AI Assisted Coding Lab 6.4

Veekshitha Adharasani

2303A52175

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:

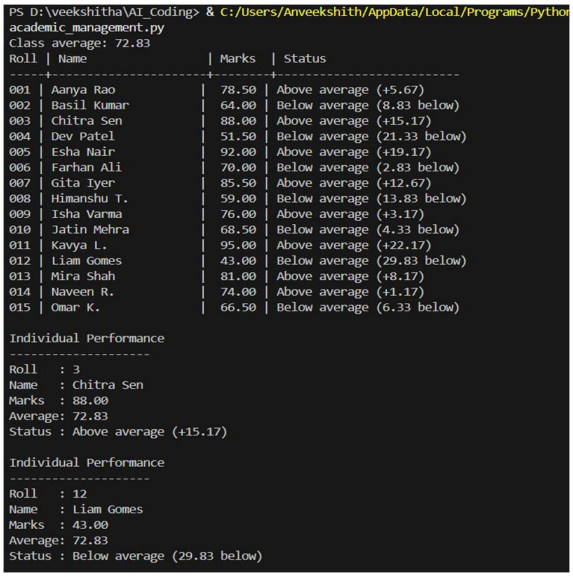
```python
class Student:
    """Represents a student with a name, roll number and marks."""

    def __init__(self, name: str, roll_number: int, marks: float):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_details(self) -> str:
        """Return a neatly formatted string with the student's details."""
        return f"{self.roll_number:03d} | {self.name:<20} | {self.marks:6.2f}"

    def performance_against_average(self, class_average: float) -> str:
        """Return a message indicating whether this student's marks are above the class a

        Uses a simple conditional check and returns a human-friendly message.
        """
        if self.marks > class_average:
            return f"Above average (+{self.marks - class_average:0.2f})"
        elif self.marks < class_average:
            return f"Below average ({class_average - self.marks:0.2f} below)"
        else:
            return "At class average"
```

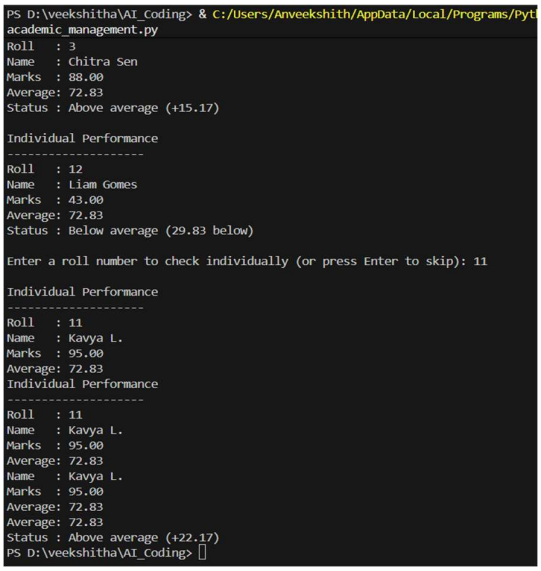
```python
def compute_class_average(students: list[Student]) -> float:
    """Compute and return the average marks for the given students."""
    if not students:
        return 0.0
    return sum(s.marks for s in students) / len(students)


def display_all_students(students: list[Student]) -> None:
    """Display all students with a neat table and performance status."""
    avg = compute_class_average(students)
    header = "Roll | Name                 | Marks  | Status"
    sep = "-----+----------------------+--------+------------------------"
    print("Class average: {:.2f}".format(avg))
    print(header)
    print(sep)
    for s in students:
        print(f"{s.display_details()} | {s.performance_against_average(avg)}")


def find_student_by_roll(students: list[Student], roll: int) -> Student | None:
    """Return the Student with the given roll number, or None if not found."""
    for s in students:
        if s.roll_number == roll:
            return s
    return None


def individual_performance_checker(students: list[Student], roll: int) -> None:
    """Print the individual student's details and performance compared to class average."""
    avg = compute_class_average(students)
    student = find_student_by_roll(students, roll)
    if not student:
        print(f"No student found with roll number {roll}")
        return
    print("\nIndividual Performance")
    print("----------------------")
    print(f"Roll    : {student.roll_number}")
```

```python
def find_student_by_roll(students: list[Student], roll: int) -> Student | None:
    """Return the Student with the given roll number, or None if not found."""
    for s in students:
        if s.roll_number == roll:
            return s
    return None


def individual_performance_checker(students: list[Student], roll: int) -> None:
    """Print the individual student's details and performance compared to class average."""
    avg = compute_class_average(students)
    student = find_student_by_roll(students, roll)
    if not student:
        print(f"No student found with roll number {roll}")
        return
    print("\nIndividual Performance")
    print("----------------------")
    print(f"Roll    : {student.roll_number}")
    print(f"Name    : {student.name}")
    print(f"Marks   : {student.marks:.2f}")
    print(f"Average: {avg:.2f}")
    print(f"Status  : {student.performance_against_average(avg)}")


def _static_students() -> list[Student]:
    """Return a list of 15 statically-defined students."""
    raw = [
        ("Aanya Rao", 1, 78.5),
        ("Basil Kumar", 2, 64.0),
        ("Chitra Sen", 3, 88.0),
        ("Dev Patel", 4, 51.5),
        ("Esha Nair", 5, 92.0),
        ("Farhan Ali", 6, 70.0),
        ("Gita Iyer", 7, 85.5),
        ("Himanshu T.", 8, 59.0),
```

```python
def _static_students() -> list[Student]:
        ("Himanshu T.", 8, 59.0),
        ("Isha Varma", 9, 76.0),
        ("Jatin Mehra", 10, 68.5),
        ("Kavya L.", 11, 95.0),
        ("Liam Gomes", 12, 43.0),
        ("Mira Shah", 13, 81.0),
        ("Naveen R.", 14, 74.0),
        ("Omar K.", 15, 66.5),
    ]
    return [Student(name, roll, marks) for name, roll, marks in raw]


def main() -> None:
    students = _static_students()
    display_all_students(students)

    # Demonstrate individual checker for a couple of examples
    individual_performance_checker(students, 3)
    individual_performance_checker(students, 12)

    # Optional interactive check (only runs when user wants to)
    try:
        resp = input('\nEnter a roll number to check individually (or press Enter to skip): ').strip()
        if resp:
            try:
                r = int(resp)
                individual_performance_checker(students, r)
            except ValueError:
                print('Invalid roll number input; skipping individual check.')
    except (EOFError, KeyboardInterrupt):
        # Non-interactive environment: ignore
        pass


if __name__ == '__main__':
    main()
```

OUTPUT:





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:

```python
# Take user input for number of sensor readings
num_readings = int(input("Enter the number of sensor readings: "))

# Initialize list to store sensor readings
sensor_readings = []

# Take sensor readings from user
for i in range(num_readings):
    reading = int(input(f"Enter sensor reading {i + 1}: "))
    sensor_readings.append(reading)

# Process sensor readings
print("\n" + "="*50)
print("SENSOR READING ANALYSIS - EVEN NUMBERS")
print("="*50)

for reading in sensor_readings:
    # Use modulus operator to check if number is even
    if reading % 2 == 0:
        square = reading ** 2
        print(f"Sensor Reading: {reading:>5} | Square: {square:>8} | Status: EVE
    else:
        print(f"Sensor Reading: {reading:>5} | Status: ODD (Skipped)")

print("="*50)
```

OUTPUT:

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

================================================
SENSOR READING ANALYSIS - EVEN NUMBERS
================================================
Sensor Reading:    10 | Square:      100 | Status: EVEN
================================================
Sensor Reading:    10 | Square:      100 | Status: EVEN
Sensor Reading:    10 | Square:      100 | Status: EVEN
Sensor Reading:     5 | Status: ODD (Skipped)
Sensor Reading:    45 | Status: ODD (Skipped)
Sensor Reading:    45 | Status: ODD (Skipped)
Sensor Reading:    20 | Square:      400 | Status: EVEN
Sensor Reading:    20 | Square:      400 | Status: EVEN
Sensor Reading:    46 | Square:     2116 | Status: EVEN
================================================
PS D:\veekshitha\AI_Coding>
```

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:
    # Constructor to initialize account holder and balance
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance

    # Method to deposit amount
    def deposit(self, amount):
        # Check if amount is positive
        if amount > 0:
            # Add amount to balance
            self.balance += amount
            # Print confirmation message
            print(f"✓ Deposit successful! ${amount:.2f} added to account.")
            print(f"  Current balance: ${self.balance:.2f}\n")
        else:
            print("X Invalid amount. Please enter a positive value.\n")

    # Method to withdraw amount
    def withdraw(self, amount):
        # Check if amount is positive
        if amount <= 0:
            print("X Invalid amount. Please enter a positive value.\n")
        # Check if balance is sufficient
        elif amount > self.balance:
            print("X Insufficient balance. Withdrawal denied.")
            print(f"  Available balance: ${self.balance:.2f}\n")
        else:
            # Subtract amount from balance
            self.balance -= amount
            # Print updated balance
            print(f"✓ Withdrawal successful! ${amount:.2f} withdrawn from account.")
            print(f"  Current balance: ${self.balance:.2f}\n")

    # Method to display account details
    def display_balance(self):
        print(f"Account Holder: {self.account_holder}")
```

```python
name = input("Enter account holder name: ")
# Get initial balance
initial_balance = float(input("Enter initial balance: $"))

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

print("\n" + "="*40)
print("BANKING SYSTEM")
print("="*40 + "\n")

# Display initial account details
account.display_balance()

# Menu-driven operations
while True:
    print("1. Deposit")
    print("2. Withdraw")
    print("3. Check Balance")
    print("4. Exit")

    choice = input("Select an option (1-4): ")

    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":
        account.display_balance()
    elif choice == "4":
        print("Thank you for using our banking system. Goodbye!")
        break
    else:
        print("X Invalid option. Please try again.\n")
```

OUTPUT:

```
Enter account holder name: vicky
Enter initial balance: $300

========================================
BANKING SYSTEM
========================================

Account Holder: vicky
Current Balance: $300.00

1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Select an option (1-4): 1
Enter deposit amount: $300
✓ Deposit successful! $300.00 added to account.
  Current balance: $600.00

1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Select an option (1-4): 3
Account Holder: vicky
Current Balance: $600.00

1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Select an option (1-4): 2
Enter withdrawal amount: $100
✓ Withdrawal successful! $100.00 withdrawn from account.
  Current balance: $500.00

1. Deposit
2. Withdraw
1. Deposit
```

```
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Select an option (1-4): 2
Enter withdrawal amount: $100
✓ Withdrawal successful! $100.00 withdrawn from account.
  Current balance: $500.00

1. Deposit
2. Withdraw
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Select an option (1-4): 4
Thank you for using our banking system. Goodbye!
PS D:\veekshitha\AI_Coding>
```

JUSTIFICATION:

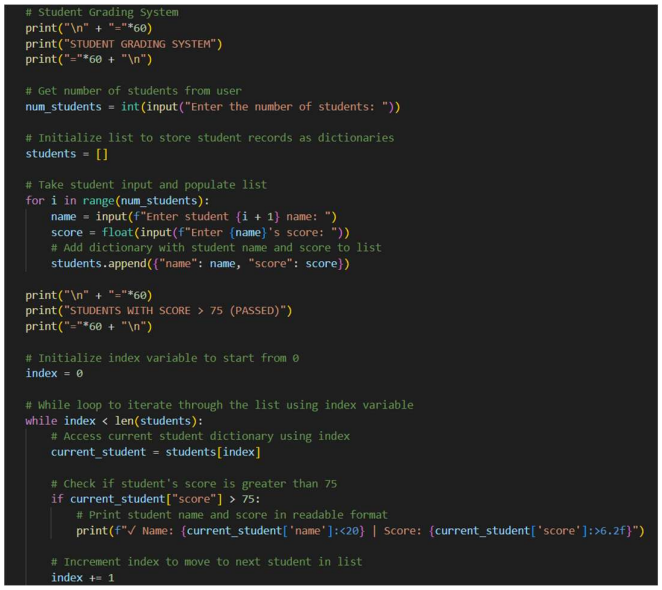
The solution applies object-oriented programming by encapsulating account data and behavior inside the BankAccount class, which improves structure and clarity. The deposit and withdraw methods correctly use self to access and update class attributes, ensuring proper state management. Conditional logic 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

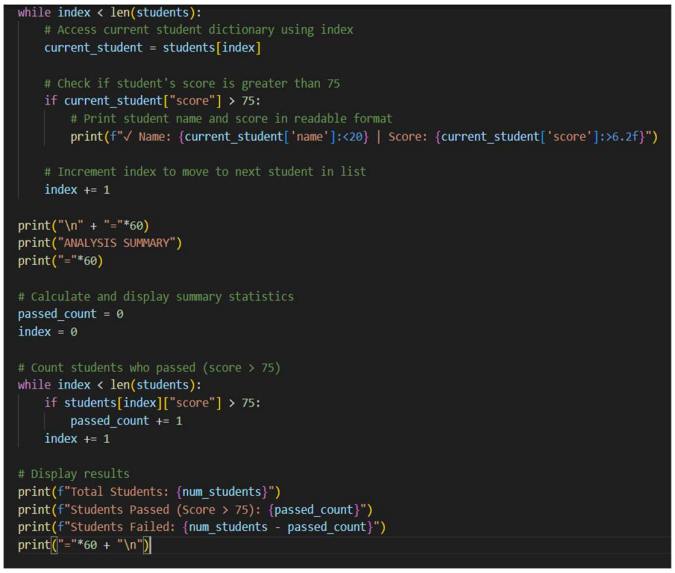
Prompt:

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:

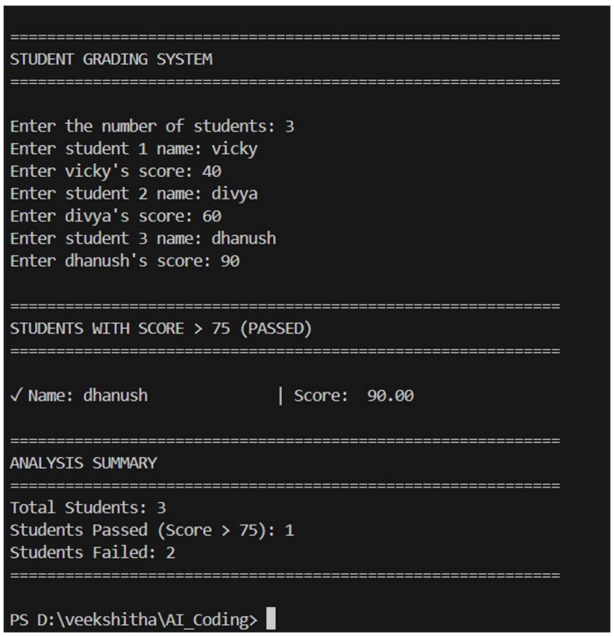


```
# Student Grading System
print("\n" + "="*60)
print("STUDENT GRADING SYSTEM")
print("="*60 + "\n")

# Get number of students from user
num_students = int(input("Enter the number of students: "))

# Initialize list to store student records as dictionaries
students = []

# Take student input and populate list
for i in range(num_students):
    name = input(f"Enter student {i + 1} name: ")
    score = float(input(f"Enter {name}'s score: "))
    # Add dictionary with student name and score to list
    students.append({"name": name, "score": score})

print("\n" + "="*60)
print("STUDENTS WITH SCORE > 75 (PASSED)")
print("="*60 + "\n")

# Initialize index variable to start from 0
index = 0

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

    # Check if student's score is greater than 75
    if current_student["score"] > 75:
        # Print student name and score in readable format
        print(f"✓ Name: {current_student['name']:<20} | Score: {current_student['score']:>6.2f}")

    # Increment index to move to next student in list
    index += 1
```



```
while index < len(students):
    # Access current student dictionary using index
    current_student = students[index]

    # Check if student's score is greater than 75
    if current_student["score"] > 75:
        # Print student name and score in readable format
        print(f"✓ Name: {current_student['name']:<20} | Score: {current_student['score']:>6.2f}")

    # Increment index to move to next student in list
    index += 1

print("\n" + "="*60)
print("ANALYSIS SUMMARY")
print("="*60)

# Calculate and display summary statistics
passed_count = 0
index = 0

# Count students who passed (score > 75)
while index < len(students):
    if students[index]["score"] > 75:
        passed_count += 1
    index += 1

# Display results
print(f"Total Students: {num_students}")
print(f"Students Passed (Score > 75): {passed_count}")
print(f"Students Failed: {num_students - passed_count}")
print("="*60 + "\n")
```

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:

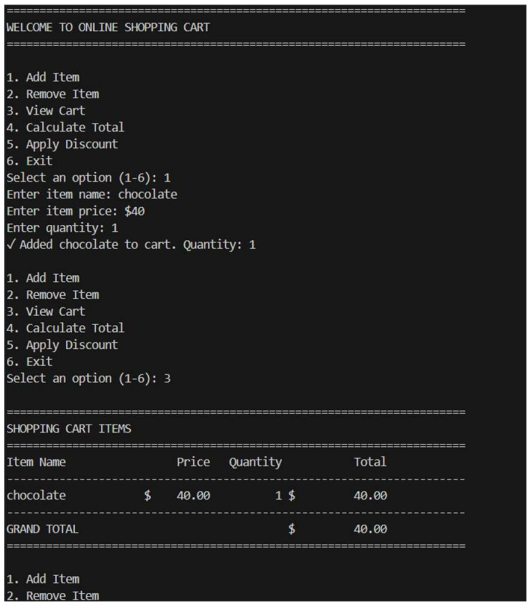
```python
class ShoppingCart:
    # Initializer to create empty cart
    def __init__(self):
        self.items = []

    # Method to add item to cart
    def add_item(self, name, price, quantity):
        # Check if item already exists in cart
        for item in self.items:
            if item["name"].lower() == name.lower():
                # Update quantity if item exists
                item["quantity"] += quantity
                print(f"✓ Updated {name}. New quantity: {item['quantity']}\n")
                return

        # Add new item to cart if it doesn't exist
        self.items.append({"name": name, "price": price, "quantity": quantity})
        print(f"✓ Added {name} to cart. Quantity: {quantity}\n")

    # Method to remove item from cart by name
    def remove_item(self, name):
        # Loop through items to find matching name
        for i, item in enumerate(self.items):
            if item["name"].lower() == name.lower():
                # Remove item from list
                removed_item = self.items.pop(i)
                print(f"✓ Removed {removed_item['name']} from cart.\n")
                return

        # Print message if item not found
        print(f"✗ Item '{name}' not found in cart.\n")

    # Method to calculate total cost of all items
    def calculate_total(self):
        # Initialize total to 0
        total = 0
```

```python
class ShoppingCart:
    def calculate_total(self):
        # Loop through each item in cart
        for item in self.items:
            # Calculate item cost (price * quantity) and add to total
            item_cost = item["price"] * item["quantity"]
            total += item_cost

        # Return total amount
        return total

    # Method to apply discount based on total amount
    def apply_discount(self, discount_threshold, discount_percent):
        # Calculate total without discount
        total = self.calculate_total()

        # Check if total exceeds discount threshold
        if total > discount_threshold:
            # Calculate discount amount
            discount_amount = total * (discount_percent / 100)
            # Calculate final price after discount
            final_total = total - discount_amount
            print(f"✓ Discount Applied!")
            print(f"  Original Total: ${total:.2f}")
            print(f"  Discount ({discount_percent}%): -${discount_amount:.2f}")
            print(f"  Final Total: ${final_total:.2f}\n")
            return final_total
        else:
            # No discount applied
            print(f"✗ Total does not exceed ${discount_threshold}. No discount applied.")
            print(f"  Current Total: ${total:.2f}\n")
            return total

    # Method to display all items in cart
    def display_cart(self):
        # Check if cart is empty
        if not self.items:
```

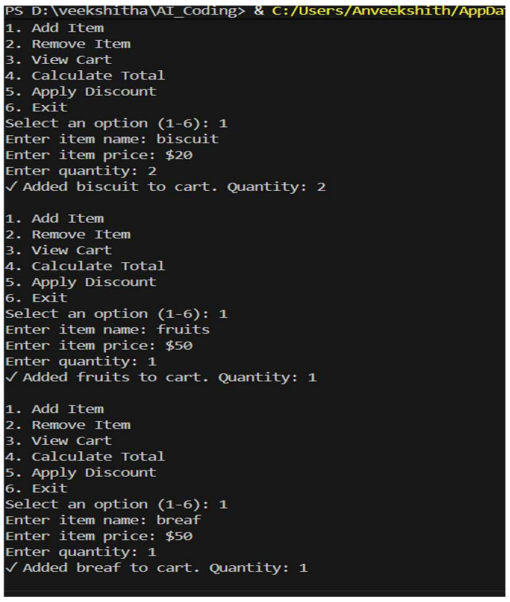
```python
class ShoppingCart:
    # Method to display all items in cart
    def display_cart(self):
        # Check if cart is empty
        if not self.items:
            print("✗ Cart is empty.\n")
            return

        # Display cart header
        print("\n" + "="*70)
        print("SHOPPING CART ITEMS")
        print("="*70)
        print(f"{'Item Name':<20} {'Price':>10} {'Quantity':>10} {'Total':>15}")
        print("-"*70)

        # Loop through items and display each one
        for item in self.items:
            item_total = item["price"] * item["quantity"]
            print(f"{item['name']:<20} ${item['price']:>9.2f} {item['quantity']:>10} ${item_total:>14.2f}")

        print("-"*70)
        # Display grand total
        print(f"{'GRAND TOTAL':<20} {' ':>10} {' ':>10} ${self.calculate_total():>14.2f}")
        print("="*70 + "\n")


# Main program - Shopping Cart System
if __name__ == "__main__":
    # Create shopping cart object
    cart = ShoppingCart()

    print("="*70)
    print("WELCOME TO ONLINE SHOPPING CART")
    print("="*70 + "\n")

    # Menu-driven shopping system
```

```python
        choice = input("Select an option (1-6): ")

        if choice == "1":
            # Add item to cart
            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":
            # Remove item from cart
            name = input("Enter item name to remove: ")
            cart.remove_item(name)

        elif choice == "3":
            # Display cart contents
            cart.display_cart()

        elif choice == "4":
            # Show total without discount
            total = cart.calculate_total()
            print(f"\nCart Total: ${total:.2f}\n")

        elif choice == "5":
            # Apply discount
            threshold = float(input("Enter discount threshold amount: $"))
            percent = float(input("Enter discount percentage: "))
            cart.apply_discount(threshold, percent)

        elif choice == "6":
            print("Thank you for shopping! Goodbye!\n")
            break

        else:
            print("✗ Invalid option. Please try again.\n")
```

OUTPUT:



```
============================================================
WELCOME TO ONLINE SHOPPING CART
============================================================

1. Add Item
2. Remove Item
3. View Cart
4. Calculate Total
5. Apply Discount
6. Exit
Select an option (1-6): 1
Enter item name: chocolate
Enter item price: $40
Enter quantity: 1
✓ Added chocolate to cart. Quantity: 1

1. Add Item
2. Remove Item
3. View Cart
4. Calculate Total
5. Apply Discount
6. Exit
Select an option (1-6): 3

============================================================
SHOPPING CART ITEMS
============================================================
Item Name                Price   Quantity          Total
------------------------------------------------------------
chocolate            $    40.00          1 $         40.00
------------------------------------------------------------
GRAND TOTAL                                $         40.00
============================================================

1. Add Item
2. Remove Item
```



```
PS D:\veekshitha\AI_Coding> & C:/Users/Anveekshith/AppDa
1. Add Item
2. Remove Item
3. View Cart
4. Calculate Total
5. Apply Discount
6. Exit
Select an option (1-6): 1
Enter item name: biscuit
Enter item price: $20
Enter quantity: 2
✓ Added biscuit to cart. Quantity: 2

1. Add Item
2. Remove Item
3. View Cart
4. Calculate Total
5. Apply Discount
6. Exit
Select an option (1-6): 1
Enter item name: fruits
Enter item price: $50
Enter quantity: 1
✓ Added fruits to cart. Quantity: 1

1. Add Item
2. Remove Item
3. View Cart
4. Calculate Total
5. Apply Discount
6. Exit
Select an option (1-6): 1
Enter item name: breaf
Enter item price: $50
Enter quantity: 1
✓ Added breaf to cart. Quantity: 1
```

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.