SR-UNIVERSITY

ASSIGNMENT:6.4

B.NO :10

Task 1: Student Performance Evaluation System

Scenario

You are building a simple academic management module for a university system where student performance needs to be evaluated automatically.

Task Description

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:

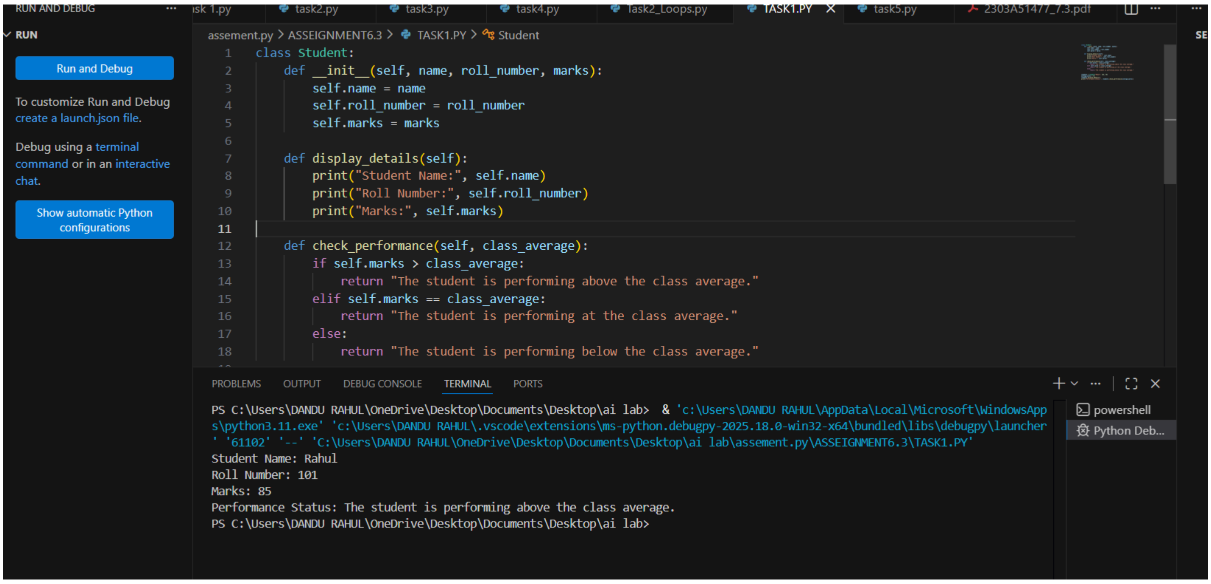
Create a Python class Student with attributes name, roll_number, marks.

Add a method display_details() to print all student information.

Add a method check_performance(class_average) that returns if marks are above, equal, or below the class average.

Code:

```
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("Student Name:", self.name)
        print("Roll Number:", self.roll_number)
        print("Marks:", self.marks)
    def check_performance(self, class_average):
        if self.marks > class_average:
            return "The student is performing above the class average."
        elif self.marks == class_average:
            return "The student is performing at the class average."
        else:
            return "The student is performing below the class average."


student1 = Student("Rahul", 101, 85)
average_marks = 75
student1.display_details()
print("Performance Status:", student1.check_performance(average_marks))
```


RUN AND DEBUG
RUN
Run and Debug
To customize Run and Debug create a launch.json file.
Debug using a terminal command or in an interactive chat.
Show automatic Python configurations
ısk 1.py
task2.py
task3.py
task4.py
Task2_Loops.py
TASK1.PY
task5.py
2303A51477_7.3.pdf
assement.py > ASSEIGNMENT6.3 > TASK1.PY > Student
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
Python Deb...

```
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("Student Name:", self.name)
        print("Roll Number:", self.roll_number)
        print("Marks:", self.marks)

    def check_performance(self, class_average):
        if self.marks > class_average:
            return "The student is performing above the class average."
        elif self.marks == class_average:
            return "The student is performing at the class average."
        else:
            return "The student is performing below the class average."
```

```
PS C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab>  & 'c:\Users\DANDU RAHUL\AppData\Local\Microsoft\WindowsApp
s\python3.11.exe' 'c:\Users\DANDU RAHUL\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher
' '61102' '--' 'C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab\assement.py\ASSEIGNMENT6.3\TASK1.PY'
Student Name: Rahul
Roll Number: 101
Marks: 85
Performance Status: The student is performing above the class average.
PS C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab>
```

analysis:

The `Student` class stores a student's name, roll number, and marks.

`display_details()` shows student info, and `check_performance()` compares marks with the class average.

It uses objects, methods, and `if-elif-else` to evaluate and report performance.

Task 2: 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

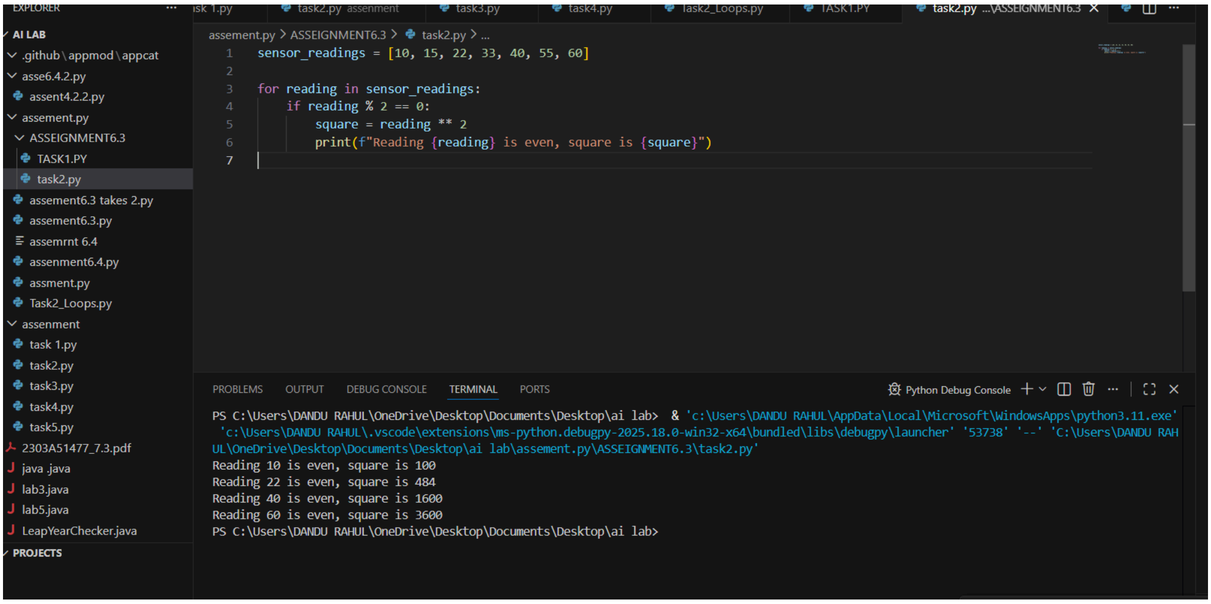
- A complete for loop generated by Copilot
- Use of:
  - o Modulus operator to identify even numbers
  - o Conditional statements
- Correct and formatted output for valid inputs

prompt:

# For each sensor reading in the list, check if it is even.
# If it is even, calculate its square and print: "Reading X is even, square is Y"

Code:

```
sensor_readings = [10, 15, 22, 33, 40, 55, 60]

for reading in sensor_readings:
    if reading % 2 == 0:
        square = reading ** 2
        print(f"Reading {reading} is even, square is {square}")
```



```
sensor_readings = [10, 15, 22, 33, 40, 55, 60]

for reading in sensor_readings:
    if reading % 2 == 0:
        square = reading ** 2
        print(f"Reading {reading} is even, square is {square}")

```

```
PS C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab>  & 'c:\Users\DANDU RAHUL\AppData\Local\Microsoft\WindowsApps\python3.11.exe'
 'c:\Users\DANDU RAHUL\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '53738' '--' 'C:\Users\DANDU RAH
UL\OneDrive\Desktop\Documents\Desktop\ai lab\assement.py\ASSEIGNMENT6.3\task2.py'
Reading 10 is even, square is 100
Reading 22 is even, square is 484
Reading 40 is even, square is 1600
Reading 60 is even, square is 3600
PS C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab>
```

Analysis:

The program goes through sensor readings and picks only the even numbers.It calculates the square of each even reading and prints it.

It uses `%` to check even numbers and `if` to control the logic

Task 3: 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:

Create a Python class BankAccount with attributes account_holder and balance.

Add a deposit(amount) method to add money.

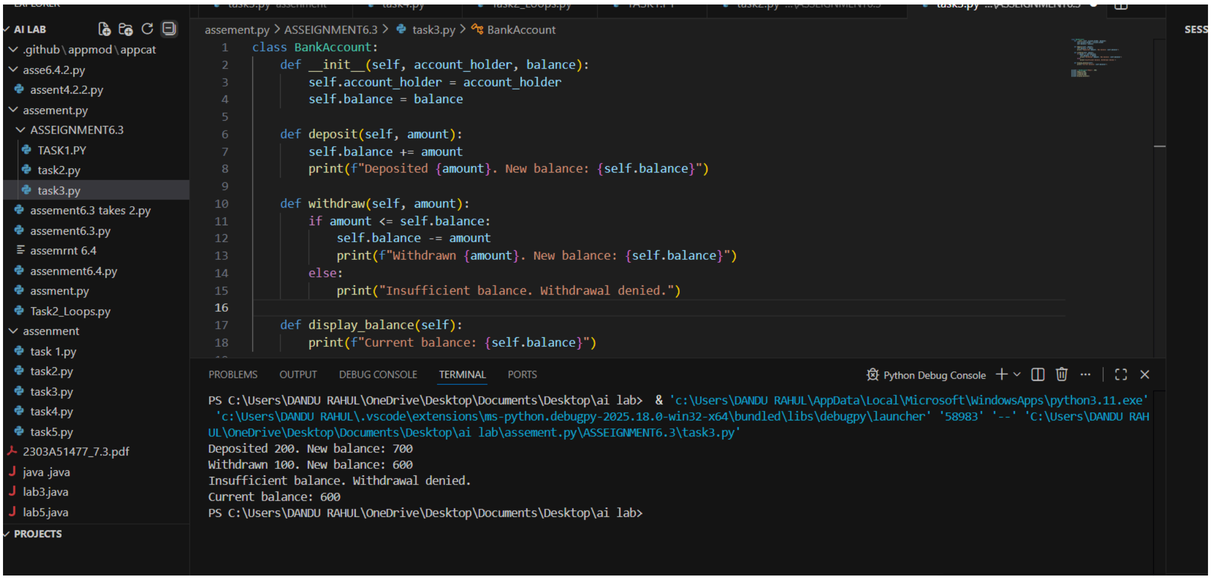
Add a withdraw(amount) method that only allows withdrawal if balance is sufficient, otherwise prints "Insufficient balance".

Add a display_balance() method to show the current balance.

Code:

```
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. New balance: {self.balance}")

    def withdraw(self, amount):
        if amount <= self.balance:
```

```python
            self.balance -= amount
            print(f"Withdrawn {amount}. New balance: {self.balance}")
        else:
            print("Insufficient balance. Withdrawal denied.")

    def display_balance(self):
        print(f"Current balance: {self.balance}")



account = BankAccount("Rahul", 500)
account.deposit(200)
account.withdraw(100)
account.withdraw(700)
account.display_balance()
```



```python
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. New balance: {self.balance}")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawn {amount}. New balance: {self.balance}")
        else:
            print("Insufficient balance. Withdrawal denied.")

    def display_balance(self):
        print(f"Current balance: {self.balance}")
```

```
PS C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab>  & 'c:\Users\DANDU RAHUL\AppData\Local\Microsoft\WindowsApps\python3.11.exe'
 'c:\Users\DANDU RAHUL\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58983' '--' 'C:\Users\DANDU RAH
UL\OneDrive\Desktop\Documents\Desktop\ai lab\assement.py\ASSEIGNMENT6.3\task3.py'
Deposited 200. New balance: 700
Withdrawn 100. New balance: 600
Insufficient balance. Withdrawal denied.
Current balance: 600
PS C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab>
```

Analysis:

The `BankAccount` class stores the account holder’s name and balance.
The `deposit()` method adds money, while `withdraw()` only allows withdrawals if the balance is enough, otherwise it shows a message.
`display_balance()` shows the current balance, using `self` attributes and `if-else` for safe operations.

Task 4: 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:

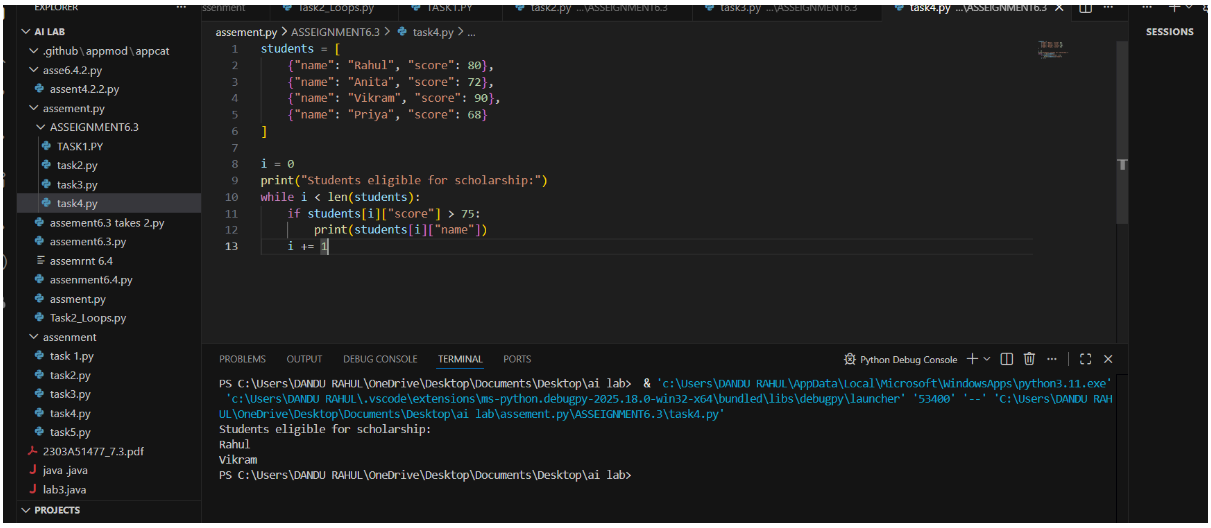
Create a while loop to go through the list of students.

Print the name of each student whose score is greater than 75.

Make sure to handle the index correctly and increment it each time.

Code:

```
students = [

    {"name": "Rahul", "score": 80},

    {"name": "Anita", "score": 72},
    {"name": "Vikram", "score": 90},

    {"name": "Priya", "score": 68}

]



i = 0

print("Students eligible for scholarship:")

while i < len(students):

    if students[i]["score"] > 75:

        print(students[i]["name"])

    i += 1
```



```
students = [
    {"name": "Rahul", "score": 80},
    {"name": "Anita", "score": 72},
    {"name": "Vikram", "score": 90},
    {"name": "Priya", "score": 68}
]

i = 0
print("Students eligible for scholarship:")
while i < len(students):
    if students[i]["score"] > 75:
        print(students[i]["name"])
    i += 1
```

```
PS C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab>  & 'c:\Users\DANDU RAHUL\AppData\Local\Microsoft\WindowsApps\python3.11.exe'
 'c:\Users\DANDU RAHUL\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '53400' '--' 'C:\Users\DANDU RAH
UL\OneDrive\Desktop\Documents\Desktop\ai lab\assement.py\ASSEIGNMENT6.3\task4.py'
Students eligible for scholarship:
Rahul
Vikram
PS C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab>
```

Analysis:

The program stores students as dictionaries in a list with their name and score.
It uses a `while` loop to go through the list and checks if each student scored more than 75.
Eligible students' names are printed, using index-based access and conditional statements.

Task 5: 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)

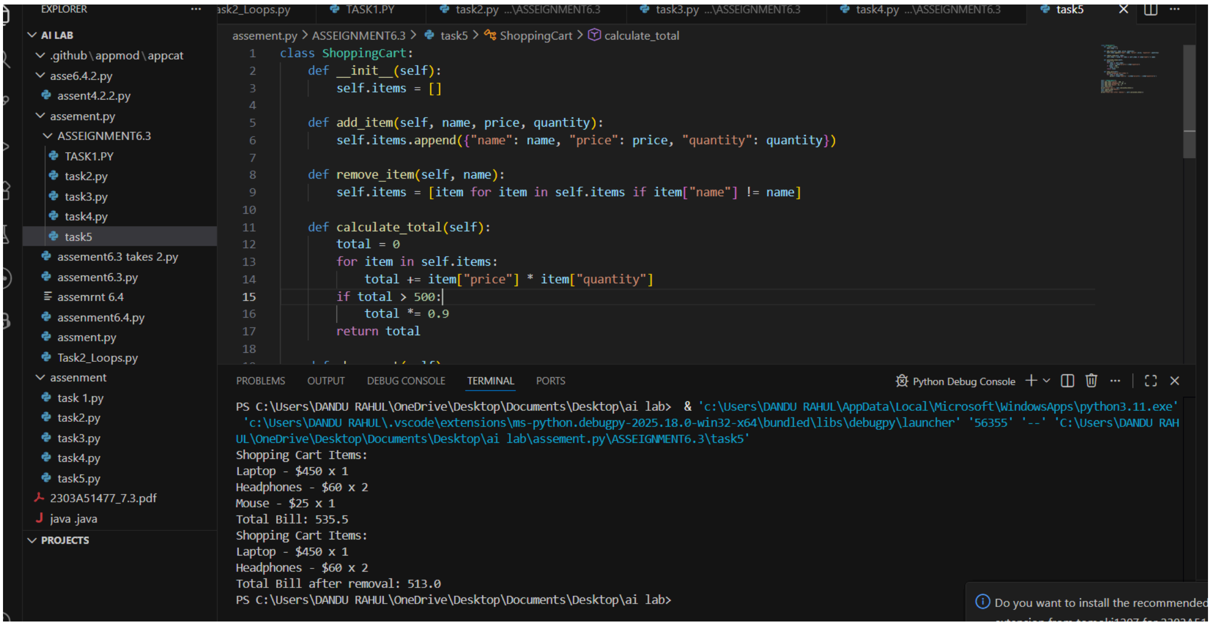
Use GitHub Copilot to generate methods that:
- Add items to the cart
- Remove items from the cart
- Calculate the total bill using a loop
- Apply conditional discounts (e.g., discount if total exceeds a certain amount)

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 functionality

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Code:

```python
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, name, price, quantity):
        self.items.append({"name": name, "price": price, "quantity": quantity})

    def remove_item(self, name):
        self.items = [item for item in self.items if item["name"] != name]

    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]
        if total > 500:
            total *= 0.9
        return total

    def show_cart(self):
        print("Shopping Cart Items:")
        for item in self.items:
            print(f'{item["name"]} - ${item["price"]} x
{item["quantity"]}')


cart = ShoppingCart()
cart.add_item("Laptop", 450, 1)
cart.add_item("Headphones", 60, 2)
cart.add_item("Mouse", 25, 1)
cart.show_cart()
print("Total Bill:", cart.calculate_total())
cart.remove_item("Mouse")
cart.show_cart()
print("Total Bill after removal:", cart.calculate_total())
```



```python
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, name, price, quantity):
        self.items.append({"name": name, "price": price, "quantity": quantity})

    def remove_item(self, name):
        self.items = [item for item in self.items if item["name"] != name]

    def calculate_total(self):
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]
        if total > 500:
            total *= 0.9
        return total

```

```
PS C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab>  & 'c:\Users\DANDU RAHUL\AppData\Local\Microsoft\WindowsApps\python3.11.exe'
 'c:\Users\DANDU RAHUL\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '56355' '--' 'C:\Users\DANDU RAH
UL\OneDrive\Desktop\Documents\Desktop\ai lab\assement.py\ASSEIGNMENT6.3\task5'
Shopping Cart Items:
Laptop - $450 x 1
Headphones - $60 x 2
Mouse - $25 x 1
Total Bill: 535.5
Shopping Cart Items:
Laptop - $450 x 1
Headphones - $60 x 2
Total Bill after removal: 513.0
PS C:\Users\DANDU RAHUL\OneDrive\Desktop\Documents\Desktop\ai lab>
```

Analysis:

The `ShoppingCart` class manages items as dictionaries with name, price, and quantity.
It allows adding and removing items, displaying the cart, and calculating the total bill.
A loop sums the item costs, and conditional logic applies a discount if the total exceeds a set amount, ensuring correct billing.