islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

Fundamental of Computing

70% Individual Coursework

Submission : Final Submission

Academic Semester: Summer Semester 2025

Credit: 15 credit semester long module

Student Name: EIJKEYAL PAKHRIN

London Met ID: 24**49**

College ID: NP**********22**

Assignment Due Date: Wednesday, August 27, 2025

Assignment Submission Date: Friday, August 8, 2025

Submitted To: Pratik Panta & Aryan Thapa

Islington College,Nepal

11% Overall Similarity

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

Match Groups

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

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

1 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

5%	Internet sources
0%	Publications
10%	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.

Eijkeyal.docx

Document Details

Submission ID **trn:oid:::3618:109612663**

Submission Date
Aug 25, 2025, 9:38 PM GMT+5:45

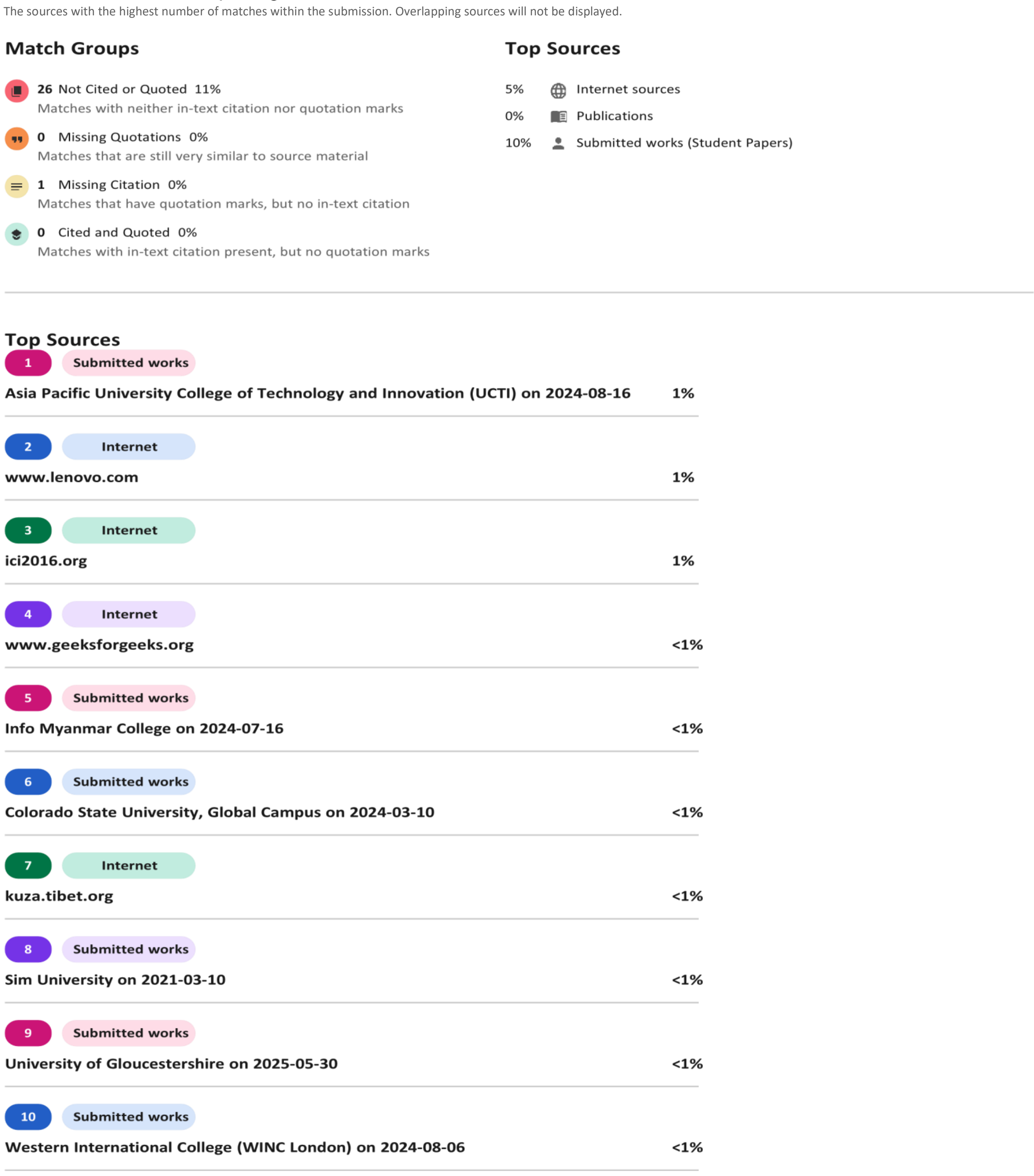
Download Date
Aug 25, 2025, 9:41 PM GMT+5:45 File Name **Eijkeyal.docx** File Size
20.6 KB

10 Pages

3,045 Words

15,956 Characters

Fundamental of Computing



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

Match Groups

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

5%	Internet sources
0%	Publications
10%	Submitted works (Student Papers)

Top Sources

1	Submitted works	Asia Pacific University College of Technology and Innovation (UCTI) on 2024-08-16	1%
2	Internet	www.lenovo.com	1%
3	Internet	ici2016.org	1%
4	Internet	www.geeksforgeeks.org	<1%
5	Submitted works	Info Myanmar College on 2024-07-16	<1%
6	Submitted works	Colorado State University, Global Campus on 2024-03-10	<1%
7	Internet	kuza.tibet.org	<1%
8	Submitted works	Sim University on 2021-03-10	<1%
9	Submitted works	University of Gloucestershire on 2025-05-30	<1%
10	Submitted works	Western International College (WINC London) on 2024-08-06	<1%

Contents

INTRODUCTION

This is the simple python project based on the development of Shoes Wholesale Management System for SpeedzWear. This coursework manage manual inventory and streamline sales operations. Manual handling of stock records, invoices and transactions can be time consuming error-prone and difficult to track. To address this kind of problem, we created a efficient ways to manage inventory system by developing the Shoes Wholesale Management System for the SpeedzWear, a private wholesaler of both domestics and international shoe brands.

This provides a structured way to record, manage and monitor both sales and restock transactions. It reads the inventory of shoes in a text file, which contains details such type , brand, available quantity, unit price and origin of manufacture either domestics or international brands. The program automatically reads this file, processes user inputs, and updates stock whenever a transaction is completed.

One of the key concepts that I applied in the project is file handling in python, which ensures that stock is updated persistently. Another important concept is the use of data structure such as list, int, float, set, tuple, Boolean, string and dictionaries to temporarily store the process and shoe information during the runtime. The project also uses string manipulation to handle the texts-based file records and input validation to ensure that errors are minimized when administrators enter transaction details.

This system has automatic generation of invoices feature in .txt format. Each invoices contains transaction details such as shoe type, brand, quantity, rate, customer name, discount applied and the final total. For sales, discounts are applied If customers purchase in bulk(eg,5% discount for more than 10 items purchased) and additional discounts are given for domestic products(7% discount on bulk purchases). For stocks, invoices summarize the vendor, items restocked and the cost incurred.

This coursework follows the principle of modular programming, where the system is divided into separate functions responsible for file reading/writing transaction handling, discount calculation and invoice generation. This approach improves program clarity, makes testing easier, and allows individual functions to be reused or modified without affecting the entire system.

AIMS AND OBJECTIVE OF THE PROJECT

The main objective of this project is to design and execute a wholesale management application that simplifies the process of managing stocks and transactions in speedzwear. The specific objectives are:

I. It shows what shoes we have in a nice, clear list right from the file, sells them by automatically taking them out of stock and giving the right discount based on the brand.

II. It lets us add new shoes from vendors and update the main list so we always know when to restock.

III. It makes a special receipt file for every sale that says exactly what was bought, the price, the customer's name, the discount, and the final total.

IV. The whole system is built in separate, sturdy blocks so one part can't easily break the others, and it runs safely in a loop until we tell it to shut down.

TECHNOLOGY USED WHILE DOING COURSEWORK

PYTHON(IDLE)

In this coursework used Python IDLE while developing the whole program of the shoe sales management system. Python IDLE (Integrated Development and Learning Environment) is an integrated development for Python that provides convenient tools for writing, testing, and debugging Python code. It provides a graphical user interface (GUI) that helps to simplify the coding process and helps users to manage their projects efficiently. And it also enables you to write and execute a single line of code, much like how Shell writes and edits code. The idle interactive interpreter, often referred to simply as the interpreter, is a fundamental component of many programming languages, including Python. It allows the user to execute commands or statements one at a time, providing a script or program as a whole. It provides an interactive interpreter commonly accessed through the Python shell or IDLE. We can start the Python interpreter by opening a terminal or command prompt and typing python or python3, depending on your Python version. (Python Software Foundation(n.d.). IDLE (Integrated Development and Learning Environment). Python.org)

Figure 1 Python (IDLE)

MS-WORD 4

Microsoft Word is a word processing application developed by Microsoft. I used MS Word for documenting the report of the overall program because it provides different varieties and different tools for designing, creating, editing, and formatting text documents and reports. MS Word provides easy-to-format, professional support for diagrams and screenshots and allows detailed explanations, which make my shoe sales management program documentation clear and presentable. (Microsoft. (n.d.). Microsoft Word. Microsoft.)

Figure 2 MS word

NOTEPAD

Notepad is a simple text editor that comes pre-installed on most of the operating systems. It allows you to create and edit plain text files, such as notes, scripts, and hypertext markup language (HTML) code. But I used Notepad to store the stock file and to save the generated invoice bill and details about the shoe sales management system. It can edit text files (bearing the ".txt" filename extension) and compatible formats, such as batch files, INI files, and log files. Notepad offers only the most basic text manipulation functions, such as finding and replacing text. (Microsoft. (n.d.). Notepad. Microsoft.)

Figure 3 Notepad

DRAW.IO

Draw.io, also known as diagrams.net, is an open-source and cross-platform diagramming tool for creating various types of diagrams, such as flowcharts, network diagrams, UML, and mind maps. It is available as a web application and desktop application. Users can start from scratch or use templates and export diagrams in multiple formats like PNG, JPEG, SVG, and PDF. I used this platform for designing a flowchart of shoe sales management and its motives to show how it works and how it runs or flows through the entire programming till the end. (JGraph. (n.d.). draw.io – Online Diagramming Tool. diagrams.net.)

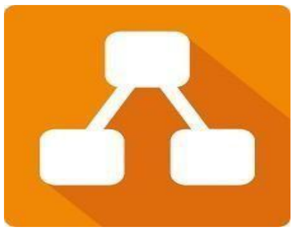


Figure 4 Draw.io

ALGORITHIM FOR SHOES WHOLESALES MANAGEMENT SYSTEM

Step 1: Start.

Step 2: Read stock from text file.

Step 3: Display current stock items in tabular format.

Step 4: Display the four choices.

Step 5: If the option is to select 1, then go to step 6.

If the option is to select 2, go to step 15.

If the option selected is 3, then go to step 22.

If the option is to select 4, then go to step 23.

Step 6: Ask for the customer's name.

Step 7: Select the valid shoe name. If the shoe name and quantity are valid, then go to Step 9. Else go to step 8.

Step 8: If the shoe name is invalid, show the “shoe not found” message and ask again to buy or finish.

Step 9: Update the stock list and selected shoe list in text files.

Step 10: Calculate the total and apply a discount based on the number of quantities selected and the origin of the shoes.

Step 11: If the customer wants to buy again, then go to step 7. Otherwise, go to step 12.

Step 12: Display the bill in the terminal and generate the invoice bill in text files.

Step 13: Open the stock file to update with new quantities.

Step 14: Save the sold bill to a new file and return to step 1.

Step 15: Ask vendor for restock.

Step 16: Ask the shoe type if it is valid, go to 17, else, go to 15.

Step 17: Enter the quantity of shoes.
Step 18: Update product quantity and bought list.

Step 19: Calculate the total purchase amount and display the purchased bill.

Step 20: Open stock txt files in write mode and update new stock.

Step 21: Open a new text file and generate a purchased bill in the file, then go to step 1.

Step 22: Displayed updated stock in terminal. Then go to step 1.

Step 23: Display exiting… Goodbye!

FLOWCHART OF THE PROGRAM

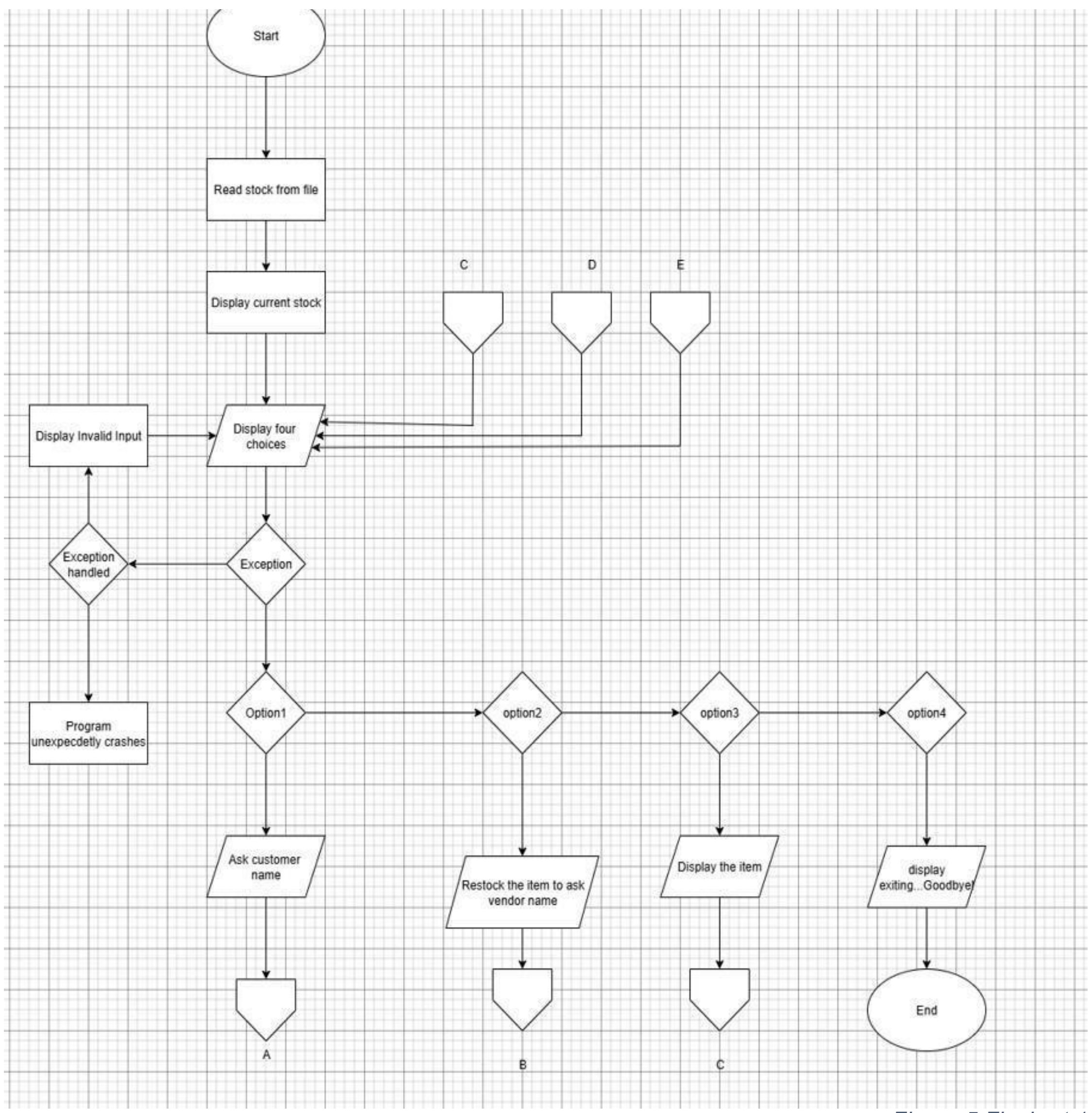


Figure 5 Flochart 1

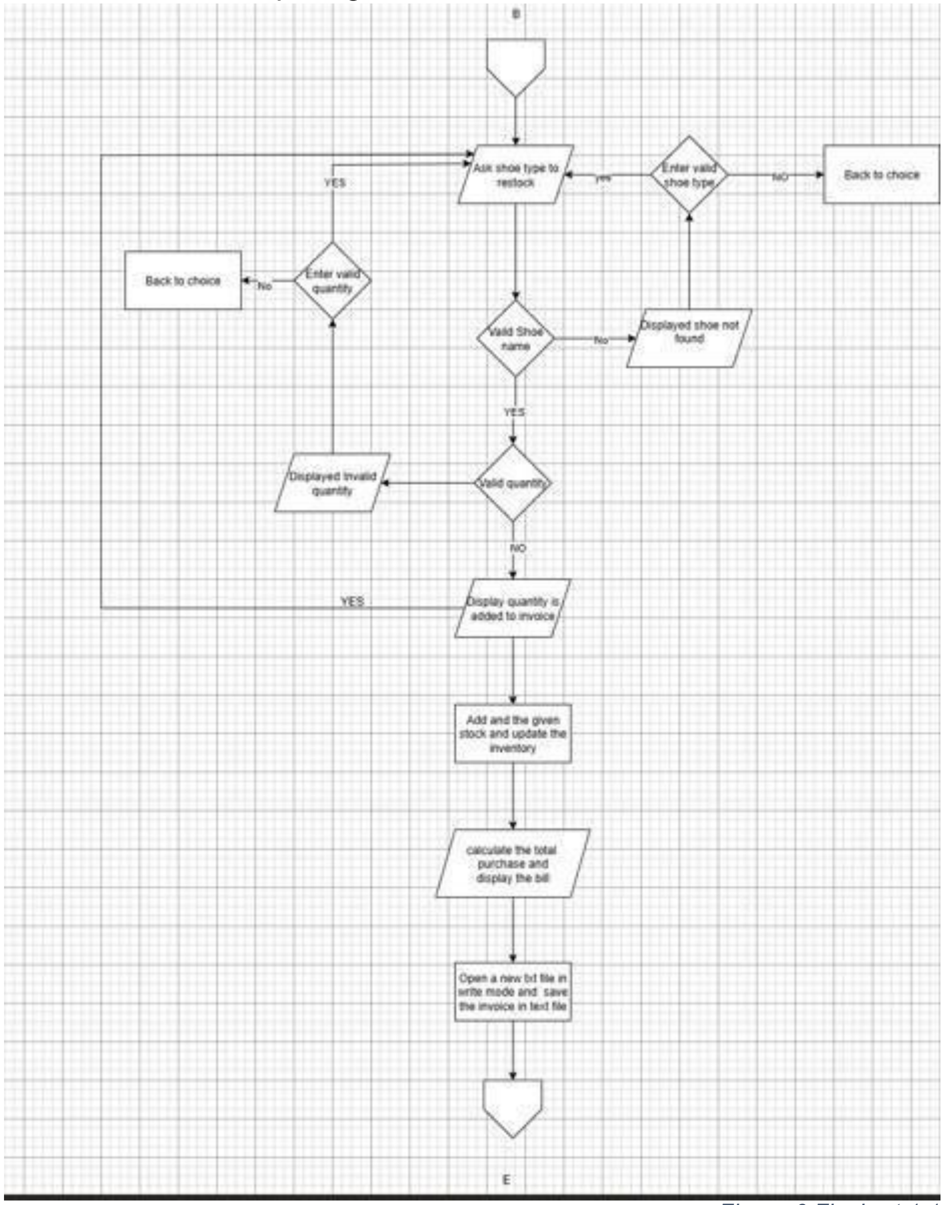

B
Ask shoe type to restock
yes
Enter valid shoe type
NO
Back to choice
YES
Back to choice
No
Enter valid quantity
Valid Shoe name
No
Displayed shoe not found
YES
Displayed Invalid quantity
Valid quantity
NO
YES
Display quantity is added to invoice
Add and the given stock and update the inventory
calculate the total purchase and display the bill
Open a new txt file in write mode and save the invoice in text file
E

Figure 6 Flochart 1.1

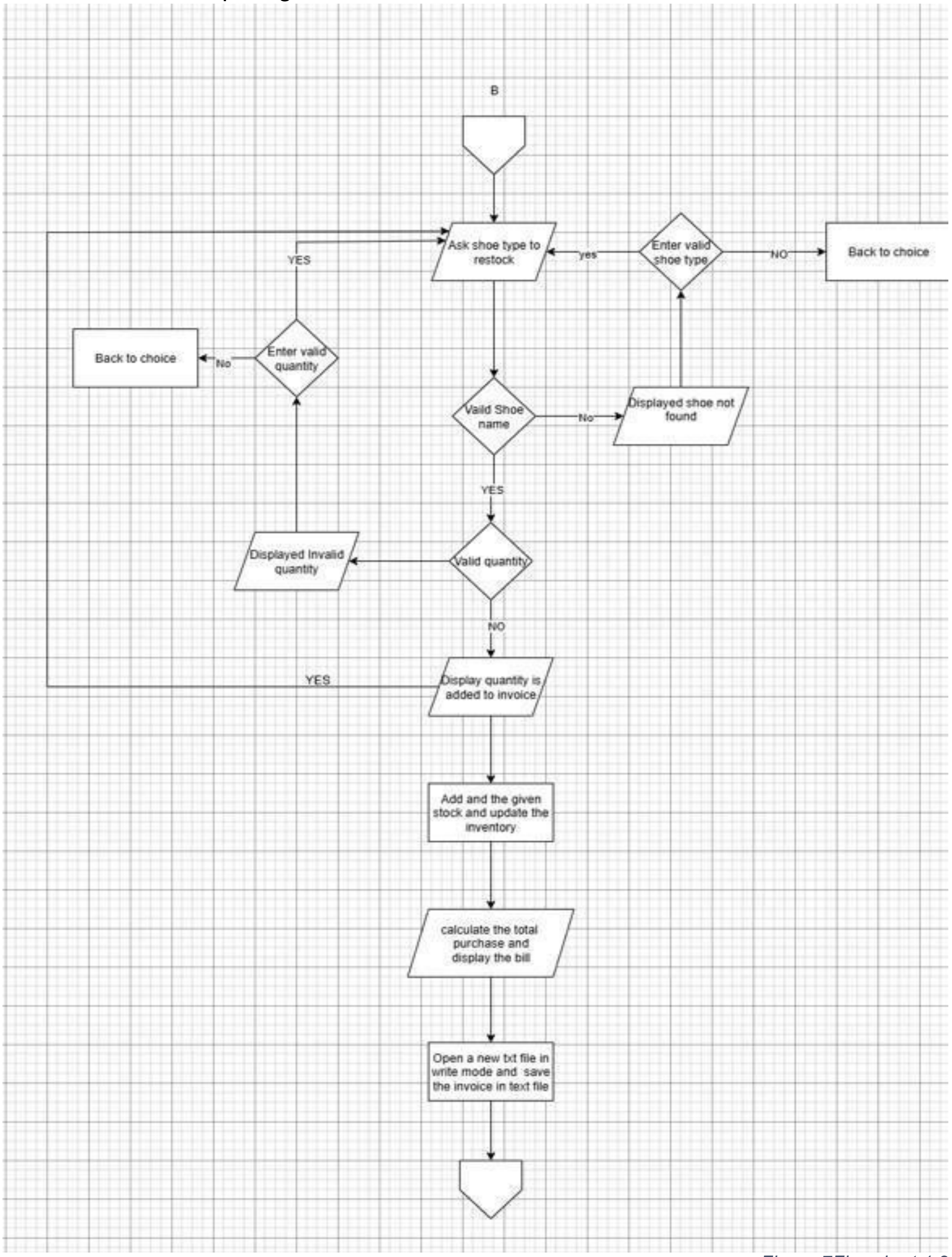

B
Ask shoe type to restock
yes
Enter valid shoe type
NO
Back to choice
YES
Back to choice
No
Enter valid quantity
Vaild Shoe name
No
Displayed shoe not found
YES
Displayed Invalid quantity
Valid quantity
NO
YES
Display quantity is added to invoice
Add and the given stock and update the inventory
calculate the total purchase and display the bill
Open a new txt file in write mode and save the invoice in text file

Figure 7Flowchart 1.2

PSEUDOCODE

Pseudocode is a high level description of program that allows programmer to do based on this pseudocode. It is not written in a specific programming language, but uses structured statements to explain the logic of the program in a way that is easy to read and understand.

Read _file.py

```
START

DEFINE an empty list called shoes

TRY
    OPEN the file with the given filename for reading

    FOR each line in the file
        IF line is empty, SKIP it

        SPLIT the line by tab into parts
        IF number of parts is not 5, PRINT "Skipping invalid line" and SKIP it

        ASSIGN parts to shoe_type, brand, qty, price, origin

        TRY
            CONVERT qty to integer
            CONVERT price to float
            CREATE a shoe dictionary with type, brand, quantity, price, origin
            ADD the shoe dictionary to shoes list
        CATCH conversion error
            PRINT "Skipping invalid line"

CATCH file not found error
    PRINT "File not found"

RETURN shoes list

END
```

Write_file.py

```
Function print_sale_invoice(filename, items, customer):
Print "SALE INVOICE"
Print current date and time
Print customer name
Print table headers: Type, Brand, Qty, Price, Origin, Discount, Total
total_amount = 0 For each item in items: total_amount += item total Print
item details in table format

Print total amount
Open file with given filename in write mode
Write all the same info to the file (invoice table and totals)
Close file
End Function

Print Restock Invoice
Function print_restock_invoice(vendor, items, filename):
Print "RESTOCK INVOICE"
Print current date and time
Print vendor name
Print table headers: Type, Brand, Qty, Price, Origin
total_amount = 0 For each item in items: line_total
= quantity * price total_amount += line_total Print
item details in table format

Print total amount
Open file with given filename in write mode
Write all the same info to the file (invoice table and totals)
Close file
End Function

Update Stock File
Function write_stock_file(filename, shoes):
Open file with given filename in write mode For
each shoe in shoes:
Write shoe info: type, brand, quantity, price, origin (tab separated)
Close file
End Function
```

Operation_file.py

```
Function find_shoe(shoes, shoe_name):
For each shoe in shoes:
If shoe name matches shoe_name:
```

```
Return shoe
Return "not found"
End Function

Function sell_shoes(shoes):
Repeat:
Ask user: "Enter customer name"
Until name is not empty

sold_items = empty list

Loop:
Ask user: "Enter shoe type to sell (or type 'done' to finish)"
If input is "done", stop loop

Find shoe in stock If shoe
not found:
Print "Shoe not in stock"
Continue to next

Ask user: "Enter quantity to sell" If quantity
is invalid or more than stock:
Print "Invalid quantity"
Continue to next

# Apply discount
If quantity >= 10 AND origin is international:
discount = 5%
Else if quantity >= 10 AND origin is domestic:
discount = 7% Else: discount = 0%

total_price = quantity * price * (1 - discount)
Add shoe details to sold_items
Reduce stock by quantity

If sold_items is not empty:
Print invoice on screen
Save invoice to file
Update stock file
End Function
Function restock_shoes(shoes): Ask user:
"Enter vendor name"  restocked_items =
empty list
```

```
Loop:
Ask user: "Enter shoe type to restock (or type 'done' to finish)"
If input is "done", stop loop

Find shoe in stock If shoe
found:
Ask user: "Enter quantity to add" If quantity
is valid:
Add quantity to stock
Add shoe info to restocked_items Print
confirmation Else:
Print "Invalid quantity" Else:
Print "Shoe not found"

If restocked_items is not empty:
Print invoice on screen
Save invoice to file
Update stock file
Print "Restock completed"
End Function
```

Main_file.py

```
START

LOAD shoes from file
DISPLAY stock

LOOP forever
SHOW menu: Sell, Restock, Display, Exit
GET user choice

IF choice is Sell
```

```
PROCESS sale
SAVE shoes to file
ELSE IF choice is Restock
PROCESS restock
SAVE shoes to file
ELSE IF choice is Display
LOAD shoes from file
DISPLAY stock
ELSE IF choice is Exit
PRINT "Goodbye"
STOP loop
ELSE
PRINT "Invalid choice"

END LOOP

END
```

DATA STRUCTURE

A data structure is the way of organizing and storing the data in a computer so that it can be used efficiently.

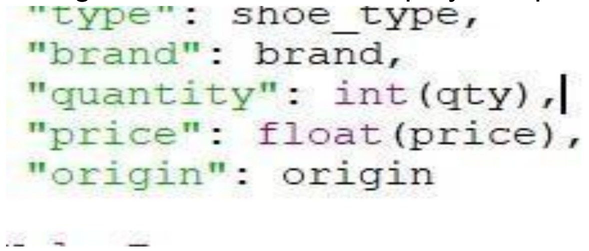
I. It allows to store multiple pieces of data in an organized way.

II. It gives access to modify and process the data.

III. Choosing the right data type or data structure affects the performance and simplicity of your program.

There is different types of data structure that I used in this coursework.

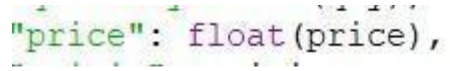
Integer (Int): The integer data type represents the whole numbers without decimal points. In this program, integers are used to store the quantity of shoes available in stock. Integer Values allows the program to perform operations like counting, comparisons and conditional checks efficiently. For example, the quantity of a particular shoe is stores as an integer so that it can be displayed, updated or checked against purchase conditions.



```
"type": shoe_type,
"brand": brand,
"quantity": int(qty),|
"price": float(price),
"origin": origin
```

Figure 8 Integer

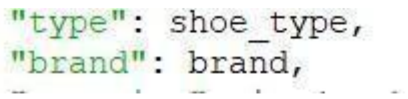
Float(float): The data type represents decimal numbers, which are used for precise measurements for monetary values or scientific values. In this program, the price of each shoe is stored as a float. This allows the program to display the price correctly with two decimal places and perform calculations such as applying discounts or totaling purchase amounts.



```
"price": float(price),
```

Figure 9 Float

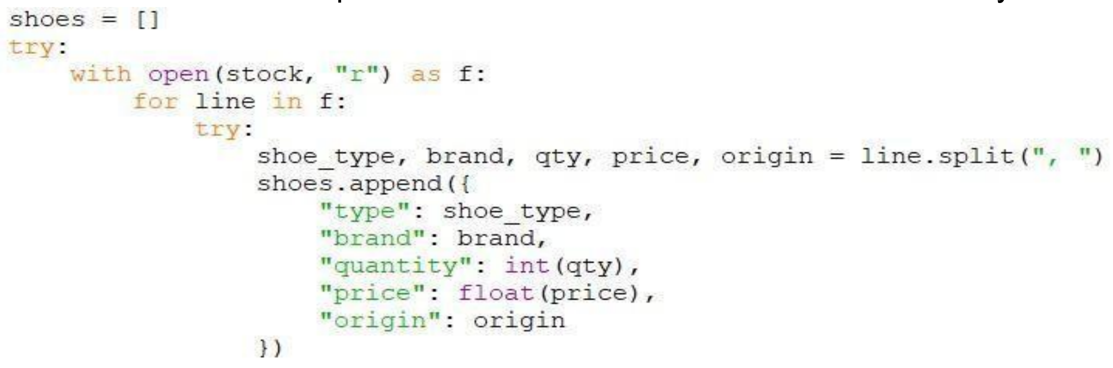
String(str): The string data type is used to store sequences of characters, such as text. In this program, stings store information like shoe type, brand and category. String operations, including splitting lines from the file, capitalizing words and removing newline characters, are used to process and format the data correctly for display and storage.



```
"type": shoe_type,
"brand": brand,
```

Figure 10 String

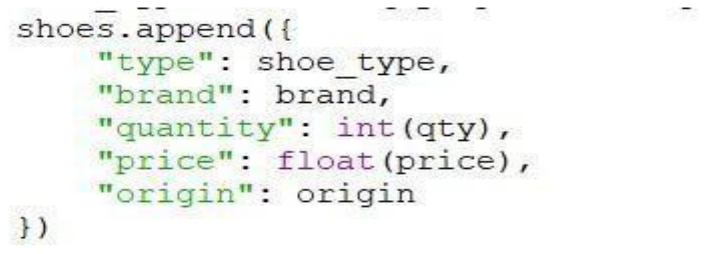
List: A list is an ordered collection of items that can store multiple elements. In this program, a list is used to store all the store records(stock_list). Lists allow the program to dynamically add new shoe entries and iterate through them when displaying the stock. Each element of the list represents one shoe record stored as a dictionary.



```
shoes = []
try:
    with open(stock, "r") as f:
        for line in f:
            try:
                shoe_type, brand, qty, price, origin = line.split(", ")
                shoes.append({
                    "type": shoe_type,
                    "brand": brand,
                    "quantity": int(qty),
                    "price": float(price),
                    "origin": origin
                })
```

Figure 11 List

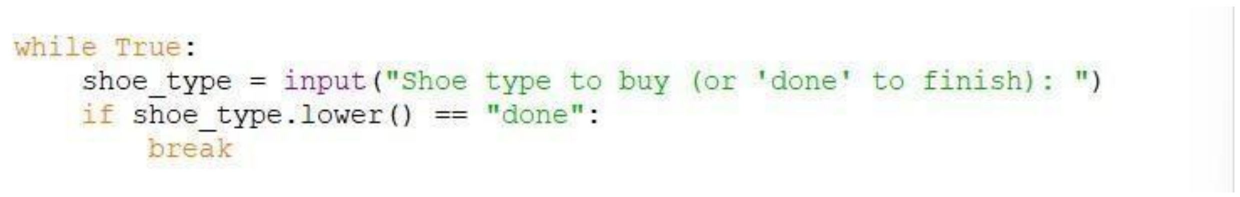
Dictionary: A dictionary is a collection of key:Values pairs that allows storing structured data. Each shoe is stored as a dictionary with the key values like “Type”, “Brand”, ”Quantity”,” Price” and “Category”. This structure allow easy access to each attributes of a shoe by referencing its key, which is helpful for processing, updating and displaying information.



```
shoes.append({
    "type": shoe_type,
    "brand": brand,
    "quantity": int(qty),
    "price": float(price),
    "origin": origin
})
```

Figure 12 Dictionary

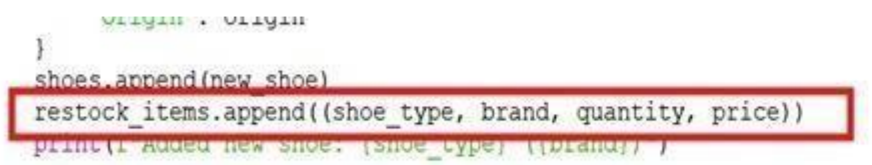
Boolean(bool): The Boolean data type represents two values either True or False. In this program, Booleans are used in conditional statement to make decisions, such as checking whether the user wants to buy shoes. Booleans values help control the flow of the program based on logical conditions.



```
while True:
    shoe_type = input("Shoe type to buy (or 'done' to finish): ")
    if shoe_type.lower() == "done":
        break
```

Figure 13 boolean

Tuple(tuple): A tuple is an ordered, immutable collection of items. Tuples can store the multiple elements like list but cannot be changed once created. While this program primarily uses lists and dictionaries, tuples can be used to represent fixed data such as, shoe’s type, price, brand name, discount percent ensuring the pair cannot be accidentally modified.



```
    "origin": origin
}
shoes.append(new_shoe)
restock_items.append((shoe_type, brand, quantity, price))
print(f"Added new shoe: {shoe_type} ({brand})")
```

Figure 14 Tuple

Set(set): A set is an unordered collection of unique items. Sets can be used to remove the duplication values from the customers ID. For example, a set could be used in this program to find all unique shoe brands available in the stock.

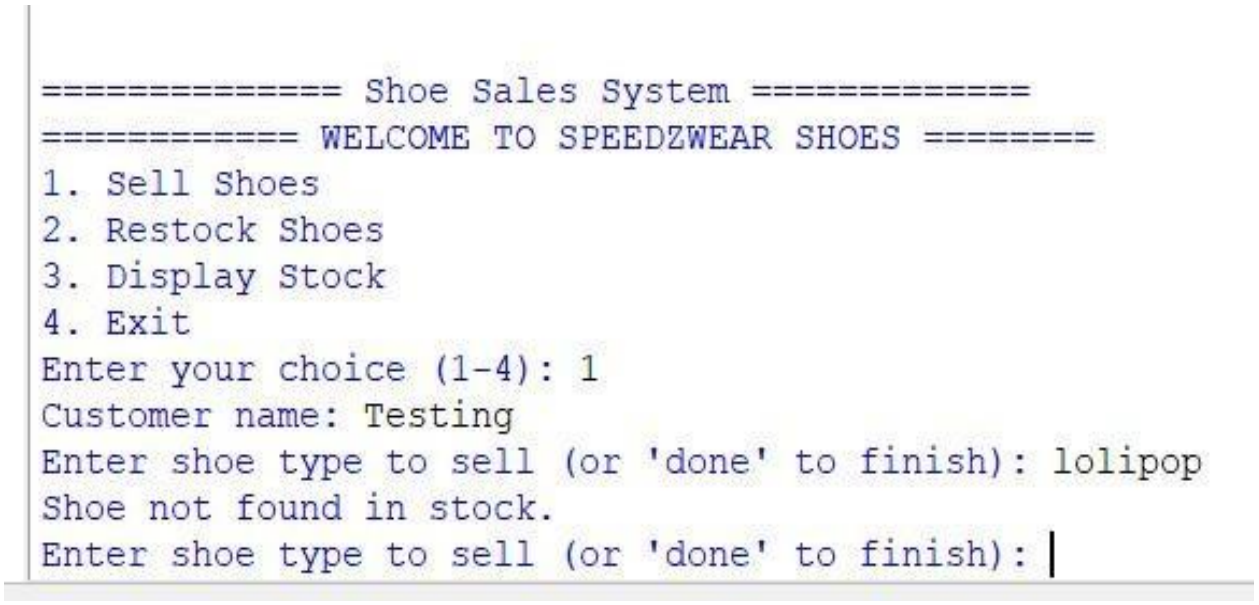
PROGRAM

This program is specially designed to manage products, handle purchases and sales, generate invoices, and terminate gracefully while managing and selling the inventory of the Speedz shoes. It maintains a list of products with details such as type, brand, price, and stock, allowing users to view the current inventory at any time. Users can restock or update inventory to date. For sales, the program enables selling multiple products in a single transaction, validating stock availability, and calculating the total price for each item. After a sale, a text-based invoice is created containing the customer name, product details, quantities, prices, and total. This invoice is also displayed in the terminal or shell. The program operates a menu system where users can choose to display stock, purchase, sell, restock, and exit, with the exit option terminating the program gracefully.

TESTING

Objective	To make sure or handle invalid inputs without crashing
Action	Enter an invalid quantity purchase or restock. Eg-1, -20
Expected Result	It detects the invalid input and display invalid messages without crashing.
Actual Result	Display “Invalid input or quantity”
Conclusion	This program handles invalid quantity using try-except and preventing crashes and ensuring data integrity.

Test 1: Try/Except implementation

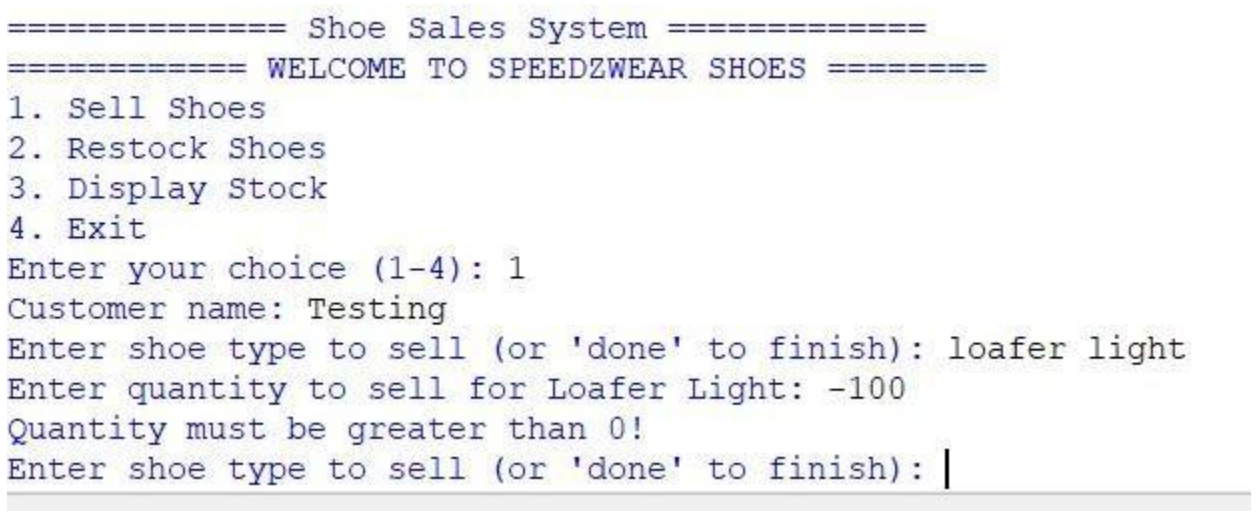


```
============== Shoe Sales System =============
============ WELCOME TO SPEEDZWEAR SHOES ========
1. Sell Shoes
2. Restock Shoes
3. Display Stock
4. Exit
Enter your choice (1-4): 1
Customer name: Testing
Enter shoe type to sell (or 'done' to finish): lolipop
Shoe not found in stock.
Enter shoe type to sell (or 'done' to finish):
```

Figure 15 Handles Invalid input

Test 2: Selection of purchase and sale

Objective	To verify the program validates the negative and invalid inputs during purchase and sale.
Action	Enter negative number when purchasing or selling that doesn't exist in the inventory.
Expected Result	Handles Invalid inputs and quantities displaying appropriate error messages.
Actual Result	Display appropriate messages to handle crashes and no changes are made to the inventory.
Conclusion	Correct input validation for purchases and sale operation is functioning correctly.

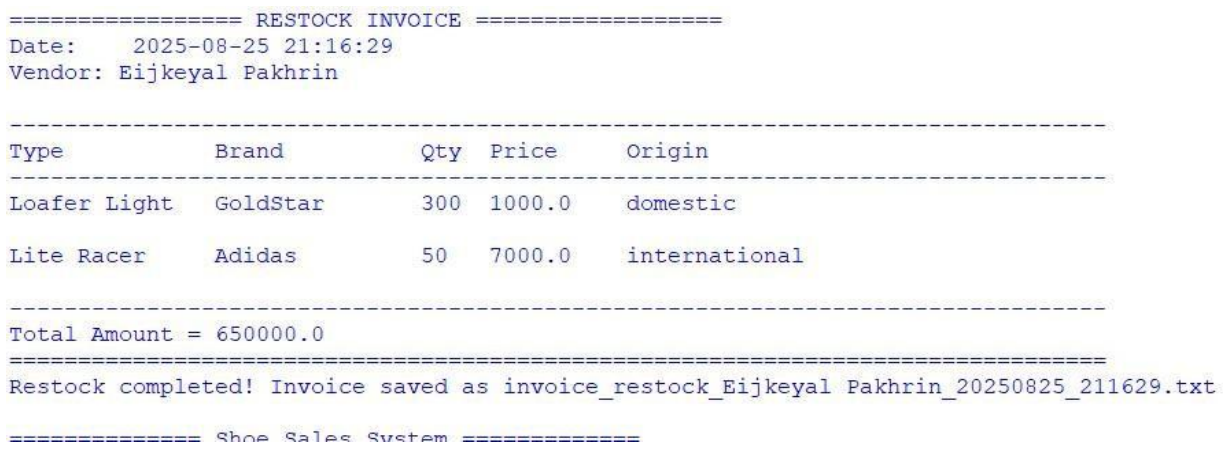


```
============== Shoe Sales System =============
============ WELCOME TO SPEEDZWEAR SHOES ========
1. Sell Shoes
2. Restock Shoes
3. Display Stock
4. Exit
Enter your choice (1-4): 1
Customer name: Testing
Enter shoe type to sell (or 'done' to finish): loafer light
Enter quantity to sell for Loafer Light: -100
Quantity must be greater than 0!
Enter shoe type to sell (or 'done' to finish):
```

Figure 16 Invalid details

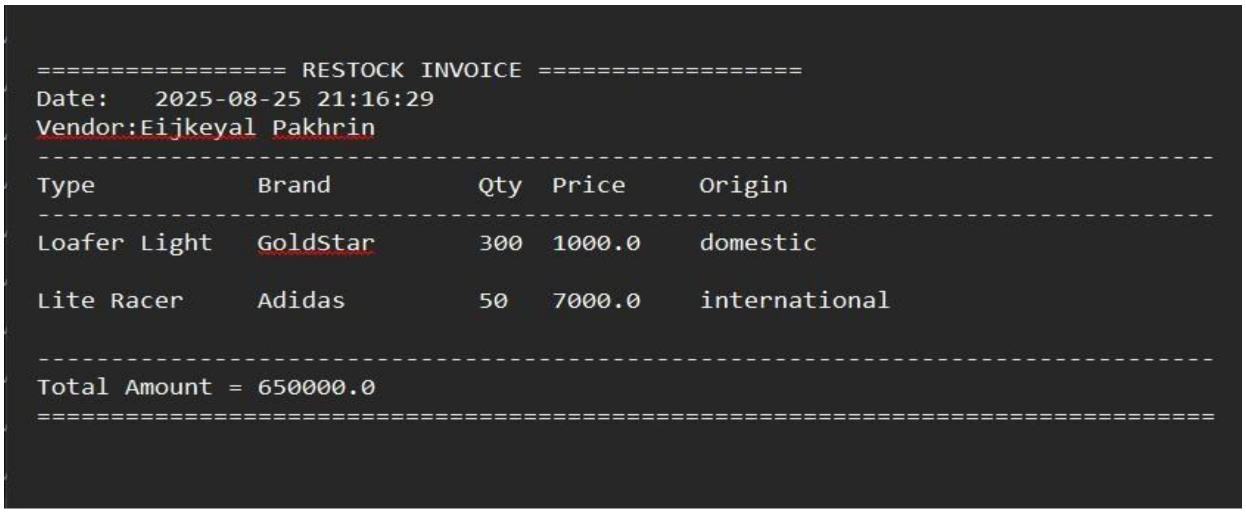
Test 3: File Generation for Purchase

Objective	To ensure that purchasing multiple products updates inventory and also generates bill to correct file.
Action	After purchased of products it should add to inventory and display updated stock.
Expected Result	Purchased bill should be displayed on terminal as well as text file with updated stock.
Actual Result	The file is created correctly, and the shell and txt file shows the purchase details.
Conclusion	Restock invoice generation and stock update for purchase operation are working properly.



```
================= RESTOCK INVOICE ==================
Date:    2025-08-25 21:16:29
Vendor: Eijkeyal Pakhrin

--------------------------------------------------------------------------------
Type           Brand          Qty  Price     Origin
--------------------------------------------------------------------------------
Loafer Light   GoldStar       300  1000.0    domestic

Lite Racer     Adidas         50   7000.0    international

--------------------------------------------------------------------------------
Total Amount = 650000.0
================================================================================
Restock completed! Invoice saved as invoice_restock_Eijkeyal Pakhrin_20250825_211629.txt

============== Shoe Sales System =============
```

Figure 17 Display restock bill

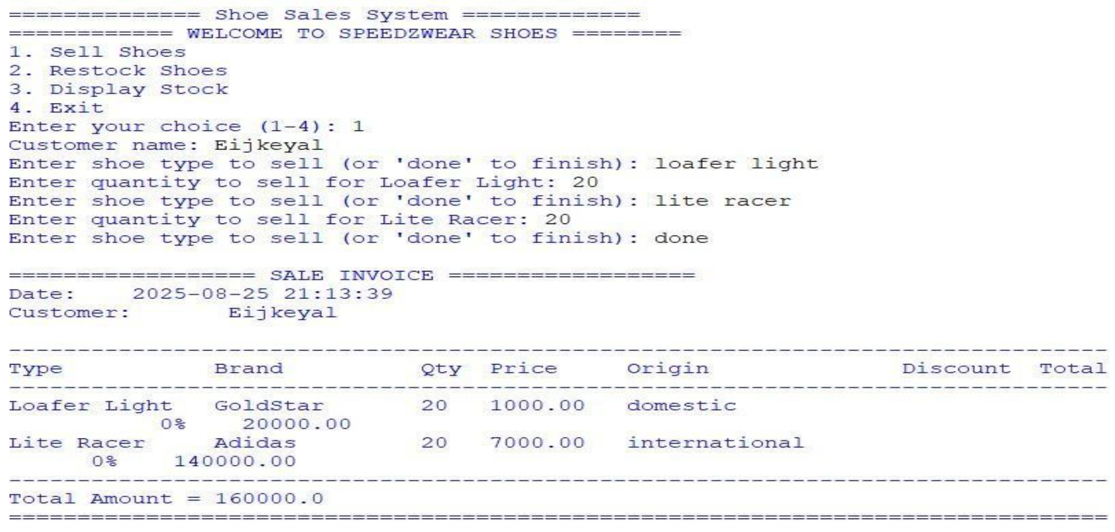


```
================= RESTOCK INVOICE ==================
Date:    2025-08-25 21:16:29
Vendor:Eijkeyal Pakhrin
--------------------------------------------------------------------------------
Type           Brand          Qty  Price     Origin
--------------------------------------------------------------------------------
Loafer Light   GoldStar       300  1000.0    domestic

Lite Racer     Adidas         50   7000.0    international

--------------------------------------------------------------------------------
Total Amount = 650000.0
================================================================================
```

Figure 18 Generate restock Bills to the file

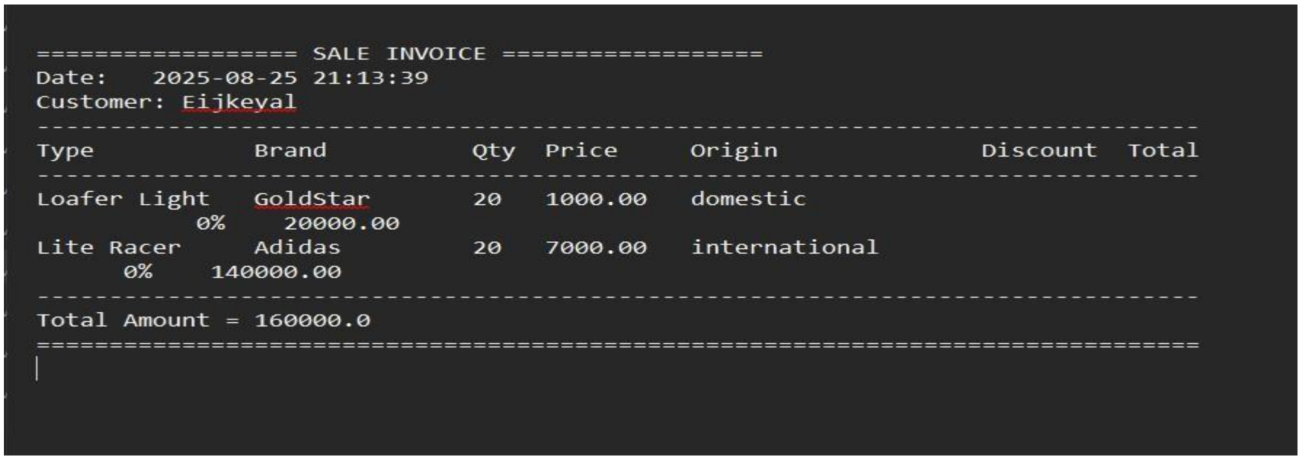
Test 4: File Generation for sale

Objective	It make sure that selling multiple products updates inventory and generates a correct sale invoice bill.
Action	Sales bill should be displayed in the shell as well as text files.
Expected Result	Sold product details with discounted amount should be displayed in the shell and a .txt file with customer name, products details, qty, price and total amount.
Actual Result	The file generated successfully, and the shell shows the sale details.
Conclusion	Sale process and invoice generations perform correctly.



```
============== Shoe Sales System =============
============ WELCOME TO SPEEDZWEAR SHOES ========
1. Sell Shoes
2. Restock Shoes
3. Display Stock
4. Exit
Enter your choice (1-4): 1
Customer name: Eijkeyal
Enter shoe type to sell (or 'done' to finish): loafer light
Enter quantity to sell for Loafer Light: 20
Enter shoe type to sell (or 'done' to finish): lite racer
Enter quantity to sell for Lite Racer: 20
Enter shoe type to sell (or 'done' to finish): done

================== SALE INVOICE ==================
Date:     2025-08-25 21:13:39
Customer:        Eijkeyal

--------------------------------------------------------------------------------
Type           Brand          Qty  Price     Origin              Discount  Total
--------------------------------------------------------------------------------
Loafer Light   GoldStar       20   1000.00   domestic
           0%    20000.00
Lite Racer     Adidas         20   7000.00   international
      0%    140000.00
--------------------------------------------------------------------------------
Total Amount = 160000.0
================================================================================
```

Figure 19 Displaying sales bills

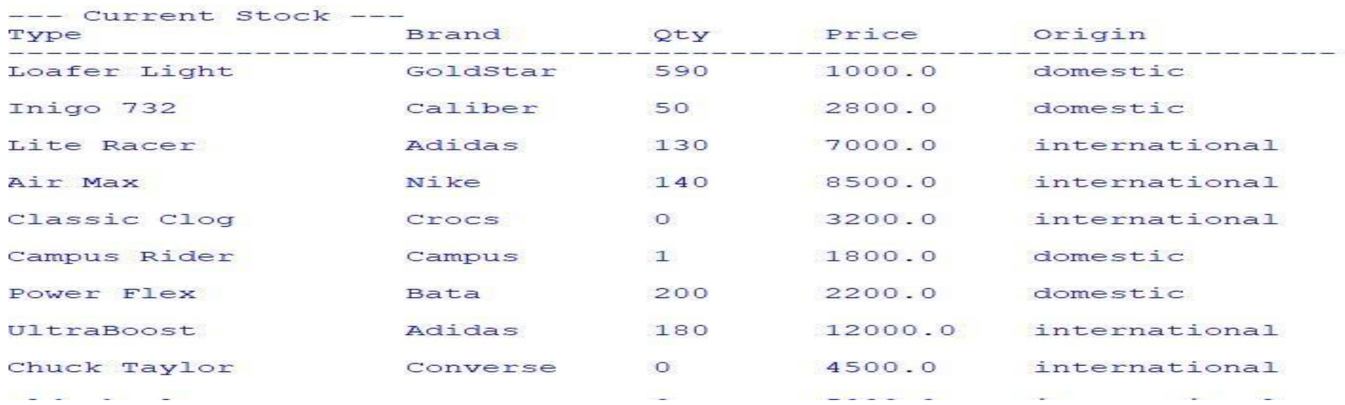


```
================== SALE INVOICE ==================
Date:   2025-08-25 21:13:39
Customer: Eijkeyal
--------------------------------------------------------------------------------
Type           Brand          Qty  Price     Origin              Discount  Total
--------------------------------------------------------------------------------
Loafer Light   GoldStar       20   1000.00   domestic
           0%    20000.00
Lite Racer     Adidas         20   7000.00   international
      0%    140000.00
--------------------------------------------------------------------------------
Total Amount = 160000.0
================================================================================
```

Figure 20 Generate sells bills to the file

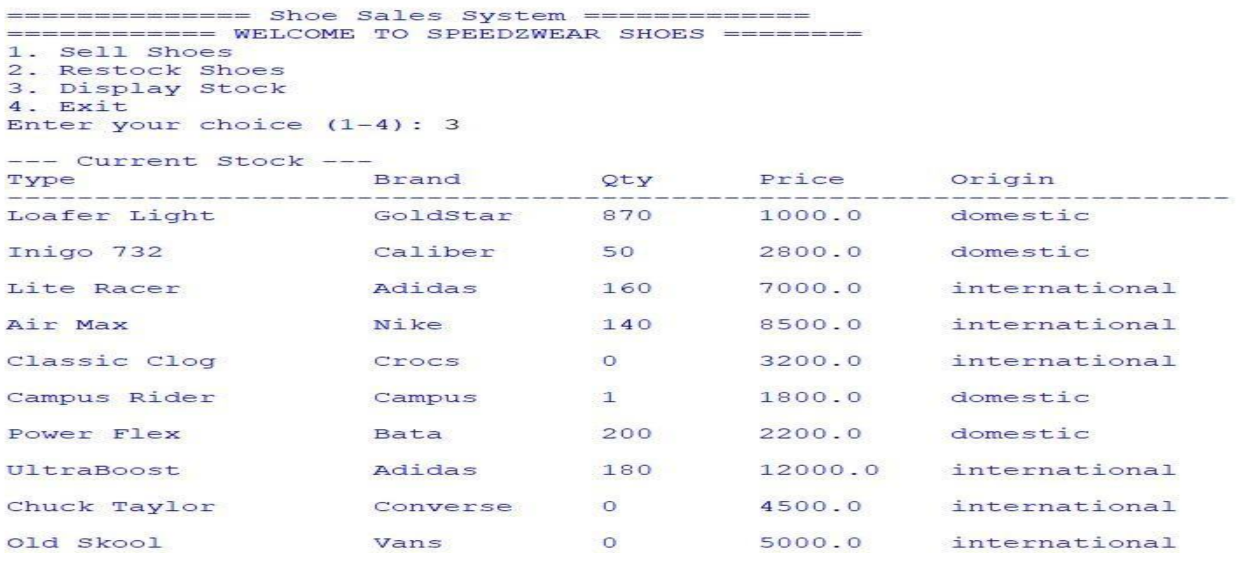
Test 5: stock update verification

Objective	To verify stock updates correctly after each and every transactions like sales or purchased.
Action	If purchase products and confirms that stock increses and sell the product confirms that stock decreases and shoed in terminal and a .txt files.
Expected Result	Inventory must be updated each and every transactions of buy and sells and generate invoice bills.
Actual Result	Stock changes appear correctly in the program and the files.
Conclusion	Inventory update mechanism is working correctly for both purchase and sale operations.



```
--- Current Stock ---
Type                    Brand         Qty      Price      Origin
-------------------------------------------------------------------------------
Loafer Light            GoldStar      590      1000.0     domestic
Inigo 732               Caliber       50       2800.0     domestic
Lite Racer              Adidas        130      7000.0     international
Air Max                 Nike          140      8500.0     international
Classic Clog            Crocs         0        3200.0     international
Campus Rider            Campus        1        1800.0     domestic
Power Flex              Bata          200      2200.0     domestic
UltraBoost              Adidas        180      12000.0    international
Chuck Taylor            Converse      0        4500.0     international
```

Figure 21 Curent stock before update



```
============== Shoe Sales System =============
============ WELCOME TO SPEEDZWEAR SHOES ========
1. Sell Shoes
2. Restock Shoes
3. Display Stock
4. Exit
Enter your choice (1-4): 3

--- Current Stock ---
Type                    Brand         Qty      Price      Origin
--------------------------------------------------------------------------
Loafer Light            GoldStar      870      1000.0     domestic
Inigo 732               Caliber       50       2800.0     domestic
Lite Racer              Adidas        160      7000.0     international
Air Max                 Nike          140      8500.0     international
Classic Clog            Crocs         0        3200.0     international
Campus Rider            Campus        1        1800.0     domestic
Power Flex              Bata          200      2200.0     domestic
UltraBoost              Adidas        180      12000.0    international
Chuck Taylor            Converse      0        4500.0     international
Old Skool               Vans          0        5000.0     international
```

Figure 22 After updated stock

CONCLUSION

Overall, to conclude the inventory and billing system functions as intended, effectively managing product stock, handling purchases and sales and generating accurate invoices in text files. It ensures data integrity by validating user inputs and using exception handling to prevent crashes. This system updates stock correctly after transaction, reflects changes both-on-screen and in text files, and provides a clear user friendly interface through menu driven workflow. Overall, this system demonstrates reliable performance, proper error handling, and complete traceability of transactions, making it suitable for small-scale inventory and sales management.

References

JGraph. (n.d.). draw.io – Online Diagramming Tool. diagrams.net. (n.d.). Draw.io. Retrieved August 25, 2025, from https://www.diagrams.net/

Microsoft. (n.d.). Microsoft Word. Microsoft. (n.d.). MS-WORD. Retrieved August 25, August, from https://www.microsoft.com/en-us/microsoft-365/word

Microsoft. (n.d.). Notepad. Microsoft. (n.d.). Notepad. Retrieved August 25, 2025, from https://apps.microsoft.com/detail/notepad/9msmlrh6lzf3

Python Software Foundation(n.d.). IDLE (Integrated Development and Learning Environment). Python.org. (n.d.). Python. Retrieved August 25, 2025, from https://docs.python.org/3/library/idle.html

APPENDIX

```
#Read_file



def read_shoes_file(filename):

   shoes = []      try:         with

open(filename, "r") as file:

for line in file:              if line ==

"\n":

continue
```

```
                parts = line.split("\t")

if

len(parts) != 5:

                   print("Skipping invalid line:", line)

continue



               shoe_type, brand, qty, price, origin = parts
                try:

                   shoes.append({

                       "type": shoe_type,

                       "brand": brand,

                       "quantity": int(qty),

                       "price": float(price),

                       "origin": origin

                   })

               except ValueError:

                   print("Skipping invalid line:", line)

except FileNotFoundError:

       print("File not found:", filename)



    return shoes
#write_file from datetime import datetime def print_sale_invoice(filename,

items, customer):

    print("\n================== SALE INVOICE ==================")

print("Date:\t", datetime.now().strftime("%Y-%m-%d %H:%M:%S"))     print("Customer:\t"
+ customer + "\n")     print("-" * 80)

print(f"{'Type':<15}{'Brand':<15}{'Qty':<5}{'Price':<10}"

f"{'Origin':<20}{'Discount':<10}{'Total':<10}")     print("-" * 80)


   total_amount = 0     for

item in items:

      total_amount += item['total']

print(f"{item['type']:<15}{item['brand']:<15}{item['quantity']:<5}"

f"{item['price']:<10.2f}{item['origin']:<20}"             f"{item['discount']}%{'':<4}{item['total']:<10.2f}")


   print("-" * 80)     print("Total Amount

=", total_amount)     print("=" * 80)


   # Save invoice to file      with
open(filename, "w") as f:

      f.write("\n================== SALE INVOICE ==================\n")

      f.write("Date:\t" + datetime.now().strftime("%Y-%m-%d %H:%M:%S") + "\n")

      f.write("Customer: " + customer + "\n")

      f.write("-" * 80 + "\n")

      f.write(f"{'Type':<15}{'Brand':<15}{'Qty':<5}{'Price':<10}"

f"{'Origin':<20}{'Discount':<10}{'Total':<10}\n")

      f.write("-" * 80 + "\n")


      for item in items:
            f.write(f"{item['type']:<15}{item['brand']:<15}{item['quantity']:<5}"

f"{item['price']:<10.2f}{item['origin']:<20}"

f"{item['discount']}%{'':<4}{item['total']:<10.2f}\n")


        f.write("-" * 80 + "\n")

        f.write("Total Amount = " + str(total_amount) + "\n")

        f.write("=" * 80 + "\n")




def print_restock_invoice(vendor, items, filename):

    print("================= RESTOCK INVOICE ==================")

print("Date:\t", datetime.now().strftime("%Y-%m-%d %H:%M:%S")) print("Vendor:\t" +

vendor + "\n")      print("-"*80)

print(f"{'Type':<15}{'Brand':<15}{'Qty':<5}{'Price':<10}{'Origin':<20}")     print("-"*80)


    total_amount = 0     for

item in items:

        line_total = item['quantity'] * item['price']         total_amount

+= line_total


print(f"{item['type']:<15}{item['brand']:<15}{item['quantity']:<5}{item['price']:<10}{item['origi
n']:<20}")


    print("-"*80)     print("Total Amount =", total_amount)

print("="*80)
```

```
   # Save invoice to file     with open(filename,
"w") as f:
        f.write("\n================== RESTOCK INVOICE ===================\n")
        f.write("Date:\t" + datetime.now().strftime("%Y-%m-%d %H:%M:%S") + "\n")
        f.write("Vendor:" + vendor + "\n")
        f.write("-"*80 + "\n")
        f.write(f"{'Type':<15}{'Brand':<15}{'Qty':<5}{'Price':<10}{'Origin':<20}\n")
f.write(""*80 + "\n")          for item in items:
            line_total = item['quantity'] * item['price']

f.write(f"{item['type']:<15}{item['brand']:<15}{item['quantity']:<5}{item['price']:<10}{item['ori
gin']:<20}\n")
        f.write("-"*80 + "\n")
        f.write("Total Amount = " + str(total_amount) + "\n")
        f.write("="*80 + "\n")
"


# ---------------- UPDATE STOCK FILE ----------------
def write_stock_file(filename, shoes):     with
open(filename, "w") as f:         for shoe in shoes:

f.write(f"{shoe['type']}\t{shoe['brand']}\t{shoe['quantity']}\t{shoe['price']}\t{shoe['origin']}\n")
#operation_file


from datetime import datetime from write_file import print_sale_invoice,
print_restock_invoice, write_stock_file
```

```
def find_shoe(shoes, shoe_type):     """Find shoe
in stock list"""     for shoe in shoes:         if
shoe['type'].lower() == shoe_type.lower():
            return shoe
return None


def process_sale(shoes):
    """Handle selling multiple shoes with discount and proper validation."""


    while True:
        customer = input("Customer name: ")         if
customer == "" or customer == " ":
print("Error: Customer name must be
valid.")

        else:
            break


    sold_items = []

    while True:
        shoe_type = input("Enter shoe type to sell (or 'done' to finish): ")         if
shoe_type.lower() == "done":
```

```
            break


        # Find shoe          shoe = find_shoe(shoes,
shoe_type)
        if not shoe:             print("Shoe not found
in stock.")             continue


        # Validate quantity
        try:               qty = int(input(f"Enter quantity to sell for
{shoe['type']}: "))             if qty <= 0:                 print("Quantity
must be greater than 0!")                 continue           except
ValueError:
            print("Quantity must be a number.")             continue


        if qty > shoe['quantity']:
            print(f"Not enough stock. Available quantity: {shoe['quantity']}")             continue

        # Apply discount logic          discount = 0
if qty >= 10:             if shoe['origin'].lower() ==
"international":                  discount = 0.05  #
5% discount         elif shoe['origin'].lower() ==
"domestic":              discount = 0.07  # 7%
discount
```

```
         total = qty * shoe['price']

discounted_total = total * (1 - discount)



        sold_items.append({

            "type": shoe['type'],

            "brand": shoe['brand'],

            "quantity": qty,
            "price": shoe['price'],

            "origin": shoe['origin'],

            "discount": discount * 100,

            "total": discounted_total

        })



        # Update stock         shoe['quantity']

-= qty



    if sold_items:
        invoice_file = "sale_invoice.txt"  # or generate dynamically if you want

print_sale_invoice(invoice_file, sold_items, customer)        write_stock_file("stock.txt",

shoes)



def process_restock(shoes):

    """Handle restocking multiple shoes"""

    try:
```

```
    vendor = input("Vendor name: ")

restocked_items = []        item_ids = set()



    while True:

        shoe_type = input("Shoe type to restock (or 'done' to finish): ")
if shoe_type.lower() == "done":

            break



        shoe = find_shoe(shoes, shoe_type)

        if shoe:               try:

                qty = int(input(f"Enter quantity to add for {shoe['type']}: "))

if qty <= 0:                          print("Quantity must be greater than 0!")

continue


                shoe['quantity'] += qty

item_id = len(item_ids) + 1

item_ids.add(item_id)



                restocked_items.append({

                    "id": item_id,

                    "type": shoe['type'],

                    "brand": shoe['brand'],

                    "quantity": qty,

                    "price": shoe['price'],

                    "origin": shoe['origin']
```

```
          })                    print(f"Restocked {qty}
{shoe['type']}.")

except ValueError:

          print("Invalid input! Please enter a number.")           else:

        print("Shoe not found in stock!")


    if    restocked_items:                                          filename        =

f"invoice_restock_{vendor}_{datetime.now().strftime('%Y%m%d_%H%M%S')}.txt

"

print_restock_invoice(vendor,                        restocked_items,                        filename)

write_stock_file("shoes.txt", shoes)             print("Restock completed! Invoice saved as",

filename)

  except Exception as e:

    print("Error in process_restock:", e)


  return shoes




#main_file from operation import process_sale,

process_restock from read_file import

read_shoes_file  from write_file import

write_stock_file
```

```
def display_stock(shoes):     if
not shoes:
        print("\nNo shoes in stock.\n")         return     print("\n--- Current Stock ---")     print("{:<20}
{:<12} {:<8} {:<10} {:<12}".format("Type", "Brand", "Qty", "Price", "Origin"))     print("-"*70)
for shoe in shoes:
        print("{:<20} {:<12} {:<8} {:<10} {:<12}".format(             shoe["type"], shoe["brand"],
shoe["quantity"], shoe["price"], shoe["origin"]
        ))
    print("-"*70 + "\n")


def main():
    # Load stock from file     shoes = read_shoes_file("shoes.txt")


    # Display stock once at program start     display_stock(shoes)


    while  True:                   print("\n==============  Shoe  Sales  System
=============")                        print("============   WELCOME   TO
SPEEDZWEAR SHOES ========")
        print("1. Sell Shoes")
print("2. Restock Shoes")
print("3. Display Stock")         print("4.
Exit")


        choice = input("Enter your choice (1-4): ")
```

```
      if choice == "1":
         process_sale(shoes)
write_stock_file("shoes.txt", shoes)       elif choice
== "2":
         process_restock(shoes)
write_stock_file("shoes.txt", shoes)          elif
choice == "3":
         # Refresh stock from file and display
shoes = read_shoes_file("shoes.txt")
display_stock(shoes)          input("Press Enter to
return to the main menu...")       elif choice == "4":
         print("Exiting... Goodbye!")          break
      else:
         print("Invalid choice! Please try again.") main()
```