**Optimizing Stock Handling: A Data Structures Approach in Python**

**In Partial Fulfillment of the Requirement in**

**CS 131 - Data Structure And Algorithm**

**By:**

**Collantes, Harvy A.**

**Rudio, Khing Patrick**

**Melandro V. FLoro**

**April 2025**

### College of Informatics and Computing Sciences

1. **Optimizing Stock Handling: A Data Structures Approach in Python**
2. **Project Overview:**

In this final project, students will work in groups of three to design and develop an algorithmic model using Python programming. The model should utilize various data structures (e.g., lists, sets, tuples, dictionaries, linked lists, stacks and queues, hash tables, and trees) and algorithmic techniques (e.g., searching and sorting). Students are expected to apply critical thinking, problem-solving, and logical structuring to develop a working Python module that demonstrates real-world functionality. Pseudocode and flowcharts are to be used during the planning stage.

1. **Project Requirements**
   1. **Group Formation**:
      1. Team Leader (Name): Collantes, Harvy A.
      2. Team Member 1(Name): Rudio, Khing Patrick
   2. **Project Title and Description:**

**“Optimizing Stock Handling: A Data Structures Approach in Python”**

The goal of this project, "Optimizing Stock Handling: A Data Structures Approach in Python," is to create an inventory system that is simple but effective and uses the First-In, First-Out (FIFO) method of stock management. To guarantee that the oldest stock is sold or removed first, the system is built using basic Python data structures, particularly lists and queue operations.

The program allows users to perform out essential inventory functions like processing sales, adding new stock, and viewing current inventory through a command-line interface. Since object-oriented programming is not used in the implementation of the FIFO logic, it is lightweight and simple to understand, especially for beginners. By maintaining an organized, automated, and consistent inventory flow, the main goal is to maximize stock movement. It improves overall stock accuracy and reduces waste from unused or expired goods. The system only focuses on effective data manipulation using Python's built-in abilities, avoiding the need for external APIs, barcode scanners, or graphical user interfaces.

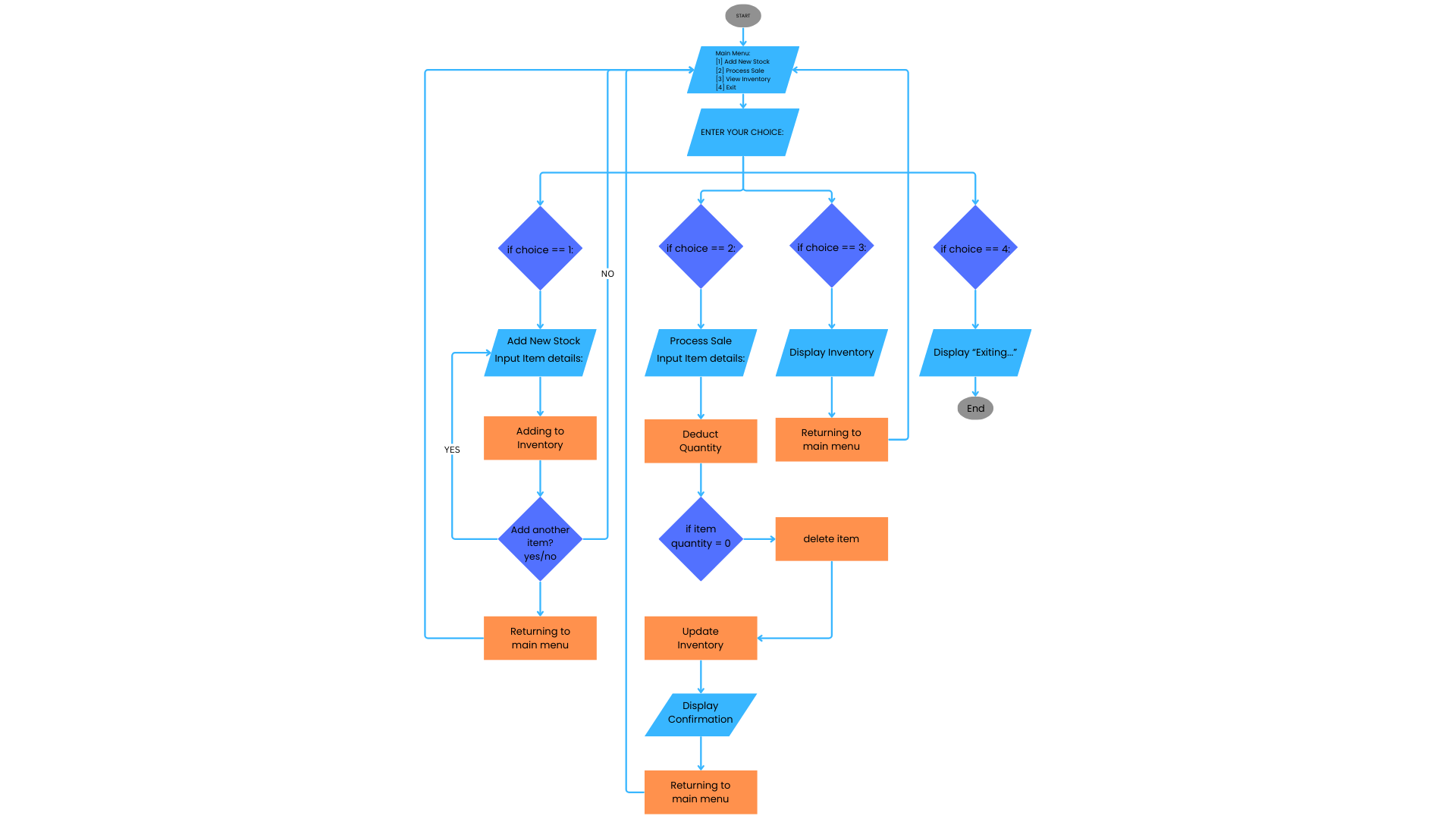
Essentially, the project's goal is to show how a functional and optimized inventory system that is appropriate for small-scale or academic settings can be achieved through the smart use of fundamental programming tools, in this case data structures.

* 1. **Deliverables**:
     1. **Python File**:

Python was used to create the main program, which is flexible in structure and has different functions for each of the inventory system's main features. To keep things simple and accessible for beginners. The following modules are part of the system:

* main\_menu(): Manages the program's main navigation and asks the user to select one of the available options.
* add\_new\_stock(): function lets the user enter the item name, quantity, and purchase date. The datetime module is used to validate the input and make sure it is formatted correctly.
* process\_sale(): Manages the sales process by determining whether the item is available and whether the quantity is adequate. It then uses a sorted list (FIFO logic) to deduct inventory based on the earliest purchase dates.
* view\_inventory(): Provides a graphical representation of all inventory entries, including item numbers, names, quantities, and dates of purchase.

All stock data is stored by the program in a global list called inventory, where each entry is a dictionary with the keys "name," "quantity," and "date." After every sale, the inventory list is cleaned to eliminate items with zero quantity. The logic makes sure that sales remove items beginning with the oldest stock entries. The code is easy to run in academic or lightweight deployment settings because it is made to run in a command-line interface (CLI) environment.

* + 1. **Documentation**:

**Flowchart:**

*This flowchart visually represents the logical flow of the inventory system, illustrating how the program handles stock input, sales processing, and inventory display based on user selections.*

Here’s the link of the flowchart: <https://www.canva.com/design/DAGjo3-FeBU/fuv9uEfU4VWrwiSVY-vlRA/edit?utm_content=DAGjo3-FeBU&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton>

**Pseudocode:**

START

FUNCTION main\_menu

LOOP infinitely

DISPLAY "---INVENTORY SYSTEM---"

DISPLAY menu options:

[1] Add New Stock

[2] Process Sale

[3] View Inventory

[4] Exit

PROMPT user for choice

IF choice is "1"

CALL add\_new\_stock

ELSE IF choice is "2"

CALL process\_sale

ELSE IF choice is "3"

CALL view\_inventory

ELSE IF choice is "4"

DISPLAY "Exiting..."

BREAK the loop

ELSE

DISPLAY "Invalid selection"

FUNCTION add\_new\_stock

LOOP

PROMPT for item name

LOOP UNTIL valid quantity is entered

PROMPT for quantity

IF quantity is not a number

DISPLAY "Invalid input"

ELSE

CONVERT quantity to integer

BREAK loop

LOOP UNTIL valid date is entered

PROMPT for purchase date (YYYY-MM-DD)

IF date is not in valid format or contains invalid values

DISPLAY "Invalid date format"

ELSE

BREAK loop

ADD item (name, quantity, date) to inventory

LOOP UNTIL valid yes/no input is given

PROMPT "Add another item? (yes/no)"

IF input is "yes"

CONTINUE loop

ELSE IF input is "no"

DISPLAY "Stock added successfully!"

RETURN

ELSE

DISPLAY "Invalid input"

FUNCTION process\_sale

LOOP

PROMPT for item name

IF item not found in inventory

DISPLAY "Item not found"

CONTINUE loop

LOOP UNTIL valid quantity is entered

PROMPT for quantity to sell

IF not a number

DISPLAY "Invalid input"

ELSE

CONVERT to integer

BREAK

CALCULATE total available quantity for that item

IF available >= quantity to sell

SET remaining = quantity to sell

SORT inventory items manually by date (oldest first)

FOR each item in inventory

IF item name matches

IF item quantity <= remaining

SUBTRACT item quantity from remaining

SET item quantity to 0

ELSE

SUBTRACT remaining from item quantity

SET remaining to 0

IF remaining == 0

BREAK

REMOVE all items from inventory with quantity == 0

DISPLAY "Sale processed successfully!"

BREAK

ELSE

DISPLAY "Insufficient stock"

FUNCTION view\_inventory

DISPLAY "Inventory List"

IF inventory is empty

DISPLAY "No items in inventory"

RETURN

SORT inventory by purchase date in ascending order (oldest first)

DISPLAY table headers

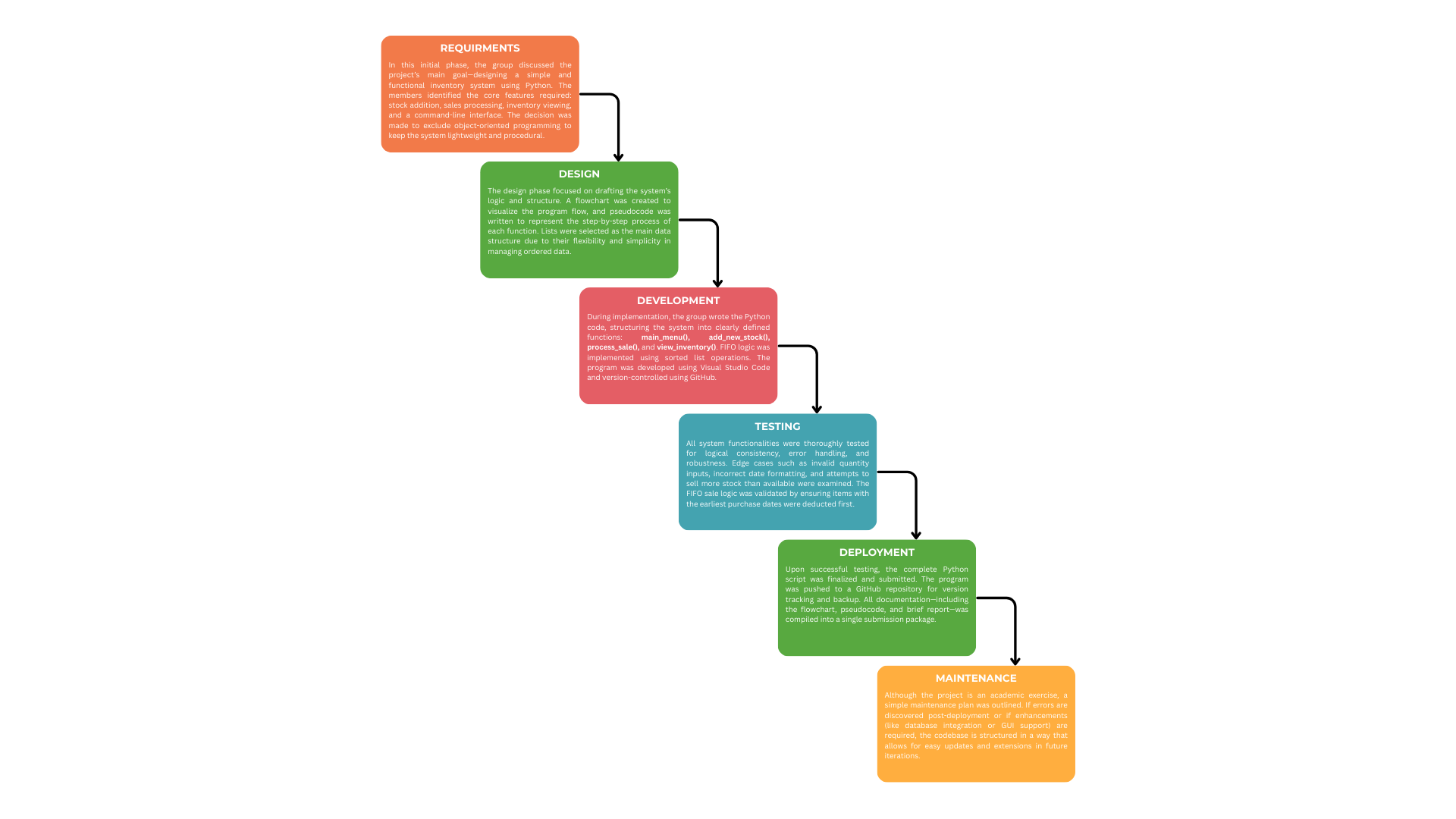
FOR each item in inventory with index

DISPLAY item number, name, quantity, and purchase date

CALL main\_menu

END

*This pseudocode provides a step-by-step representation of the system’s logic and algorithm, demonstrating how user inputs are handled and how core inventory operations like adding, selling, and viewing stock are executed.*

**Brief Report:**

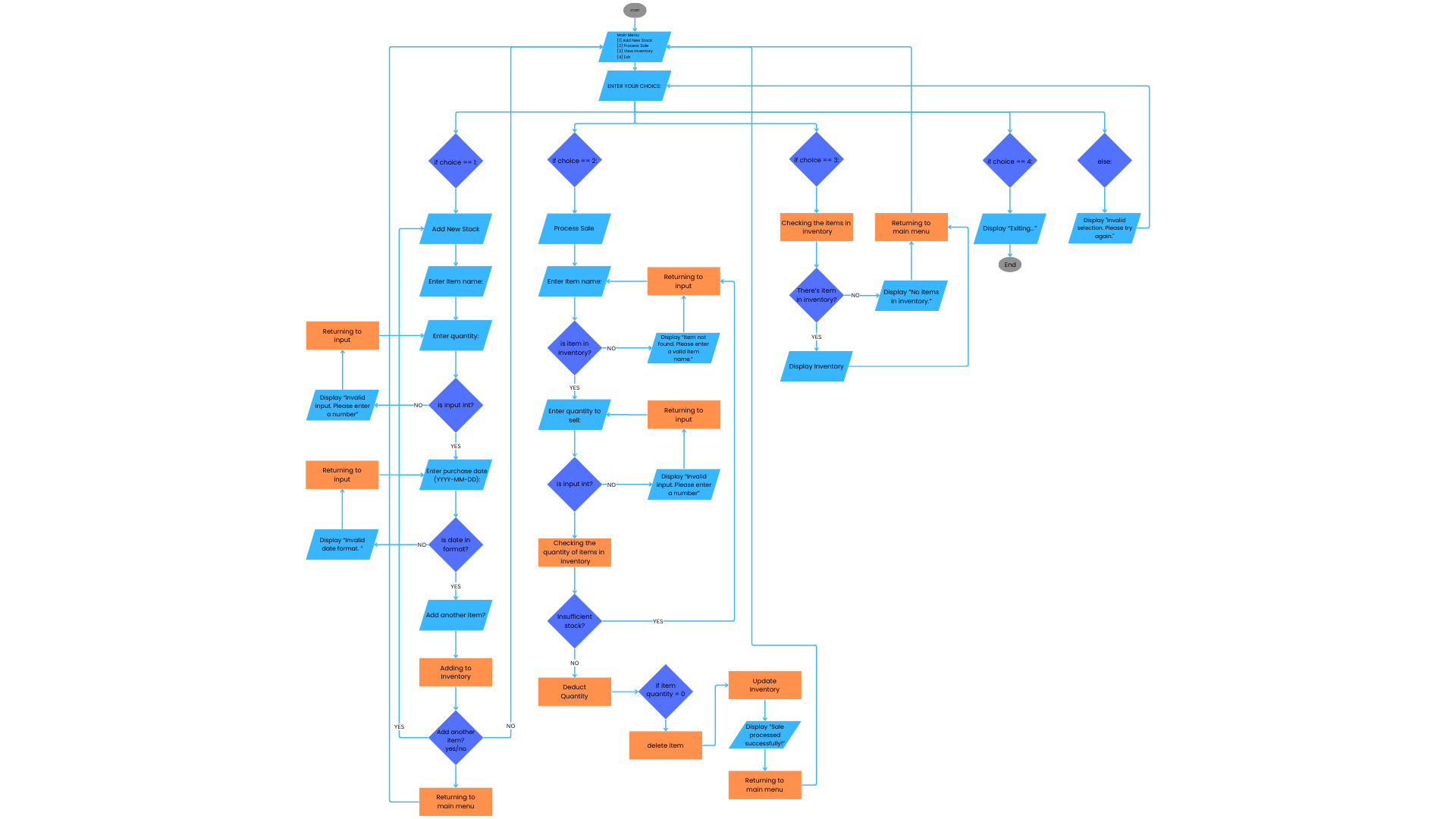
*This brief report summarizes the group’s development process using the Waterfall Model, detailing the steps from requirements gathering to maintenance, and highlighting the structured planning, function design, and task distribution involved in building the inventory system.*

Here's the link of the Waterfall Model: <https://www.canva.com/design/DAGlDnS4SaI/ALkosLaNfbaxuTaLrSBC3Q/edit?utm_content=DAGlDnS4SaI&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton>

1. **Project Phases** 
   1. **Planning and Design**:

This project's main goal is to solve the ineffective and prone to mistakes manual inventory tracking process, which can result in poor inventory control, increased waste, and wrong stock levels. Through an automated and structured process, the system is intended to help users manage stock transactions, including adding new inventory, processing sales, and viewing available items.

Because lists are easy to manipulate in Python and have built-in ordering, the group decided to use them as the main data structure for storing inventory data. Lists are perfect for keeping stock entries in chronological order because they make appending and sorting simple. While sorting functions were used to maintain the proper order during stock deductions, conditional logic and loops were used to control program flow and implement input validation.

* + - * Draft:

System Flow Diagram:

*This system flow diagram illustrates the inventory management process using optimized data structures in Python to handle stock addition, sales processing, and inventory checks efficiently.*

Here’s the link of the System Flow Diagram:

<https://www.canva.com/design/DAGlcHsNu1A/Z6Q1_DcVKoapFUTFAfebHw/edit?utm_content=DAGlcHsNu1A&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton>

* 1. **Development:**

1. Write Python code implementing the algorithm model.
2. Ensure use of:
   * Relevant data structures and algorithm techniques.
   * Modular code with functions, comments, and clear variable names.

c. **Testing and Review**:

1. Conduct thorough testing of each function/module.
2. Perform peer review within the group to ensure:
   * Code readability and logical consistency.
   * Efficient and correct use of data structures.
   * Accuracy of algorithm implementation.
3. **Grading Rubric** 
   1. **Logical Structure and Planning (25%):** 
      1. Clear, well-thought pseudocode and flowchart.

ii. Logical structuring of the module

* 1. **Implementation (50%):** 
     1. Proper use of data structures ii. Effective application of algorithms iii. Code readability and functionality
  2. **Creativity and Design (25%):** 
     1. Innovative application or solution **ii.** Well-structured and documented code

1. **Submission Details** 
   1. **Deadline**: May 09, 2025
   2. **Submission**: Submit a .zip file that contains the following:
      1. Python source files ii. Pdf/Doc format of flowchart, pseudocode, and report

Prepared by:

**Mr. MELANDRO FLORO**

Course Facilitator

Validated by:

**Mr. DIONECES O. ALIMOREN**

Program Chairperson

Approved by:

**Dr. RYNDEL V. AMORADO**

Dean, CICS