Assignment-6.4

Name: Sushma Purella

H. No:2303A52188

Batch:44

Task 1: Student Performance Evaluation System

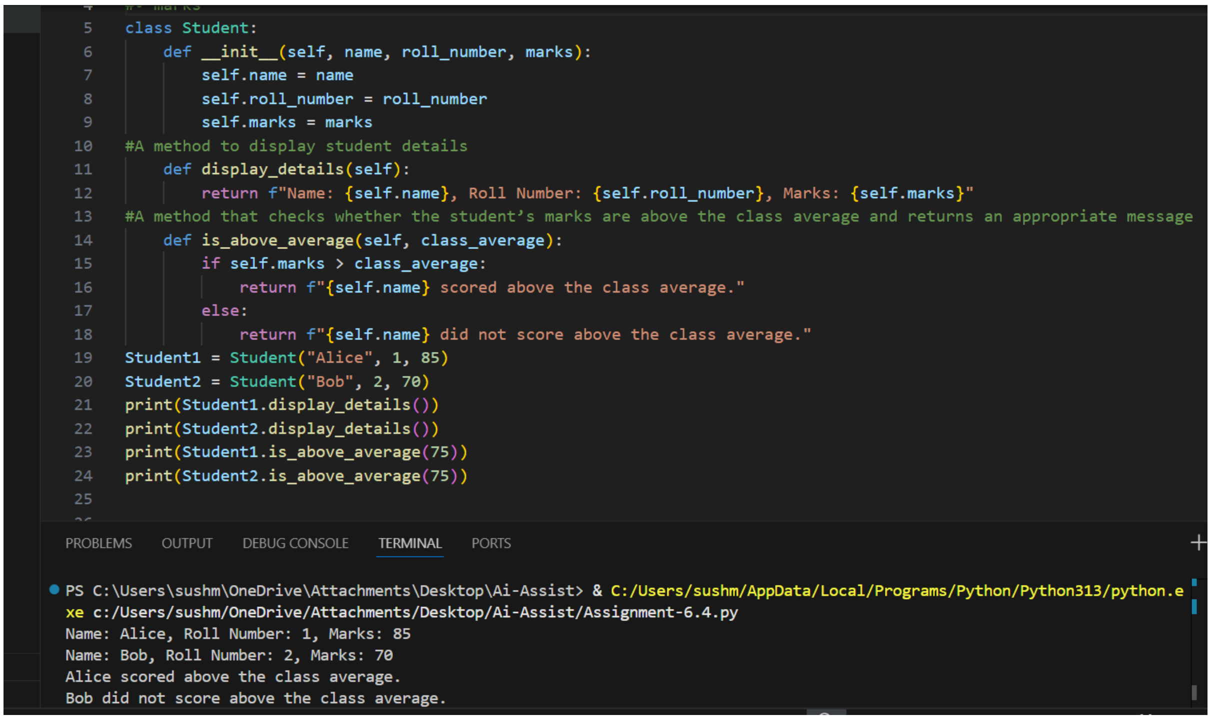
You are building a simple academic management module for a university

system where student performance needs to be evaluated automatically.

Prompt:

Create a Python class Student with attributes name, roll_number, and marks. Add methods to display student details and check if the marks are above the class average. Create objects and test the methods.

Code:



```python
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks
#A method to display student details
    def display_details(self):
        return f"Name: {self.name}, Roll Number: {self.roll_number}, Marks: {self.marks}"
#A method that checks whether the student’s marks are above the class average and returns an appropriate message
    def is_above_average(self, class_average):
        if self.marks > class_average:
            return f"{self.name} scored above the class average."
        else:
            return f"{self.name} did not score above the class average."
Student1 = Student("Alice", 1, 85)
Student2 = Student("Bob", 2, 70)
print(Student1.display_details())
print(Student2.display_details())
print(Student1.is_above_average(75))
print(Student2.is_above_average(75))
```

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Python/Python313/python.e
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment-6.4.py
Name: Alice, Roll Number: 1, Marks: 85
Name: Bob, Roll Number: 2, Marks: 70
Alice scored above the class average.
Bob did not score above the class average.
```

Observation:

The Student class was implemented successfully using object-oriented programming concepts.

The constructor correctly initializes the student attributes. The display_details() method outputs the student’s name, roll number, and marks in a clear format.

The is_above_average() method correctly compares the student’s marks with the given class average and returns an appropriate message.

The program produces correct results when tested with multiple student objects, confirming proper class behavior and method functionality.

Task 2: Data Processing in a Monitoring System

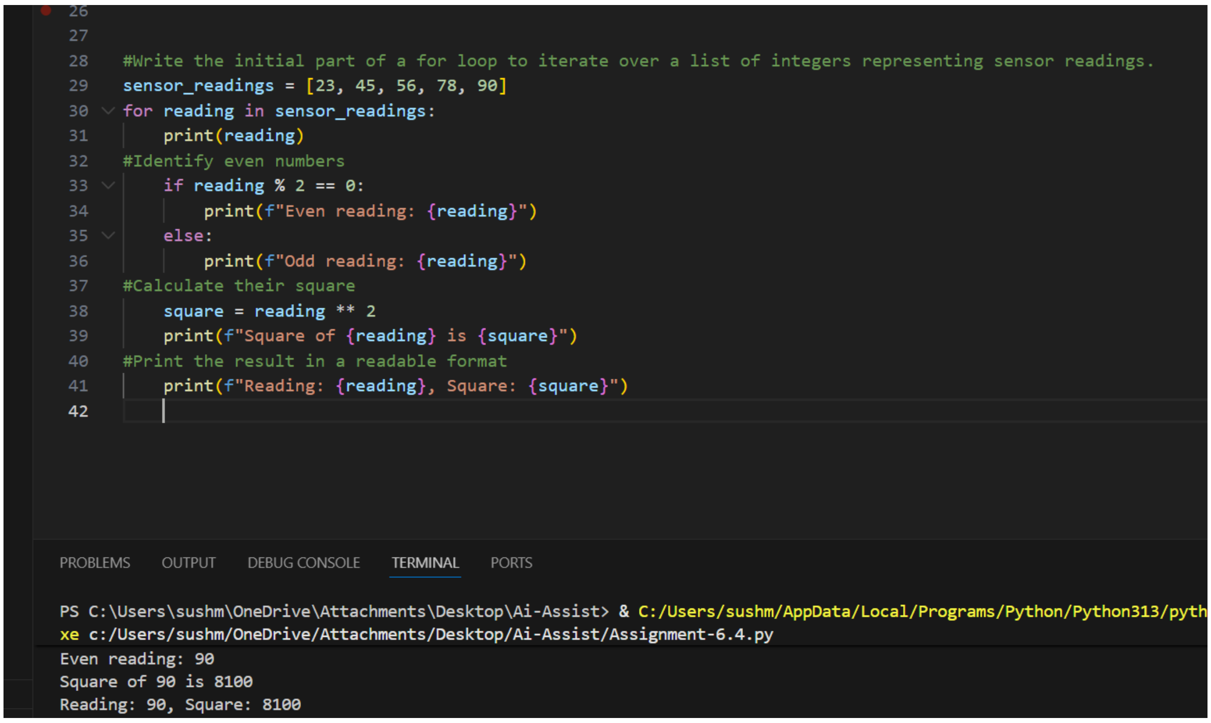
You are working on a basic data monitoring script where sensor readings

are collected as numbers. Only even readings need further processing.

Prompt:

Write the initial part of a for loop to iterate over a list of integers representing sensor readings. Identify whether each reading is even or odd, calculate its square, and print the results in a readable format.

Code:



```python

#Write the initial part of a for loop to iterate over a list of integers representing sensor readings.
sensor_readings = [23, 45, 56, 78, 90]
for reading in sensor_readings:
    print(reading)
#Identify even numbers
    if reading % 2 == 0:
        print(f"Even reading: {reading}")
    else:
        print(f"Odd reading: {reading}")
#Calculate their square
    square = reading ** 2
    print(f"Square of {reading} is {square}")
#Print the result in a readable format
    print(f"Reading: {reading}, Square: {square}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Python/Python313/pyth
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment-6.4.py
Even reading: 90
Square of 90 is 8100
Reading: 90, Square: 8100
```

Observation:

The for loop successfully iterates through each value in the sensor_readings list one by one. For every sensor reading, the program correctly checks whether the value is even or odd using the modulus operator.

It then calculates the square of each reading using the exponent operator.

All results are displayed clearly with descriptive messages, making the output easy to understand.

This confirms proper use of looping, conditional statements, and arithmetic operations in Python.

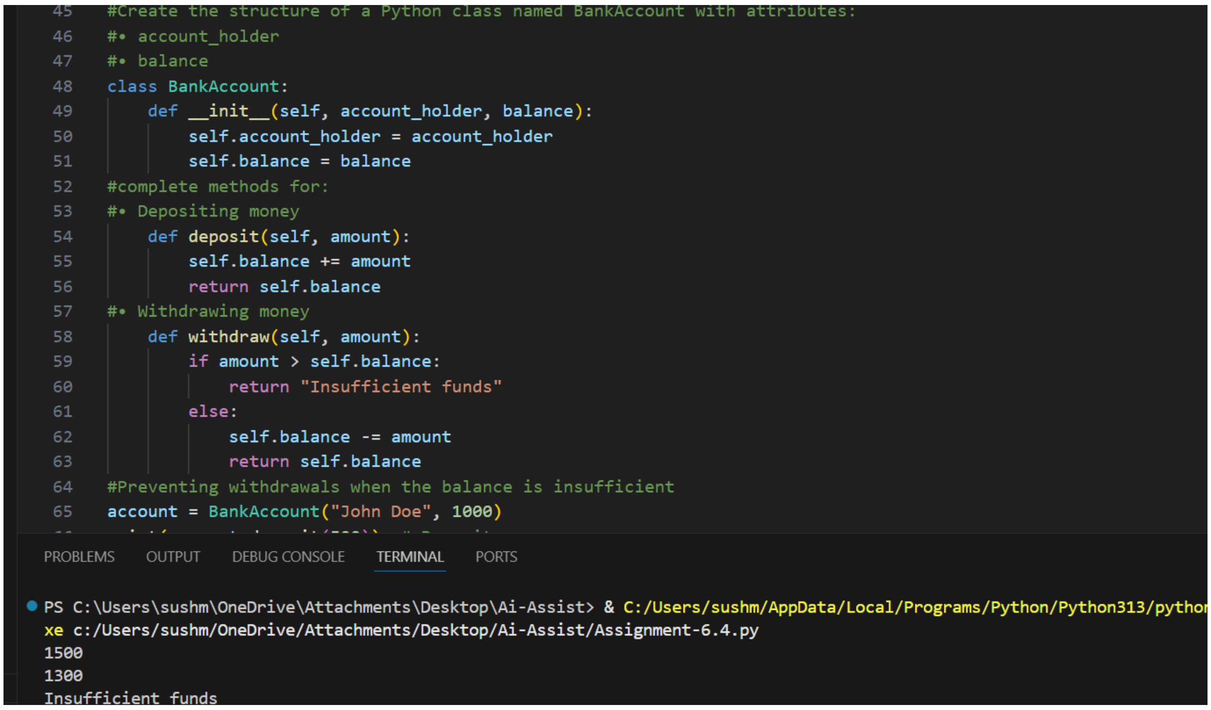
Task 3: Banking Transaction Simulation

You are developing a basic banking module that handles deposits and withdrawals for customers.

Prompt:

Create a Python class named BankAccount with attributes account_holder and balance. Implement methods to deposit money and withdraw money. Ensure that withdrawals are prevented when the balance is insufficient. Create an object of the class and demonstrate all operations.

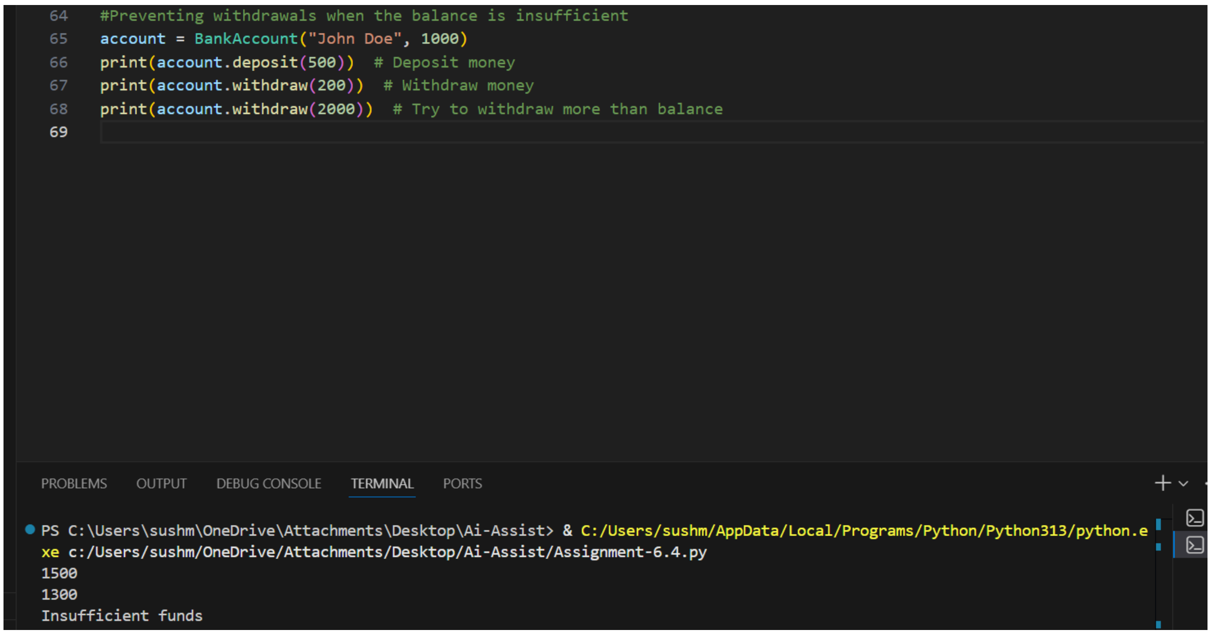
Code:



```python
#Create the structure of a Python class named BankAccount with attributes:
#• account_holder
#• balance
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance
#complete methods for:
#• Depositing money
    def deposit(self, amount):
        self.balance += amount
        return self.balance
#• Withdrawing money
    def withdraw(self, amount):
        if amount > self.balance:
            return "Insufficient funds"
        else:
            self.balance -= amount
            return self.balance
#Preventing withdrawals when the balance is insufficient
account = BankAccount("John Doe", 1000)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Python/Python313/pythor
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment-6.4.py
1500
1300
Insufficient funds
```



```python
#Preventing withdrawals when the balance is insufficient
account = BankAccount("John Doe", 1000)
print(account.deposit(500))  # Deposit money
print(account.withdraw(200))  # Withdraw money
print(account.withdraw(2000))  # Try to withdraw more than balance
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Python/Python313/python.e
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment-6.4.py
1500
1300
Insufficient funds
```

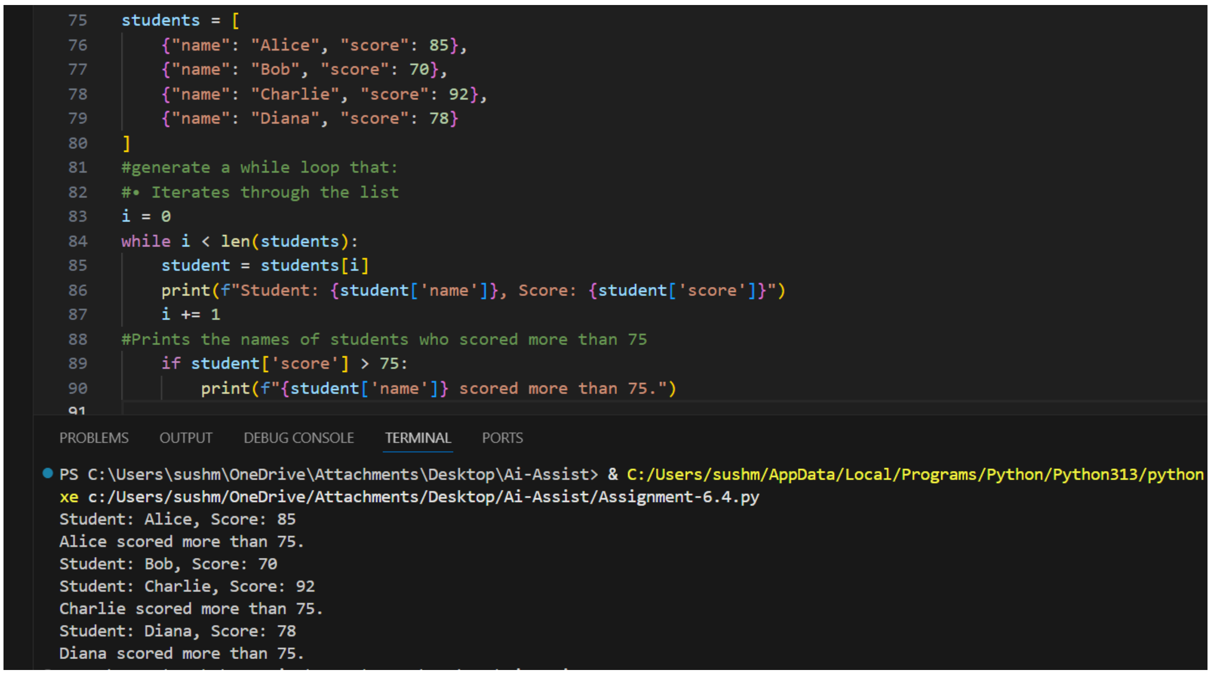
Task 4: Student Scholarship Eligibility Check

A university wants to identify students eligible for a merit-based scholarship based on their scores.

Prompt:

Define a list of dictionaries where each dictionary represents a student with name and score. Use a while loop to iterate through the list and print student details. Also, print the names of students who scored more than 75

Code:



```python
students = [
    {"name": "Alice", "score": 85},
    {"name": "Bob", "score": 70},
    {"name": "Charlie", "score": 92},
    {"name": "Diana", "score": 78}
]
#generate a while loop that:
#• Iterates through the list
i = 0
while i < len(students):
    student = students[i]
    print(f"Student: {student['name']}, Score: {student['score']}")
    i += 1
#Prints the names of students who scored more than 75
    if student['score'] > 75:
        print(f"{student['name']} scored more than 75.")
```

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Python/Python313/python
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment-6.4.py
Student: Alice, Score: 85
Alice scored more than 75.
Student: Bob, Score: 70
Student: Charlie, Score: 92
Charlie scored more than 75.
Student: Diana, Score: 78
Diana scored more than 75.
```

Observation:

The list of dictionaries correctly stores student names and their respective scores.

The while loop successfully iterates through each student using an index variable. During each iteration, the program prints the student's name and score in a readable format.

The conditional statement accurately identifies students who scored more than 75 and prints their names accordingly.

This demonstrates effective use of lists, dictionaries, while loops, indexing, and conditional logic in Python

Task 5: Online Shopping Cart Module

You are designing a simplified shopping cart system for an e-commerce website that supports item management and discount calculation

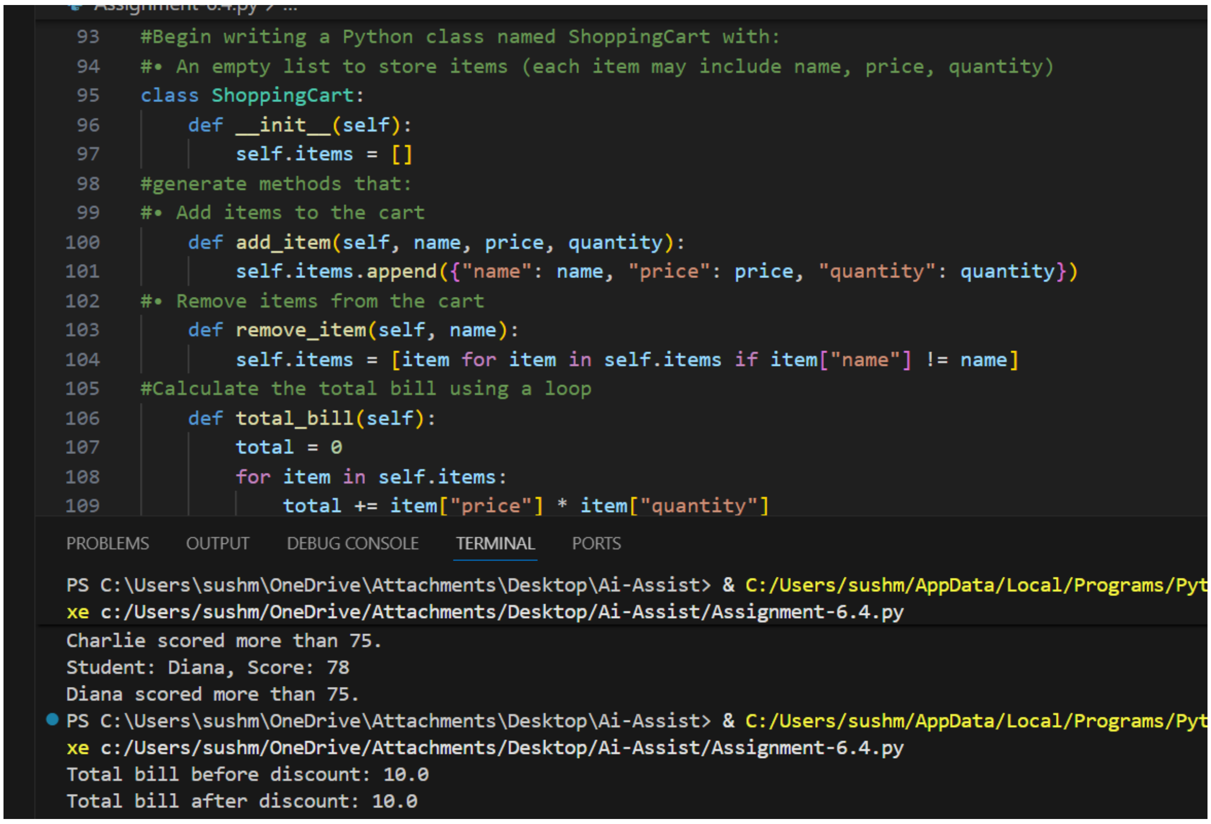
Prompt:

Begin writing a Python class named ShoppingCart with an empty list to store items.

Implement methods to add items to the cart, remove items from the cart, calculate the total bill using a loop, and apply a conditional discount if the total exceeds a specified amount.

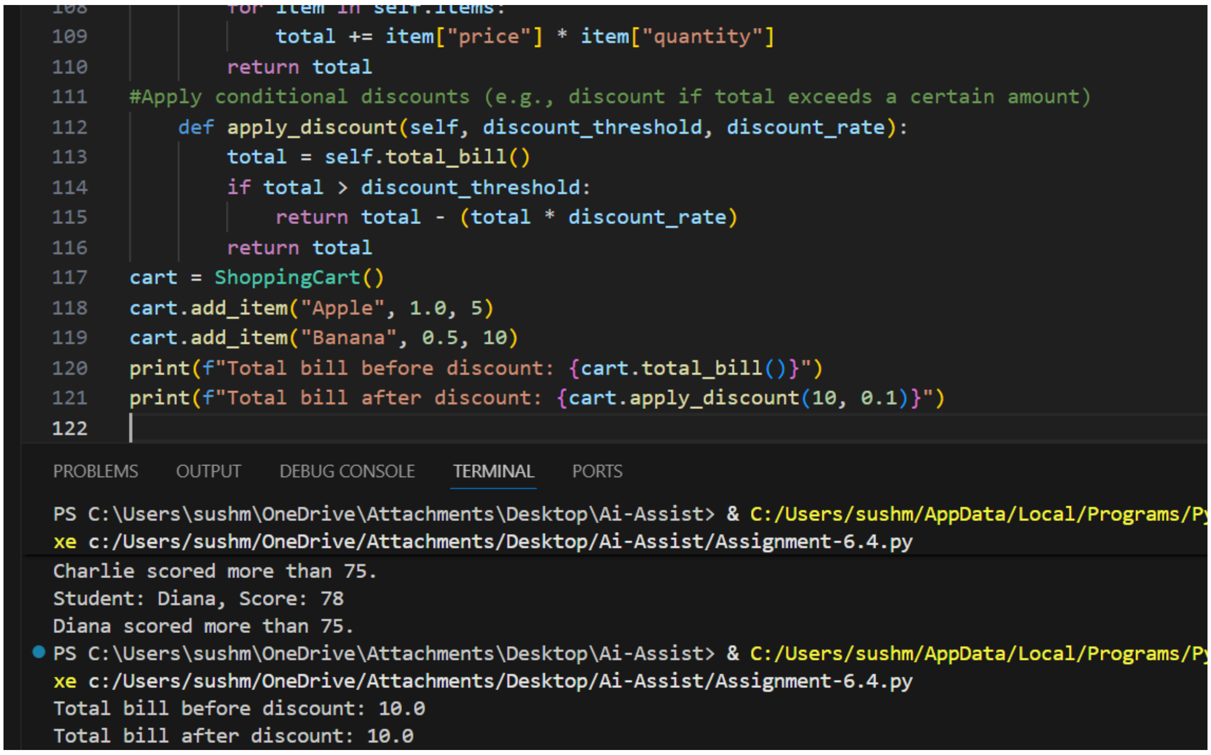
Demonstrate the functionality by adding items and calculating the final bill.

Code:



```python
#Begin writing a Python class named ShoppingCart with:
#• An empty list to store items (each item may include name, price, quantity)
class ShoppingCart:
    def __init__(self):
        self.items = []
#generate methods that:
#• Add items to the cart
    def add_item(self, name, price, quantity):
        self.items.append({"name": name, "price": price, "quantity": quantity})
#• Remove items from the cart
    def remove_item(self, name):
        self.items = [item for item in self.items if item["name"] != name]
#Calculate the total bill using a loop
    def total_bill(self):
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]
```

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Pyt
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment-6.4.py
Charlie scored more than 75.
Student: Diana, Score: 78
Diana scored more than 75.
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Pyt
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment-6.4.py
Total bill before discount: 10.0
Total bill after discount: 10.0
```



```python
            total += item["price"] * item["quantity"]
        return total
#Apply conditional discounts (e.g., discount if total exceeds a certain amount)
    def apply_discount(self, discount_threshold, discount_rate):
        total = self.total_bill()
        if total > discount_threshold:
            return total - (total * discount_rate)
        return total
cart = ShoppingCart()
cart.add_item("Apple", 1.0, 5)
cart.add_item("Banana", 0.5, 10)
print(f"Total bill before discount: {cart.total_bill()}")
print(f"Total bill after discount: {cart.apply_discount(10, 0.1)}")
```

```
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Py
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment-6.4.py
Charlie scored more than 75.
Student: Diana, Score: 78
Diana scored more than 75.
PS C:\Users\sushm\OneDrive\Attachments\Desktop\Ai-Assist> & C:/Users/sushm/AppData/Local/Programs/Py
xe c:/Users/sushm/OneDrive/Attachments/Desktop/Ai-Assist/Assignment-6.4.py
Total bill before discount: 10.0
Total bill after discount: 10.0
```

Observation:

The ShoppingCart class was implemented successfully with an empty list to store cart items. The add_item() method correctly adds items with name, price, and quantity to the cart, while the remove_item() method removes items based on their name.

The total_bill() method accurately calculates the total cost by iterating through all items and summing the product of price and quantity.

The apply_discount() method correctly applies a discount when the total bill exceeds the specified threshold.

The program produces correct results when tested, demonstrating effective use of lists, dictionaries, loops, conditional statements, and class methods in Python