CS4051NI/CC4059NI Fundamentals of Computing

70% Individual Coursework

FINAL SUBMISSION

2024/25 Spring

Student Name: YathechyaShrestha

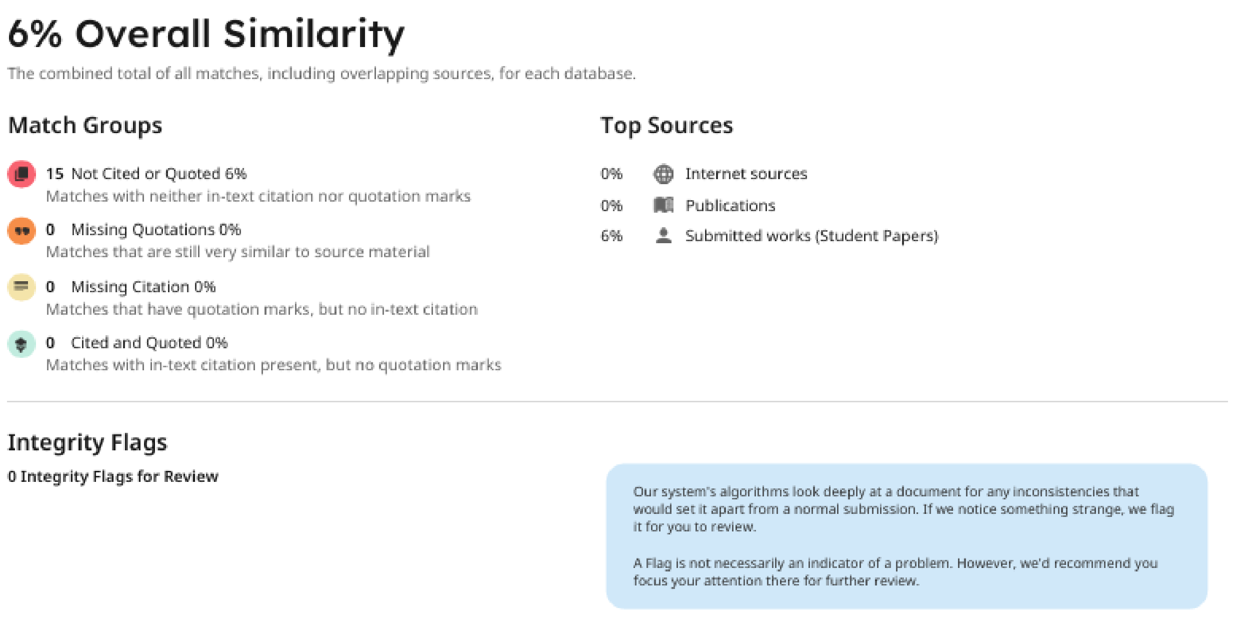
London Met ID: 24046938

College ID: np01cp4a240061

Assignment Due Date: Wednesday, May 14, 2025

Assignment Submission Date: Wednesday, May 14, 2025

1 Table of Contents



6% Overall Similarity

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

Match Groups

15 Not Cited or Quoted 6%
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

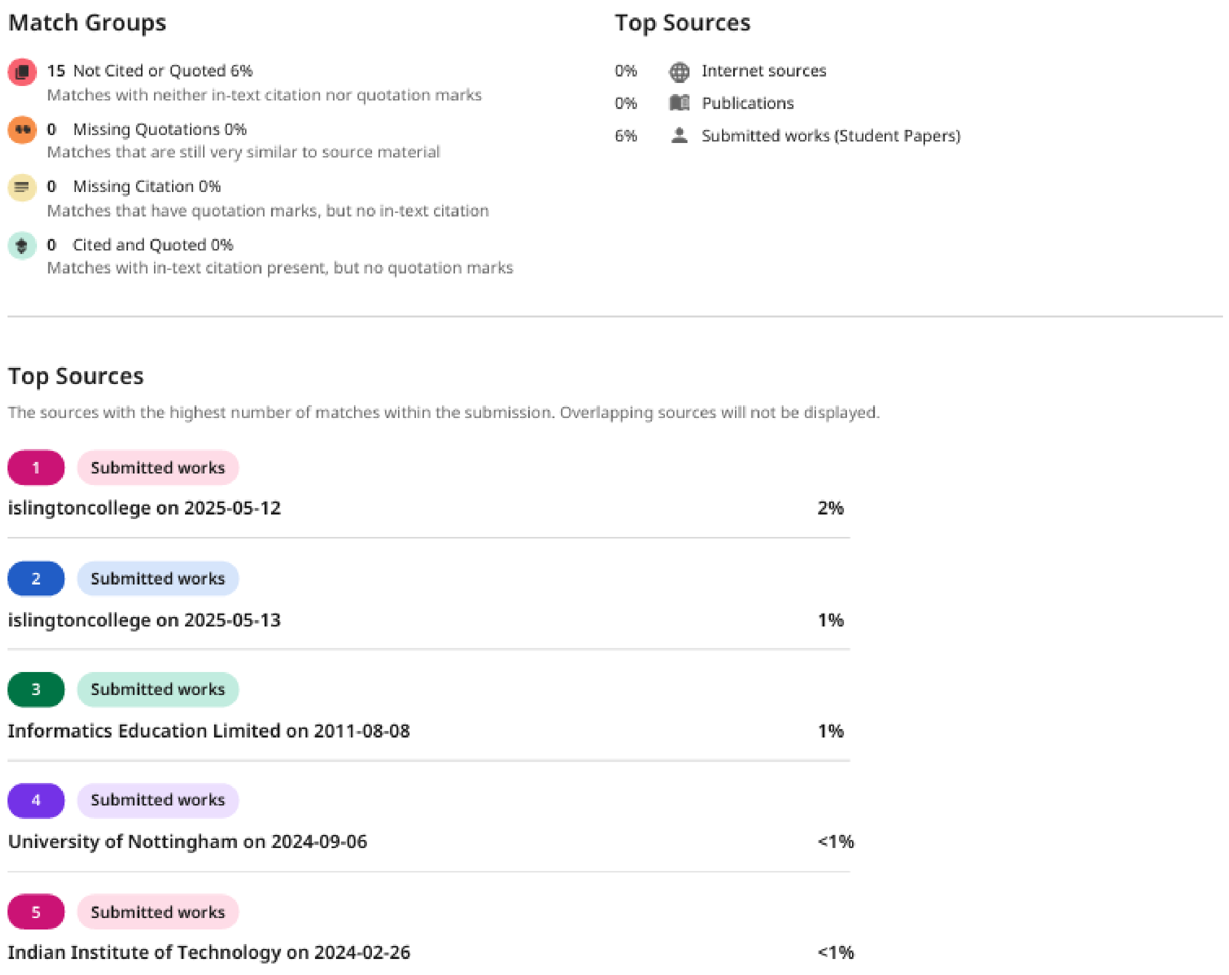
0%	Internet sources
0%	Publications
6%	Submitted works (Student Papers)

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

15 Not Cited or Quoted 6%
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

0%	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	
	islingtoncollege on 2025-05-12	2%
2	Submitted works	
	islingtoncollege on 2025-05-13	1%
3	Submitted works	
	Informatics Education Limited on 2011-08-08	1%
4	Submitted works	
	University of Nottingham on 2024-09-06	<1%
5	Submitted works	
	Indian Institute of Technology on 2024-02-26	<1%

Introduction:

Project Overview:

This project concerned the inventory management system for WeCare a beauty product shop WeCare. The proposed product management system would read product data (name, brand, quantity, cost, origin from products.txt, store it in python collections, and display it formatted in a table, manages inventory, processes sales and restocking transactions, and generates invoices.

The selling price is set at a 200% markup from the cost price. The program has error handling implemented to prevent the program from crashing due to invalid inputs

1.1 Goals and Objective:

the primary goal of this project is to create an inventory management system that automates product transactions and invoice generation for We Care store. The objectives include:

- Reading and displaying product details from a text file.
- Facilitating sales and restocking with real-time stock updates.
- Generating invoices for each transaction.
- Implementing error handling to prevent crashes due to invalid inputs.
- Ensuring modularity through structured file system (Read, Write, Operations, and Main).
- Display products with prices multiplied by two (cost x 2)

1.2 Tools and Technologies Used

The project was developed using the following tools and technologies:

- **Python 3:** The programming language used for its simplicity, readability, and robust library, which supports file handling and data processing. Python's major popularity is largely due to it being a high-level, interpreted, and general-purpose language that is simple, readable, and versatile. First made available in 1991 by Guido van Rossum, Python was created with a clear focus on making code easy to read and developers more productive. The syntax of Python is straightforward and easy as code blocks are indicated by indentation rather than braces, making it simpler to write and maintain than a lot of other languages. Python is dynamically typed and therefore does not require explicit type declarations, which supports more flexible and faster programming. Being an interpreted language, Python executes code line-by-line with the interpreter, so developers can quickly prototype and test without any need for a separate compilation step.

its one of Python's key benefits that its standard library includes many built-in modules for tasks such as file processing, mathematical operations, date and time manipulation, and networking. Besides this, Python supports not only procedural but also object-oriented and functional programming, so it can be used in a variety of coding styles for different projects.

- **Python IDLE**: The Integrated Development and Learning Environment used for writing, testing, and debugging the code. IDLE provides a lightweight interface with features like syntax highlighting and an interactive shell.The IDLE also provides elementary debugging tools which include stepping over code, use of breakpoints which helps users to find their errors and rectify them in an easy manner. Although, it does not have the advanced features that come with the more robust IDE's such as PyCharm or VS Code, IDLE's simplicity and ease of use make it a prudent option for quick scripting, education or small projects like the WeCare Cosmetics system.

- **Text Editor (Notepad):** Used to create and edit the products.txt file containing product details.

- **Windows Operating System:** Windows is the most recognized operating system in the world. Allowing the user do their required tasks as per their needs. The development and testing environment for this program to make sure its compatibility with standard file systems for invoice generation.

- **Microsoft Word**: Ms Word is one of the most frequently applied word processing program. It is used to create documents, letters, reports, and other It also allows one to edit and modify newly created or already existing documents. The Word document is saved with a .docx extension. It’s used here to create and compile the report providing evidence of its implementation and error handling

2 Analysis

2.1 Algorithm

The algorithm outlines the high-level steps for the Skin Care Product Sale System:

1. Initialize the system by loading product data from a text file into a collection.
2. Display a menu with options: display products, process sale, restock products, or exit.
3. Based on user input:

   3.2) If "display products," show all product details in a formatted manner.

   3.3) If "process sale," prompt for customer name and product IDs, validate inputs, apply the "buy three get one free" policy, update stock, and generate a sale invoice.

   3.4) If "restock products," prompt for supplier name and product IDs, validate inputs, update stock, and generate a restock invoice.

   3.5) If "exit," save the updated product data to the text file and terminate.
4. Handle invalid inputs gracefully, prompting the user to retry.
5. Repeat until the user chooses to exit.

2.2 FLOWCHART

The flowchart represents the algorithm graphically, starting by loading the product data, presenting the main menu, Decision nodes control user decisions by guiding the path to show, goods, process sales, replace them, or get out. Loops make it so the menu exists until exit, and error, handling paths manage invalid inputs.

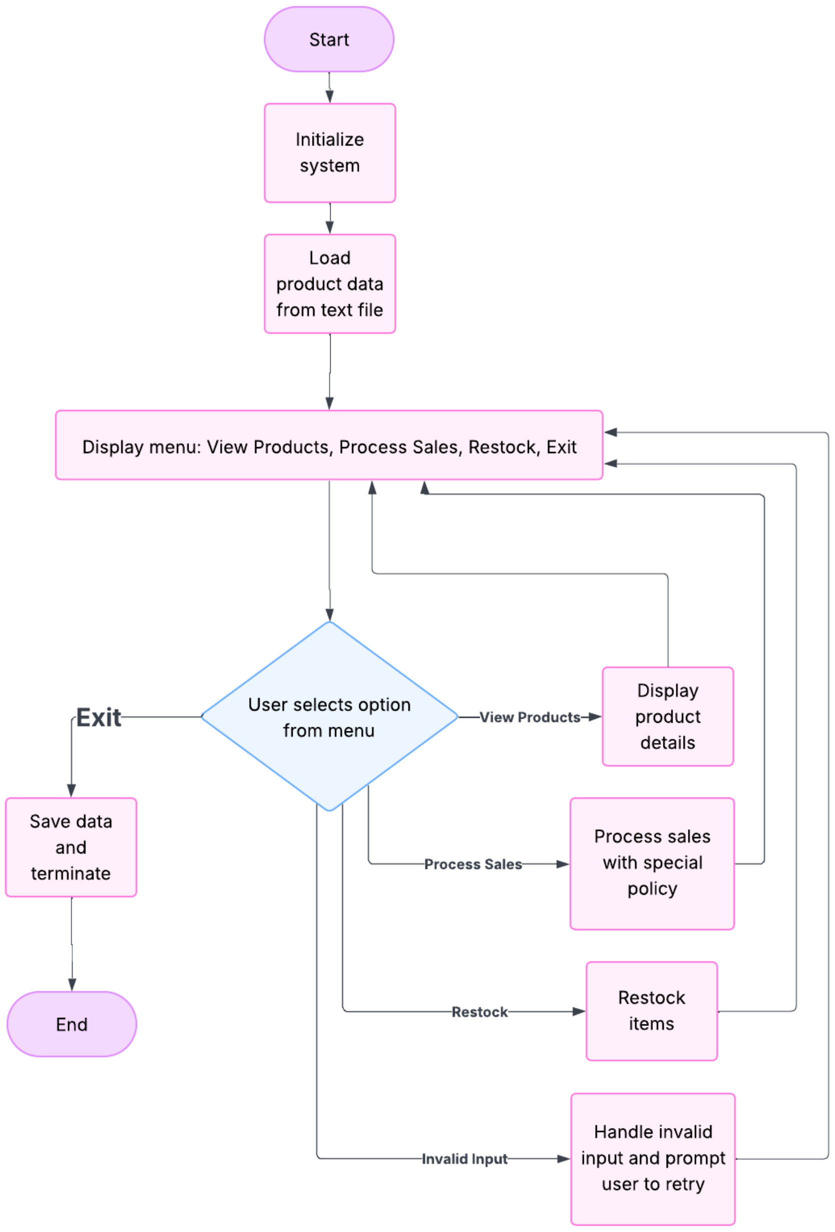

Start
Initialize system
Load product data from text file
Display menu: View Products, Process Sales, Restock, Exit
User selects option from menu
Exit
View Products
Display product details
Process Sales
Process sales with special policy
Restock
Restock items
Invalid Input
Handle invalid input and prompt user to retry
Save data and terminate
End

Figure 1:Flow chart

2.3 PSEUDOCODE:

The pseudocode is separated by module which is a reflection of the modular structure of the code:

```
READ.py
FUNCTION load_products(filename)
    product_list = empty list
    OPEN file for reading
    FOR each line in file
        SPLIT line into parts by comma
        IF parts length < 5 THEN
            SKIP line
        ENDIF
    TRIM spaces from parts
    CREATE dictionary with id, name, brand, quantity, cost, origin
    APPEND dictionary to product_list
CLOSE file
RETURN product_list
```

```
write.py:
FUNCTION display_products(products)
    PRINT header
    FOR each product in products
        FORMAT id, name, brand, quantity, price (cost * 2)
        PRINT formatted product details
    PRINT footer
END FUNCTION
```

operations.py:

```
FUNCTION process_sale(products)
       CALL display_products(products)
       GET customer_name
       cart = empty list
       total_amount = 0
       WHILE true
        GET product_id
        FOR product in products
        IF product_id matches product
                TRY
                        GET quantity
                CATCH ValueError
                        PRINT error
                BREAK
        END TRY
                IF quantity > stock THEN
                        PRINT error
                BREAK
       ENDIF
       CALCULATE free items (quantity / 3)
       CALCULATE total items and price
       ADD item to cart
       UPDATE stock
       INCREMENT total_amount
       ENDIF
IF product not found THEN
       PRINT error
ENDIF
IF no more items THEN
```

```
       BREAK
ENDIF
END WHILE
IF cart not empty THEN
       CALL generate_invoice_sale(customer_name, cart, total_amount)
ENDIF
END FUNCTION

FUNCTION restock_products(products)
       CALL display_products(products)
       GET supplier_name
       cart = empty list
       total_amount = 0
       WHILE true
        GET product_id
              FOR product in products
        IF product_id matches product
       TRY
        GET quantity
       CATCH ValueError
       PRINT error
       BREAK
       END TRY
       UPDATE stock
       ADD item to cart
       INCREMENT total_amount
ENDIF
IF product not found THEN
       PRINT error
ENDIF
       IF no more items THEN
```

```
BREAK
ENDIF
END WHILE
       IF cart not empty THEN
        CALL generate_invoice_restock(supplier_name, cart, total_amount)
        CALL save_products(products)
       ENDIF
END FUNCTION
```

main.py:

```
FUNCTION main
       products = CALL load_products()
       WHILE true
        DISPLAY menu
        GET choice
        IF choice is invalid THEN
                PRINT error
        ELSE IF choice is display THEN
                CALL display_products(products)
        ELSE IF choice is sale THEN
                CALL process_sale(products)
        ELSE IF choice is restocked THEN
                CALL restock_products(products)
        ELSE IF choice is exit THEN
                CALL save_products(products)
        BREAK
        ENDIF
END WHILE
END FUNCTION
```

2.4 Data Structures

Data structures in python can be defined as a way to store and organize data that can be accessed and modified.Python has many built in functions eg, list, tuple, dictionary,set, string, byte and bytearray.

The project implements for the storage of the product data a list of dictionaries because of its flexibility and manipulation. Every dictionary is a product with keys, id, name, brand, quantity, cost, and origin. With this structure, there is easy access and change of product at tributes. The products.txt is a comma separated text file, read into this data structure, during initialization.

```
Eg:

product _ list.append({
’id’:product _ id,
’name’:name,
’brand’:brand,
’quantity’:int(quantity),
’cost’:int(cost) ,
’origin’:origin} )
```

String: The programming language utilizes strings as a data type to handle text value.

Boolean: The value within Algebraic Boolean datatype contains only two options which are True or False values.

Other data types that have not been used are:

Tuples: are immutable data structure meaning that they cannot be changed after creation, it has odered elements also allowing duplicate elements

Sets: are mutable elements that are unordered sets usually remove duplicates

Classes and Objects: a class is a blueprint for creating objects basically defining attributes and methods while a object is an instance of a class holding the data and behaviour

3 Program

3.1 Implementation Overview

The WeCare Skin Care Product Sale System is organized in five Python files: By organizing the system into main.py, read.py, write.py, operations.py, and utils.py, we uphold modularity and follow the requirements of the course. The program keeps running in a loop, showing the products and responding to user input until the user decides to exit.

Key features include:

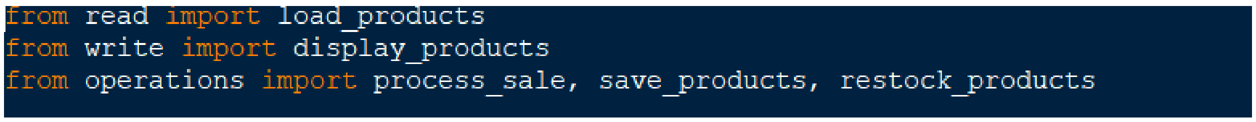
Modularity: Functions are created to serve a particular purpose (e.g., load_products for reading, process_sale for sales), and none are nested.

Error Handling: Malformed input or file problems are caught by try-except blocks in process_sale, restock_products, and save_products so the program does not crash.

User Interface: The software's usability is increased by offering a clear menu and formatted product tables and invoices.

Implementation Details:

main.py: The program is initialized, products are loaded by load_products, and then a menu loop is provided. It makes sure user input is correct and invokes the correct functions, including display_products and process_sale.



```
from read import load_products
from write import display_products
from operations import process_sale, save_products, restock_products
```

read.py: load_products imports products.txt as a list of dictionaries, removes unnecessary spaces, and makes sure each line is valid.

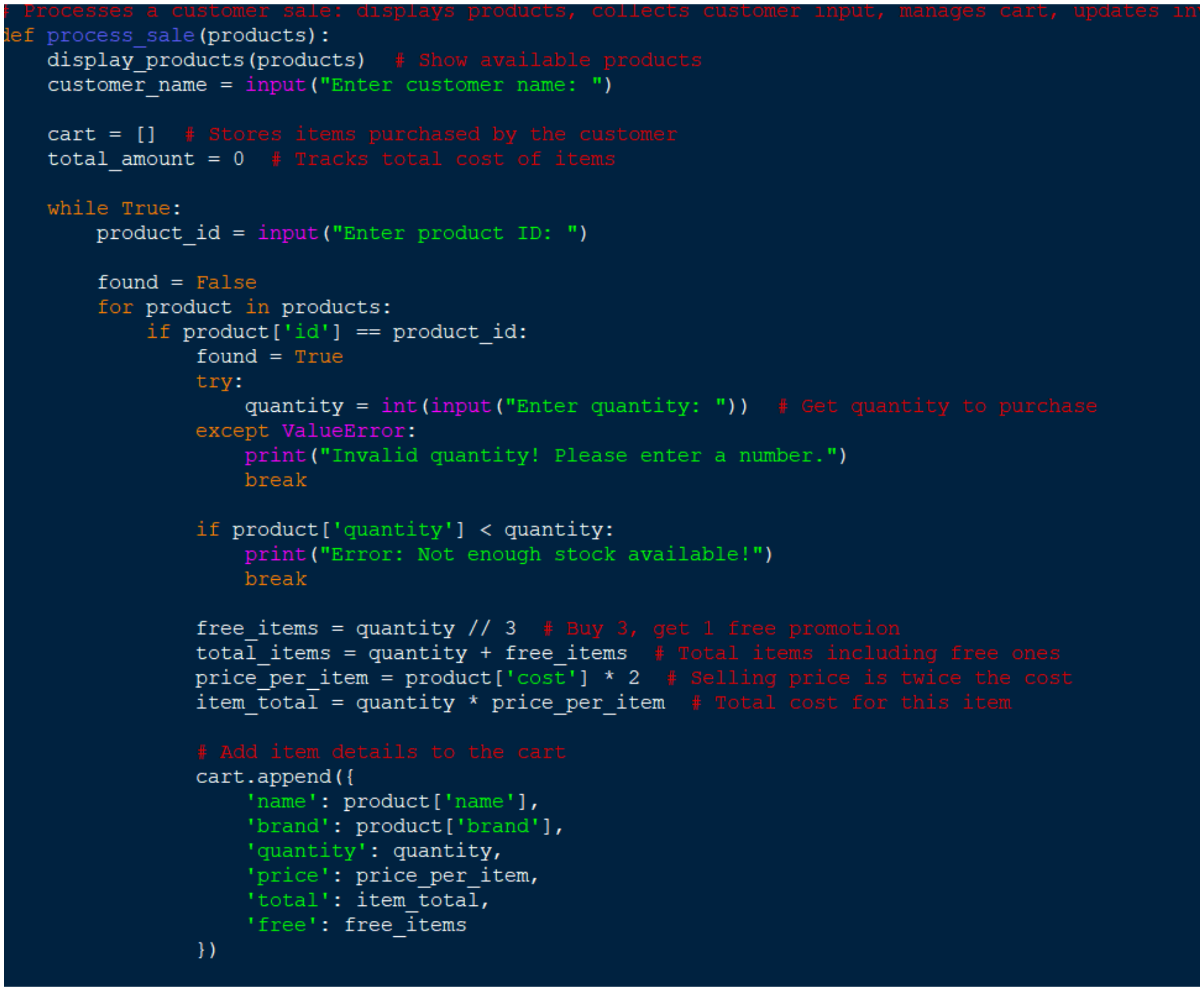
write.py: The display_products function takes the product details, applies a 200% markup, and uses pad_right to align them.

operations.py: Handles transactions:

process_sale function enacts the ‘buy three get one free’ policy and creates sale invoices

restock_products updates the product inventory and issues restock invoices.

save_products ensures data is formatted and saved in a txt file.



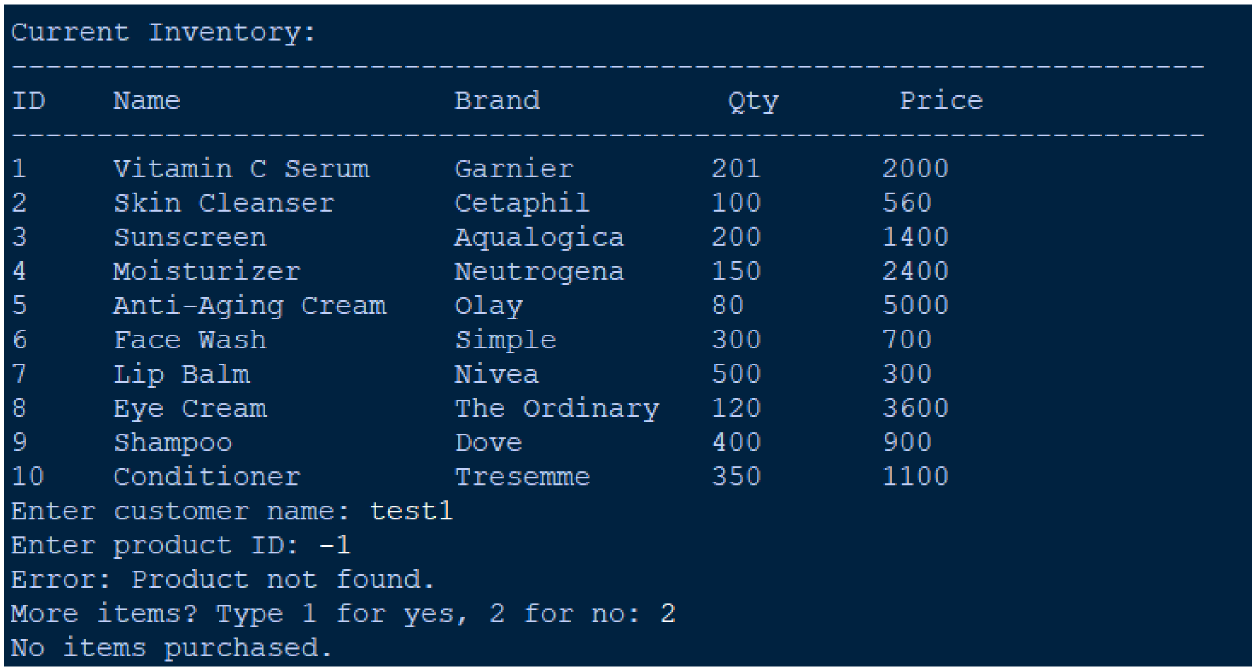
```python
# Processes a customer sale: displays products, collects customer input, manages cart, updates in
def process_sale(products):
    display_products(products)  # Show available products
    customer_name = input("Enter customer name: ")

    cart = []  # Stores items purchased by the customer
    total_amount = 0  # Tracks total cost of items

    while True:
        product_id = input("Enter product ID: ")

        found = False
        for product in products:
            if product['id'] == product_id:
                found = True
                try:
                    quantity = int(input("Enter quantity: "))  # Get quantity to purchase
                except ValueError:
                    print("Invalid quantity! Please enter a number.")
                    break

                if product['quantity'] < quantity:
                    print("Error: Not enough stock available!")
                    break

                free_items = quantity // 3  # Buy 3, get 1 free promotion
                total_items = quantity + free_items  # Total items including free ones
                price_per_item = product['cost'] * 2  # Selling price is twice the cost
                item_total = quantity * price_per_item  # Total cost for this item

                # Add item details to the cart
                cart.append({
                    'name': product['name'],
                    'brand': product['brand'],
                    'quantity': quantity,
                    'price': price_per_item,
                    'total': item_total,
                    'free': free_items
                })
```

utils.py: pad_right and pad_left are provided by utils.py to ensure uniform formatting in product tables and invoices, which increases usability.

4 TESTING

4.1 Test1:

Objective	Implementation of try except
Action	Provide invalid input(ID) and show the message
Expected result	Prompt the user,value being invalid and ask to enter again
Actual Result	Prompts the user,value being invalid and asks if they want another item
Conclusion	Test success

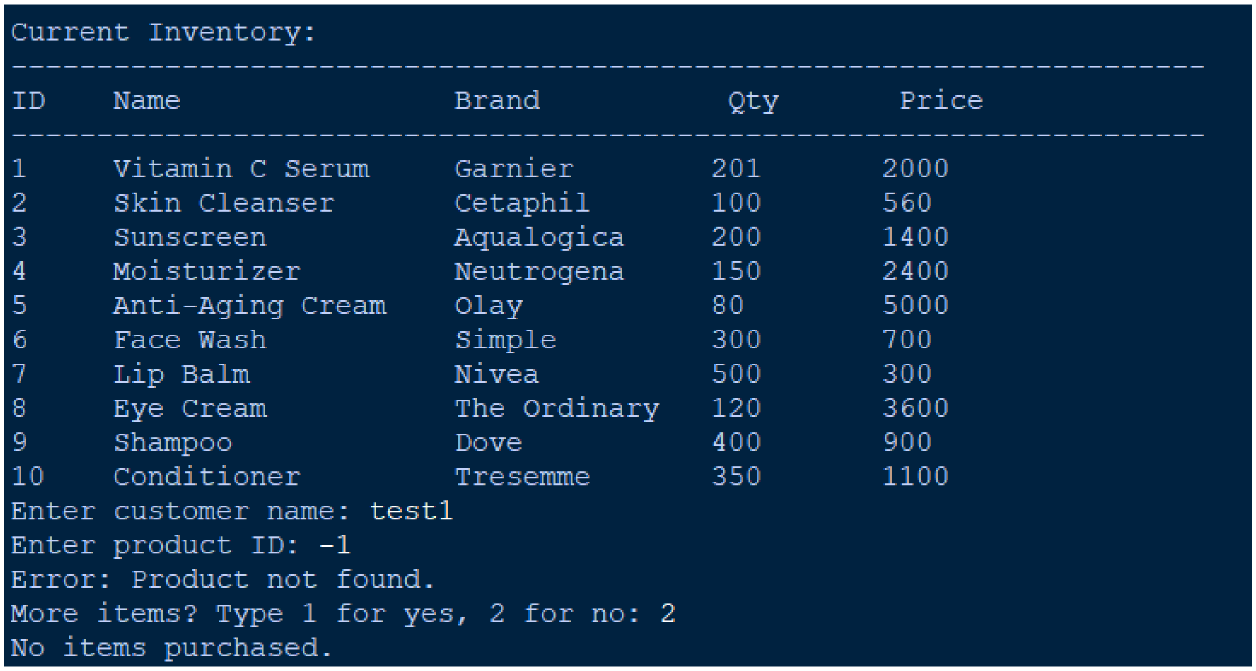


```
Current Inventory:
----------------------------------------------------------------------
ID    Name                Brand           Qty       Price
----------------------------------------------------------------------
1     Vitamin C Serum     Garnier        201       2000
2     Skin Cleanser       Cetaphil       100       560
3     Sunscreen           Aqualogica     200       1400
4     Moisturizer         Neutrogena     150       2400
5     Anti-Aging Cream    Olay           80        5000
6     Face Wash           Simple         300       700
7     Lip Balm            Nivea          500       300
8     Eye Cream           The Ordinary   120       3600
9     Shampoo             Dove           400       900
10    Conditioner         Tresemme       350       1100
Enter customer name: test1
Enter product ID: -1
Error: Product not found.
More items? Type 1 for yes, 2 for no: 2
No items purchased.
```

Figure 2test1

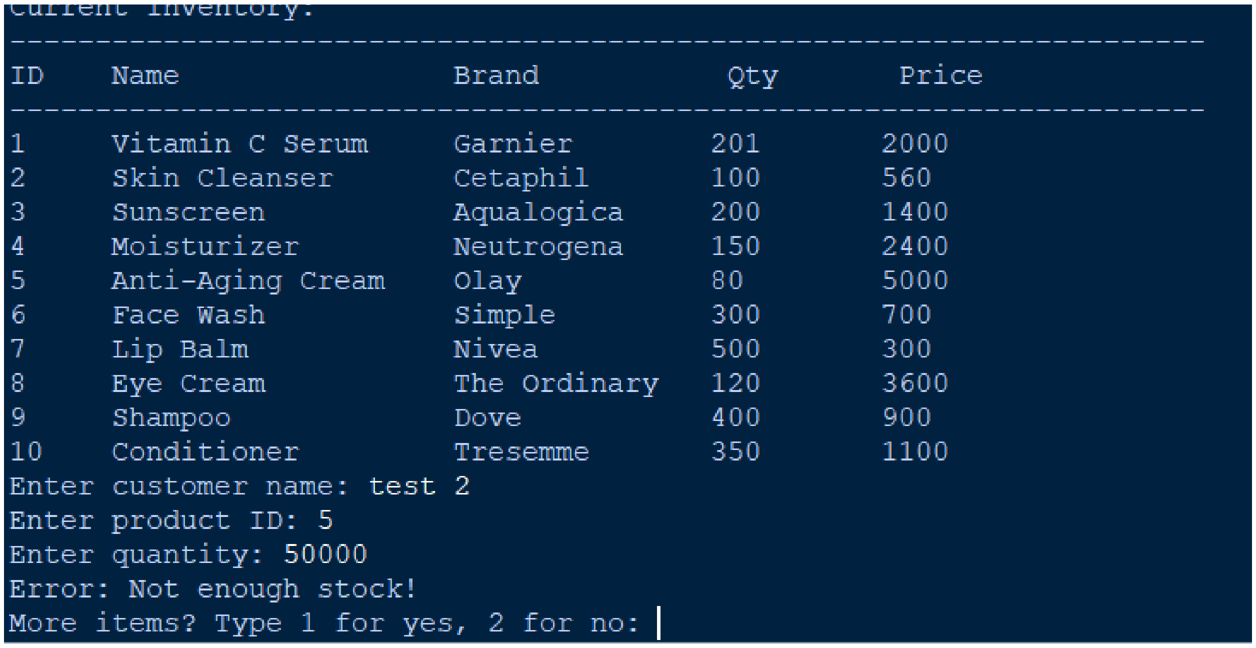
4.2 Test2:

Objective	Implementation of try except
Action	Provide invalid, negative input and show the message
Expected result	Prompt the user,value being invalid and ask to enter again
Actual Result	Prompts the user,value being invalid and asks if they want another item
Conclusion	Test success



```
Current Inventory:
----------------------------------------------------------------------
ID    Name                 Brand            Qty        Price
----------------------------------------------------------------------
1     Vitamin C Serum      Garnier         201        2000
2     Skin Cleanser        Cetaphil        100        560
3     Sunscreen            Aqualogica      200        1400
4     Moisturizer          Neutrogena      150        2400
5     Anti-Aging Cream     Olay            80         5000
6     Face Wash            Simple          300        700
7     Lip Balm             Nivea           500        300
8     Eye Cream            The Ordinary    120        3600
9     Shampoo              Dove            400        900
10    Conditioner          Tresemme        350        1100
Enter customer name: test1
Enter product ID: -1
Error: Product not found.
More items? Type 1 for yes, 2 for no: 2
No items purchased.
```

Figure 3test2.1

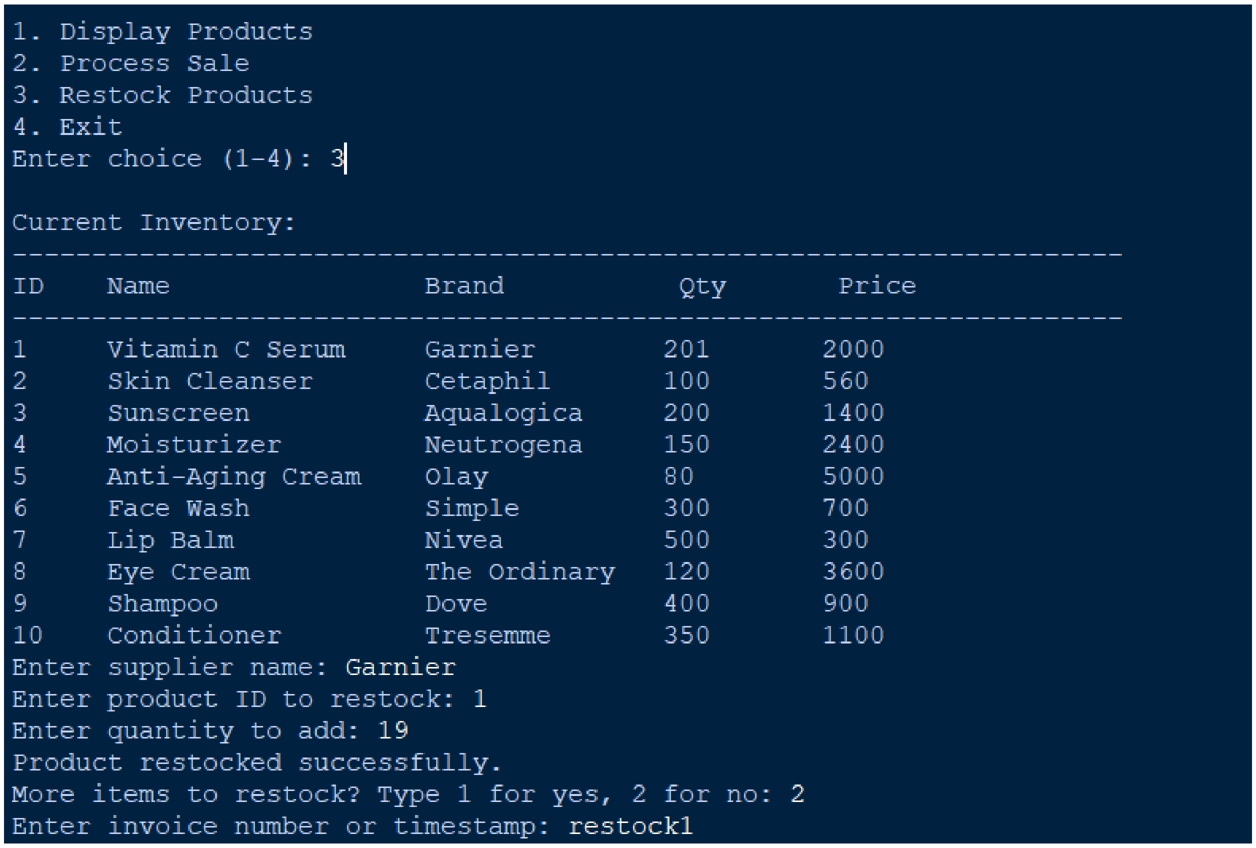


```
Current Inventory:
----------------------------------------------------------------------
ID    Name                 Brand            Qty        Price
----------------------------------------------------------------------
1     Vitamin C Serum      Garnier         201        2000
2     Skin Cleanser        Cetaphil        100        560
3     Sunscreen            Aqualogica      200        1400
4     Moisturizer          Neutrogena      150        2400
5     Anti-Aging Cream     Olay            80         5000
6     Face Wash            Simple          300        700
7     Lip Balm             Nivea           500        300
8     Eye Cream            The Ordinary    120        3600
9     Shampoo              Dove            400        900
10    Conditioner          Tresemme        350        1100
Enter customer name: test 2
Enter product ID: 5
Enter quantity: 50000
Error: Not enough stock!
More items? Type 1 for yes, 2 for no: |
```

Figure 4 test2.2

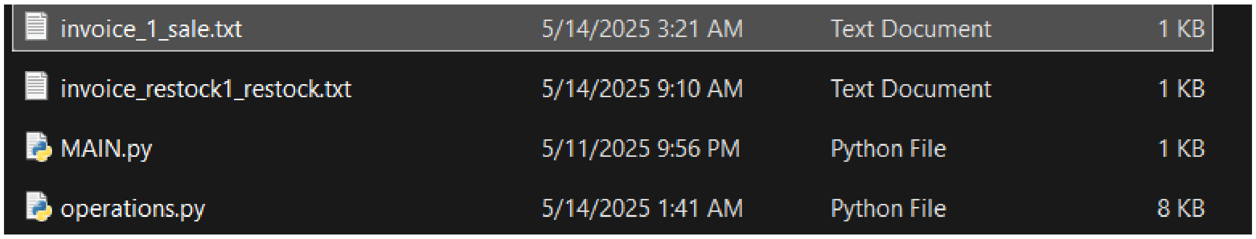
4.3 Test3:

Objective	To make sure sure that multiple products are purchased, displayed in the console and stored in txt file
Action	Filling the required fields and viewing the generated invoice
Expected result	Program should display the fields of choice, display invoice in console
Actual Result	The program displayed the fields of choice and generated the invoice accordingly
Conclusion	Test success



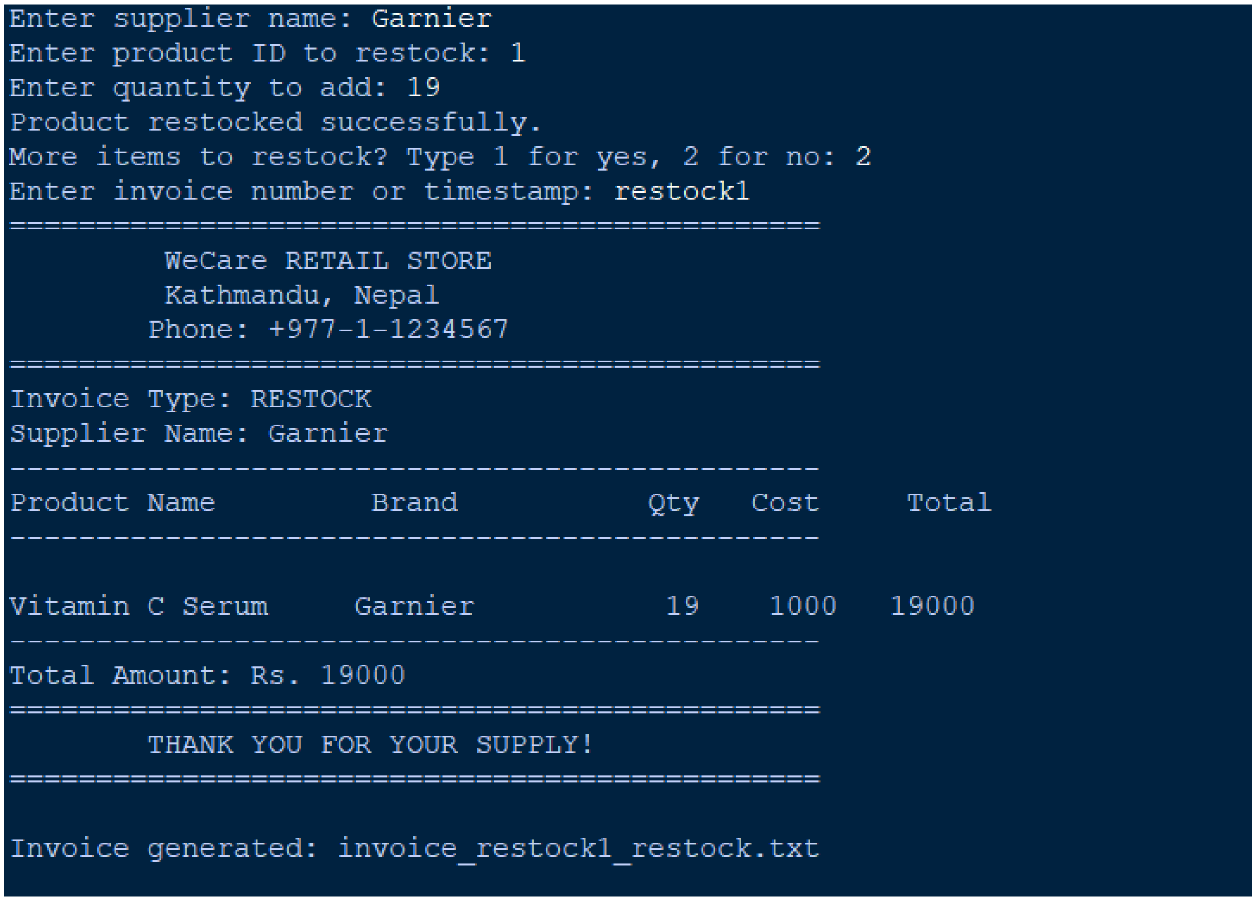
```
1. Display Products
2. Process Sale
3. Restock Products
4. Exit
Enter choice (1-4): 3

Current Inventory:
----------------------------------------------------------------------
ID     Name                Brand           Qty        Price
----------------------------------------------------------------------
1      Vitamin C Serum     Garnier        201        2000
2      Skin Cleanser       Cetaphil       100        560
3      Sunscreen           Aqualogica     200        1400
4      Moisturizer         Neutrogena     150        2400
5      Anti-Aging Cream    Olay           80         5000
6      Face Wash           Simple         300        700
7      Lip Balm            Nivea          500        300
8      Eye Cream           The Ordinary   120        3600
9      Shampoo             Dove           400        900
10     Conditioner         Tresemme       350        1100
Enter supplier name: Garnier
Enter product ID to restock: 1
Enter quantity to add: 19
Product restocked successfully.
More items to restock? Type 1 for yes, 2 for no: 2
Enter invoice number or timestamp: restock1
```

Figure 5 test 3.1



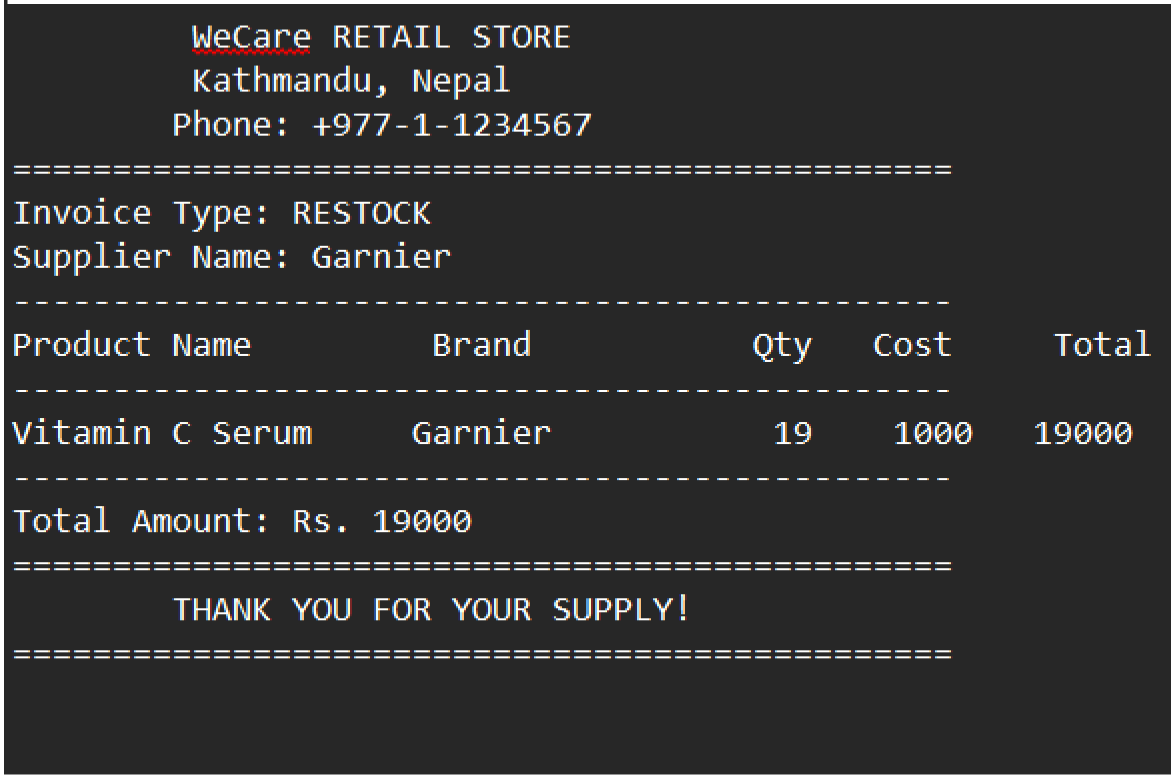
```
invoice_1_sale.txt              5/14/2025 3:21 AM    Text Document    1 KB
invoice_restock1_restock.txt    5/14/2025 9:10 AM    Text Document    1 KB
MAIN.py                         5/11/2025 9:56 PM    Python File      1 KB
operations.py                   5/14/2025 1:41 AM    Python File      8 KB
```

Figure 6 test 3.2 file



```
Enter supplier name: Garnier
Enter product ID to restock: 1
Enter quantity to add: 19
Product restocked successfully.
More items to restock? Type 1 for yes, 2 for no: 2
Enter invoice number or timestamp: restock1
==============================================
         WeCare RETAIL STORE
         Kathmandu, Nepal
        Phone: +977-1-1234567
==============================================
Invoice Type: RESTOCK
Supplier Name: Garnier
----------------------------------------------
Product Name          Brand           Qty   Cost     Total
----------------------------------------------

Vitamin C Serum     Garnier           19    1000   19000
----------------------------------------------
Total Amount: Rs. 19000
==============================================
        THANK YOU FOR YOUR SUPPLY!
==============================================

Invoice generated: invoice_restock1_restock.txt
```

Figure 7test 3.3 invoice

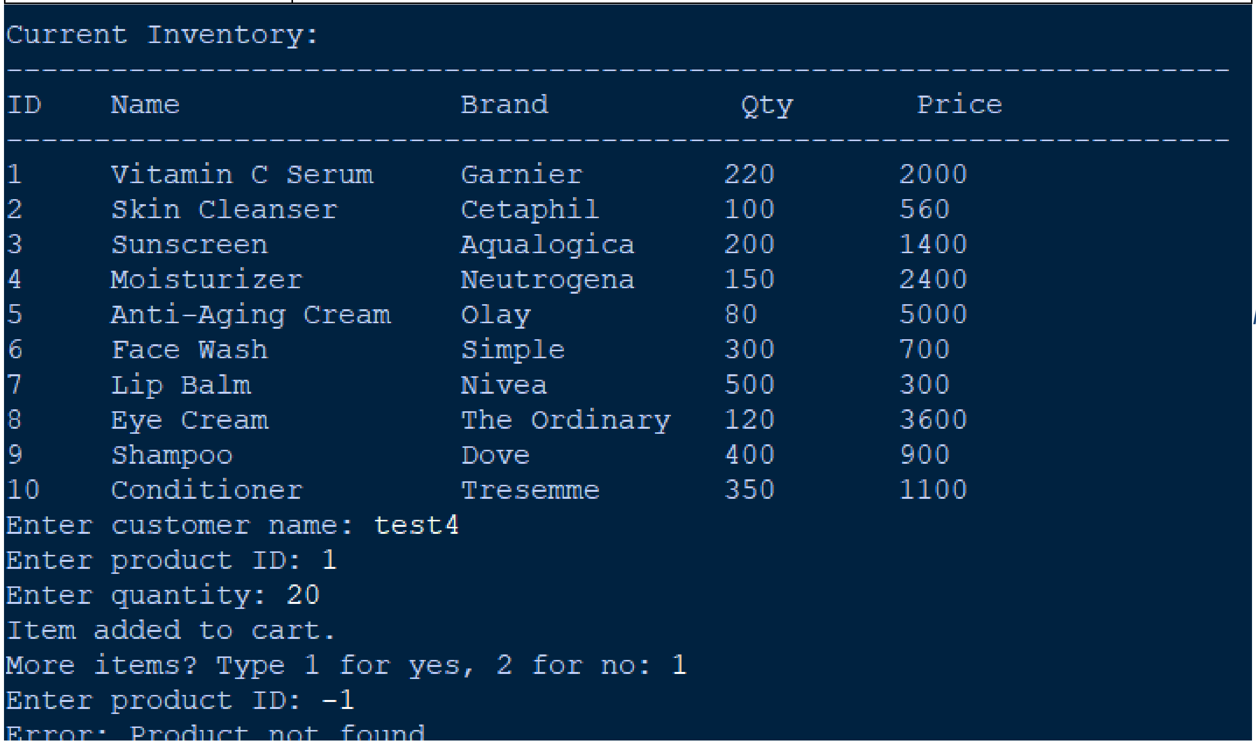


```
         WeCare RETAIL STORE
         Kathmandu, Nepal
        Phone: +977-1-1234567
==============================================
Invoice Type: RESTOCK
Supplier Name: Garnier
----------------------------------------------
Product Name          Brand           Qty   Cost     Total
----------------------------------------------
Vitamin C Serum     Garnier           19    1000   19000
----------------------------------------------
Total Amount: Rs. 19000
==============================================
        THANK YOU FOR YOUR SUPPLY!
==============================================
```

Figure 8test 3.3 invoice

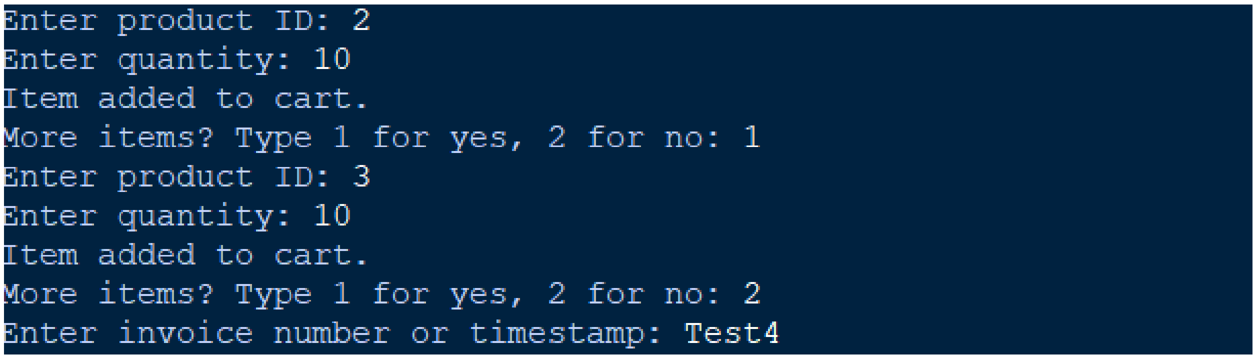
4.4 Test4:

Objective	multiple products are sold and correctly processed, displayed in the console and stored in a new .txt file I filled the required fields then printed an invoice
Action	The program gave me fields to enter which I filled, handled the exceptions, and displayed the invoice.
Expected result	The program should display fields to enter the choice, handle exception (if an occurred), generate invoice and display the invoice in console.
Actual Result	The program should display fields to enter the choice, handle exception (if an occurred), generate invoice and display the invoice in console.
Conclusion	Test success

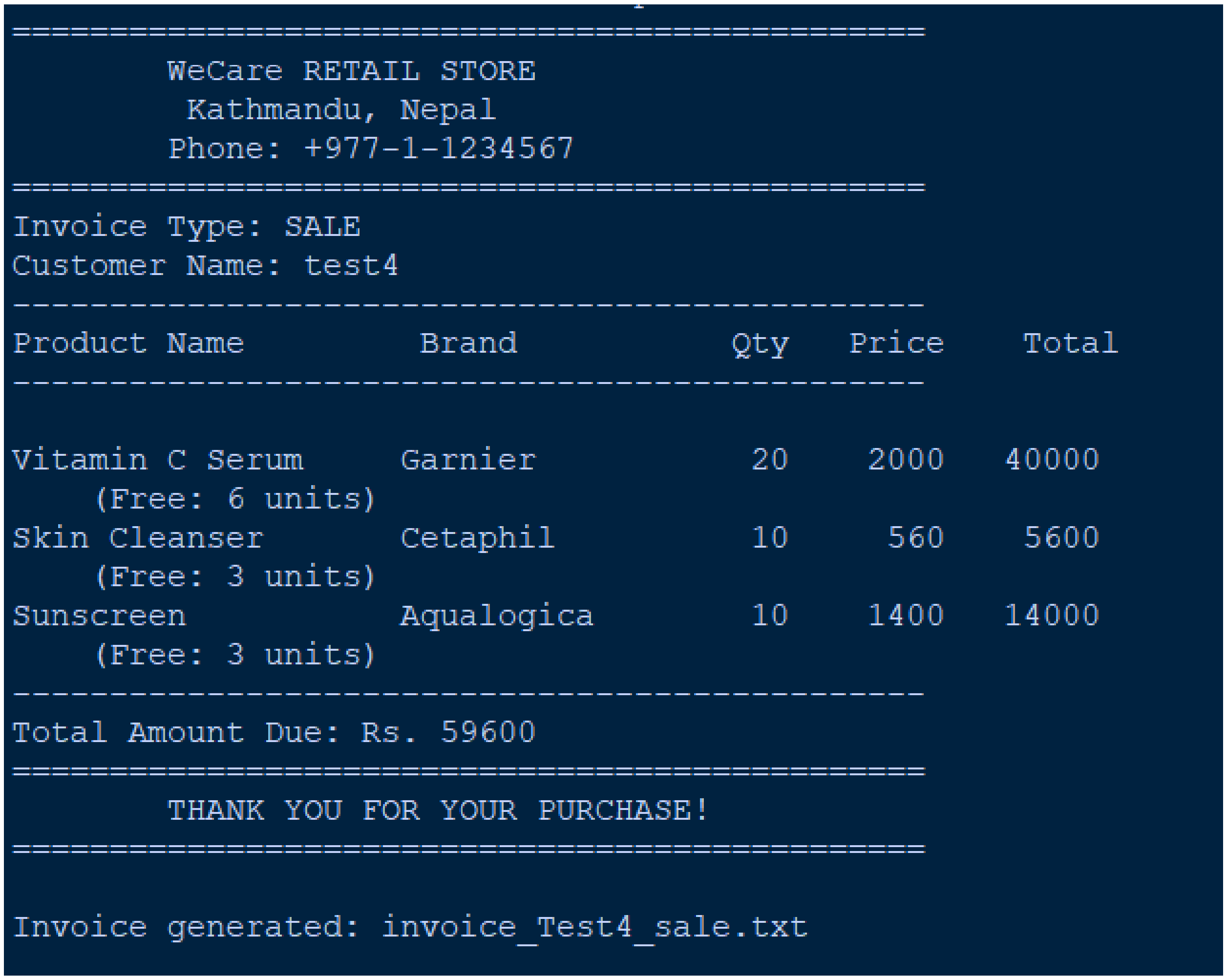


```
Current Inventory:
----------------------------------------------------------------------
ID     Name                Brand           Qty        Price
----------------------------------------------------------------------
1      Vitamin C Serum     Garnier        220        2000
2      Skin Cleanser       Cetaphil       100        560
3      Sunscreen           Aqualogica     200        1400
4      Moisturizer         Neutrogena     150        2400
5      Anti-Aging Cream    Olay           80         5000
6      Face Wash           Simple         300        700
7      Lip Balm            Nivea          500        300
8      Eye Cream           The Ordinary   120        3600
9      Shampoo             Dove           400        900
10     Conditioner         Tresemme       350        1100
Enter customer name: test4
Enter product ID: 1
Enter quantity: 20
Item added to cart.
More items? Type 1 for yes, 2 for no: 1
Enter product ID: -1
Error: Product not found
```

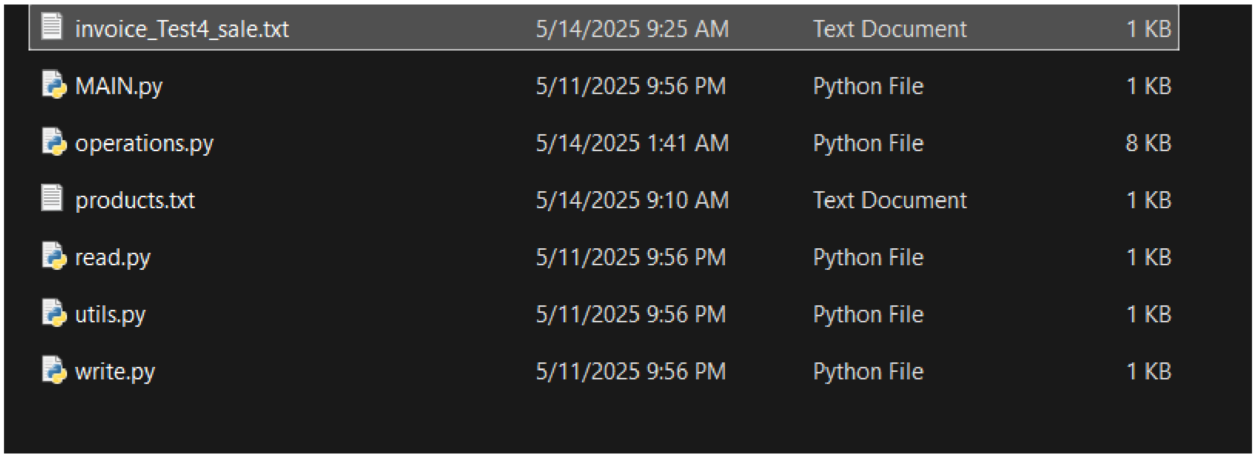
Figure 9: test 4



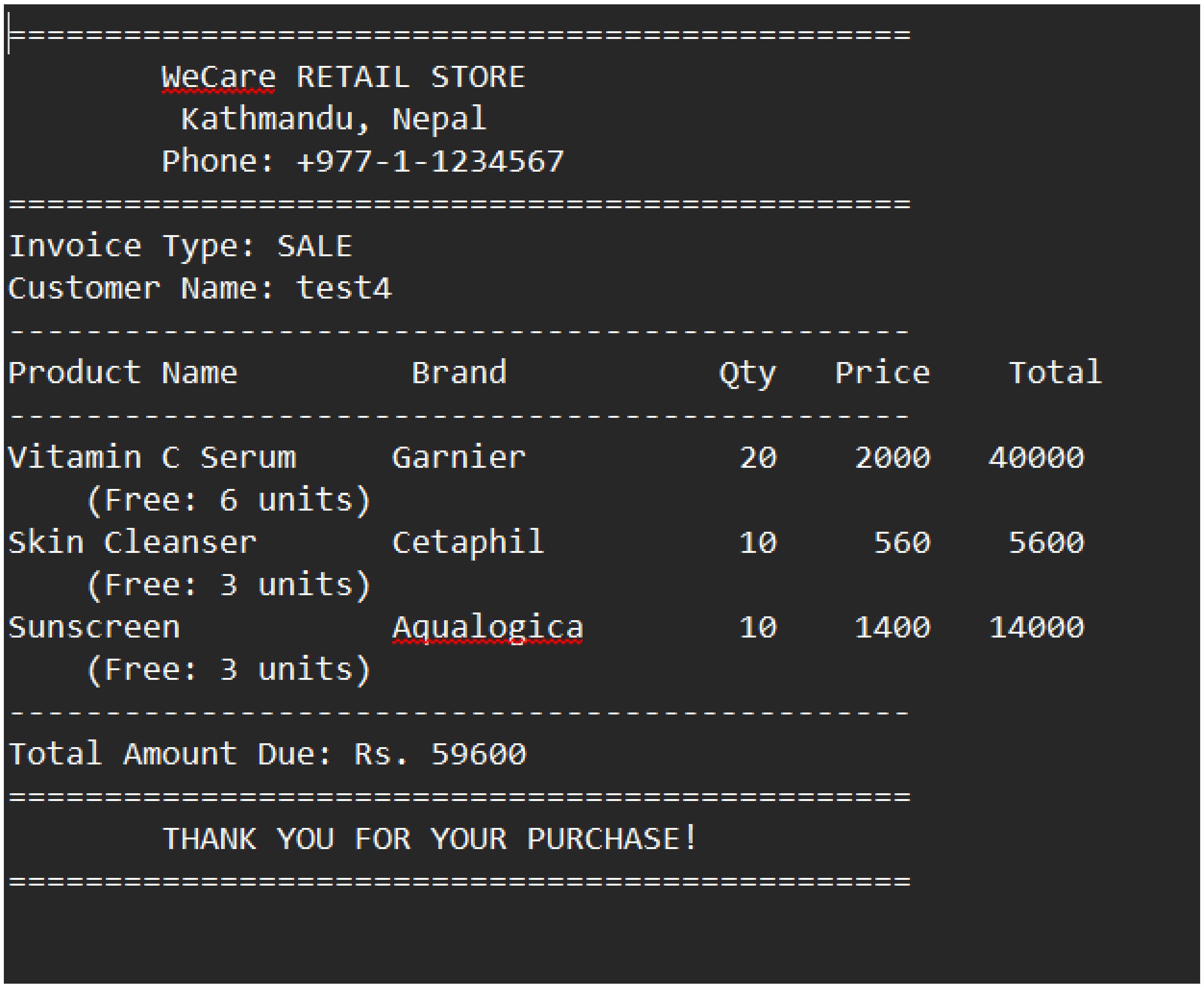
```
Enter product ID: 2
Enter quantity: 10
Item added to cart.
More items? Type 1 for yes, 2 for no: 1
Enter product ID: 3
Enter quantity: 10
Item added to cart.
More items? Type 1 for yes, 2 for no: 2
Enter invoice number or timestamp: Test4
```



```
===============================================
        WeCare RETAIL STORE
         Kathmandu, Nepal
        Phone: +977-1-1234567
===============================================
Invoice Type: SALE
Customer Name: test4
-----------------------------------------------
Product Name         Brand           Qty   Price    Total
-----------------------------------------------

Vitamin C Serum     Garnier           20    2000   40000
    (Free: 6 units)
Skin Cleanser       Cetaphil          10     560    5600
    (Free: 3 units)
Sunscreen           Aqualogica        10    1400   14000
    (Free: 3 units)
-----------------------------------------------
Total Amount Due: Rs. 59600
===============================================
        THANK YOU FOR YOUR PURCHASE!
===============================================

Invoice generated: invoice_Test4_sale.txt
```



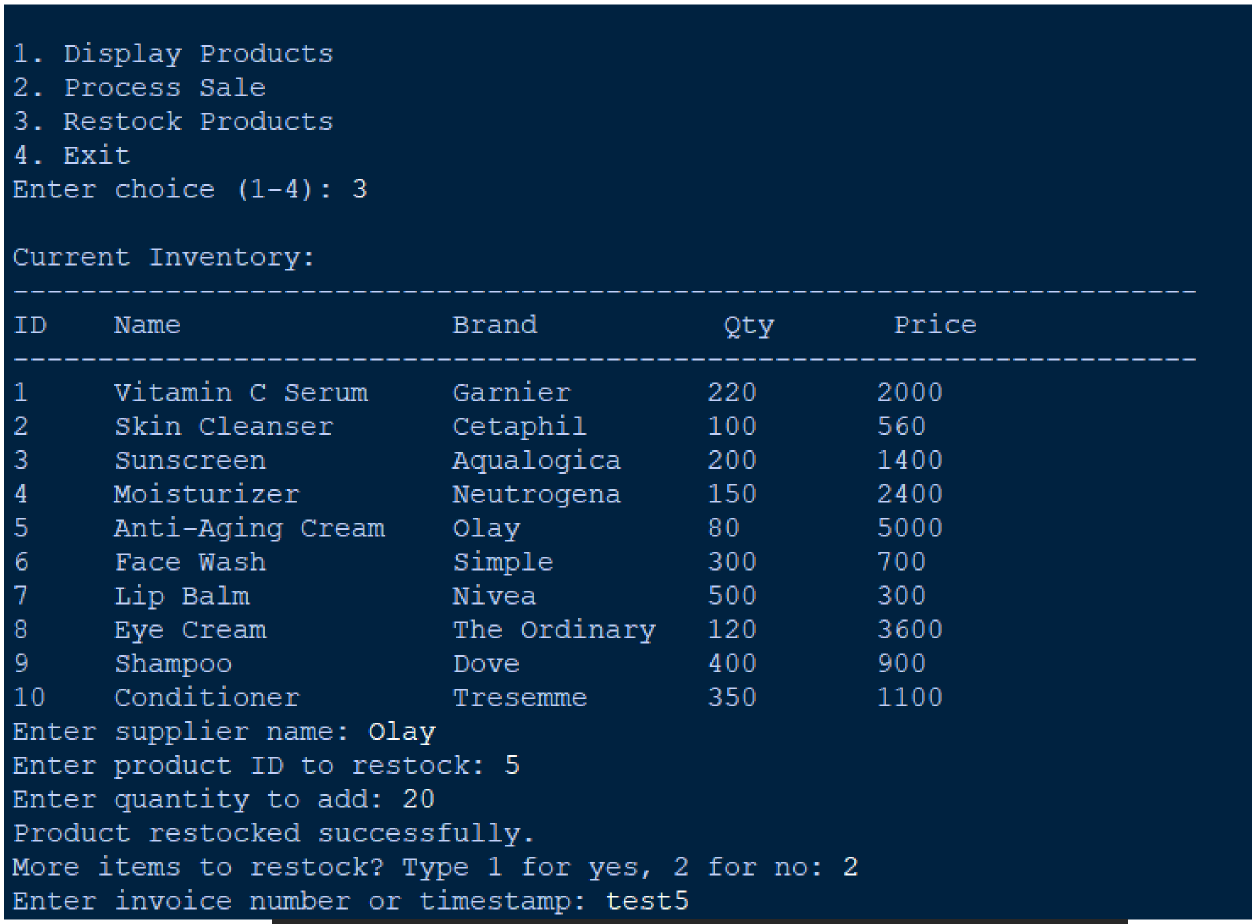
invoice_Test4_sale.txt	5/14/2025 9:25 AM	Text Document	1 KB
MAIN.py	5/11/2025 9:56 PM	Python File	1 KB
operations.py	5/14/2025 1:41 AM	Python File	8 KB
products.txt	5/14/2025 9:10 AM	Text Document	1 KB
read.py	5/11/2025 9:56 PM	Python File	1 KB
utils.py	5/11/2025 9:56 PM	Python File	1 KB
write.py	5/11/2025 9:56 PM	Python File	1 KB



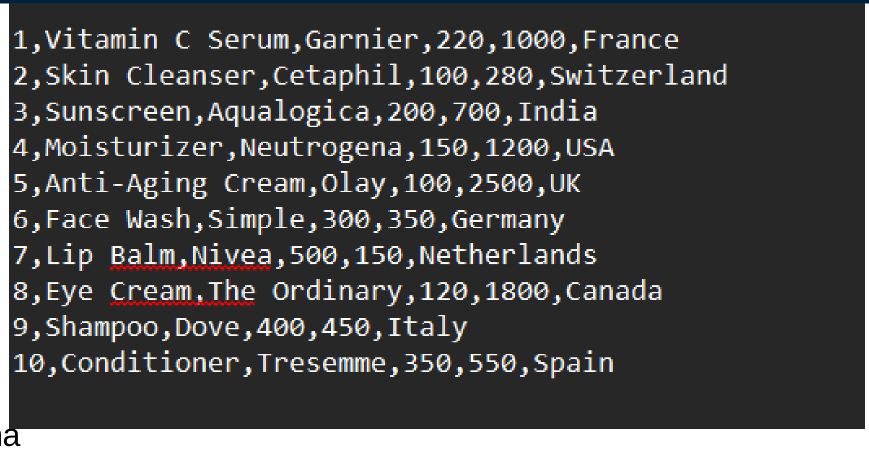
```
===============================================
        WeCare RETAIL STORE
         Kathmandu, Nepal
        Phone: +977-1-1234567
===============================================
Invoice Type: SALE
Customer Name: test4
-----------------------------------------------
Product Name         Brand           Qty   Price    Total
-----------------------------------------------
Vitamin C Serum     Garnier           20    2000   40000
    (Free: 6 units)
Skin Cleanser       Cetaphil          10     560    5600
    (Free: 3 units)
Sunscreen           Aqualogica        10    1400   14000
    (Free: 3 units)
-----------------------------------------------
Total Amount Due: Rs. 59600
===============================================
        THANK YOU FOR YOUR PURCHASE!
===============================================
```

4.5 Test4:

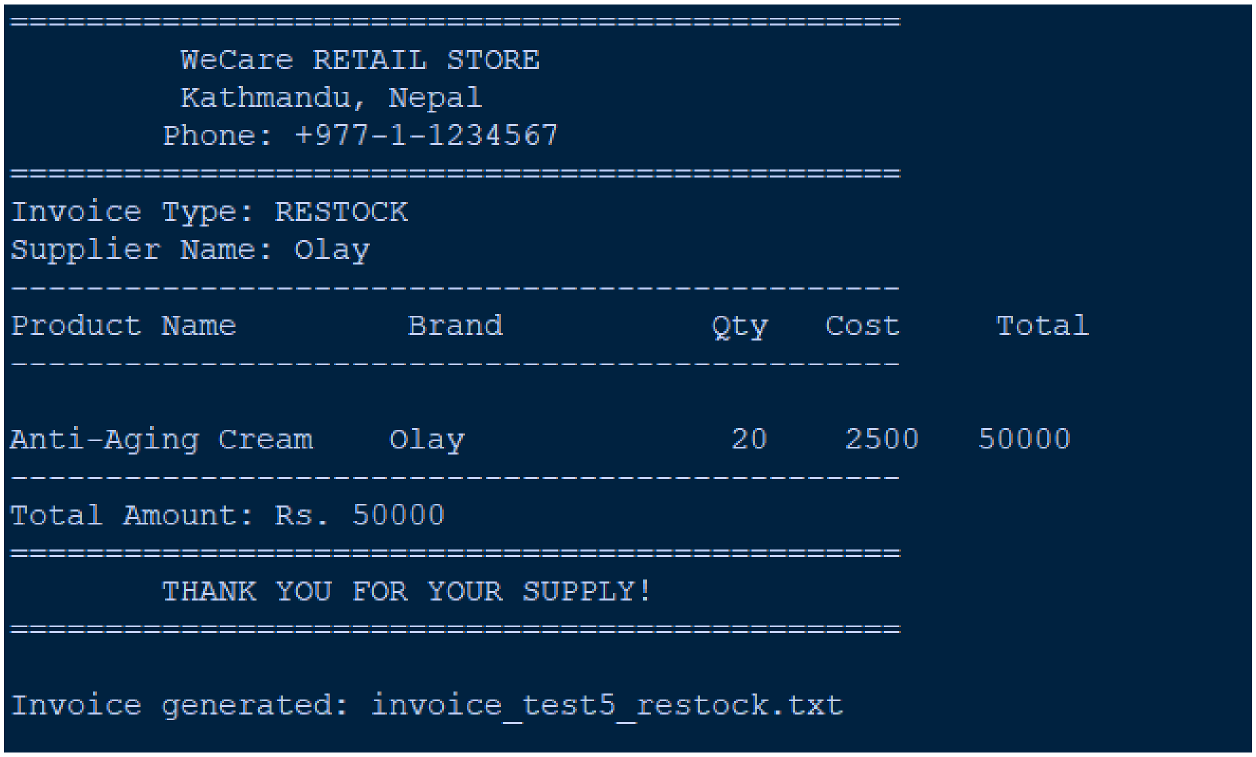
Objective	To confirm that the updated values are reflected on the product.txt file as well (Restocking)
Action	The program gave me fields to enter which I filled, handled the exceptions, and displayed the invoice.
Expected result	The program should bring changes to products.txt file when the user restocks a product.
Actual Result	The program showed changes when the user restocked the product.
Conclusion	Test success



```
1. Display Products
2. Process Sale
3. Restock Products
4. Exit
Enter choice (1-4): 3

Current Inventory:
----------------------------------------------------------------------
ID    Name                Brand           Qty        Price
----------------------------------------------------------------------
1     Vitamin C Serum     Garnier        220        2000
2     Skin Cleanser       Cetaphil       100        560
3     Sunscreen           Aqualogica     200        1400
4     Moisturizer         Neutrogena     150        2400
5     Anti-Aging Cream    Olay           80         5000
6     Face Wash           Simple         300        700
7     Lip Balm            Nivea          500        300
8     Eye Cream           The Ordinary   120        3600
9     Shampoo             Dove           400        900
10    Conditioner         Tresemme       350        1100
Enter supplier name: Olay
Enter product ID to restock: 5
Enter quantity to add: 20
Product restocked successfully.
More items to restock? Type 1 for yes, 2 for no: 2
Enter invoice number or timestamp: test5
```



```
1,Vitamin C Serum,Garnier,220,1000,France
2,Skin Cleanser,Cetaphil,100,280,Switzerland
3,Sunscreen,Aqualogica,200,700,India
4,Moisturizer,Neutrogena,150,1200,USA
5,Anti-Aging Cream,Olay,100,2500,UK
6,Face Wash,Simple,300,350,Germany
7,Lip Balm,Nivea,500,150,Netherlands
8,Eye Cream,The Ordinary,120,1800,Canada
9,Shampoo,Dove,400,450,Italy
10,Conditioner,Tresemme,350,550,Spain
```



```
===============================================
         WeCare RETAIL STORE
         Kathmandu, Nepal
        Phone: +977-1-1234567
===============================================
Invoice Type: RESTOCK
Supplier Name: Olay
-----------------------------------------------
Product Name         Brand          Qty   Cost     Total
-----------------------------------------------

Anti-Aging Cream    Olay             20    2500   50000
-----------------------------------------------
Total Amount: Rs. 50000
===============================================
        THANK YOU FOR YOUR SUPPLY!
===============================================

Invoice generated: invoice_test5_restock.txt
```

5 CONCLUSION

The program, divided into four Python files (read. Using a strong try-except block, the system is protected against invalid inputs and remains stable. The adoption of good naming practices and modular organization makes the code easier to read and maintain. Including 200% selling price markup, and the ability to deal with multiple products in a single transaction incorporated useful business features into the system.

The code is broken down into four separate Python files—read.py, write.py, operations.py, and MAIN.py—which helps make it easier to manage and scale. Every file is responsible for something specific, like reading and writing data or managing sales and restocking, which keeps things organized. The program does everything needed, including displaying what's available, processing sales and purchases, updating the main text file with new stock levels, and generating invoices for each transaction.

From the program's design that handling multiple products and generating one complete bill enhances its efficiency and user-friendliness. The program's responsiveness to different needs is shown by the availability of optional features, such as invoice formats and supplier information for restocking. All considered, the project shows how structured programming results in a dependable, efficient, and retail management system.

6 Lessons Learned

Constructing this retail management system gave me many valuable insights into programming and software design. A major lesson I learned was how valuable modularity is in organizing code. By organizing the program into four files, each with its own job, I learned how to structure a project for easier maintenance and expansion. Breaking up the logic into different functions highlighted why it is essential to create code with a single purpose, as this increases both readability and ease of maintenance. Using try-except blocks was a vital part of learning how to handle errors in advance, making it possible for the program to respond to incorrect inputs, such as non-numeric quantities, instead of failing. This process made it clear that systems should be able to handle user errors. Proper use of naming standards, including the exclusion of ambiguous or culturally biased terms, was another important lesson, since it highlighted how clear naming helps others understand the code.
modularity is an important method for organizing code. Splitting the program into four files, each assigned a particular role, taught me how to organize a project for simpler upkeep and further development. Arranging the logic in different functions showed me that having a single focus for each function is necessary, as it improves both readability and maintenance. By making use of try-except blocks, I understood how to handle errors ahead of time, which enabled the program to smoothly respond to incorrect inputs like non-numeric quantities, rather than crashing. It became obvious during this process that systems must be able to withstand user errors and remain reliable.

7 APPENDIX

7.1 MAIN.py:

```
from read import load_products
from write import display_products
from operations import process_sale, save_products, restock_products

if __name__ == "__main__":
    products = load_products()
    while True:
        print("\n1. Display Products")
        print("2. Process Sale")
        print("3. Restock Products")
        print("4. Exit")
        choice = input("Enter choice (1-4): ")
        if choice in ['1', '2', '3', '4']:
            if choice == '1':
                display_products(products)
            elif choice == '2':
                process_sale(products)
            elif choice == '3':
                restock_products(products)
            elif choice == '4':
                save_products(products)
                print("Exiting program. Changes saved.")
                break
        else:
            print("Invalid choice! Please enter 1, 2, 3, or 4.")
```

7.2 Operations.py:

```
from write import display_products

def process_sale(products):
    display_products(products)
    customer_name = input("Enter customer name: ")

    cart = []  # List to store purchased items
    total_amount = 0

    while True:
        product_id = input("Enter product ID: ")

        found = False
        for product in products:
            if product['id'] == product_id:
                found = True
                try:
                    quantity = int(input("Enter quantity: "))
                except ValueError:
                    print("Invalid quantity!")
                    break

                if product['quantity'] < quantity:
                    print("Error: Not enough stock!")
                    break
                free_items = quantity // 3
                total_items = quantity + free_items
                price_per_item = product['cost'] * 2
                item_total = quantity * price_per_item

                # Add to cart
                cart.append({
                    'name': product['name'],
                    'brand': product['brand'],
                    'quantity': quantity,
                    'price': price_per_item,
                    'total': item_total,
                    'free': free_items
                })

                total_amount += item_total

                # Deduct from inventory
                product['quantity'] -= total_items

                print("Item added to cart.")
                break

        if not found:
            print("Error: Product not found.")

        more_items = input("More items? Type 1 for yes, 2 for no: ")
        if more_items == '2':
            break
```

```
    if not cart:
        print("No items purchased.")
        return
    generate_invoice_sale(customer_name, cart, total_amount)
    save_products(products)

def pad_right(text, length):
    text = str(text)
    while len(text) < length:
        text = text + ' '
    return text

def pad_left(text, length):
    text = str(text)
    while len(text) < length:
        text = ' ' + text
    return text

def generate_invoice_sale(customer_name, cart, total_amount):
    timestamp = input("Enter invoice number or timestamp: ")
    filename = "invoice_" + timestamp + "_sale.txt"
    file = open(filename, 'w')

    # Header
    header = "==================================================\n"
    header += "        WeCare RETAIL STORE\n"
    header += "         Kathmandu, Nepal\n"
    header += "        Phone: +977-1-1234567\n"
    header += "==================================================\n"
    header += "Invoice Type: SALE\n"
    header += "Customer Name: " + customer_name + "\n"
    header += "--------------------------------------------\n"
    header += "Product Name        Brand          Qty   Price    Total\n"
    header += "--------------------------------------------\n"

    print(header)
    file.write(header)

    for item in cart:
        name = pad_right(item['name'], 20)
        brand = pad_right(item['brand'], 15)
        qty = pad_left(item['quantity'], 5)
        price = pad_left(item['price'], 8)
        total = pad_left(item['total'], 8)

        line = name + brand + qty + price + total
        print(line)
        file.write(line + "\n")

        if item['free'] > 0:
            free_line = "    (Free: " + str(item['free']) + " units)"
            print(free_line)
            file.write(free_line + "\n")

    footer = "--------------------------------------------\n"
    footer += "Total Amount Due: Rs. " + str(total_amount) + "\n"
    footer += "================================================\n"
    footer += "        THANK YOU FOR YOUR PURCHASE!\n"
    footer += "================================================\n"

    print(footer)
    file.write(footer)
    file.close()
    print("Invoice generated: " + filename)

def generate_invoice_restock(supplier, cart, total_amount):
    timestamp = input("Enter invoice number or timestamp: ")
    filename = "invoice_" + timestamp + "_restock.txt"
    file = open(filename, 'w')

    header = "==================================================\n"
    header += "         WeCare RETAIL STORE\n"
    header += "         Kathmandu, Nepal\n"
    header += "        Phone: +977-1-1234567\n"
    header += "==================================================\n"
    header += "Invoice Type: RESTOCK\n"
    header += "Supplier Name: " + supplier + "\n"
    header += "--------------------------------------------------\n"
    header += "Product Name         Brand           Qty   Cost     Total\n"
    header += "--------------------------------------------------\n"
    print(header)
    file.write(header)

    for item in cart:
        name = pad_right(item['name'], 20)
        brand = pad_right(item['brand'], 15)
        qty = pad_left(item['quantity'], 5)
        cost = pad_left(item['cost'], 8)
        total = pad_left(item['total'], 8)
        line = name + brand + qty + cost + total
        print(line)
        file.write(line + "\n")
    footer = "--------------------------------------------------\n"
    footer += "Total Amount: Rs. " + str(total_amount) + "\n"
    footer += "==================================================\n"
    footer += "        THANK YOU FOR YOUR SUPPLY!\n"
    footer += "==================================================\n"
    print(footer)
    file.write(footer)
    file.close()
    print("Invoice generated: " + filename)

def save_products(products, filename="products.txt"):
    try:
        with open(filename, 'w') as file:
            for product in products:
                line = product['id'] + "," + product['name'] + "," + product['brand'] + "," +
str(product['quantity']) + "," + str(product['cost']) + "," + product['origin'] + "\n"
                file.write(line)
    except Exception as e:
        print("Error saving products:", e)

def restock_products(products):
    display_products(products)
    supplier = input("Enter supplier name: ")
    cart = []
    total_amount = 0

    while True:
        product_id = input("Enter product ID to restock: ")
        found = False
        for product in products:
            if product['id'] == product_id:
                found = True
                try:
                    quantity = int(input("Enter quantity to add: "))
                except ValueError:
                    print("Invalid quantity!")
                    break
                product['quantity'] += quantity
                total_cost = quantity * product['cost']
                cart.append({
                    'name': product['name'],
                    'brand': product['brand'],
                    'quantity': quantity,
                    'cost': product['cost'],
                    'total': total_cost
                })
                total_amount += total_cost
                print("Product restocked successfully.")
                break
        if not found:
            print("Error: Product not found.")
        more_items = input("More items to restock? Type 1 for yes, 2 for no: ")
        if more_items == '2':
            break

    if cart:
        generate_invoice_restock(supplier, cart, total_amount)
        save_products(products)
    else:
        print("No products restocked.")
```

7.3 Read.py:

```
def load_products(filename="products.txt"):
    product_list = []

    def space(i):
        while i and i[0] == ' ':
            i = i[1:]
        while i and i[-1] == ' ':
            i = i[:-1]
        return i

    file = open(filename, 'r')
    for line in file:
        parts = line.split(',')
        if len(parts) < 5:
            continue  # skip invalid lines

        product_id = space(parts[0])
        name = space(parts[1])
        brand = space(parts[2])
        quantity = space(parts[3])
        cost = space(parts[4])
        origin = space(parts[5].replace('\n', ''))

        product_list.append({
            'id': product_id,
            'name': name,
            'brand': brand,
            'quantity': int(quantity),
            'cost': int(cost),
            'origin': origin
        })

    file.close()
    return product_list
```

7.4 Write.py:

```
from utils import pad_right

def display_products(products):
    print("\nCurrent Inventory:")
    print("-" * 70)
    print("ID    Name                Brand           Qty       Price     ")
    print("-" * 70)
    for product in products:
        pid = product['id']
        name = product['name']
        brand = product['brand']
        qty = str(product['quantity'])
        price = str(product['cost'] * 2)

        pad_id = pad_right(pid, 6)
        pad_name = pad_right(name, 20)
        pad_brand = pad_right(brand, 15)
        pad_qty = pad_right(qty, 10)

        print(pad_id + pad_name + pad_brand + pad_qty + price)
```

7.5 Utlis.py:

```
def pad_right(text, length):
    text = str(text)
    while len(text) < length:
        text = text + ' '
    return text

def pad_left(text, length):
    text = str(text)
    while len(text) < length:
        text = ' ' + text
    return text
```