Import and export

import csv

import os

PRODUCTS\_FILE = 'products.csv'

BILL\_FILE = 'bill.csv'

# Create files with headers if they don't exist

if not os.path.exists(PRODUCTS\_FILE):

    with open(PRODUCTS\_FILE, 'w', newline='') as f:

        writer = csv.writer(f)

        writer.writerow(['Name', 'Price', 'Stock'])

if not os.path.exists(BILL\_FILE):

    with open(BILL\_FILE, 'w', newline='') as f:

        writer = csv.writer(f)

        writer.writerow(['Product', 'Price', 'Quantity', 'Total'])

def check():

    person = input("Who are you (owner/dealer/customer)? ").lower()

    if person == "owner":

        return Owner()

    elif person == "dealer":

        return Dealer()

    elif person == "customer":

        return Customer()

    else:

        print("Invalid role.")

        return None

class ProductManager:

    @staticmethod

    def read\_all():

        if not os.path.exists(PRODUCTS\_FILE):

            return []

        with open(PRODUCTS\_FILE, newline='') as f:

            reader = csv.reader(f)

            next(reader, None)  # Skip header

            return [[row[0], float(row[1]), int(row[2])] for row in reader]

    @staticmethod

    def write\_all(products):

        with open(PRODUCTS\_FILE, 'w', newline='') as f:

            writer = csv.writer(f)

            writer.writerow(['Name', 'Price', 'Stock'])  # Write header

            writer.writerows(products)

class Owner:

    def menu(self):

        while True:

            print("\nOwner Menu:\n1. Create\n2. Update\n3. Delete\n4. Read\n5. Exit")

            choice = input("Enter choice: ")

            if choice == '1':

                self.create\_product()

            elif choice == '2':

                self.update\_product()

            elif choice == '3':

                self.delete\_product()

            elif choice == '4':

                self.read\_products()

            elif choice == '5':

                break

            else:

                print("Invalid choice.")

    def create\_product(self):

        name = input("Product name: ")

        try:

            price = float(input("Price: "))

            stock = int(input("Stock: "))

        except ValueError:

            print("Invalid input. Please enter numeric values.")

            return

        products = ProductManager.read\_all()

        products.append([name, price, stock])

        ProductManager.write\_all(products)

        print("Product added.")

    def update\_product(self):

        products = ProductManager.read\_all()

        name = input("Enter product name to update: ")

        found = False

        for product in products:

            if product[0].lower() == name.lower():

                new\_name = input("Enter new product name (or press Enter to keep the same): ")

                if new\_name.strip() != "":

                    product[0] = new\_name

                try:

                    product[1] = float(input("New price: "))

                    product[2] = int(input("New stock: "))

                except ValueError:

                    print("Invalid input.")

                    return

                found = True

                break

        if found:

            ProductManager.write\_all(products)

            print("Product updated.")

        else:

            print("Product not found.")

    def delete\_product(self):

        products = ProductManager.read\_all()

        name = input("Enter product name to delete: ")

        new\_products = [p for p in products if p[0].lower() != name.lower()]

        if len(new\_products) == len(products):

            print("Product not found.")

        else:

            ProductManager.write\_all(new\_products)

            print("Product deleted.")

    def read\_products(self):

        products = ProductManager.read\_all()

        if not products:

            print("No products found.")

            return

        print("\nAvailable Products:")

        for p in products:

            print(f"Name: {p[0]}, Price: Rs.{p[1]}, Stock: {p[2]}")

class Dealer:

    def input\_data(self):

        name = input("Product name to add/update: ")

        try:

            stock = int(input("Stock to add: "))

        except ValueError:

            print("Invalid stock input.")

            return

        products = ProductManager.read\_all()

        found = False

        for product in products:

            if product[0].lower() == name.lower():

                product[2] += stock

                found = True

                break

        if not found:

            try:

                price = float(input("New product price: "))

            except ValueError:

                print("Invalid price.")

                return

            products.append([name, price, stock])

            print("New product added.")

        else:

            print("Stock updated for existing product.")

        ProductManager.write\_all(products)

class Customer:

    def view\_and\_buy(self):

        products = ProductManager.read\_all()

        if not products:

            print("No products available.")

            return

        print("\nAvailable Products:")

        for i, p in enumerate(products):

            print(f"{i+1}. {p[0]} - Rs.{p[1]} (Stock: {p[2]})")

        try:

            choice = int(input("Select product number to buy: ")) - 1

            quantity = int(input("Enter quantity: "))

        except ValueError:

            print("Invalid input.")

            return

        if 0 <= choice < len(products):

            product = products[choice]

            if quantity <= product[2]:

                total = product[1] \* quantity

                product[2] -= quantity

                ProductManager.write\_all(products)

                self.\_generate\_bill(product[0], product[1], quantity, total)

                print(f"Purchased {quantity} x {product[0]} for Rs.{total}")

            else:

                print("Not enough stock.")

        else:

            print("Invalid product number.")

    def \_generate\_bill(self, name, price, qty, total):

        with open(BILL\_FILE, 'a', newline='') as f:

            writer = csv.writer(f)

            writer.writerow([name, price, qty, total])

# Main interaction

user = check()

if isinstance(user, Owner):

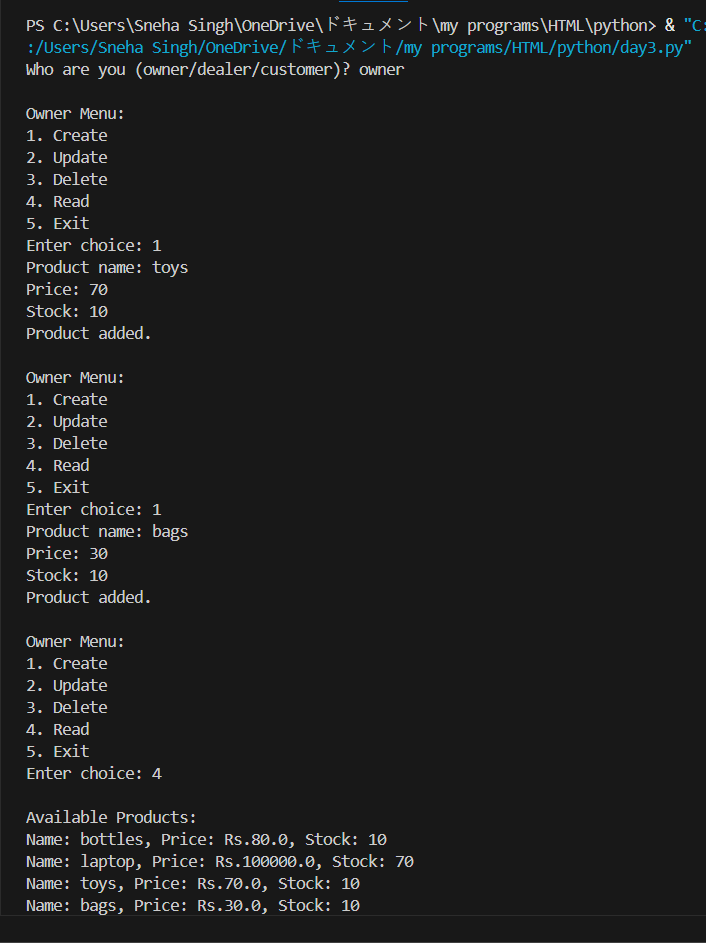
    user.menu()

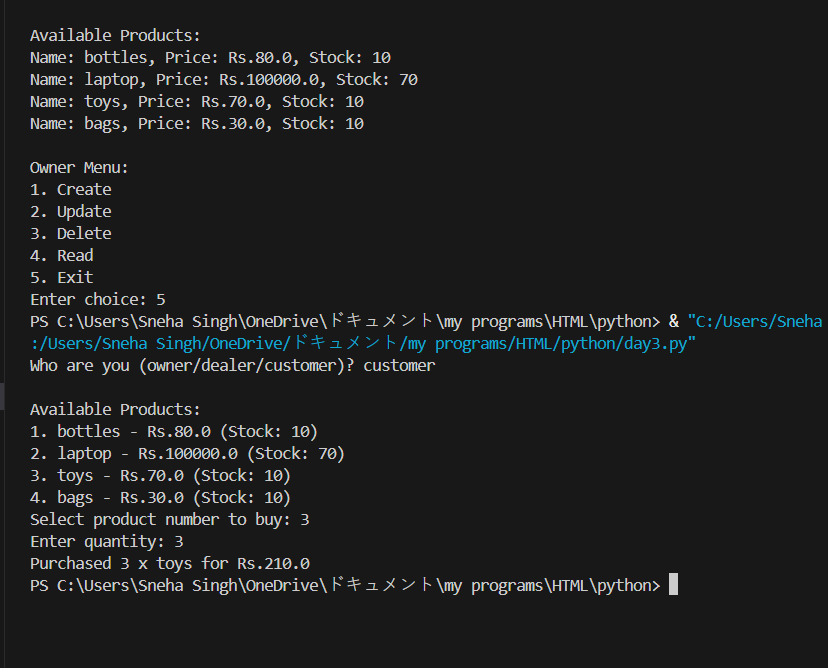
elif isinstance(user, Dealer):

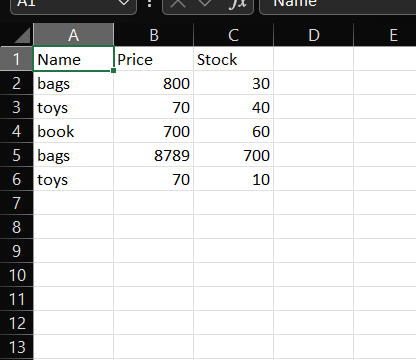
    user.input\_data()

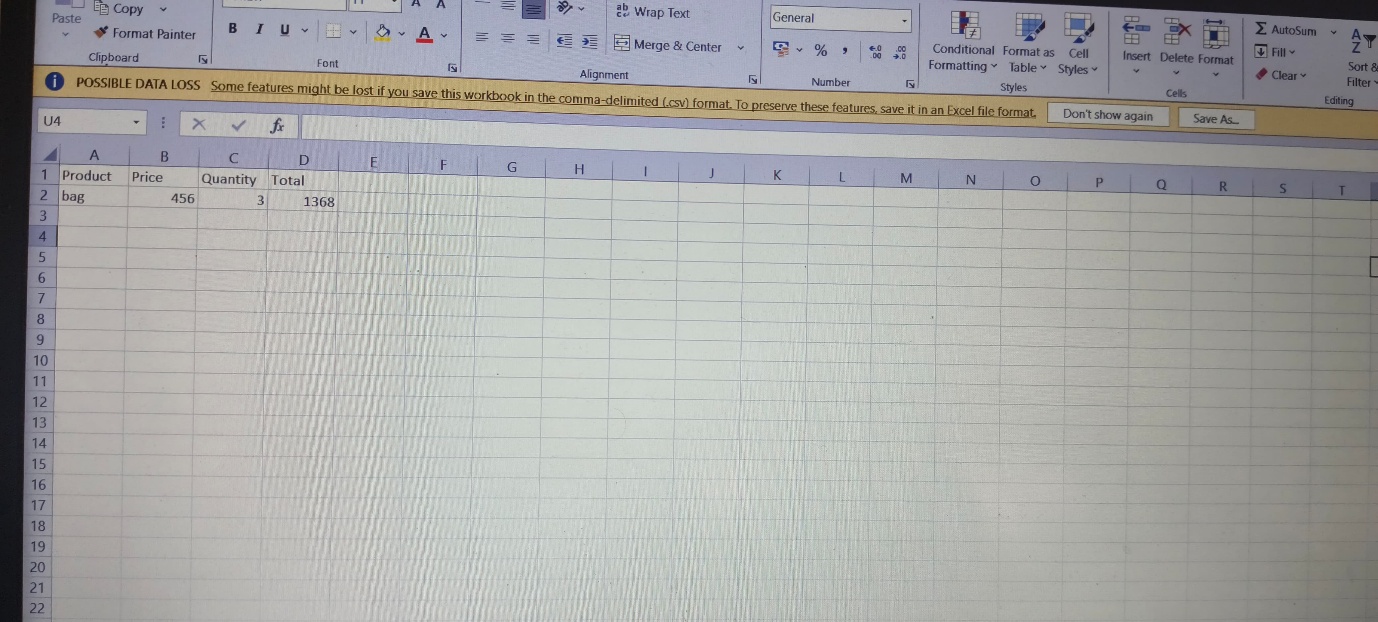
elif isinstance(user, Customer):

    user.view\_and\_buy()









Auditorium

n = int(input("Enter number of rows: "))

a = [[0 for \_ in range(5)] for \_ in range(n)]

b = [[0 for \_ in range(10)] for \_ in range(n)]

c = [[0 for \_ in range(5)] for \_ in range(n)]

section = input("Enter which section you want (a/b/c): ").lower()

row = int(input("Enter row number: "))

column = int(input("Enter column number: "))

seat = f"{section}[{row}][{column}]"

print("Selected seat:", seat)

if section == 'a':

    if 0 <= row < n and 0 <= column < 5:

        a[row][column] = "\*"

    else:

        print("Invalid seat in section a.")

elif section == 'b':

    if 0 <= row < n and 0 <= column < 10:

        b[row][column] = "\*"

    else:

        print("Invalid seat in section b.")

elif section == 'c':

    if 0 <= row < n and 0 <= column < 5:

        c[row][column] = "\*"

    else:

        print("Invalid seat in section c.")

else:

    print("Choose the correct section (a/b/c).")

print("\nSeating Layout:")

for i in range(n):

    rowa = "".join(str(v) for v in a[i])

    rowb = "".join(str(v) for v in b[i])

    rowc = "".join(str(v) for v in c[i])

    print(f"{rowa} {rowb} {rowc}")

