

# Project Report: Smart Inventory and Billing System

## 1. GitHub Repository

Project Source Code:

<https://github.com/AdiChakote/ds-billingsystem>

This repository contains the complete Python implementation of the **Smart Inventory and Billing System**, including inventory management, billing logic, and role-based access for customers and shopkeepers.

## 2. Introduction

The **Smart Inventory and Billing System** is a console-based Python application designed to simulate a basic **retail store environment**.

The system supports two roles:

- **Customer** – for purchasing items and generating bills
- **Shopkeeper** – for managing stock and adding new items

This project demonstrates real-world concepts such as **inventory management, billing logic, discounts, and role-based access** using Python.

---

## 3. Objective

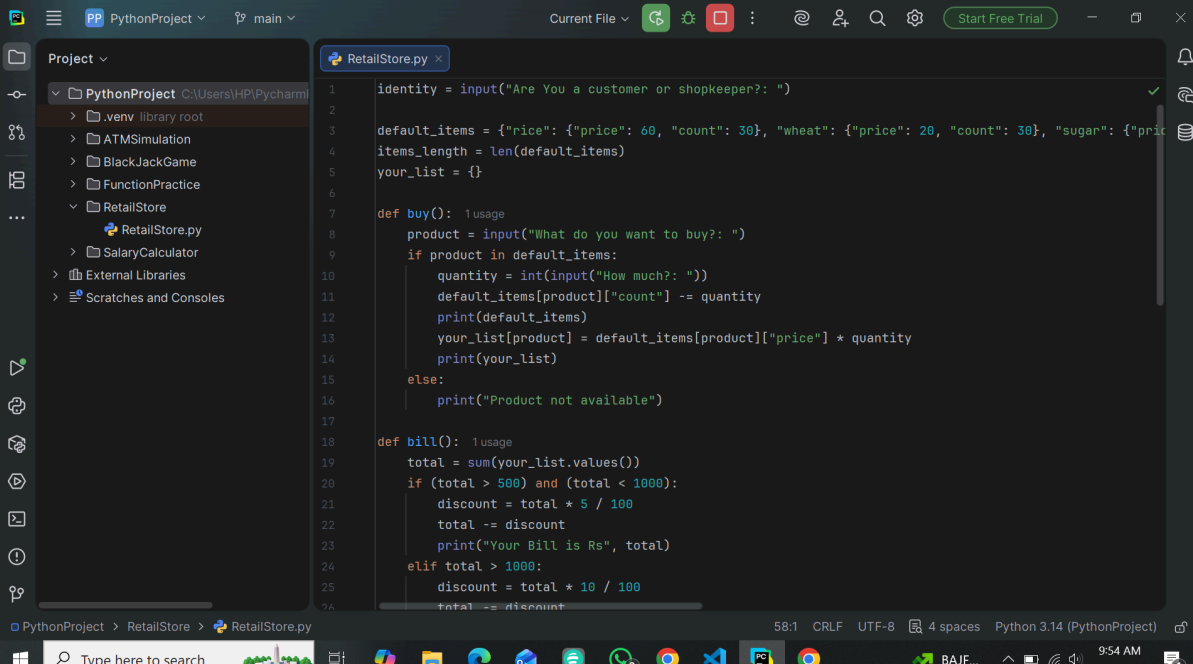
The main objectives of this project are:

- To manage store inventory using Python dictionaries
  - To allow customers to purchase items and generate bills
  - To provide discount logic based on total purchase amount
  - To allow shopkeepers to view and update stock
  - To demonstrate modular programming using functions
-

## 4. Technologies Used

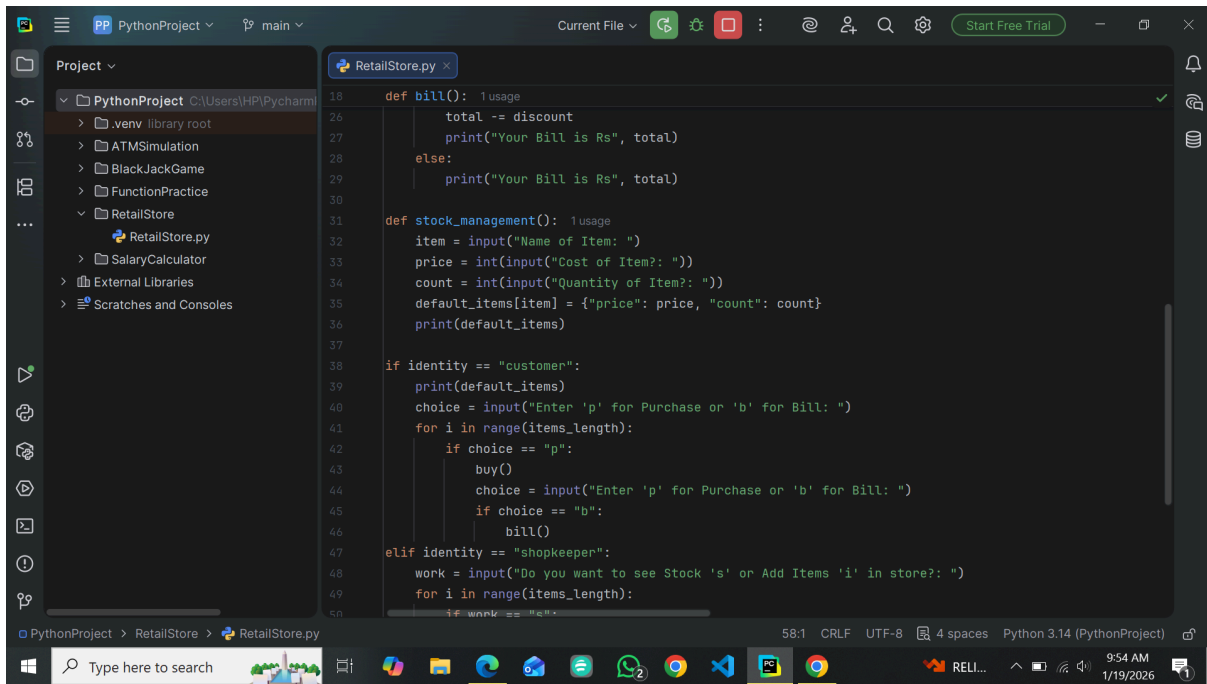
- **Programming Language:** Python
  - **Concepts Applied:**
    - Dictionaries (nested dictionaries)
    - Functions
    - Conditional statements
    - Loops
    - User input/output
  - **Interface:** Command Line (CLI)
- 

## 5. Source Code

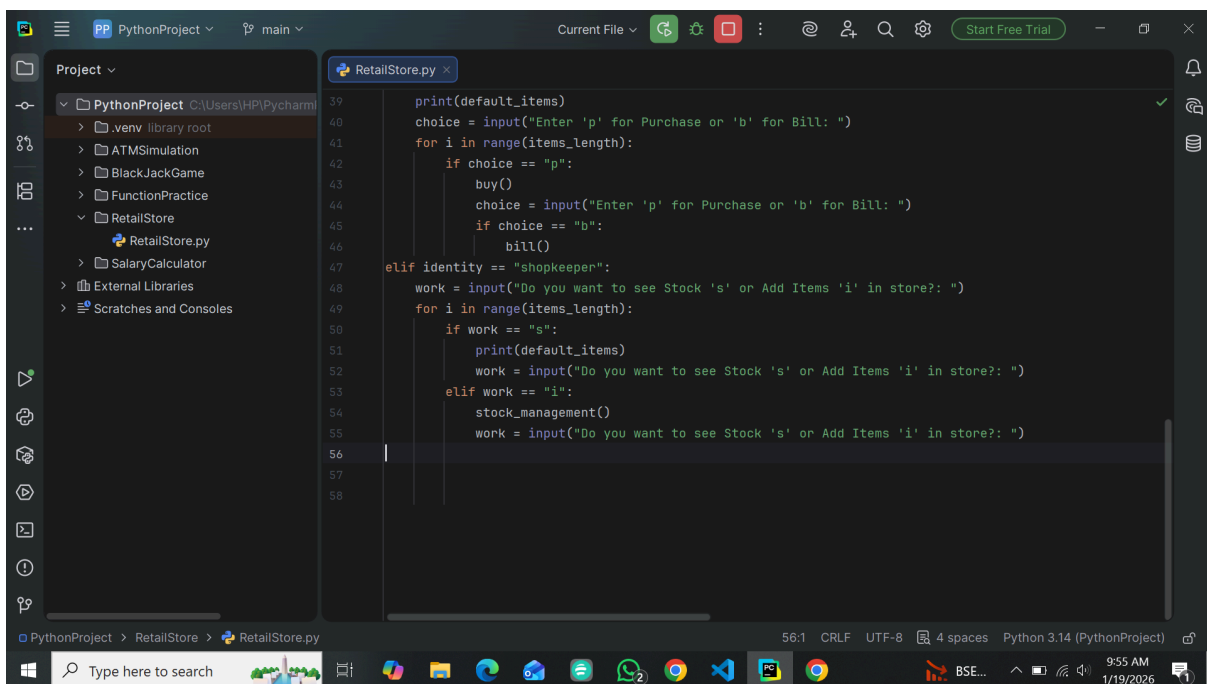


```
1 identity = input("Are You a customer or shopkeeper?: ")
2
3 default_items = {"rice": {"price": 60, "count": 30}, "wheat": {"price": 20, "count": 30}, "sugar": {"price": 10, "count": 30}}
4 items_length = len(default_items)
5 your_list = {}
6
7 def buy(): 1 usage
8     product = input("What do you want to buy?: ")
9     if product in default_items:
10         quantity = int(input("How much?: "))
11         default_items[product]["count"] -= quantity
12         print(default_items)
13         your_list[product] = default_items[product]["price"] * quantity
14         print(your_list)
15     else:
16         print("Product not available")
17
18 def bill(): 1 usage
19     total = sum(your_list.values())
20     if (total > 500) and (total < 1000):
21         discount = total * 5 / 100
22         total -= discount
23         print("Your Bill is Rs", total)
24     elif total > 1000:
25         discount = total * 10 / 100
26         total -= discount
```

The screenshot shows a PyCharm IDE with a project named 'PythonProject'. The file explorer on the left shows a directory structure with files like 'ATMSimulation', 'BlackJackGame', 'FunctionPractice', 'RetailStore', 'SalaryCalculator', and 'Scratches and Consoles'. The main editor window displays the 'RetailStore.py' file, which contains Python code for a retail store simulation. The code includes a dictionary for default items, a 'buy()' function to handle purchases, and a 'bill()' function to calculate the total bill with discounts. The status bar at the bottom indicates the file is at line 58, column 1, using CRLF line endings, UTF-8 encoding, and 4 spaces for indentation. The Python version is 3.14.



```
18 def bill(): 1 usage
26     total -= discount
27     print("Your Bill is Rs", total)
28 else:
29     print("Your Bill is Rs", total)
30
31 def stock_management(): 1 usage
32 item = input("Name of Item: ")
33 price = int(input("Cost of Item?: "))
34 count = int(input("Quantity of Item?: "))
35 default_items[item] = {"price": price, "count": count}
36 print(default_items)
37
38 if identity == "customer":
39     print(default_items)
40     choice = input("Enter 'p' for Purchase or 'b' for Bill: ")
41     for i in range(items_length):
42         if choice == "p":
43             buy()
44             choice = input("Enter 'p' for Purchase or 'b' for Bill: ")
45             if choice == "b":
46                 bill()
47 elif identity == "shopkeeper":
48     work = input("Do you want to see Stock 's' or Add Items 'i' in store?: ")
49     for i in range(items_length):
50         if work == "s":
51             print(default_items)
52             work = input("Do you want to see Stock 's' or Add Items 'i' in store?: ")
53         elif work == "i":
54             stock_management()
55             work = input("Do you want to see Stock 's' or Add Items 'i' in store?: ")
56
57
58
```



```
39     print(default_items)
40     choice = input("Enter 'p' for Purchase or 'b' for Bill: ")
41     for i in range(items_length):
42         if choice == "p":
43             buy()
44             choice = input("Enter 'p' for Purchase or 'b' for Bill: ")
45             if choice == "b":
46                 bill()
47 elif identity == "shopkeeper":
48     work = input("Do you want to see Stock 's' or Add Items 'i' in store?: ")
49     for i in range(items_length):
50         if work == "s":
51             print(default_items)
52             work = input("Do you want to see Stock 's' or Add Items 'i' in store?: ")
53         elif work == "i":
54             stock_management()
55             work = input("Do you want to see Stock 's' or Add Items 'i' in store?: ")
56
57
58
```

 **Smart Inventory and Billing System Code**

## 6. Code Overview

```
identity = input("Are You a customer or shopkeeper?: ")
```

```
default_items = {  
    "rice": {"price": 60, "count": 30},  
    "wheat": {"price": 20, "count": 30},  
    "sugar": {"price": 40, "count": 70},  
    "ghee": {"price": 600, "count": 40},  
    "soap": {"price": 30, "count": 100},  
    "toothpaste": {"price": 50, "count": 80}  
}  
  
your_list = {}
```

---

## 7. Data Structure Explanation

### 7.1 Inventory Dictionary (**default\_items**)

- Stores product name as key
- Each product has:
  - **price**
  - **count** (available stock)

Example:

```
"rice": {"price": 60, "count": 30}
```

### 7.2 Cart Dictionary (**your\_list**)

- Stores purchased items and their total cost
  - Used for bill calculation
- 

## 8. Function-wise Explanation

### 8.1 buy() Function

```
def buy():  
    product = input("What do you want to buy?: ")  
    if product in default_items:  
        quantity = int(input("How much?: "))  
        default_items[product]["count"] -= quantity  
        your_list[product] = default_items[product]["price"] *  
quantity  
    else:  
        print("Product not available")
```

#### Purpose:

- Allows customer to purchase products

#### Logic:

- Checks if product exists
  - Reduces stock count
  - Adds item to customer cart
- 

### 8.2 bill() Function

```
def bill():  
    total = sum(your_list.values())  
    if (total > 500) and (total < 1000):  
        discount = total * 5 / 100  
        total -= discount  
    elif total > 1000:  
        discount = total * 10 / 100  
        total -= discount  
    print("Your Bill is Rs", total)
```

**Purpose:**

- Calculates final bill amount

**Discount Rules:**

- ₹500 – ₹999 → **5% discount**
- Above ₹1000 → **10% discount**
- Below ₹500 → No discount

---

### 8.3 stock\_management() Function

```
def stock_management():  
    item = input("Name of Item: ")  
    price = int(input("Cost of Item?: "))  
    count = int(input("Quantity of Item?: "))  
    default_items[item] = {"price": price, "count": count}
```

**Purpose:**

- Allows shopkeeper to add or update items in inventory
- 

## 9. Role-Based Logic

### 9.1 Customer Flow

```
if identity == "customer":
```

Customer can:

- View available items
  - Purchase products
  - Generate bill
- 

### 9.2 Shopkeeper Flow

```
elif identity == "shopkeeper":
```

Shopkeeper can:

- View stock
  - Add new items
  - Update inventory
- 

## 10. Sample Execution

**Example:**

Are You a customer or shopkeeper?: customer

What do you want to buy?: rice

How much?: 5

Your Bill is Rs 300

---

## 11. Features of the System

- Role-based access (Customer / Shopkeeper)
  - Real-time stock update
  - Discount-based billing system
  - Modular and reusable functions
  - Dictionary-based inventory storage
- 

## 12. Limitations

- No validation for negative or excess quantity
  - No loop exit option
  - Stock can go negative
  - No persistent storage (data resets on restart)
  - No authentication for shopkeeper
- 

## 13. Future Scope / Enhancements

The system can be enhanced by:

1. **Input Validation**



- Prevent negative values
- Check stock availability before purchase

## **2. Persistent Storage**

- Save data using files or databases

## **3. Multiple Customers**

- Support multiple users and sessions

## **4. Authentication System**

- Login system for shopkeeper

## **5. Advanced Billing**

- GST, tax, invoice generation

## **6. GUI / Web Application**

- Convert to Tkinter / Flask / Django app

---

# **14. Conclusion**

The Smart Inventory and Billing System effectively demonstrates real-world retail logic using Python. It applies core programming concepts such as dictionaries, functions, and conditional logic in a practical scenario. This project serves as a strong foundation for building more advanced inventory and billing systems.