COS 125 – HMWK4 – Inventory Management

You can reference the textbook (in BrightSpace), class materials, and tutoring. Do not use AI. Include the comment header in all homework assignment files, specify any collaboration you had.

I. General Rules & Academic Integrity

- **Topics:** Practicing elements and concepts covered in class: Getting user input, data type conversion, manipulating lists/strings, implementing decision/control structures (if/else) and loops.
- **Concepts Used: Only use concepts presented in the course to-date.** Using advanced concepts or concepts from outside the course does not demonstrate understanding of the required material and may result in a zero for the assignment.
- **Resources:** You may reference the textbook, class materials, and tutoring.
- **Academic Integrity: Do not use AI.** Specify any collaboration you had within the comment heade, if you went to Boardman 138, state that.
- **Support:** Seek help if you get stuck. Boardman 138 has staff available to help.

II. Submission Requirements

- **File Name:** yourName_Inventory.py (replace "yourName" with your actual name, e.g., lauraGurney_ Inventory.py).
- **Comment Header:** Include the required comment header in your file and ensure data within the header is updated.
- **Folder Name:** Place the .py file into a folder named yourName_complexData.
- **Submission:** Keep python files inside your assignment folder. Compress the folder (.zip) and submit the compressed file.

Part 1: Initial Warehouse Stock Setup

This program focuses on creating and manipulating a dataset (the warehouse inventory) using nested lists and loops.

Required Data Structure:

Your program must use a single main nested list for inventory data, it will hold the entire company inventory, including which warehouse items are located within. This structure must adhere to the following hierarchy:

1. The main list holds elements representing:
2. Each Warehouse as named by user's input.
3. Within each Warehouse must contain:
    - A list of **Categories** within that warehouse. (e.g. Tools, Paint, etc.) as entered by the user.

4. Each Category element must contain:
    - A list of **Items** within that category as entered by the user and the initial number of items (stock count). . (e.g. Hammer, 50; Masks, 150, etc.)

Setup Instructions:

1. **Determine Structure Size:** Prompt the user to enter the **number of warehouses** for the organization.
2. **Nested Input Loops:** Use nested loops to iterate through the process of creating the inventory structure:
    - Loop through each warehouse, prompting the user for its **name**.
    - Inside the warehouse loop, prompt the user for the **number of categories** for that warehouse.
    - Loop through the categories, prompting for the **category name**.
    - Inside the category loop, prompt the user for the **number of items** in that category.
    - Loop through the items, prompting for the **item name** and the **initial stock count** (integer).
3. **Final Output:** Print the final, updated list with all user entries. Format the output to clearly display the hierarchy, similar to the example shown below:

```
The Total Inventory
Warehouse A:
    Tools: 6 Hammer, 150 Screwdriver
    Hardware: 5 Hinges, 17 Wall Anchors
Warehouse B:
    Paint: 8 Blue, 7 Yellow
```

Etc... (based on user entry)

Part 2: Search and Transaction

After the inventory is stocked, allow the user to search for an item and perform a transaction that modifies the inventory number.

1. **Get Search Input:** Prompt the user to enter the **name of the item** they want to search for.
2. **Initialize Variables:** Properly initialize or reset any variables used for tracking the search status for use in the searching loops.
3. **Implement Nested Search Logic:** Use nested loops to step through the entire data list structure (nested lists) to find a match. **DO NOT use the *break* statement** to exit loops early; use logic to manage the search process.
4. **If Found (Match):**
    - Print a success message showing the **Location (Warehouse)**, **Category**, and **Initial Stock**.
    - **Transaction Prompt:** Prompt the user to enter the **type of transaction** (e.g., 'add' or 'remove') and the **quantity** (integer).

- **Constraint Enforcement:** If the transaction type is 'remove', implement logic to ensure the new stock count **does not fall below zero**. If it would, print an error and refuse the transaction.
- **Update List:** If the transaction is valid, the code must **modify the stock count value** for the item within the main nested list structure.
- **Final Output:** Display a final message showing the **new, updated stock level** in the inventory.

5. **If Not Found (No Match):**
   - If the nested search completes and no match is found, print a clear error message.

```
Updated Total Inventory
Warehouse A:
    Tools: 10 Hammer, 10 Screwdriver
    Hardware: 5 Hinges, 1 Wall Anchors
Warehouse B:
    Paint: 0 Blue, 57 Yellow
```

Grading will be based on:
Implementing course content correctly.
Implementing logic, loops, and data types properly.

Criterion	Exceeds	Meets	Partially Meets	Does not Meet
Implements Course Content Correctly and Accurately.				
Interpretation	The code is concise, clear, and efficient. It uses insightful representations to model the problem. The student has a deep and thoughtful understanding of the project and problem requirements.	The student accurately converts the problem's requirements into code. All inputs are correctly interpreted, and the processes are successful and comprehensive.	The student attempts to model the problem but makes minor errors in their interpretation. For instance, they might correctly model the loop but make a small error.	The student's code fundamentally misunderstands the problem. They may attempt to use programmatic structures but fail to correctly represent the problem model.
Logical Structures	The program correctly uses logical decision structures (e.g., if/elif/else) to handle all	The program correctly uses if/elif/else to handle the core logic and ensure the program	The program attempts to use conditional logic but makes minor errors that may cause the program to fail or	The program fails to use conditional logic.

	possible scenarios, including edge cases. The use of these structures is efficient and well-placed.	doesn't crash on invalid input, if applicable.	produce incorrect logical outputs under certain conditions.	
Looping	The student implements looping structures correctly to complete required processes perfectly. Loop(s) terminate correctly and efficiently, and the variables are updated within and nest, if used is correctly implemented.	The student correctly implements loop(s), functions as intended, terminating when the initial cost is covered. Minor issues may be present.	Loop(s) not implemented or contain logical errors (e.g., infinite loop, incorrect termination condition) that prevent from functioning properly. Not implemented correctly to solve specified challenge.	The student fails to implement a while loop or uses an incorrect loop type and or structure.
Data Handling	All inputs are stored in self-documenting variables, and the program demonstrates robust data conversion and handling of program data and user inputs.	The program correctly prompts the user for all inputs, stores them in descriptive variables, and converts data types correctly for calculations. Handles program data properly is minimal errors.	The program prompts for some but not all inputs, or the variable names are not clear. There may be errors in data type conversion that led to incorrect output.	The program does not prompt for all inputs or fails to store them correctly. Data type conversion is either not attempted or is consistently incorrect.
Communication & Formatting	The program's output is highly effective and easy to read. It uses excellent	The program successfully adheres to all output specification	The program's output does not effectively meet the overall purpose of the program /	The program fails to provide adequately formatted output. It may use vague, hard to read, or the

	formatting (\t, \n) to display results of program clearly and concisely.	standards. The readability and user experience output could be more clearly formatted and effective.	problem. The output is presented without a clear explanation of what is meant, lacking formatting and clarity.	output is completely absent or incomprehensible.
File & Folder Organization	The file and folder naming and organization are professional, clear, and logical.	The file and folder naming and organization are correct.	The file or folder naming is slightly off, but the file is still recognizable.	The file and folder naming and organization are incorrect, making the file difficult to find or identify.
Header Comments Zero grade if not included	The student includes a comment header that is well-formatted and includes all of the required information along with a brief description of the program's functionality.		Comment header in present without required information and is significantly lacking relevant information.	A comment header is not included.