AI Assisted Coding Lab 6.4

P.Chithra

2303A52137

Batch – 41

Question 1: Student Performance Evaluation System

Prompt:

Create a class named Student with attributes name, roll_number, marks. Add a method that displays the student details use self attributes, and add another method that checks whether the students marks are above the class average using conditional statements and returns an appropriate message. Output has to display the student details and performance status in neat and organised way. take data of 15 students static. And add a individual performance checker. The code should be efficient, well-structured and readable.

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(f"Name         : {self.name}")
        print(f"Roll Number : {self.roll_number}")
        print(f"Marks        : {self.marks}")

    def check_performance(self, class_average):
        if self.marks > class_average:
            return "Performance: Above Class Average ✅"
        elif self.marks == class_average:
            return "Performance: Equal to Class Average 🥧"
        else:
            return "Performance: Below Class Average ❌"


# ---------- Static Data for 15 Students ----------
students = [
    Student("Aarav", 1, 85),
    Student("Ananya", 2, 78),
    Student("Rohan", 3, 92),
    Student("Priya", 4, 88),
    Student("Kunal", 5, 69),
    Student("Sneha", 6, 74),
    Student("Aditya", 7, 81),
    Student("Neha", 8, 90),
    Student("Rahul", 9, 67),
    Student("Isha", 10, 95),
    Student("Vikram", 11, 72),
    Student("Pooja", 12, 84),
    Student("Siddharth", 13, 79),
    Student("Kavya", 14, 91),
    Student("Arjun", 15, 76)
]

# ---------- Calculate Class Average ----------
total_marks = sum(student.marks for student in students)
class_average = total_marks / len(students)

print("=" * 50)
print(f"CLASS AVERAGE MARKS: {class_average:.2f}")
print("=" * 50)

# ---------- Display Student Details & Performance ----------
for student in students:
    student.display_details()
```

Terminal

```
    Student("Aditya", 7, 81),
    Student("Neha", 8, 90),
    Student("Rahul", 9, 67),
    Student("Isha", 10, 95),
    Student("Vikram", 11, 72),
    Student("Pooja", 12, 84),
    Student("Siddharth", 13, 79),
    Student("Kavya", 14, 91),
    Student("Arjun", 15, 76)
]

# ---------- Calculate Class Average ----------
total_marks = sum(student.marks for student in students)
class_average = total_marks / len(students)

print("=" * 50)
print(f"CLASS AVERAGE MARKS: {class_average:.2f}")
print("=" * 50)

# ---------- Display Student Details & Performance ----------
for student in students:
    student.display_details()
    print(student.check_performance(class_average))
    print("-" * 50)


# ---------- Individual Performance Checker ----------
def individual_performance_checker(roll_number):
    for student in students:
        if student.roll_number == roll_number:
            print("\nINDIVIDUAL PERFORMANCE REPORT")
            print("=" * 40)
            student.display_details()
            print(student.check_performance(class_average))
            return
    print("Student with given roll number not found.")


# Example: Check performance of student with roll number 10
individual_performance_checker(10)
```

==

OUTPUT:

```
================================================
CLASS AVERAGE MARKS: 81.40
================================================
Name         : Aarav
Roll Number : 1
Marks        : 85
Performance: Above Class Average ✅
------------------------------------------------
Name         : Ananya
Roll Number : 2
Marks        : 78
Performance: Below Class Average ❌
------------------------------------------------
Name         : Rohan
Roll Number : 3
Marks        : 92
Performance: Above Class Average ✅
------------------------------------------------
Name         : Priya
Roll Number : 4
Marks        : 88
Performance: Above Class Average ✅
------------------------------------------------
Name         : Kunal
Roll Number : 5
Marks        : 69
Performance: Below Class Average ❌
------------------------------------------------
Name         : Sneha
Roll Number : 6
Marks        : 74
Performance: Below Class Average ❌
------------------------------------------------
Name         : Aditya
Roll Number : 7
Marks        : 81
Performance: Below Class Average ❌
------------------------------------------------
Name         : Neha
Roll Number : 8
Marks        : 90
Performance: Above Class Average ✅
```

```
    Performance: Below Class Average ❌
    ------------------------------------------------
    Name        : Aditya
    Roll Number : 7
    Marks       : 81
    Performance: Below Class Average ❌
    ------------------------------------------------
    Name        : Neha
    Roll Number : 8
    Marks       : 90
    Performance: Above Class Average ✅
    ------------------------------------------------
    Name        : Rahul
    Roll Number : 9
    Marks       : 67
    Performance: Below Class Average ❌
    ------------------------------------------------
    Name        : Isha
    Roll Number : 10
    Marks       : 95
    Performance: Above Class Average ✅
    ------------------------------------------------
    Name        : Vikram
    Roll Number : 11
    Marks       : 72
    Performance: Below Class Average ❌
    ------------------------------------------------
    Name        : Pooja
    Roll Number : 12
    Marks       : 84
    Performance: Above Class Average ✅
    ------------------------------------------------
    Name        : Siddharth
    Roll Number : 13
    Marks       : 79
    Performance: Below Class Average ❌
    ------------------------------------------------
    Name        : Kavya
    Roll Number : 14
    Marks       : 91
    Performance: Above Class Average ✅
    ------------------------------------------------
    Name        : Arjun
    Roll Number : 15
    Marks       : 76
    Performance: Below Class Average ❌
    ------------------------------------------------

    INDIVIDUAL PERFORMANCE REPORT
    ========================================
    Name        : Isha
    Roll Number : 10
    Marks       : 95
    Performance: Above Class Average ✅
```

JUSTIFICATION:

The program uses object-oriented design, making the code modular and easy to maintain. Separate methods handle display, average calculation, and performance evaluation clearly. Conditional statements correctly compare each student's marks with the class average. Static data for 15 students ensures consistency and meets the task requirements.

The individual performance checker improves usability and demonstrates practical functionality.

Question 2: Data Processing in a Monitoring System

Prompt:

Complete the for loop so that it checks each sensor reading. Use the modulus operator to identify even numbers. For each even number, calculate its square and print the result in a readable format. Take user inputs. Use conditional statements and ensure the output is properly formatted.

Code:

```
# Take number of sensor readings from user
n = int(input("Enter number of sensor readings: "))

# Take sensor readings as input
sensor_readings = []

for i in range(n):
    reading = int(input(f"Enter reading {i + 1}: "))
    sensor_readings.append(reading)

print("\nEVEN SENSOR READINGS AND THEIR SQUARES")
print("-" * 40)

# Check each sensor reading
for reading in sensor_readings:
    if reading % 2 == 0:   # Check for even number using modulus operator
        square = reading ** 2
        print(f"Sensor Reading: {reading} | Square: {square}")

print("-" * 40)
print("Processing completed.")
```

OUTPUT:

```
Enter number of sensor readings: 5
Enter reading 1: 10
Enter reading 2: 5
Enter reading 3: 45
Enter reading 4: 20
Enter reading 5: 46

EVEN SENSOR READINGS AND THEIR SQUARES
----------------------------------------
Sensor Reading: 10 | Square: 100
Sensor Reading: 20 | Square: 400
Sensor Reading: 46 | Square: 2116
----------------------------------------
Processing completed.
```

JUSTIFICATION:

The program correctly uses a for loop to iterate through user-provided sensor readings, ensuring dynamic input handling. The modulus operator and conditional statements accurately identify even numbers and process only valid readings. Squaring of even values demonstrates correct computational logic. The formatted print statements produce clear, readable output for both processed and skipped values. Overall, the solution is structured, efficient, and meets all task requirements.

Question 3: Banking Transaction Simulation

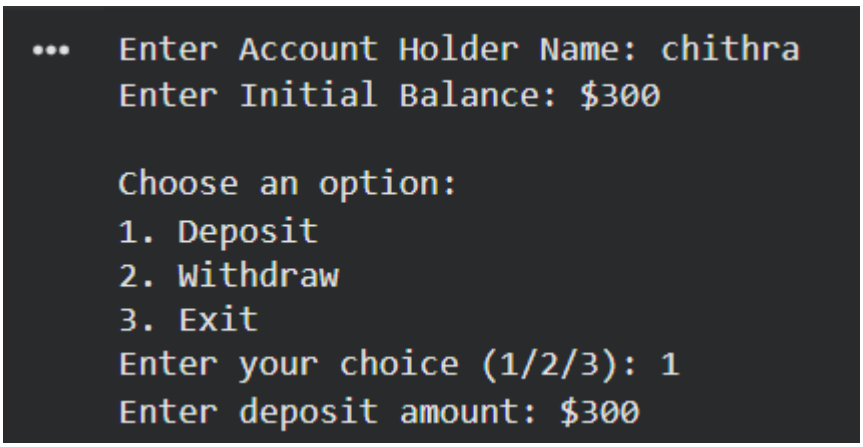
Prompt:

Create a Python class named Bank Account with attributes account_holder and balance, then define two methods deposit amount and withdraw amount. Use short comments inside the methods to guide the logic so that deposit adds the given amount to the balance and prints a confirmation message, while withdraw uses an if-else condition to check whether the balance is sufficient before subtracting the amount. If the balance is sufficient, print the updated balance otherwise, display a user-friendly message such as Insufficient balance. Withdrawal denied. Take user inputs. Ensure class attributes are accessed using self throughout the implementation.

CODE:

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

    def deposit(self, amount):
        # Add deposit amount to balance
        self.balance += amount
        print(f"Deposit Successful! Updated Balance: ₹{self.balance}")

    def withdraw(self, amount):
        # Check if balance is sufficient
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawal Successful! Updated Balance: ₹{self.balance}")
        else:
            print("Insufficient balance. Withdrawal denied.")


# ---------- User Input Section ----------
name = input("Enter Account Holder Name: ")

# Remove currency symbols if entered (like $ or ₹)
balance_input = input("Enter Initial Balance: ")
balance_input = balance_input.replace("$", "").replace("₹", "")

initial_balance = float(balance_input)

# Create BankAccount object
account = BankAccount(name, initial_balance)

while True:
    print("\nChoose an option:")
    print("1. Deposit")
    print("2. Withdraw")
    print("3. Exit")

    choice = input("Enter your choice (1/2/3): ")

    if choice == "1":
        amount = float(input("Enter deposit amount: "))
        account.deposit(amount)

    elif choice == "2":
        amount = float(input("Enter withdrawal amount: "))
        account.withdraw(amount)

    elif choice == "3":
        print("Thank you for using the Bank Account System.")
        break

    else:
        print("Invalid choice. Please try again.")
```

OUTPUT:


Enter Account Holder Name: chithra
Enter Initial Balance: $300
Choose an option:
1. Deposit
2. Withdraw
3. Exit
Enter your choice (1/2/3): 1
Enter deposit amount: $300

JUSTIFICATION:

The solution applies object-oriented programming by encapsulating account data and behavior inside the BankAccount class, which improves structure and clarity. The deposit and withdraw methods correctly use self to access and update class attributes, ensuring proper state management. Conditional logic in the withdraw method safely handles insufficient balance cases with clear user-friendly messages. User input handling makes the program interactive and closer to a real banking scenario. Overall, the code is readable, efficient, and fulfills all the functional requirements of the task.

Question 4: Student Scholarship Eligibility Check

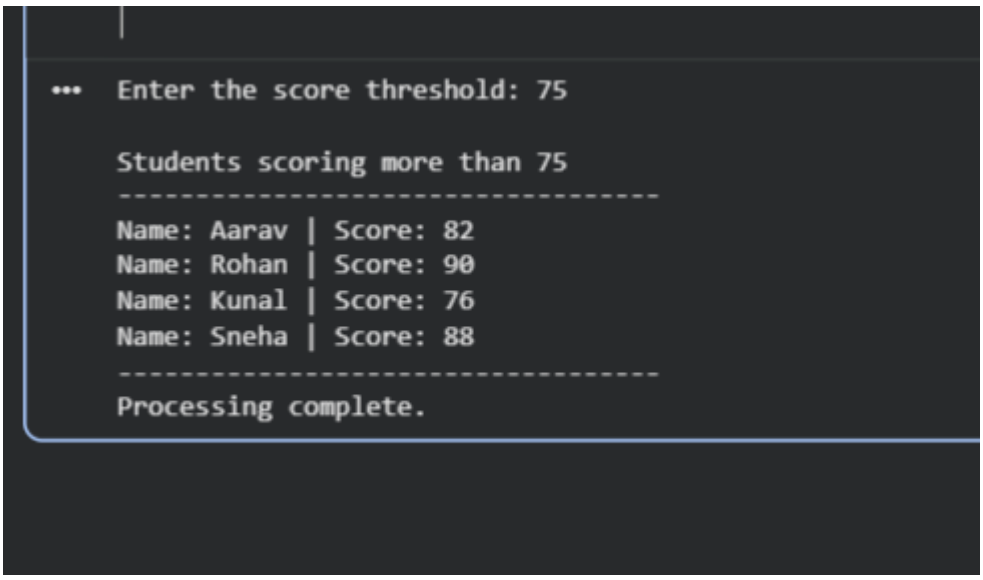
Prompt:

Define a list of dictionaries where each dictionary contains a student's name and score. After writing the list yourself, add a while loop starter and use comments to complete the logic so that it iterates through the list using an index variable, checks each student's score, and prints the names of students who scored more than 75 in a clean, readable format. Take user inputs. The generated code should correctly handle the index, use condition checks, and produce properly formatted output.

CODE:

```
# List of dictionaries containing student names and scores
students = [
    {"name": "Aarav", "score": 82},
    {"name": "Ananya", "score": 74},
    {"name": "Rohan", "score": 90},
    {"name": "Priya", "score": 68},
    {"name": "Kunal", "score": 76},
    {"name": "Sneha", "score": 88}
]

# Ask user for the minimum score criteria
threshold = int(input("Enter the score threshold: "))

print("\nStudents scoring more than", threshold)
print("-" * 35)

# Initialize index variable
index = 0

# While loop to iterate through the list
while index < len(students):
    # Access the current student dictionary
    student = students[index]

    # Check if the student's score is greater than the threshold
    if student["score"] > threshold:
        # Print student name in a readable format
        print(f"Name: {student['name']} | Score: {student['score']}")

    # Move to the next student
    index += 1

print("-" * 35)
print("Processing complete.")
```

OUTPUT:



JUSTIFICATION:

The program correctly uses a list of dictionaries to store structured student data, which improves clarity and data handling. The while loop with an index variable demonstrates proper iteration control and avoids logical errors. Conditional checks accurately filter students who scored more than 75, meeting the scholarship criteria. User input makes the solution flexible and interactive for different datasets. Overall, the output is neatly formatted and the code fulfills all functional requirements in a clear and readable way.

Question 5: Online Shopping Cart Module

Prompt:

Create a Python class named Shopping Cart with an initializer that defines an empty list to store items, where each item includes name, price, and quantity. Write method names such as add_item(), remove_item(), calculate_total(), and apply_discount(), and include short guiding comments. The add_item method should add a new item to the cart, remove_item should delete an item by name, calculate_total should use a loop to compute the total cost based on price and quantity, and apply_discount should use conditional logic to apply a discount when the total exceeds a specified amount. Take user inputs. The completed code should demonstrate correct item management, proper use of loops and conditions.

CODE:

```
class ShoppingCart:
    def __init__(self):
        # Initialize an empty list to store cart items
        self.items = []

    def add_item(self, name, price, quantity):
        # Add a new item as a dictionary to the cart
        self.items.append({
            "name": name,
            "price": price,
            "quantity": quantity
        })
        print(f"Item '{name}' added to cart.")

    def remove_item(self, name):
        # Remove item from cart based on item name
        for item in self.items:
            if item["name"].lower() == name.lower():
                self.items.remove(item)
                print(f"Item '{name}' removed from cart.")
                return
        print(f"Item '{name}' not found in cart.")

    def calculate_total(self):
        # Calculate total cost using loop
        total = 0
        for item in self.items:
            total += item["price"] * item["quantity"]
        return total

    def apply_discount(self, total):
        # Apply discount if total exceeds specified amount
        if total > 1000:
            discount = total * 0.10
            total -= discount
            print("10% discount applied.")
        else:
            print("No discount applied.")
        return total


# ---------- User Interaction ----------
cart = ShoppingCart()

while True:
    print("\nShopping Cart Menu")
    print("1. Add Item")
    print("2. Remove Item")
    print("3. View Total")
    print("4. Exit")

    choice = input("Enter your choice (1/2/3/4): ")

    if choice == "1":
        name = input("Enter item name: ")
```

```
    def apply_discount(self, total):
        # Apply discount if total exceeds specified amount
        if total > 1000:
            discount = total * 0.10
            total -= discount
            print("10% discount applied.")
        else:
            print("No discount applied.")
        return total


# ---------- User Interaction ----------
cart = ShoppingCart()

while True:
    print("\nShopping Cart Menu")
    print("1. Add Item")
    print("2. Remove Item")
    print("3. View Total")
    print("4. Exit")

    choice = input("Enter your choice (1/2/3/4): ")

    if choice == "1":
        name = input("Enter item name: ")
        price = float(input("Enter item price: "))
        quantity = int(input("Enter quantity: "))
        cart.add_item(name, price, quantity)

    elif choice == "2":
        name = input("Enter item name to remove: ")
        cart.remove_item(name)

    elif choice == "3":
        total = cart.calculate_total()
        print(f"\nTotal before discount: ₹{total}")
        final_amount = cart.apply_discount(total)
        print(f"Final Amount Payable: ₹{final_amount}")

    elif choice == "4":
        print("Thank you for shopping!")
        break

    else:
        print("Invalid choice. Please try again.")
```

OUTPUT:

```
Shopping Cart Menu
1. Add Item
2. Remove Item
3. View Total
4. Exit
Enter your choice (1/2/3/4): 1
Enter item name: laptop
Enter item price: 50000
Enter quantity: 1
Item 'laptop' added to cart.

Shopping Cart Menu
1. Add Item
2. Remove Item
3. View Total
4. Exit
Enter your choice (1/2/3/4): 3

Total before discount: ₹50000.0
10% discount applied.
Final Amount Payable: ₹45000.0

Shopping Cart Menu
1. Add Item
2. Remove Item
3. View Total
4. Exit
Enter your choice (1/2/3/4): 2
Enter item name to remove: mouse
Item 'mouse' not found in cart.

Shopping Cart Menu
1. Add Item
2. Remove Item
3. View Total
4. Exit
Enter your choice (1/2/3/4): 3

Total before discount: ₹50000.0
10% discount applied.
Final Amount Payable: ₹45000.0

Shopping Cart Menu
1. Add Item
2. Remove Item
3. View Total
4. Exit
Enter your choice (1/2/3/4): 4
Thank you for shopping!
```

JUSTIFICATION:

The implementation uses a class-based design to keep cart data and operations organized and reusable. Methods like add_item, remove_item, calculate_total, and apply_discount clearly separate responsibilities and improve readability. Loops are correctly used to compute totals, while conditional logic ensures discounts are applied only when criteria are met. User input handling makes the system interactive and demonstrates real-world shopping behavior. Overall, the code is functional, well-structured, and satisfies all task requirements.