("Invalid input! Please enter a number.")  # Handle invalid input

# Save product data back to the file after any operation
save_products(product)



# read.py

# Function to read products from the file and return them as a dictionary
def read_products():
```

```
    product = {}  # Initialize an empty dictionary to store products
    with open("product.txt", "r") as file:  # Open the product file in read mode
        data = file.readlines()  # Read all lines from the file

    pid = 1  # Starting product ID for assigning to products
    for line in data:
        line = line.replace("\n", "").split(",")  # Split each line by commas
        line[2] = int(line[2])  # Quantity (convert to integer)
        line[3] = int(line[3])  # Price (convert to integer)
        product[pid] = line  # Add the product to the dictionary with a unique ID
        pid = pid + 1  # Increment product ID for the next product

    return product  # Return the dictionary of products




import datetime  # Importing datetime module for timestamping invoices

# Function to save updated product data to the file
def save_products(product):
    with open("product.txt", "w") as file:  # Open product file in write mode
        for pid in product:  # Iterate over the product dictionary
            line = product[pid][0] + "," + product[pid][1] + "," + str(product[pid][2]) + "," +
str(product[pid][3]) + "," + product[pid][4] + "\n"
            # Prepare each product's data in comma-separated format
            file.write(line)  # Write each product's data to the file
    print("Stock updated and saved!")  # Notify the user that data was saved

# Function to generate a sales invoice after a transaction
```

```
def generate_invoice(customer_name, timestamp, sold_items, total_price,
vat_percentage, shipping_fee, total_with_vat):
    invoice_name = "Sales_Invoices/sale_invoice_" + customer_name + "_" +
str(timestamp) + ".txt"

    # Create a list to store invoice lines for both file and terminal output
    invoice_lines = []

    # Header section
    header1 = "+" + "-" * 60 + "+"
    header2 = "|" + " " * 22 + "SALES INVOICE" + " " * 22 + "|"
    header3 = "+" + "-" * 60 + "+"
    header4 = "| Customer Name: " + customer_name
    header5 = "| Date: " + str(timestamp)

    invoice_lines.append(header1)
    invoice_lines.append(header2)
    invoice_lines.append(header3)
    invoice_lines.append(header4)
    invoice_lines.append(header5)
    invoice_lines.append(header3)

    # Column headings - fixed width formatting
    table_header1 = "+-----------------+---------------+-----+------+------------+------------+"
    table_header2 = "| Product         | Brand         | Qty | Free | Unit Price | Amount     |"

    invoice_lines.append(table_header1)
    invoice_lines.append(table_header2)
    invoice_lines.append(table_header1)

    # Process each sold item's detail
```

```
    for item in sold_items:
        product = item[0]
        brand = item[1]
        qty = str(item[3])
        free = str(item[4])
        unit_price = str(int(item[6] / item[3]))  # derived from total amount / quantity
        amount = str(item[6])

        # Format each column with proper spacing manually
        product_col = (product + " " * 15)[:15]
        brand_col = (brand + " " * 13)[:13]
        qty_col = " " * (3 - len(qty)) + qty
        free_col = " " * (4 - len(free)) + free
        price_col = " " * (10 - len(unit_price)) + unit_price
        amount_col = " " * (10 - len(amount)) + amount

        # Build the line with proper spacing
        line = "| " + product_col + " | " + brand_col + " | " + qty_col + " | " + free_col + " | " +
price_col + " | " + amount_col + " |"
        invoice_lines.append(line)

    # Footer line
    invoice_lines.append(table_header1)

    # Totals section
    vat_amount = (total_price * vat_percentage) / 100
    grand_total = total_with_vat + shipping_fee

    subtotal_str = str(total_price)
    vat_str = str(vat_amount)
    shipping_str = str(shipping_fee)
```

```
    grand_total_str = str(grand_total)

    subtotal_line = "| Subtotal      : Rs. " + " " * (10 - len(subtotal_str)) + subtotal_str + " |"
    vat_line = "| VAT (" + str(vat_percentage) + "%)     : Rs. " + " " * (10 - len(vat_str)) +
vat_str + " |"
    shipping_line = "| Shipping      : Rs. " + " " * (10 - len(shipping_str)) + shipping_str + "
|"
    total_line = "| GRAND TOTAL   : Rs. " + " " * (10 - len(grand_total_str)) +
grand_total_str + " |"

    invoice_lines.append(subtotal_line)
    invoice_lines.append(vat_line)
    invoice_lines.append(shipping_line)
    invoice_lines.append(header3)
    invoice_lines.append(total_line)
    invoice_lines.append(header3)

    # Print invoice to terminal
    print("\n=== INVOICE PREVIEW ===")
    for line in invoice_lines:
        print(line)

    # Write invoice to file
    with open(invoice_name, "w") as invoice_file:
        for line in invoice_lines:
            invoice_file.write(line + "\n")

    print("\nSale invoice created successfully!")
    print("Invoice saved as: " + invoice_name)
    print("Total amount: Rs. " + str(grand_total))
```

```
# operation.py
import datetime  # Importing datetime for timestamping
from write import generate_invoice

# Function to display all products with details
def display_products(product):
    print("-" * 80)
    print("ID" + " " * 5 + "|" + "Name" + " " * 15 + "|" + "Brand" + " " * 11 + "|" + "Qty" + " " *
8 + "|" + "Price" + " " * 7 + "|" + "Country")
    for i in product.items():
        print("-" * 80)
        pid = str(i[0])  # Product ID
        name = i[1][0]  # Product Name
        brand = i[1][1]  # Product Brand
        qty = str(i[1][2])  # Product Quantity
        price = str(i[1][3])  # Product Price
        country = i[1][4]  # Product Country

        # Print each product's details in a table format
        print("|" + pid + " " * (5 - len(pid)),
              "|" + name + " " * (18 - len(name)),
              "|" + brand + " " * (15 - len(brand)),
              "|" + qty + " " * (10 - len(qty)),
              "|" + price + " " * (11 - len(price)),
              "|" + country)
    print("-" * 80)

# Function to handle sale operations (customer buying products)
def handle_sale(product):
    try:
        customer_name = input("Enter customer name: ")
```

```
    if not isinstance(customer_name, str) or customer_name.strip().isdigit():
        print("Invalid customer name. A name cannot be a number. Please enter a valid
name.")
        return "error"
    timestamp = datetime.datetime.now()
    total_price = 0  # Initialize total price of sale
    sold_items = []  # List to store sold items for the invoice
    vat_percentage = 13  # Example VAT percentage

    while True:
        product_id = int(input("Enter product ID to buy (0 to finish): "))  # Get product ID
        if product_id == 0:  # If user enters 0, end the sale
            break

        if product_id not in product:  # If product ID is invalid
            print("Invalid Product ID!")
            continue

        name, brand, stock, cost_price, country = product[product_id]  # Get product
details
        quantity = int(input("Enter quantity to buy for " + name + ": "))  # Get quantity to
buy




        free_items = quantity // 3  # Calculate free items based on the "Buy 3 Get 1
Free" rule
        total_items = quantity + free_items  # Total items being purchased (including
free items)

        if stock < total_items:  # Check if enough stock is available
```

```
            print("Not enough stock!")
            continue

        selling_price = cost_price * 2  # Selling price is double the cost price
        price_for_product = quantity * selling_price  # Calculate price for the sold
quantity
        total_price += price_for_product  # Add to total sale price
        product[product_id][2] -= total_items  # Update stock after sale

        # Add sold item to sold items list for invoice generation
        sold_items.append([name, brand, country, quantity, free_items, total_items,
price_for_product])

    if len(sold_items) == 0:  # If no items were sold, notify the user
        print("No valid items were purchased.")
        return

    print("\nTransaction Successful!")
    print("Total Price: Rs. " + str(total_price))

    # Calculate VAT and total with VAT
    vat_amount = (total_price * vat_percentage) / 100
    total_with_vat = total_price + vat_amount

    # Ask if the user wants to add shipping charge
    shipping_charge = input("Do you want to add shipping charge? (yes/no): ").lower()
    if shipping_charge == "yes":
        shipping_fee = float(input("Enter shipping charge: "))
    else:
        shipping_fee = 0.0
```

```
        final_total = total_with_vat + shipping_fee  # Calculate the final total

        # Generate the invoice timestamp and filename
        time = datetime.datetime.now()



        # Call function to generate the invoice
        generate_invoice(customer_name, timestamp, sold_items, total_price,
vat_percentage, shipping_fee, final_total)

    except ValueError:
        print("Invalid input! Please enter valid numbers.")  # Handle invalid inputs

# Function to handle product restocking operations
def handle_restock(product):
    try:
        supplier_name = input("Enter supplier name: ")  # Get supplier name
        total_cost = 0  # Initialize total cost of restocking
        restock_items = []  # List to store restocked items

        # Get VAT percentage
        vat_percentage = float(input("Enter VAT percentage: "))

        while True:
            product_id = int(input("Enter product ID to restock (0 to finish): "))  # Get product
ID
            if product_id == 0:  # If user enters 0, end the restocking
                break

            if product_id not in product:  # If product ID is invalid
                print("Invalid Product ID!")
```

```
            continue

        name, brand, stock, cost_price, country = product[product_id]  # Get product details
        restock_quantity = int(input("Enter restock quantity for " + name + ": "))  # Get restock quantity
        new_cost_price = int(input("Enter new cost price for " + name + ": "))  # Get new cost price

        product[product_id][2] += restock_quantity  # Update the stock quantity
        product[product_id][3] = new_cost_price  # Update the cost price

        item_cost = restock_quantity * new_cost_price  # Calculate total cost for the restocked items
        total_cost += item_cost  # Add to the total restock cost
        restock_items.append([name, brand, country, restock_quantity, new_cost_price, item_cost])

    if len(restock_items) == 0:  # If no items were restocked, notify the user
        print("No products restocked.")
        return

    # Calculate VAT and total with VAT
    vat_amount = (total_cost * vat_percentage) / 100
    total_with_vat = total_cost + vat_amount

    print("\nRestock Successful!")
    print("Total Restock Cost: Rs. " + str(total_cost))
    print("VAT Amount (" + str(vat_percentage) + "%): Rs. " + str(vat_amount))
    print("Total Cost with VAT: Rs. " + str(total_with_vat))
```

```
        # Generate restock invoice timestamp and filename
        time = datetime.datetime.now()



        # Create restock invoice and write it to a file
        restock_invoice = "Restock_Invoices/restock_invoice_" + supplier_name + "_" +
str(time).replace(":", "-").replace(" ", "_") + ".txt"
        with open(restock_invoice, "w") as restock_file:
            # Write invoice header to file
            restock_file.write("Supplier Name: " + supplier_name + "\n")
            restock_file.write("Date: " + str(time) + "\n")
            restock_file.write("-" * 40 + "\n")

            # Print invoice header to terminal
            print("\n" + "=" * 50)
            print("RESTOCK INVOICE")
            print("=" * 50)
            print("Supplier Name: " + supplier_name)
            print("Date: " + str(time))
            print("-" * 50)

            for item in restock_items:
                # Write item details to file
                restock_file.write("Product: " + item[0] + "\n")
                restock_file.write("Brand: " + item[1] + "\n")
                restock_file.write("Country: " + item[2] + "\n")
                restock_file.write("Quantity Restocked: " + str(item[3]) + "\n")
                restock_file.write("New Cost Price: Rs. " + str(item[4]) + "\n")
                restock_file.write("Item Total Cost: Rs. " + str(item[5]) + "\n")
                restock_file.write("-" * 40 + "\n")
```

```
            # Print item details to terminal
            print("Product: " + item[0])
            print("Brand: " + item[1])
            print("Country: " + item[2])
            print("Quantity Restocked: " + str(item[3]))
            print("New Cost Price: Rs. " + str(item[4]))
            print("Item Total Cost: Rs. " + str(item[5]))
            print("-" * 50)

        # Write summary to file
        restock_file.write("Total Cost: Rs. " + str(total_cost) + "\n")
        restock_file.write("VAT Amount (" + str(vat_percentage) + "%): Rs. " +
str(vat_amount) + "\n")
        restock_file.write("Total Cost with VAT: Rs. " + str(total_with_vat) + "\n")

        # Print summary to terminal
        print("Total Cost: Rs. " + str(total_cost))
        print("VAT Amount (" + str(vat_percentage) + "%): Rs. " + str(vat_amount))
        print("Total Cost with VAT: Rs. " + str(total_with_vat))
        print("=" * 50)

    print("Restock invoice saved as: " + restock_invoice)  # Notify user of the restock
invoice creation

except ValueError:
    print("Invalid input! Please enter valid numbers.")  # Handle invalid inputs
```