A logo with blue and grey text

Description automatically generated

**Module Code & Module Title:**

**CS4051NT Fundamentals of Computing**

**Assessment Weightage & Type:**

**60% Individual Coursework**

**Year and Semester:**

**2024 Spring**

**Student Name: Abhishek Sharma**

**London Met ID: 23056498**

**College ID: NP05CP4S240023**

**Assignment Due Date: August 18, 2024**

**Word Count:**

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

Table of Contents

[1 Introduction 6](#_Toc174111566)

[1.1 Introduction to Python 6](#_Toc174111567)

[1.2 Features of Python 7](#_Toc174111568)

[1.3 Introduction to IDLE 8](#_Toc174111569)

[1.4 Introduction to coursework 9](#_Toc174111570)

[1.5 Goals and Objectives 9](#_Toc174111571)

[2 Algorithm 10](#_Toc174111572)

[2.1 Algorithm of overall code 11](#_Toc174111573)

[2.2 Detailed Explanation of each Functions 13](#_Toc174111574)

[2.2.1 main() function 13](#_Toc174111575)

[2.2.2 read\_furniture\_data() function 13](#_Toc174111576)

[2.2.3 read\_furniture\_products() function 13](#_Toc174111577)

[2.2.4 read\_inventory() function 13](#_Toc174111578)

[2.2.5 write\_inventory(inventory) function 14](#_Toc174111579)

[2.2.6 order\_Invoice(furniture\_ids,manufacturers,product\_names,quantities,transactions,price\_per\_units,item\_totals,subtotal,shipping\_cost,grand\_total,filename,vat\_amount) function 14](#_Toc174111580)

[2.2.7 sell\_invoice(filename,customer\_name,transactions,subtotal,vat\_amount,shipping\_cost,grand\_total) function 14](#_Toc174111581)

[2.2.8 update\_inventory(furniture\_id, quantity\_change) function 15](#_Toc174111582)

[2.2.9 add\_new\_product(furniture\_id, manufacturer, product\_name, quantity, price) function 15](#_Toc174111583)

[2.2.10 order\_furniture(furniture\_id, quantity, employee\_name) function 15](#_Toc174111584)

[2.2.11 generate\_invoice(transactions) function 16](#_Toc174111585)

[2.2.12 sell\_furniture(customer\_name, furniture\_id, quantity) function 16](#_Toc174111586)

[3 Pseudocode 17](#_Toc174111587)

[3.1 Pseudocode for main.py code 17](#_Toc174111588)

[3.2 Pseudocode for read.py 19](#_Toc174111589)

[3.3 Pseudocode for write.py 21](#_Toc174111590)

[3.4 Pseudocode for operation.py 25](#_Toc174111591)

[3.5 Pseudocode for submain.py 29](#_Toc174111592)

[4 Flowchart 33](#_Toc174111593)

[5 Data Structures 34](#_Toc174111594)

[6 Program 35](#_Toc174111595)

[7 Testing 35](#_Toc174111596)

[8 Conclusion 36](#_Toc174111597)

[9 References 37](#_Toc174111598)

[10 Appendix 38](#_Toc174111599)

**Table of Figures:**

[Figure 1: Python Logo 6](#_Toc173080570)

[Figure 2: IDLE Logo 8](#_Toc173080571)

[Figure 3: Pictorial example of Algorithm 10](#_Toc173080572)

**Table of Tables:**

# Introduction

The "Fundamentals of Computing" module (CS4051NT), led by Mr. Ajayraj Bhattrai, focuses on Python programming, and covers essential computing concepts including, algorithms, flowcharts, and software development practices. This semester-long course assesses our performance through coursework, which accounts for 60% of our grade, and an in-course test, making up the remaining 40%. Designed to build our foundational knowledge and enhance critical thinking skills, the module combines lectures, tutorials, workshops, practical assignments, and coursework to provide a comprehensive learning experience in computing.

## Introduction to Python

Python is an admired general purpose; high level language with multifunctional uses and is acknowledged for its ease of use. Standardized by Guido van Rossum in 1991 and consistently developed by the Python Software Foundation, Python was designed and fine-tuned to be as readable and understandable, with simple and elegant syntactic structure, which would let the programmers express any idea in a clean, concise manner. Another significant aspect of Python is that it enables the quick development cycle where the members deliver fast, and it is amenable to integration. Over time, Python has evolved into two major versions: Two main groups of the programming language: Python 2 and Python 3 which differ in the features they provide, the capabilities they offer and the paradigms they support. However, both versions are still in use in multiple spheres ranging from web development, scientific computing, AI, and automation. (Geeks For Geeks, 2024)

A blue and yellow snake logo

Description automatically generated

Figure 1: Python Logo

## Features of Python

**Free and Open Source**

Python is an open-source language that can be downloaded from the official Python website and can be downloaded by just clicking on the Download Python link. Since it is an open source, the code is easily accessible to everyone for making modifications. It is free from legal protection and can be downloaded, used, and shared with anyone and everyone.

**Easy to Code**

Python is a great programming language that is relatively easier to learn than languages such as C, C#, JavaScript or Java. It will take you a few hours or days to learn how to program in Python if you are a beginner. It is intended for new users as well as advanced developers in the field and network engineering.

**Easy to Read**

Python is easy to learn since its structures are quite basic and uncluttered. Thus, the code structure in Python is very clear, and instead of semicolons or brackets, the code blocks are identified by indentations.

**Object-Oriented Language**

Python also supports Object-Oriented Programming which comprises of the notions of classes and object encapsulation. Due to this, it is an ideal option for creating intricate applications and programs.

**GUI Programming Support**

There are GUI modules available for Python like PyQt5 and PyQt4, wxPython, Tkinter and so on. In the present scenario, GUI frameworks are widely used, and out of all, PyQt5 is in great demand for developing graphical applications. And many more. (Geeks for geeks, 2024)

## Introduction to IDLE

IDLE, Interactive Development and Learning Environment for Python runs on Windows Unix as well as MacOS. It has a feature of a Python shell for entering code and the prompt for input, output, and exceptional conditions are in the colour. There are nice additional options that you can see in the text editor such as undo, highlighter for Python code and tips when writing functions. You can find specific text to replace it with or in files with a particular name. Like many programming languages, IDLE also provides the debugger to track down the errors in your code and allows you to step through your code to see how the program runs. that is convenient to work with adding that it has settings and tools included within. (Python Software Foundation, 2024)

A close-up of a paper

Description automatically generated

Figure 2: IDLE Logo

## Introduction to coursework

This coursework for the "Fundamentals of Computing" module (CS4051NT) focuses on developing a Python program to manage inventory and transactions for a furniture store. The program will read, and update information stored in a text file, displaying available furniture, and generating transaction notes for both purchases from manufacturers and sales to customers. It will incorporate features like error handling, input validation, and the calculation of VAT for sales transactions. The coursework emphasizes modular programming, using appropriate data structures for efficient data management, and includes documentation, algorithm development, and testing to ensure functionality and reliability.

## Goals and Objectives

The objectives of our coursework for the "Fundamentals of Computing" module are designed to achieve comprehensive understanding and practical skills in Python programming and software development. Our goals include:

* **Develop a Functional Inventory Management System:** Create a Python program to manage furniture inventory for a store, allowing updates for sales and orders directly impacting stock levels.
* **Implement Modular Programming:** Structure the program into separate functions for managing inventory, processing transactions, and generating invoices to ensure code reusability and clarity.
* **Utilize Effective Data Structures:** Employ appropriate data structures like dictionaries and lists in Python to efficiently store and manipulate furniture data, ensuring optimal performance for inventory operations.
* **Document and Present:** Provide clear documentation including algorithm descriptions, pseudocode, and flowcharts to illustrate program logic and functionality. Include a testing section to validate program behavior under different scenarios.

These objectives focus on developing practical Python programming skills while addressing the specific requirements of managing furniture inventory and transaction processing as outlined in our coursework.

# Algorithm

An algorithm is a set of directions used to accomplish a certain objective or solve a problem. While algorithms are mostly performed by computers, people also employ them in their daily lives. In real life, we are following an algorithm each time we carry out a certain consecutive operation, from tying the shoelaces to making coffee in the morning, for instance.

In other word it simply refers to a step-by-step procedure that can be used to solve a problem in computer science. Algorithms are used in search engines, music streaming, navigation systems, and many other systems and applications that are present in computers. (Nikolopoulou, 2024)

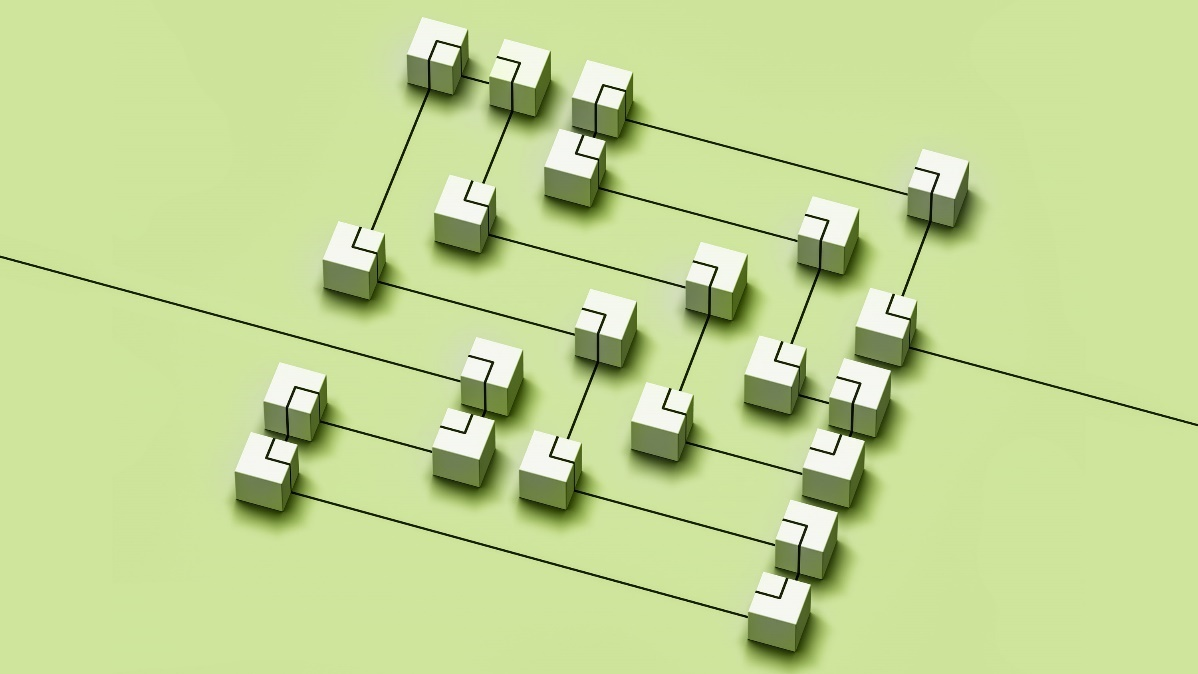


Figure 3: Pictorial example of Algorithm

## Algorithm of overall code

**Step 1: Start**

**Step 2: Display the Main Menu**

Print options:

Display available furniture

Purchase furniture from manufacturer

Sell furniture to customer

Exit

**Step 3: Prompt User for a Choice**

Read the user's choice.

**Step 4: Handle User Choice**

**If Choice is 1: Display Available Furniture**

Call read\_furniture\_data() to read and display furniture data from "inventory.txt".

**If Choice is 2: Purchase Furniture from Manufacturer**

Call for\_choice\_2() function.

**Inside for\_choice\_2():**

Prompt for employee's name.

Validate Name: Ensure the name does not contain digits.

**Enter a loop for ordering:**

Display current inventory by calling read\_furniture\_products().

Prompt for furniture ID and quantity to order.

Validate Furniture ID: Ensure it is in digit form.

Validate Quantity: Ensure it is a positive number.

Call order\_furniture() to process the order.

Ask if the user wants to enter another transaction. If "no", exit the loop.

Call generate\_invoice() to create an invoice for the orders.

**If Choice is 3: Sell Furniture to Customer**

Call for\_choice\_3() function.

**Inside for\_choice\_3():**

Prompt for customer’s name.

Validate Name: Ensure the name does not contain digits.

Read the inventory by calling read\_inventory().

**Enter a loop for selling:**

Display current inventory by calling read\_furniture\_products().

Prompt for furniture ID and quantity to sell.

Validate Quantity: Ensure it is a positive number and check inventory and Add the sale transaction to the list.

Ask if the user wants to sell another product. If "no", exit the loop.

Call sell\_furniture() to process the sale.

* **If Choice is 4: Exit**

Print a thank you message.

Exit the program.

* **Else: Invalid Choice**

Print an error message and prompt the user to try again.

**Step 4: End**

## Detailed Explanation of each Functions

A Python function is organized, self-contained programming construct intended for the performance of a particular operation. Function help in improving the factor of modularity as well as ensure extensive use of codes. In Python, a function is called by passing required data, or values into it using a parameter or argument. (Tutorials Point, 2024)

### main() function

The main() function provides a menu-driven interface for managing the BRJ Furniture Store. It runs in a loop, displaying options to view furniture, purchase from a manufacturer, sell to customers, or exit. Based on user input, it calls the appropriate function to handle each task. The loop continues until the user chooses to exit, at which point a thank you message is displayed, and the program terminates. This function ensures the menu remains active and responsive to user choices throughout the session.

### read\_furniture\_data() function

The read\_furniture\_data() function reads data from a text file named "inventory.txt" and displays it in a tabular format. The file is expected to contain comma-separated values for ID, Manufacturer, Product Name, Quantity, and Price. The function opens the file, reads its lines, and splits each line into individual values. It then formats and prints this data in a well-organized table. If the file is not found, it catches the FileNotFoundError exception and prints an error message indicating that the file does not exist.

### read\_furniture\_products() function

The read\_furniture\_products() function reads data from a file named "inventory.txt" and displays it in a formatted table. The file is expected to contain comma-separated values representing ID, Manufacturer, Product Name, Quantity, and Price. The function opens the file, reads the lines, and processes each line into a list of values. It then prints a header and the relevant columns (ID, Product Name, Quantity, and Price) in a neatly formatted table. If the file is missing, it catches the FileNotFoundError and displays an error message indicating the file's absence.

### read\_inventory() function

The read\_inventory() function reads data from a file named "inventory.txt" and returns it as a list of lists. Each line in the file is split into columns based on commas, with each inner list representing one line of the file. If the file is not found, it catches the FileNotFoundError exception and prints an error message. This function is useful for processing inventory data stored in a text file and making it accessible for further operations.

### write\_inventory(inventory) function

The write\_inventory function writes the updated inventory data back to inventory.txt. It takes a list of lists representing the inventory as its input parameter. The function opens inventory.txt in write mode and iterates over each item in the inventory. Each item is converted to a comma-separated string and written to the file. This function ensures that the inventory file is updated with the latest data.

### order\_Invoice(furniture\_ids,manufacturers,product\_names,quantities,transactions,price\_per\_units,item\_totals,subtotal,shipping\_cost,grand\_total,filename,vat\_amount) function

The order\_Invoice() function generates and writes an order invoice to a specified file. It takes various details about the order as input arguments, including furniture IDs, manufacturers, product names, quantities, and costs. The function writes these details to the file in a formatted manner, including the subtotal, VAT amount, shipping cost, and grand total. The invoice also includes the date, time, and the employee's name who processed the order. The function uses Python's file handling to create and write the invoice content, and it prints a confirmation message once the invoice is generated.

### sell\_invoice(filename,customer\_name,transactions,subtotal,vat\_amount,shipping\_cost,grand\_total) function

The sell\_invoice() function creates and writes a sales invoice to a specified file. It includes various details related to the sale, such as the customer's name, furniture IDs, brands, product names, quantities sold, prices per unit, and total costs for each item. It also calculates and displays the subtotal, VAT amount, shipping cost, and grand total. The function organizes and formats this information into a readable invoice and writes it to the file. After completing the write operation, it prints a confirmation message indicating the invoice file's name.

### update\_inventory(furniture\_id, quantity\_change) function

The update\_inventory function updates the quantity of a specific product in the inventory. It takes the product ID and the change in quantity (positive for adding stock, negative for reducing stock) as input parameters. The function reads the current inventory using read\_inventory(). It iterates over the inventory to find the item with the given product ID and updates its quantity accordingly. The updated inventory is then written back to inventory.txt using the write\_inventory() function.

### add\_new\_product(furniture\_id, manufacturer, product\_name, quantity, price) function

The add\_new\_product function adds a new product to the inventory. It takes the product ID, manufacturer, product name, quantity, and price as input parameters. The function first reads the current inventory using read\_inventory(). It then appends the new product details to the inventory list. Finally, the function writes the updated inventory back to inventory.txt using the write\_inventory() function.

### order\_furniture(furniture\_id, quantity, employee\_name) function

The order\_furniture() function handles the ordering of furniture items by updating the inventory and recording transaction details. It accepts a furniture ID, quantity, employee name, and a list of transactions.

First, the function checks if the item is in the inventory using read\_inventory(). If found, it updates the quantity with update\_inventory(), calculates the total cost, and adds the transaction details to the transactions list.

If the item is not found, it prompts the user to enter details for the new product and adds it to the inventory with add\_new\_product(). The function then recursively calls itself to process

### generate\_invoice(transactions) function

The generate\_invoice() function creates an invoice by processing transaction details, including furniture IDs, manufacturers, product names, and costs. It calculates shipping costs based on the distance to the manufacturer, computes the subtotal, VAT, and grand total. It then generates a filename using the employee's name and current timestamp, and saves the invoice to a file using the order\_Invoice() function..

### sell\_furniture(customer\_name, furniture\_id, quantity) function

The sell\_furniture() function handles furniture sales by verifying transaction details against the inventory. It calculates the subtotal, VAT, and shipping cost based on customer location. Valid transactions are processed and logged, while invalid ones are noted. The function updates the inventory, generates a filename based on the customer’s name and current date-time, and creates a sale invoice using the sell\_invoice() function.

# Pseudocode

Pseudocode is a full and easily comprehensible description of how exactly a certain program, or an algorithm should work. The script is articulated in more formal vocabulary but chosen deliberately to use the programming language’s natural structure and style, so that it can be more easily read and understood by programmers and other people involved in the project. Pseudocode is not a programming language, and one cannot compile the pseudocode to create an executable software. It only provides a skeleton for converting the logical code into a real programming language. (Sheldon, 2024)

## Pseudocode for main.py code

IMPORT read\_furniture\_data FROM read

IMPORT for\_choice\_2, for\_choice\_3 FROM submain

FUNCTION main()

WHILE True

OUTPUT "\n==================================================="

OUTPUT " Welcome to BRJ Furniture Store Management System"

OUTPUT "===================================================="

OUTPUT "\n1. Display available furniture"

OUTPUT "2. Purchase furniture from manufacturer"

OUTPUT "3. Sell furniture to customer"

OUTPUT "4. Exit"

INPUT choice

IF choice == "1" THEN

CALL read\_furniture\_data()

ELSE IF choice == "2" THEN

CALL for\_choice\_2()

ELSE IF choice == "3" THEN

CALL for\_choice\_3()

ELSE IF choice == "4" THEN

OUTPUT "\n---------------Exiting-----------------\n"

OUTPUT "\n-----------------------------------------"

OUTPUT "Thank you for choosing BRJ Furniture."

OUTPUT "-----------------------------------------\n"

BREAK

ELSE

OUTPUT "\nInvalid choice! Please try again."

END WHILE

END FUNCTION main

IF \_\_name\_\_ == '\_\_main\_\_':

CALL main()

## Pseudocode for read.py

FUNCTION read\_furniture\_data()

TRY

OPEN file "inventory.txt" FOR READING

READ all lines from the file into variable lines

SPLIT each line by comma and strip whitespace, store in variable data

OUTPUT header line: "\nID | Manufacturers | Product Name | Quantity | Price"

OUTPUT separator line: "-" \* 78

FOR each row in data

OUTPUT formatted row

OUTPUT separator line: "-" \* 78

END FOR

EXCEPT FileNotFoundError

OUTPUT "The file 'inventory.txt' does not exist."

END TRY

END FUNCTION

FUNCTION read\_furniture\_products()

TRY

OPEN file "inventory.txt" FOR READING

READ all lines from the file into variable lines

SPLIT each line by comma and space, store in variable data

OUTPUT header line: "ID | Product Name | Quantity | Price"

OUTPUT separator line: "-" \* 35

FOR each row in data

SET furniture\_id to row[0]

SET product\_name to row[2]

SET quantity to row[3]

SET price to row[4]

OUTPUT formatted row

END FOR

EXCEPT FileNotFoundError

OUTPUT "The file 'inventory.txt' does not exist."

END TRY

END FUNCTION

FUNCTION read\_inventory()

TRY

OPEN file "inventory.txt" FOR READING

READ all lines from the file into variable lines

RETURN lines split by comma and space, stripped of whitespace

EXCEPT FileNotFoundError

OUTPUT "The file 'inventory.txt' does not exist."

END TRY

END FUNCTION

## Pseudocode for write.py

IMPORT read\_inventory from read

IMPORT datatime

FUNCTION write\_inventory(inventory)

OPEN "inventory.txt" FOR WRITING AS file

FOR each item IN inventory

WRITE item joined by commas and followed by newline to file

END FOR

END FUNCTION

FUNCTION add\_new\_product(furniture\_id, manufacturer, product\_name, quantity, price)

CALL read\_inventory() TO GET inventory

APPEND [furniture\_id, manufacturer, product\_name, str(quantity), price] TO inventory

CALL write\_inventory(inventory)

END FUNCTION

FUNCTION order\_Invoice(furniture\_ids, manufacturers, product\_names, quantities, transactions, price\_per\_units, item\_totals, subtotal, shipping\_cost, grand\_total, filename, vat\_amount)

OPEN filename FOR WRITING AS file

WRITE invoice header and details to file

FOR i FROM 0 TO LENGTH(furniture\_ids) - 1

WRITE "Total Cost for Item {i+1} (ID {furniture\_ids[i]}): {item\_totals[i]}" TO file

END FOR

WRITE subtotal, VAT amount, shipping cost, and grand total to file

CLOSE file

OUTPUT "Invoice generated: {filename}"

END FUNCTION

FUNCTION read\_furniture\_data()

TRY

OPEN "inventory.txt" FOR READING AS file

READ all lines from file into variable lines

SPLIT each line by comma and strip whitespace, store in variable data

OUTPUT header line: "ID | Manufacturers | Product Name | Quantity | Price"

OUTPUT separator line: "-" \* 78

FOR each row in data

OUTPUT formatted row

OUTPUT separator line: "-" \* 78

END FOR

EXCEPT FileNotFoundError

OUTPUT "The file 'inventory.txt' does not exist."

END TRY

END FUNCTION

FUNCTION read\_furniture\_products()

TRY

OPEN "inventory.txt" FOR READING AS file

READ all lines from file into variable lines

SPLIT each line by comma and space, store in variable data

OUTPUT header line: "ID | Product Name | Quantity | Price"

OUTPUT separator line: "-" \* 35

FOR each row in data

SET furniture\_id to row[0]

SET product\_name to row[2]

SET quantity to row[3]

SET price to row[4]

OUTPUT formatted row

END FOR

EXCEPT FileNotFoundError

OUTPUT "The file 'inventory.txt' does not exist."

END TRY

END FUNCTION

FUNCTION read\_inventory()

TRY

OPEN "inventory.txt" FOR READING AS file

READ all lines from file into variable lines

RETURN list of lists (each line split by comma and space, stripped of whitespace)

EXCEPT FileNotFoundError

OUTPUT "The file 'inventory.txt' does not exist."

END TRY

END FUNCTION

## Pseudocode for operation.py

IMPORT datetime

IMPORT write\_inventory, add\_new\_product from write

IMPORT read\_inventory from read

IMPORT order\_invoice and sell\_invoice from write

FUNCTION update\_inventory(furniture\_id, quantity\_change)

CALL read\_inventory() TO GET inventory

FOR each item IN inventory

IF item[0] == furniture\_id

SET current\_quantity TO int(item[3])

SET new\_quantity TO current\_quantity + quantity\_change

SET item[3] TO str(new\_quantity)

BREAK

END IF

END FOR

CALL write\_inventory(inventory)

END FUNCTION

FUNCTION order\_furniture(furniture\_id, quantity, employee\_name, transactions)

CALL read\_inventory() TO GET product\_list

SET product\_found TO False

FOR each item IN product\_list

IF item[0] == furniture\_id

SET product\_found TO True

CALL update\_inventory(furniture\_id, quantity)

SET price\_per\_unit TO float(item[4].replace('$', ''))

SET total\_cost TO quantity \* price\_per\_unit

APPEND transaction DETAILS TO transactions

BREAK

END IF

END FOR

IF NOT product\_found

OUTPUT "This product is not in inventory. Adding the item to inventory..."

PROMPT user FOR manufacturer, product name, and price

CALL add\_new\_product(furniture\_id, manufacturer, product\_name, quantity, price)

CALL order\_furniture(furniture\_id, quantity, employee\_name, transactions)

END IF

END FUNCTION

FUNCTION generate\_invoice(transactions)

SET filename TO f'order\_invoice\_of\_{transactions[0]["employee\_name"]}\_{datetime.datetime.now().strftime("%Y%m%d%H%M")}.txt'

INITIALIZE lists and variables: furniture\_ids, manufacturers, product\_names, quantities, price\_per\_units, item\_totals, subtotal, vat\_amount

PROMPT user FOR distanceChecker

SET shipping\_cost BASED ON distanceChecker

FOR each transaction IN transactions

APPEND transaction DETAILS TO respective lists

UPDATE subtotal AND vat\_amount

END FOR

CALCULATE grand\_total

CALL order\_Invoice() WITH all collected details

END FUNCTION

FUNCTION sell\_furniture(customer\_name, transactions)

CALL read\_inventory() TO GET inventory

INITIALIZE variables: subtotal, vat\_amount, shipping\_cost, valid\_transactions, invalid\_transactions

FOR each transaction IN transactions

SET product\_found TO False

FOR each item IN inventory

IF item[0] == transaction["furniture\_id"]

SET product\_found TO True

CALCULATE total\_cost AND price\_per\_unit

CHECK inventory quantity AND UPDATE if valid

APPEND transaction TO valid\_transactions

BREAK

END IF

END FOR

IF NOT product\_found

APPEND error TO invalid\_transactions

END FOR

IF valid\_transactions EXISTS

PROMPT user FOR location

SET shipping\_cost BASED ON location

CALCULATE grand\_total

SET filename TO f'sale\_invoice\_of\_{customer\_name}\_{datetime.datetime.now().strftime("%Y%m%d%H%M")}.txt'

CALL sell\_invoice() WITH all collected details

END IF

END FUNCTION

## Pseudocode for submain.py

FUNCTION for\_choice\_2()

SET transactions TO empty list

SET employee\_name TO user input "Enter employee name: "

IF employee\_name IS digit

PRINT "Name should not contain digits."

ELSE

WHILE True

TRY

PRINT "Item already available in furniture's inventory are here: "

CALL read\_furniture\_products()

PRINT "If you want to buy products that are not in inventory, enter the next ID."

SET furniture\_id TO user input "Enter furniture ID to order: "

IF furniture\_id IS NOT digit

PRINT "Furniture ID should be in digit form."

ELSE

SET quantity TO user input "Enter quantity to order: " AS integer

IF quantity IS greater than 0

CALL order\_furniture(furniture\_id, quantity, employee\_name, transactions)

SET order\_more TO user input "Do you want to enter another transaction? (yes/no): " AS lowercase string

IF order\_more IS NOT "yes"

BREAK

END IF

ELSE

PRINT "Enter a valid Quantity (Quantity should be in positive number)."

END IF

EXCEPT ValueError

PRINT "Please, provide the information carefully."

END TRY

END WHILE

CALL generate\_invoice(transactions)

END IF

END FUNCTION

FUNCTION for\_choice\_3()

SET transactions TO empty list

SET customer\_name TO user input "Enter customer name: "

IF customer\_name IS digit

PRINT "Name should not contain digits."

ELSE

SET inventory TO CALL read\_inventory()

WHILE True

TRY

CALL read\_furniture\_products()

SET furniture\_id TO user input "Enter furniture ID to sell: "

SET quantity TO user input "Enter quantity to sell: " AS integer

IF quantity IS greater than 0

SET item\_available TO False

FOR each item IN inventory

IF item[0] == furniture\_id

SET item\_available TO True

IF quantity IS greater than item[3]

PRINT "Please enter the valid quantity."

BREAK

END IF

END IF

END FOR

IF item\_available

APPEND {

"furniture\_id": furniture\_id,

"quantity": quantity

} TO transactions

ELSE

PRINT "Product ID {furniture\_id} is not available in inventory."

END IF

SET sell\_more TO user input "Do you want to sell another product? (yes/no): " AS lowercase string

IF sell\_more IS NOT "yes"

BREAK

END IF

ELSE

PRINT "Enter a valid Quantity (Quantity should be in positive number)."

END IF

EXCEPT ValueError

PRINT "Please, provide the information carefully."

END TRY

END WHILE

IF transactions EXISTS

CALL sell\_furniture(customer\_name, transactions)

END IF

END IF

END FUNCTION

# Flowchart

# Data Structures

In developing the furniture inventory management system, several data structures in Python are utilized to efficiently manage data storage, input/output operations, and transaction processing:

**Lists**: Used to store the list of furniture items read from and written to the text file. Lists allow for sequential access and modification of data entries, making them suitable for managing dynamic inventory updates and transaction histories.

**Dictionaries**: Employed to map unique identifiers (IDs) of furniture items to detailed information such as manufacturer, product name, available quantity, and price per unit. This facilitates quick look up and retrieval of specific item details during transaction processing and inventory updates.

**Strings**: Utilized for processing and formatting textual data, such as parsing input from administrators or formatting output for transaction receipts. String manipulation functions are employed to ensure data integrity and clarity in generated invoices and transaction notes.

**File Handling**: Python's built-in file handling capabilities are employed for input/output operations, allowing the program to read furniture data from and write updated data back to the text file. This ensures persistent storage and retrieval of inventory information between program sessions.

**Custom Class**: For more complex applications, custom-defined classes can encapsulate furniture item attributes and methods, enhancing code modularity and scalability. Classes can represent furniture items as objects with properties and behaviors tailored to specific application requirements.

Each data structure is chosen based on its suitability for handling various aspects of the inventory management system, from efficient data retrieval and manipulation to ensuring robust file handling and user interaction. (Akash, 2024)

# Program

# Testing

# Conclusion

In conclusion, the construction of the BRJ Furniture Store Inventory Management System has therefore been a worthy exercise in learning Python programming and data processing. It was somewhat difficult but also very rewarding to produce a functional approach to organizing stock, take payments, and issue receipts.  
  
I maintain clean and organized code that is written in the standard manner of software development. I was able to solve problems related to organizing and processing the data of inventory with the assistance of lists and dictionaries. Since the program is capable of reading from and writing text files, data is secured and retrievable for regular use.

I also learned skills in the design of algorithms, error management and file management from this project. I also got to know about the documentation of my work and it’s testing as well for it to be confirmed that it works effectively.  
  
During planning, coding, and debugging stages I obtained the experience in such aspects as team cooperation, critical thinking, and practical use of the learned material.  
  
Therefore, the BRJ Furniture Store Inventory Management System proves my commitment and IT competency. I am glad that I accomplished such things, and I think it will be useful in the store. General, this project has prepared me for future software development related tasks, and I will use the skills acquired in my future learning and working endeavors.

# References

Akash. (2024, July 17). *data-structures-in-python*. Retrieved from edureka: https://www.edureka.co/blog/data-structures-in-python/

Geeks For Geeks. (2024, July 16). *introduction-to-python*. Retrieved from Geeks For Geeks: https://www.geeksforgeeks.org/introduction-to-python/

Geeks for geeks. (2024, July 17). *python-features*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/python-features/

Nikolopoulou, K. (2024, July 28). *what-is-an-algorithm*. Retrieved from scribbr: https://www.scribbr.com/ai-tools/what-is-an-algorithm/

Python Software Foundation. (2024, July 16). *idle.html*. Retrieved from docs.Python: https://docs.python.org/3/library/idle.html

Sheldon, R. (2024, July 28). *pseudocode*. Retrieved from techtarget: https://www.techtarget.com/whatis/definition/pseudocode

Tutorials Point. (2024, August 9). *python\_functions.htm*. Retrieved from Tutorials Point: https://www.tutorialspoint.com/python/python\_functions.htm

# Appendix