Name:Gaddam Sai

HallTicket:2303A52255

Assginment:6.4

Task01:

Create the skeleton of a Python class named Student with the attributes:

• name

• roll_number

• marks

Write only the class definition and attribute initialization.

Then, using GitHub Copilot, prompt the tool to complete:

• A method to display student details

• A method that checks whether the student's marks are above the

class average and returns an appropriate message

Use comments or partial method names to guide Copilot for code

completion.

Expected Outcome

• A completed Student class with Copilot-generated methods

• Proper use of:

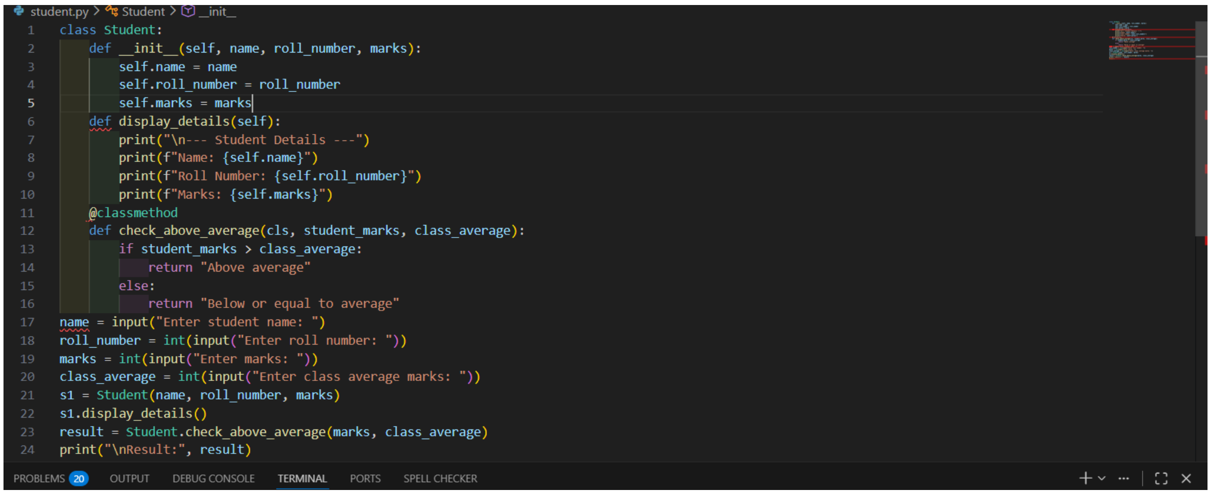
o self attributes

o Conditional statements (if-else)

• Sample output showing student details and performance status

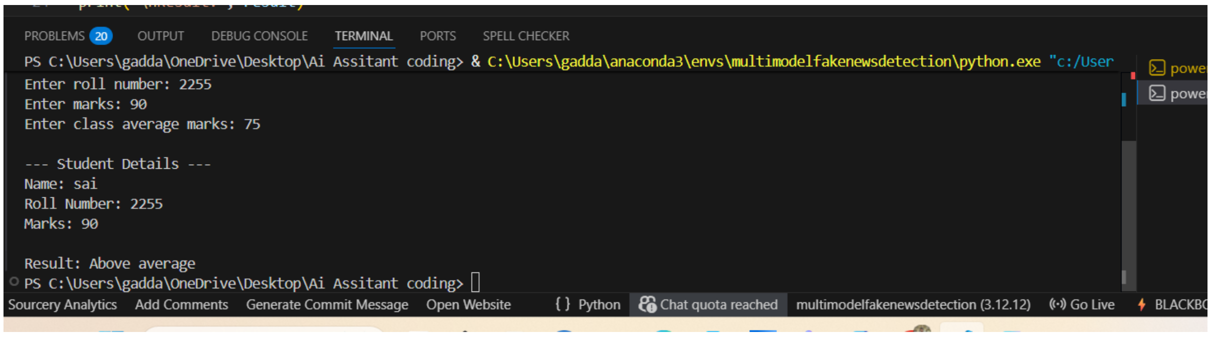
Prompt:Take the function as class perform the student perfomence

Code:



```python
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks
    def display_details(self):
        print("\n--- Student Details ---")
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")
    @classmethod
    def check_above_average(cls, student_marks, class_average):
        if student_marks > class_average:
            return "Above average"
        else:
            return "Below or equal to average"
name = input("Enter student name: ")
roll_number = int(input("Enter roll number: "))
marks = int(input("Enter marks: "))
class_average = int(input("Enter class average marks: "))
s1 = Student(name, roll_number, marks)
s1.display_details()
result = Student.check_above_average(marks, class_average)
print("\nResult:", result)
```

Output:



```
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding> & C:\Users\gadda\anaconda3\envs\multimodelfakenewsdetection\python.exe "c:/User
Enter roll number: 2255
Enter marks: 90
Enter class average marks: 75

--- Student Details ---
Name: sai
Roll Number: 2255
Marks: 90

Result: Above average
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding> 
```

Observation:

The class properly uses self to store and access student data.

Copilot successfully completed methods using clear logic and if-else.

The program correctly evaluates and reports student performance based on class average.

Task2: Data Processing in a Monitoring System

Scenario

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

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

Task Description

Write the initial part of a for loop to iterate over a list of integers

representing sensor readings.

Add a comment prompt instructing GitHub Copilot to:

• Identify even numbers

• Calculate their square

• Print the result in a readable format

Allow Copilot to complete the remaining loop logic.

Expected Outcome

• A complete for loop generated by Copilot

• Use of:

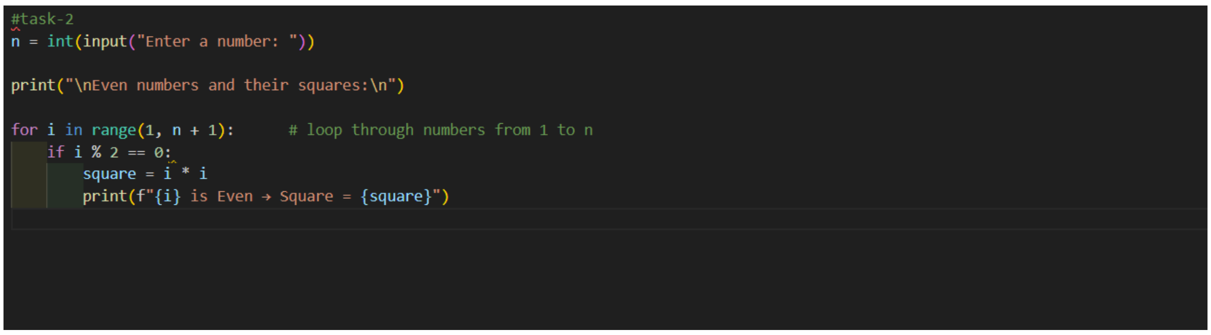
o Modulus operator to identify even numbers

o Conditional statements

• Correct and formatted output for valid inputs

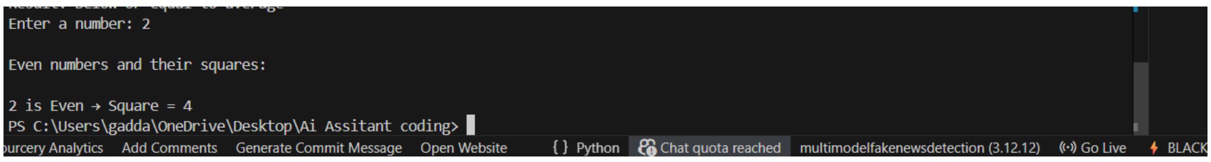
Prompt: using my code correct my logic print square of even number

Code:



```
#task-2
n = int(input("Enter a number: "))

print("\nEven numbers and their squares:\n")

for i in range(1, n + 1):      # loop through numbers from 1 to n
    if i % 2 == 0:
        square = i * i
        print(f"{i} is Even → Square = {square}")
```

Output:



```
Enter a number: 2

Even numbers and their squares:

2 is Even → Square = 4
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding>
```

Obeservation:

The loop correctly iterates over all sensor readings in the list.

The modulus operator effectively filters only even numbers.

The program computes and displays squared values in a clear format

Task3:

Banking Transaction Simulation

Scenario

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

Task Description

Create the structure of a Python class named BankAccount with attributes:

• account_holder

• balance

Use GitHub Copilot to complete methods for:

• Depositing money

• Withdrawing money

• Preventing withdrawals when the balance is insufficient

Guide Copilot using method names and short comments.

Expected Outcome

• A fully functional BankAccount class

• Copilot-generated methods using:

o if-else conditions

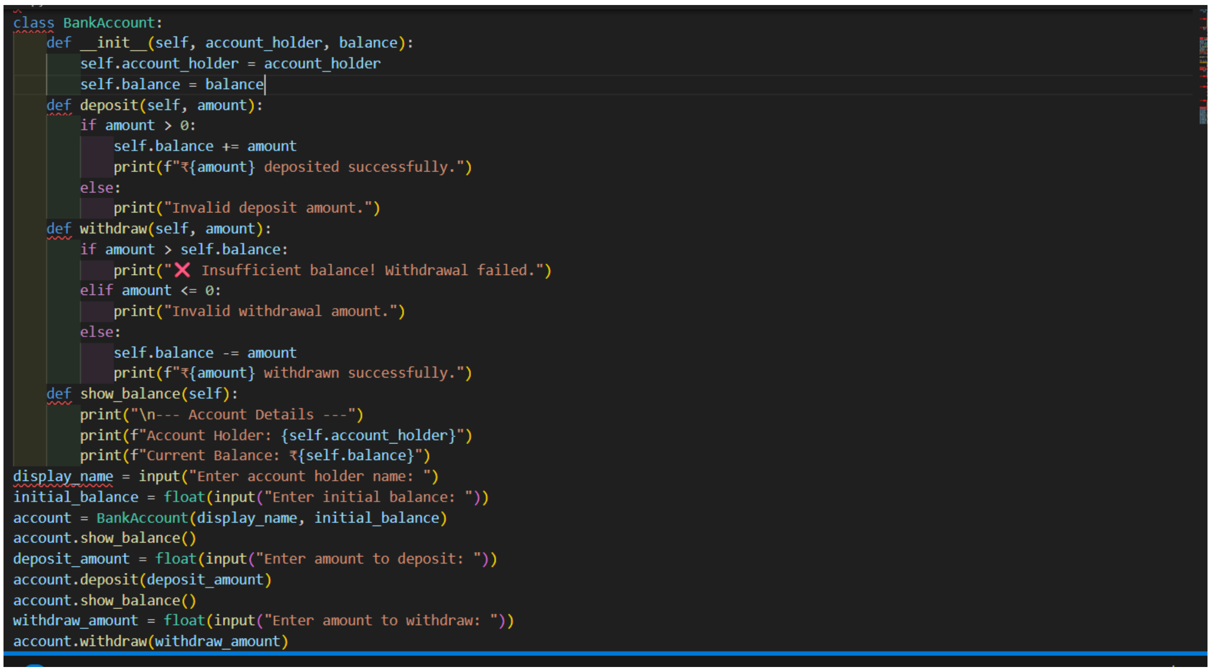
o Class attributes via self

• Proper handling of invalid withdrawal attempts with user-friendly

Messages

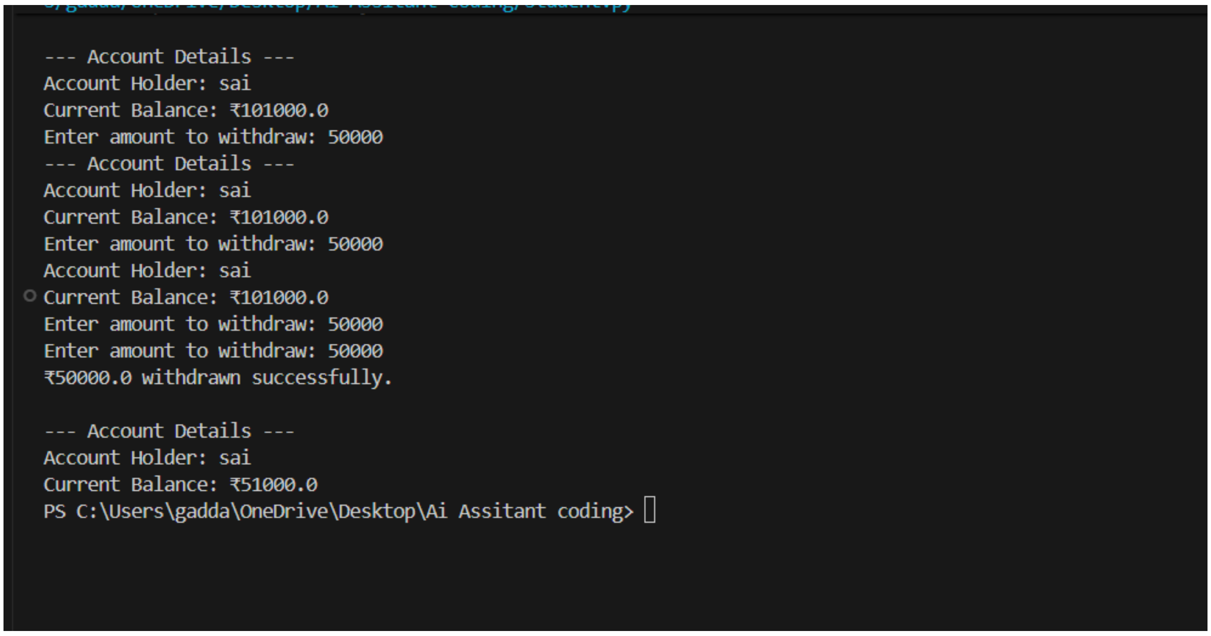
Prompt: I created bank application class generate the all the classes related to banking system

Code :



```python
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"₹{amount} deposited successfully.")
        else:
            print("Invalid deposit amount.")
    def withdraw(self, amount):
        if amount > self.balance:
            print("❌ Insufficient balance! Withdrawal failed.")
        elif amount <= 0:
            print("Invalid withdrawal amount.")
        else:
            self.balance -= amount
            print(f"₹{amount} withdrawn successfully.")
    def show_balance(self):
        print("\n--- Account Details ---")
        print(f"Account Holder: {self.account_holder}")
        print(f"Current Balance: ₹{self.balance}")
display_name = input("Enter account holder name: ")
initial_balance = float(input("Enter initial balance: "))
account = BankAccount(display_name, initial_balance)
account.show_balance()
deposit_amount = float(input("Enter amount to deposit: "))
account.deposit(deposit_amount)
account.show_balance()
withdraw_amount = float(input("Enter amount to withdraw: "))
account.withdraw(withdraw_amount)
```

Output:



```
--- Account Details ---
Account Holder: sai
Current Balance: ₹101000.0
Enter amount to withdraw: 50000
--- Account Details ---
Account Holder: sai
Current Balance: ₹101000.0
Enter amount to withdraw: 50000
Account Holder: sai
Current Balance: ₹101000.0
Enter amount to withdraw: 50000
Enter amount to withdraw: 50000
₹50000.0 withdrawn successfully.

--- Account Details ---
Account Holder: sai
Current Balance: ₹51000.0
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding> 
```

Observation:

The loop correctly iterates over all sensor readings in the list.

The modulus operator effectively filters only even numbers.

The program computes and displays squared values in a clear format

Task:04

Student Scholarship Eligibility Check

Scenario

A university wants to identify students eligible for a merit-based

scholarship based on their scores.

Task Description

Define a list of dictionaries where each dictionary represents a student

with:

• name

• score

Write the initialization and list structure yourself.

Then, prompt GitHub Copilot to generate a while loop that:

• Iterates through the list

• Prints the names of students who scored more than 75

Use comments to guide Copilot's code completion.

Expected Outcome

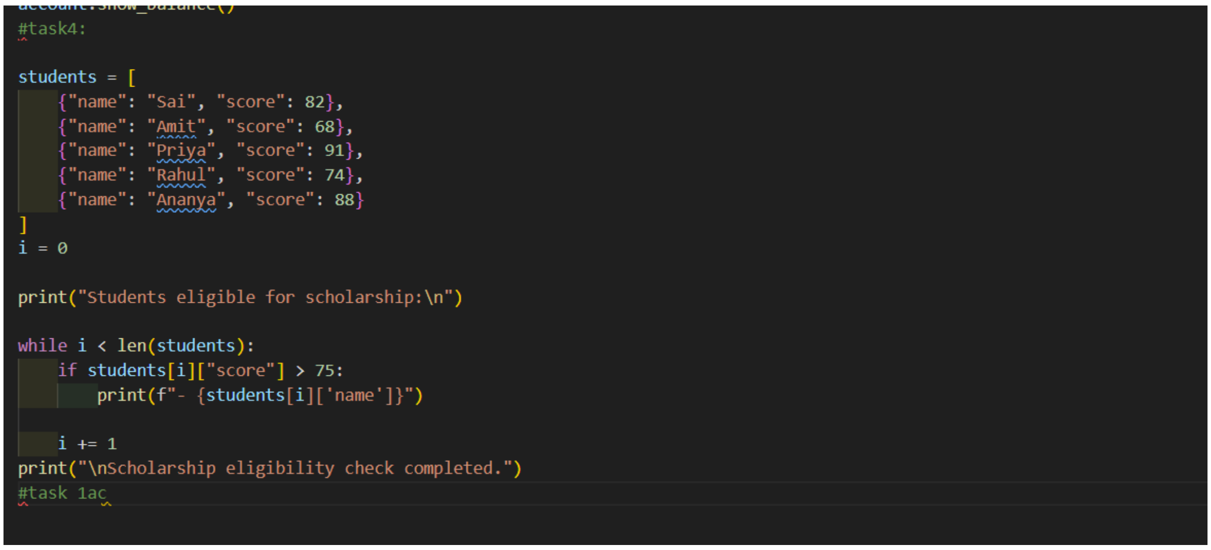
• A complete while loop generated by Copilot

• Correct index handling and condition checks

• Cleanly formatted output listing eligible students

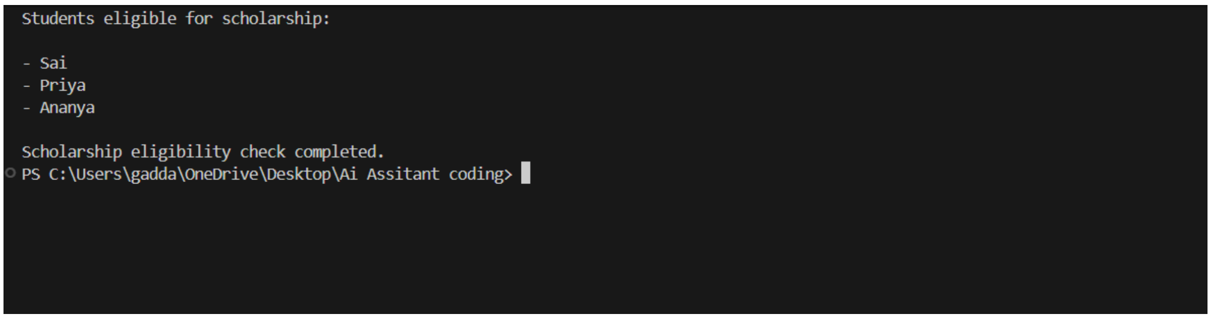
Prompt: Write this next and let Copilot complete the loop

Code:



```
#task4:

students = [
    {"name": "Sai", "score": 82},
    {"name": "Amit", "score": 68},
    {"name": "Priya", "score": 91},
    {"name": "Rahul", "score": 74},
    {"name": "Ananya", "score": 88}
]
i = 0

print("Students eligible for scholarship:\n")

while i < len(students):
    if students[i]["score"] > 75:
        print(f"- {students[i]['name']}")

    i += 1
print("\nScholarship eligibility check completed.")
#task 1ac
```

Ouput:



```
Students eligible for scholarship:

- Sai
- Priya
- Ananya

Scholarship eligibility check completed.
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding>
```

Observation:

The while loop correctly iterates using an index variable.

Conditional checking accurately filters students with scores above 75.

The output clearly lists only scholarship-eligible students in a readable format.

Task5:

Online Shopping Cart Module

Scenario

You are designing a simplified shopping cart system for an e-commerce

website that supports item management and discount calculation.

Task Description

Begin writing a Python class named ShoppingCart with:

• An empty list to store items (each item may include name, price,

quantity)

Use GitHub Copilot to generate methods that:

• Add items to the cart

• Remove items from the cart

• Calculate the total bill using a loop

• Apply conditional discounts (e.g., discount if total exceeds a

certain amount)

Use meaningful comments and method names to guide Copilot.

Expected Outcome

• A fully implemented ShoppingCart class

• Copilot-generated loops and conditional logic

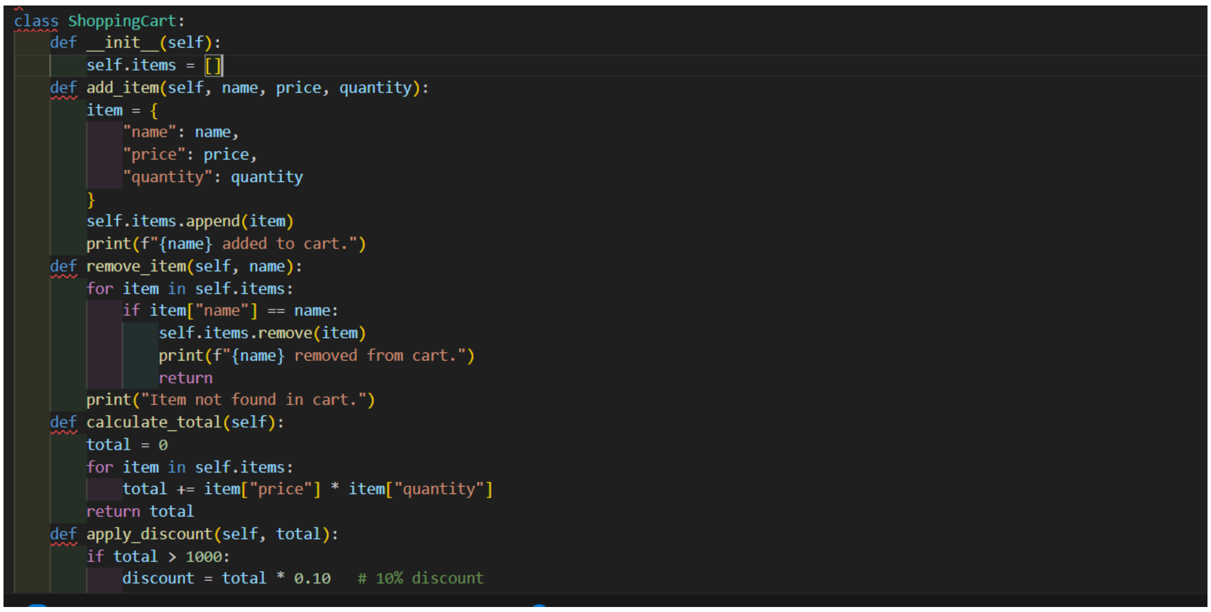
• Correct handling of item addition, removal, and discount

calculation

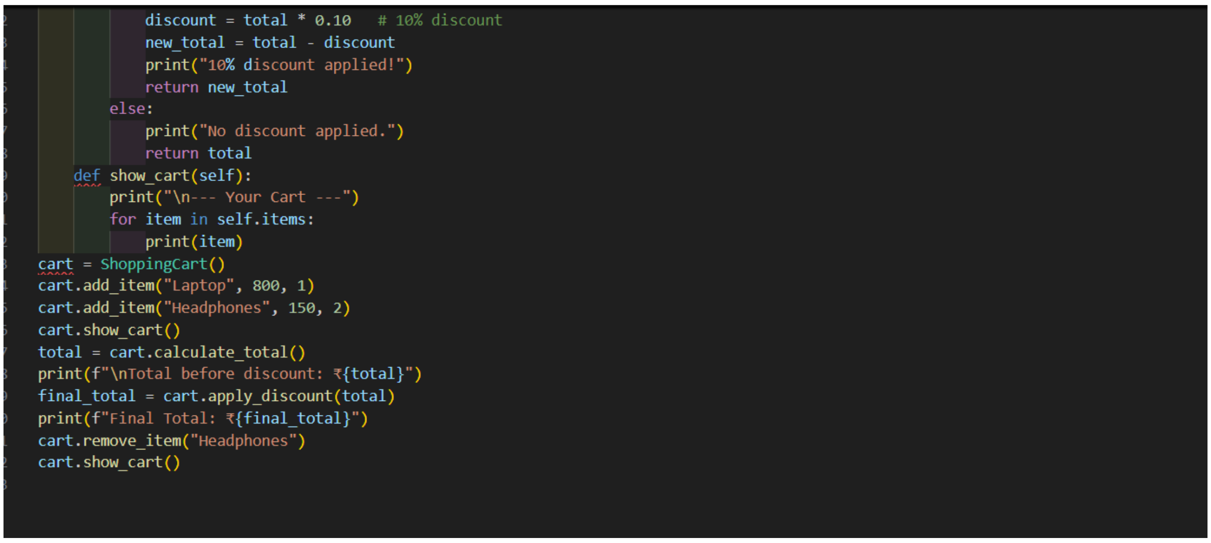
• Sample input/output demonstrating cart funct

Prompt:

Code: Create a ShoppingCart class with an empty list to store items.

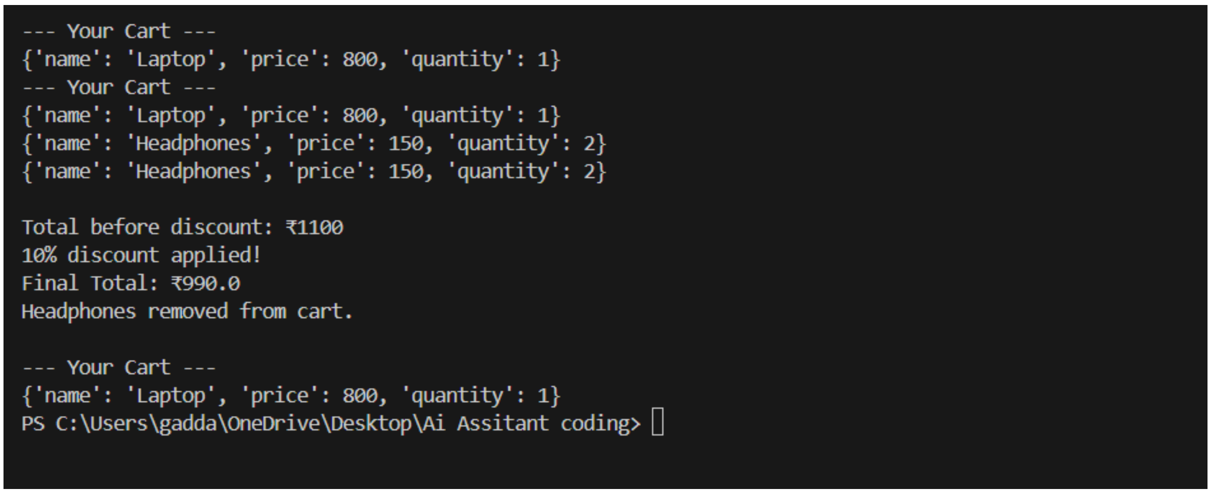


```python
class ShoppingCart:
    def __init__(self):
        self.items = []
    def add_item(self, name, price, quantity):
        item = {
            "name": name,
            "price": price,
            "quantity": quantity
        }
        self.items.append(item)
        print(f"{name} added to cart.")
    def remove_item(self, name):
        for item in self.items:
            if item["name"] == name:
                self.items.remove(item)
                print(f"{name} removed from cart.")
                return
        print("Item not found in cart.")
    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]
        return total
    def apply_discount(self, total):
        if total > 1000:
            discount = total * 0.10   # 10% discount
```



```python
            discount = total * 0.10   # 10% discount
            new_total = total - discount
            print("10% discount applied!")
            return new_total
        else:
            print("No discount applied.")
            return total
    def show_cart(self):
        print("\n--- Your Cart ---")
        for item in self.items:
            print(item)
cart = ShoppingCart()
cart.add_item("Laptop", 800, 1)
cart.add_item("Headphones", 150, 2)
cart.show_cart()
total = cart.calculate_total()
print(f"\nTotal before discount: ₹{total}")
final_total = cart.apply_discount(total)
print(f"Final Total: ₹{final_total}")
cart.remove_item("Headphones")
cart.show_cart()
```

Output:



```
--- Your Cart ---
{'name': 'Laptop', 'price': 800, 'quantity': 1}
--- Your Cart ---
{'name': 'Laptop', 'price': 800, 'quantity': 1}
{'name': 'Headphones', 'price': 150, 'quantity': 2}
{'name': 'Headphones', 'price': 150, 'quantity': 2}

Total before discount: ₹1100
10% discount applied!
Final Total: ₹990.0
Headphones removed from cart.

--- Your Cart ---
{'name': 'Laptop', 'price': 800, 'quantity': 1}
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding> 
```

Observation:

The class effectively manages items using a list of dictionaries.

Copilot-generated loops and conditions correctly compute totals and discounts.

Item addition and removal work reliably with clear user feedback.