Project Report: Python Supermarket Billing System.

1. Cover Page

(Center the following information on your first page)

- **Project Title:** Automated Supermarket Billing System
- **Language:** Python
- **Submitted by:** [Your Name]
- **Date:** [Current Date]
- **Institution/Department:** [Your Institution Name]

2. Introduction

The retail sector relies heavily on speed and accuracy. This project involves the development of a console-based **Supermarket Billing System** using Python. The application is designed to streamline the checkout process by automating price calculation, generating a detailed itemized bill, and calculating Goods and Services Tax (GST). It replaces manual calculation methods, reducing human error, and saving time for both the cashier and the customer.

3. Problem Statement

Manual billing in grocery stores and supermarkets is prone to several issues:

- **Calculation Errors:** Human error in summing up prices or calculating taxes.
- **Time Consumption:** Writing bills by hand takes time, leading to long queues.
- **Record Keeping:** Difficulty in maintaining immediate records of the current transaction.
- **Solution:** This software provides a digital interface to select items, define quantities, and automatically generate a formatted invoice with tax calculations.

4. Functional Requirements

The system provides the following functionalities based on the implemented code:

- **Menu Display:** The system displays a predefined list of available products and their unit prices (e.g., Rice, Sugar, Oil).
- **Item Selection:** Users can input specific item names and the desired quantity.
- **Cost Calculation:** The system calculates the line-item cost:

$$\text{Price} = \text{Quantity} * \text{Unit Rate}$$

- **GST Calculation:** The system automatically calculates a 5% GST on the total amount.
- **Invoice Generation:** A formatted receipt is printed showing the shop name, date, item details, total price, GST, and final payable amount.

5. Non-functional Requirements

- **Usability:** The interface is a Command Line Interface (CLI) which is simple and text-based, requiring no complex installation.
- **Accuracy:** Mathematical calculations for totals and taxes are precise.
- **Efficiency:** The system processes inputs and generates the bill instantly using in-memory data structures.
- **Reliability:** The system handles the loop of adding items until the user decides to exit or print the bill.

6. System Architecture

The system follows a procedural architecture typical of Python scripting:

1. **Input Layer:** Uses input() functions to capture Customer Name, Item Choices, and Quantities.
2. **Logic Layer:**
    a. Validates if the item exists in the items dictionary.
    b. Performs arithmetic operations for totals and GST.
    c. Appends data to lists (item list, quantity list, P_list) for temporary storage.
3. **Output Layer:** Uses print () with string formatting to generate the UI and the final Bill.

7. Design Diagrams

Use Case Diagram

- **Actor:** Cashier/User
- **Use Cases:**
    - Start System
    - View Item List
    - Add Item to Cart
    - Generate Bill

Workflow Diagram (Flowchart)

The flow of logic is as follows:

1. Start -> Input Name.
2. Display List (Optional)
3. Loop: Enter Item -> Enter Quantity -> Calculate Price -> Add to List.
4. Check: Bill requested?
5. If Yes -> Calculate GST -> Print formatted Receipt -> End.

8. Design Decisions & Rationale

- **Data Structure - Dictionary:** The code uses a dictionary item = {'Rice':75...}.
    - *Rationale:* Dictionaries provide $O(1)$ average time complexity for lookups, making it very fast to check prices when an item name is entered.

- **Data Structure - Parallel Lists:** The code uses item list, quantity list, and P_list to store the cart.
    - *Rationale:* This allows for simple iteration using a for loop to print the bill line-by-line based on the index.
- **Library - Datetime:**
    - *Rationale:* Used to automatically fetch the current date for the bill, adding authenticity to the receipt.

9. Implementation Details

The project is implemented in **Python 3**. Key technical highlights include:

- **Variables:**
    - total_price: Accumulates the sum of items.
    - GST: Calculated as. (Total * 5) / 10
- **Control Flow:**
    - A for loop iterates to allow multiple item entries.
    - if-else statements handle validation (checking if an item is available) and menu selection options.
- **String Formatting:**
    - The bill uses extensive string multiplication (e.g., print(75*"-")) to create visual separators and align columns for a professional look.

10. Screenshots / Results

(Since this is a text report, below is a simulation of the Output based on your code)

Console Output Simulation:

```
Enter your name:Sudhan
for list of items press 1:1

    Rice          Rs 75/kg
    Sugar         Rs 50/kg
    Salt          Rs 25/kg
    Oil           Rs 175/each
    Panner        Rs 350/kg
    Vermicelli    Rs 70/kg
    Maggi         Rs 99/each
    Colgate       Rs 65/each
    Fogg perfume  Rs 135/each
    Onion         Rs 50/kg
    Apple         Rs 100/kg

if you want to buy press 1 or press 2 for exist:1
Enter your items:Sugar
Enter quantity:2
can i bill the items Yes or No:No
if you want to buy press 1 or press 2 for exist:1
Enter your items:Maggi
Enter quantity:4
can i bill the items Yes or No:No
if you want to buy press 1 or press 2 for exist:1
Enter your items:Onion
Enter quantity:5
can i bill the items Yes or No:No
if you want to buy press 1 or press 2 for exist:1
Enter your items:Apple
Enter quantity:1
can i bill the items Yes or No:Yes
```

```
========================== More super market ============================
============================== Hyderabad ================================
Name: Sudhan  Date: 2025-11-23
--------------------------------------------------------------------------
S/No:    items        Quantity            price
0        Sugar             2                100
1        Maggi             4                396
2        Onion             5                250
3        Apple             1                100
--------------------------------------------------------------------------
                                                  Total Amount: Rs 846
gst amount                                                      Rs 42.3
--------------------------------------------------------------------------
                                                  final_amount: Rs 888.3
--------------------------------------------------------------------------
--------------------------------------------------------------------------
                          Thanks for Visiting
--------------------------------------------------------------------------
if you want to buy press 1 or press 2 for exist:2

Process finished with exit code 0
```

11. Testing Approach

- **Unit Testing:** Verified that the math for specific items (e.g., 2 Maggi packets at 99 each = 198) is correct.
- **Input Validation Testing:**
    - Tested entering an item not in the dictionary -> Result: System prints "Sorry you entered item is not available".
    - Tested entering option '2' -> Result: Loop breaks successfully.
- **Format Testing:** Verified that the bill prints with the correct indentation and separators.

12. Challenges Faced

- **Text Alignment:** Creating a perfectly aligned table in the console using standard print statements and spaces was challenging and required trial and error with spacing multiplication (e.g., 15*" ").
- **Case Sensitivity:** The dictionary keys are case-sensitive (e.g., 'Rice'). If a user types 'rice', the system does not recognize it.
- **Input Loops:** Designing the loop to ask for billing confirmation at the right time required careful placement of the break and input statements.

13. Learnings & Key Takeaways

- **Python Basics:** Solidified understanding of Python lists, dictionaries, and loops.
- **User Experience (UX):** Learned that even in console applications, formatting output (like the bill headers) is crucial for readability.
- **Logic construction:** Learned how to accumulate totals inside a loop and apply tax logic outside or at the end of the transaction.

14. Future Enhancements

To make this system production-ready, the following features are proposed:

1. **Database Integration:** Use SQL or a file system to store transaction history permanently.
2. **GUI Implementation:** Replace the CLI with a graphical interface using Tkinter or PyQt for better usability.
3. **Stock Management:** Automatically deduct quantities from a master inventory file when items are sold.
4. **Error Handling:** Add try-except blocks to handle cases where the user enters text instead of numbers for the quantity.

15. References

1. Python Software Foundation. (n.d.). *Python 3.10 Documentation*.
2. Source Code: Project.py (Attached in Appendix).