Name: K.Likitha

Hall Ticket: 2303A52144

Batch: 41

Task 1: Student Performance Evaluation System

Prompt:

Write a method to display student details

Write a method to check if marks are above class average and return a message

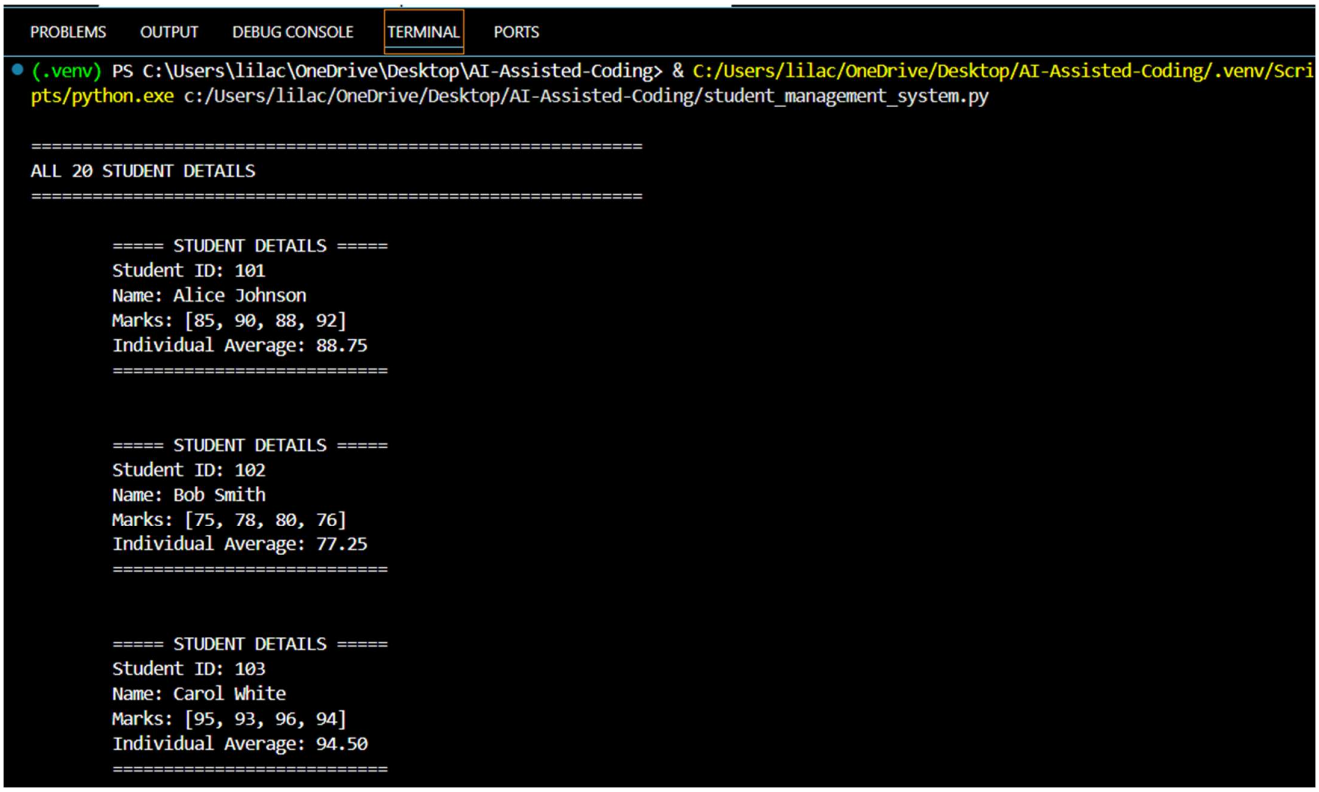
Code:

```
class Student:
    """A class to manage student information and performance."""

    def __init__(self, student_id, name, marks):
        """
        Initialize a student with ID, name, and marks.

        Args:
            student_id: Unique identifier for the student
            name: Name of the student
            marks: List of marks/scores
        """
        self.student_id = student_id
        self.name = name
        self.marks = marks

    def display_details(self):
        """
        Display comprehensive student details including name, ID, marks, and average.

        Returns:
            str: Formatted string with student information
        """
        average = sum(self.marks) / len(self.marks) if self.marks else 0
        marks_str = ", ".join(map(str, self.marks))

        details = f"""
        ===== STUDENT DETAILS =====
        Student ID: {self.student_id}
        Name: {self.name}
        Marks: [{marks_str}]
        Individual Average: {average:.2f}
        ============================
        """
        return details

    def check_above_class_average(self, class_average):
        """
        Check if student's average marks are above the class average.

        Args:
```

```
            class_average: The average marks of the class

        Returns:
            str: Message indicating whether student is above or below class average
        """
        student_average = sum(self.marks) / len(self.marks) if self.marks else 0

        if student_average > class_average:
            message = f"{self.name} ({student_average:.2f}) is ABOVE the class average
({class_average:.2f})"
            return message
        elif student_average == class_average:
            message = f"{self.name} ({student_average:.2f}) is EQUAL to the class average
({class_average:.2f})"
            return message
        else:
            message = f"{self.name} ({student_average:.2f}) is BELOW the class average
({class_average:.2f})"
            return message


# Example usage with 20 students
if __name__ == "__main__":
    # Create 20 student objects
    students = [
        Student(101, "Alice Johnson", [85, 90, 88, 92]),
        Student(102, "Bob Smith", [75, 78, 80, 76]),
        Student(103, "Carol White", [95, 93, 96, 94]),
        Student(104, "David Brown", [88, 85, 89, 87]),
        Student(105, "Emma Davis", [92, 94, 91, 93]),
        Student(106, "Frank Miller", [72, 75, 73, 74]),
        Student(107, "Grace Lee", [98, 96, 99, 97]),
        Student(108, "Henry Wilson", [80, 82, 81, 83]),
        Student(109, "Iris Taylor", [87, 89, 88, 90]),
        Student(110, "Jack Anderson", [76, 78, 77, 79]),
        Student(111, "Karen Martinez", [91, 93, 92, 94]),
        Student(112, "Leo Thomas", [84, 86, 85, 87]),
        Student(113, "Mia Jackson", [79, 81, 80, 82]),
        Student(114, "Noah White", [89, 91, 90, 92]),
        Student(115, "Olivia Harris", [94, 96, 95, 97]),
        Student(116, "Peter Martin", [81, 83, 82, 84]),
        Student(117, "Quinn Clark", [86, 88, 87, 89]),
        Student(118, "Rachel Lewis", [93, 95, 94, 96]),
        Student(119, "Sam Walker", [77, 79, 78, 80]),
        Student(120, "Tina Hall", [90, 92, 91, 93])
    ]

    # Display all student details
    print("\n" + "="*60)
    print("ALL 20 STUDENT DETAILS")
    print("="*60)
    for student in students:
        print(student.display_details())
```

```python
# Calculate class average
all_marks = []
for student in students:
    all_marks.extend(student.marks)
class_average = sum(all_marks) / len(all_marks) if all_marks else 0

print(f"\n{'='*60}")
print(f"CLASS AVERAGE (20 Students): {class_average:.2f}")
print(f"{'='*60}\n")

# Check each student against class average
print("="*60)
print("PERFORMANCE COMPARISON - ABOVE/BELOW CLASS AVERAGE")
print("="*60)
for student in students:
    print(student.check_above_class_average(class_average))
print("="*60)
```

Output:



```
(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> & C:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/.venv/Scripts/python.exe c:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/student_management_system.py
        ==========


        ===== STUDENT DETAILS =====
        Student ID: 118
        Name: Rachel Lewis
        Marks: [93, 95, 94, 96]
        Individual Average: 94.50
        ==========


        ===== STUDENT DETAILS =====
        Student ID: 119
        Name: Sam Walker
        Marks: [77, 79, 78, 80]
        Individual Average: 78.50
        ==========


        ===== STUDENT DETAILS =====
        Student ID: 120
        Name: Tina Hall
        Marks: [90, 92, 91, 93]
        Individual Average: 91.50
        ==========
```

```
============================================================
CLASS AVERAGE (20 Students): 86.89
============================================================


============================================================
PERFORMANCE COMPARISON - ABOVE/BELOW CLASS AVERAGE
============================================================
Alice Johnson (88.75) is ABOVE the class average (86.89)
Bob Smith (77.25) is BELOW the class average (86.89)
Carol White (94.50) is ABOVE the class average (86.89)
David Brown (87.25) is ABOVE the class average (86.89)
Emma Davis (92.50) is ABOVE the class average (86.89)
Frank Miller (73.50) is BELOW the class average (86.89)
Grace Lee (97.50) is ABOVE the class average (86.89)
Henry Wilson (81.50) is BELOW the class average (86.89)
Iris Taylor (88.50) is ABOVE the class average (86.89)
Jack Anderson (77.50) is BELOW the class average (86.89)
Karen Martinez (92.50) is ABOVE the class average (86.89)
Leo Thomas (85.50) is BELOW the class average (86.89)
Mia Jackson (80.50) is BELOW the class average (86.89)
Noah White (90.50) is ABOVE the class average (86.89)
Olivia Harris (95.50) is ABOVE the class average (86.89)
Peter Martin (82.50) is BELOW the class average (86.89)
Quinn Clark (87.50) is ABOVE the class average (86.89)
Rachel Lewis (94.50) is ABOVE the class average (86.89)
Sam Walker (78.50) is BELOW the class average (86.89)
Tina Hall (91.50) is ABOVE the class average (86.89)
============================================================
(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding>
```

Explanation:

The class defines basic student data using attributes and leaves method logic unfinished.
Comments act as prompts for GitHub Copilot to automatically generate methods for displaying details and evaluating performance based on class average.

Task 2: Data Processing in a Monitoring System

Prompt:

Identify even sensor readings from a list, calculate the square of each even number, and print the result in a readable format inside a for loop.

Code:

```
"""
Sensor Readings Analysis - Extract even readings and calculate their squares
"""

def analyze_sensor_readings(readings):
    """
    Analyze sensor readings to identify even values and calculate their squares.

    Args:
        readings: List of sensor reading values
    """
    print("\n" + "="*60)
    print("SENSOR READINGS ANALYSIS - EVEN NUMBERS SQUARED")
    print("="*60)
    print(f"\nOriginal Sensor Readings: {readings}")
    print(f"Total readings: {len(readings)}\n")

    print("-"*60)
    print(f"{'Reading':<15} {'Even?':<10} {'Square':<15}")
    print("-"*60)

    # Identify even readings and calculate squares
    even_readings = []
    for reading in readings:
        if reading % 2 == 0:  # Check if even
            square = reading ** 2
            even_readings.append((reading, square))
            print(f"{reading:<15} {'Yes':<10} {square:<15}")
        else:
            print(f"{reading:<15} {'No':<10} {'N/A':<15}")

    print("-"*60)
    print(f"\nTotal Even Readings Found: {len(even_readings)}")

    # Print summary of even readings and their squares
    if even_readings:
        print("\n" + "="*60)
        print("SUMMARY - EVEN READINGS AND THEIR SQUARES")
        print("="*60)
        for reading, square in even_readings:
            print(f"  {reading}² = {square}")
    else:
        print("\nNo even readings found in the sensor data.")

    print("="*60 + "\n")
```

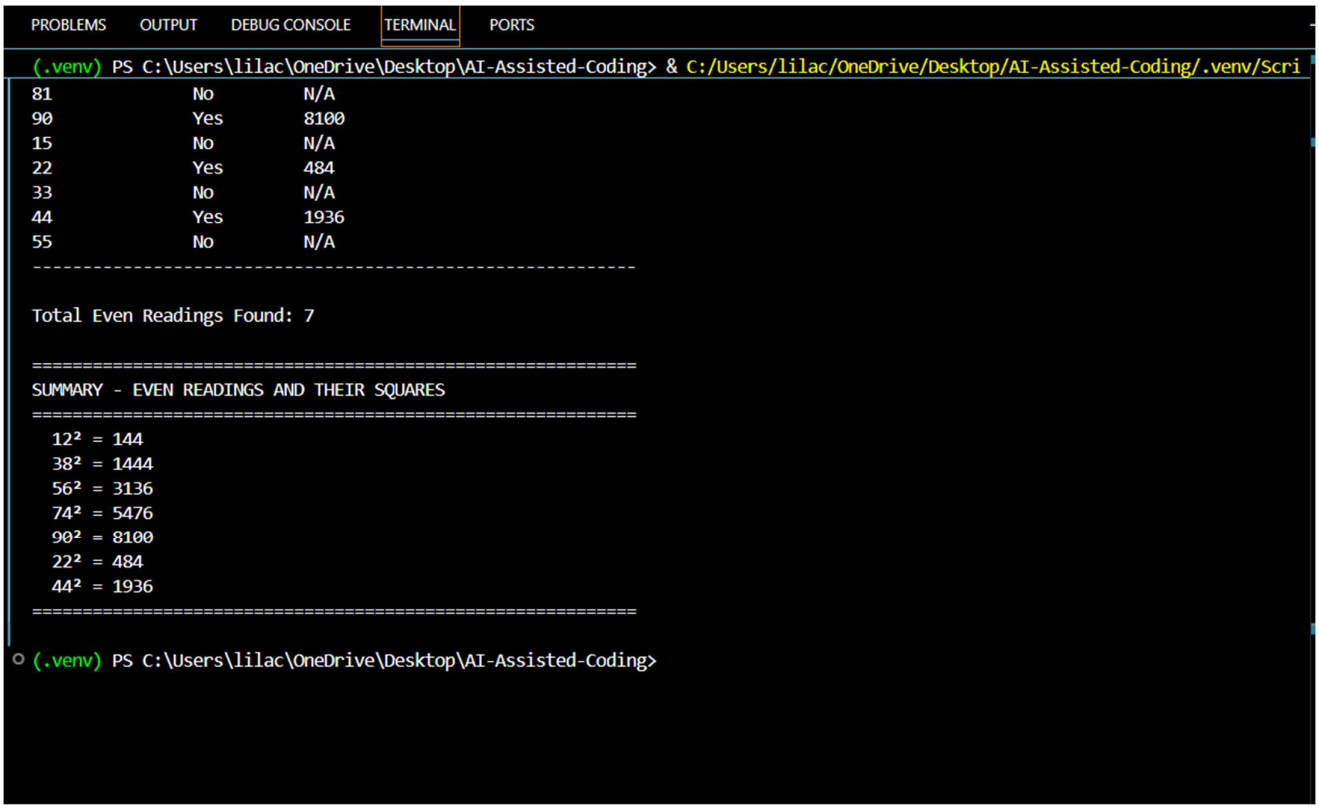
```python
    return even_readings


# Example usage with different sensor datasets
if __name__ == "__main__":
    # Example 1: Temperature sensor readings
    print("\n>>> EXAMPLE 1: TEMPERATURE SENSOR READINGS <<<")
    temperature_readings = [23, 45, 32, 67, 18, 54, 91, 28, 36, 41]
    analyze_sensor_readings(temperature_readings)

    # Example 2: Pressure sensor readings
    print("\n>>> EXAMPLE 2: PRESSURE SENSOR READINGS <<<")
    pressure_readings = [100, 105, 200, 315, 420, 502, 618, 750, 812]
    analyze_sensor_readings(pressure_readings)

    # Example 3: Mixed sensor readings
    print("\n>>> EXAMPLE 3: MIXED SENSOR READINGS <<<")
    mixed_readings = [12, 25, 38, 47, 56, 63, 74, 81, 90, 15, 22, 33, 44, 55]
    analyze_sensor_readings(mixed_readings)
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding>
(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> & C:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/.venv/Scripts/python.exe c:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/sensor_readings_analysis.py

>>> EXAMPLE 1: TEMPERATURE SENSOR READINGS <<<

============================================================
SENSOR READINGS ANALYSIS - EVEN NUMBERS SQUARED
============================================================

Original Sensor Readings: [23, 45, 32, 67, 18, 54, 91, 28, 36, 41]
Total readings: 10

------------------------------------------------------------
Reading         Even?      Square
------------------------------------------------------------
23              No         N/A
45              No         N/A
32              Yes        1024
67              No         N/A
18              Yes        324
54              Yes        2916
91              No         N/A
28              Yes        784
36              Yes        1296
41              No         N/A
------------------------------------------------------------

Total Even Readings Found: 5

============================================================
SUMMARY - EVEN READINGS AND THEIR SQUARES
============================================================
```

```
(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> & C:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/.venv/Scr
============================================================
  32² = 1024
  18² = 324
  54² = 2916
  28² = 784
  36² = 1296
============================================================


>>> EXAMPLE 2: PRESSURE SENSOR READINGS <<<

============================================================
SENSOR READINGS ANALYSIS - EVEN NUMBERS SQUARED
============================================================

Original Sensor Readings: [100, 105, 200, 315, 420, 502, 618, 750, 812]
Total readings: 9

------------------------------------------------------------
Reading         Even?       Square
------------------------------------------------------------
100             Yes         10000
105             No          N/A
200             Yes         40000
315             No          N/A
420             Yes         176400
502             Yes         252004
618             Yes         381924
750             Yes         562500
812             Yes         659344
------------------------------------------------------------

Total Even Readings Found: 7
```

```
============================================================
SUMMARY - EVEN READINGS AND THEIR SQUARES
============================================================
  100² = 10000
  200² = 40000
  420² = 176400
  502² = 252004
  618² = 381924
  750² = 562500
  812² = 659344
============================================================


>>> EXAMPLE 3: MIXED SENSOR READINGS <<<

============================================================
SENSOR READINGS ANALYSIS - EVEN NUMBERS SQUARED
============================================================

Original Sensor Readings: [12, 25, 38, 47, 56, 63, 74, 81, 90, 15, 22, 33, 44, 55]
Total readings: 14

------------------------------------------------------------
Reading         Even?       Square
------------------------------------------------------------
12              Yes         144
25              No          N/A
38              Yes         1444
47              No          N/A
56              Yes         3136
63              No          N/A
74              Yes         5476
81              No          N/A
90              Yes         8100
```



Explanation:

The loop iterates through a list of sensor readings one by one.
It checks for even values, computes their square, and prints the result in a clear and readable format.

Task 3: Banking Transaction Simulation

Prompt:

Create a Python class named BankAccount with attributes account_holder and balance. Add methods to deposit money and withdraw money, and ensure withdrawals are prevented when the balance is insufficient.

Code:

```
"""
Bank Account Management System
"""

class BankAccount:
    """A class to manage bank account operations with deposits and withdrawals."""

    def __init__(self, account_holder, initial_balance=0):
        """
        Initialize a bank account with account holder name and initial balance.

        Args:
            account_holder: Name of the account holder
            initial_balance: Starting balance (default is 0)
        """
        self.account_holder = account_holder
        self.balance = initial_balance
        self.transaction_history = []
```

```
        if initial_balance > 0:
            self.transaction_history.append(f"Account created with initial balance: ${initial_balance:.2f}")

    def deposit(self, amount):
        """
        Deposit money into the account.

        Args:
            amount: Amount to deposit (must be positive)

        Returns:
            str: Confirmation message of the deposit
        """
        if amount <= 0:
            return f"❌ Invalid deposit amount: ${amount:.2f}. Amount must be positive."

        self.balance += amount
        transaction = f"✓ Deposit: +${amount:.2f} | New Balance: ${self.balance:.2f}"
        self.transaction_history.append(transaction)
        return transaction

    def withdraw(self, amount):
        """
        Withdraw money from the account.
        Prevents withdrawal if balance is insufficient.

        Args:
            amount: Amount to withdraw

        Returns:
            str: Confirmation message or error message if withdrawal fails
        """
        if amount <= 0:
            return f"❌ Invalid withdrawal amount: ${amount:.2f}. Amount must be positive."

        if amount > self.balance:
            error_msg = f"❌ Insufficient balance! Attempted withdrawal: ${amount:.2f} | Current balance: ${self.balance:.2f}"
            self.transaction_history.append(f"Failed withdrawal attempt: ${amount:.2f}")
            return error_msg

        self.balance -= amount
        transaction = f"✓ Withdrawal: -${amount:.2f} | New Balance: ${self.balance:.2f}"
        self.transaction_history.append(transaction)
        return transaction

    def check_balance(self):
        """
        Check the current account balance.

        Returns:
            str: Formatted balance information
```

```
        """
        return f"Account Balance for {self.account_holder}: ${self.balance:.2f}"

    def display_account_info(self):
        """
        Display comprehensive account information.

        Returns:
            str: Formatted account details
        """
        info = f"""
        {'='*60}
        ACCOUNT INFORMATION
        {'='*60}
        Account Holder: {self.account_holder}
        Current Balance: ${self.balance:.2f}
        Total Transactions: {len(self.transaction_history)}
        {'='*60}
        """
        return info

    def display_transaction_history(self):
        """
        Display all transaction history for the account.
        """
        print(f"\n{'='*60}")
        print(f"TRANSACTION HISTORY - {self.account_holder}")
        print(f"{'='*60}")
        if self.transaction_history:
            for idx, transaction in enumerate(self.transaction_history, 1):
                print(f"{idx}. {transaction}")
        else:
            print("No transactions recorded.")
        print(f"{'='*60}\n")


# Example usage with multiple accounts
if __name__ == "__main__":
    print("\n" + "="*60)
    print("BANK ACCOUNT MANAGEMENT SYSTEM")
    print("="*60)

    # Create accounts
    account1 = BankAccount("John Doe", 1000)
    account2 = BankAccount("Jane Smith", 500)

    # Display account information
    print(account1.display_account_info())
    print(account2.display_account_info())

    # Perform transactions on Account 1
    print("\n>>> ACCOUNT 1: JOHN DOE - TRANSACTIONS <<<\n")
    print(account1.deposit(500))
    print(account1.deposit(250))
```

```
    print(account1.withdraw(300))
    print(account1.withdraw(200))
    print(account1.check_balance())

    # Attempt invalid withdrawal
    print(f"\n{account1.withdraw(2000)}")  # This should fail

    # Perform transactions on Account 2
    print(f"\n>>> ACCOUNT 2: JANE SMITH - TRANSACTIONS <<<\n")
    print(account2.deposit(300))
    print(account2.withdraw(100))
    print(account2.withdraw(400))  # This should fail
    print(account2.check_balance())

    # Display transaction histories
    account1.display_transaction_history()
    account2.display_transaction_history()

    # Final account status
    print("\n" + "="*60)
    print("FINAL ACCOUNT STATUS")
    print("="*60)
    print(account1.check_balance())
    print(account2.check_balance())
    print("="*60 + "\n")
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> & C:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/.venv/Scripts/python.exe c:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/bank_account_system.py

============================================================
BANK ACCOUNT MANAGEMENT SYSTEM
============================================================

        ============================================================
        ACCOUNT INFORMATION
        ============================================================
        Account Holder: John Doe
        Current Balance: $1000.00
        Total Transactions: 1
        ============================================================


        ============================================================
        ACCOUNT INFORMATION
        ============================================================
        Account Holder: Jane Smith
        Current Balance: $500.00
        Total Transactions: 1
        ============================================================


>>> ACCOUNT 1: JOHN DOE - TRANSACTIONS <<<

✓ Deposit: +$500.00 | New Balance: $1500.00
✓ Deposit: +$250.00 | New Balance: $1750.00
✓ Withdrawal: -$300.00 | New Balance: $1450.00
✓ Withdrawal: -$200.00 | New Balance: $1250.00
Account Balance for John Doe: $1250.00

❌ Insufficient balance! Attempted withdrawal: $2000.00 | Current balance: $1250.00
```

```
(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> & C:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/.venv/Scri

>>> ACCOUNT 2: JANE SMITH - TRANSACTIONS <<<

✓ Deposit: +$300.00 | New Balance: $800.00
✓ Withdrawal: -$100.00 | New Balance: $700.00
✓ Withdrawal: -$400.00 | New Balance: $300.00
Account Balance for Jane Smith: $300.00

============================================================
TRANSACTION HISTORY - John Doe
============================================================
1. Account created with initial balance: $1000.00
2. ✓ Deposit: +$500.00 | New Balance: $1500.00
3. ✓ Deposit: +$250.00 | New Balance: $1750.00
4. ✓ Withdrawal: -$300.00 | New Balance: $1450.00
5. ✓ Withdrawal: -$200.00 | New Balance: $1250.00
6. Failed withdrawal attempt: $2000.00
============================================================


============================================================
TRANSACTION HISTORY - Jane Smith
============================================================
1. Account created with initial balance: $500.00
2. ✓ Deposit: +$300.00 | New Balance: $800.00
3. ✓ Withdrawal: -$100.00 | New Balance: $700.00
4. ✓ Withdrawal: -$400.00 | New Balance: $300.00
============================================================


============================================================
FINAL ACCOUNT STATUS
============================================================
Account Balance for John Doe: $1250.00
```

```
============================================================
FINAL ACCOUNT STATUS
============================================================
Account Balance for John Doe: $1250.00
Account Balance for Jane Smith: $300.00
============================================================

(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding>
```

Explanation:

The prompt instructs Copilot to generate a BankAccount class with basic attributes and transaction methods.
It ensures safe banking operations by allowing deposits and preventing withdrawals when the account balance is insufficient.

Task 4: Student Scholarship Eligibility Check

Prompt:

Create a while loop that iterates through a list of student dictionaries and takes the scholarship cutoff score as user input.Print the names of students whose score is greater than the entered cutoff value.

Code:

```python
import random

students = []
for i in range(1, 31):
    student_name = f"Student_{i}"
    student_score = random.randint(50, 100) # Assign random scores between 50 and
100
    students.append({"name": student_name, "score": student_score})

# Initialize an index for the while loop
index = 0
# Loop through each student in the 'students' list
while index < len(students):
    # Get the current student's dictionary
    student = students[index]
    # Check if the student's 'score' is greater than 75
    if student["score"] > 75:
        # Print the 'name' of the student if they qualify
        print(f"{student['name']} (Score: {student['score']})")
    # Move to the next student
    index += 1
```

Output:

```
Student_1 (Score: 85)
Student_4 (Score: 81)
Student_6 (Score: 87)
Student_7 (Score: 88)
Student_8 (Score: 96)
Student_10 (Score: 99)
Student_11 (Score: 87)
Student_15 (Score: 79)
Student_19 (Score: 81)
Student_21 (Score: 81)
Student_22 (Score: 100)
Student_23 (Score: 98)
Student_26 (Score: 92)
Student_27 (Score: 98)
Student_29 (Score: 94)
```

Explanation:

The program stores student details in a list of dictionaries and takes a cutoff score from the user.
Using a while loop, it checks each student's score and prints the names of those who scored above the cutoff value.

Task 5: Online Shopping Cart Module

Prompt:

Create a Python class named ShoppingCart with an empty list to store cart items (name, price, quantity).
Add methods to add and remove items, calculate the total bill using a loop, and apply a discount conditionally when the total exceeds a specified amount.

Code:

```
class ShoppingCart:
    def __init__(self):
        # Initialize an empty list to store items in the cart
        # Each item will be a dictionary with keys: 'name', 'price', 'quantity'
        self.items = []

    def add_item(self):
        # Method to add an item to the shopping cart
        # This method should prompt the user for item details
        item_name = input("Enter item name: ")
        item_price = float(input(f"Enter price for {item_name}: "))
        item_quantity = int(input(f"Enter quantity for {item_name}: "))

        # Create a dictionary for the new item
        new_item = {"name": item_name, "price": item_price, "quantity":
item_quantity}
        # Add the new item to the cart's items list
        self.items.append(new_item)
        print(f"{item_quantity} x {item_name} added to cart.")

    def remove_item(self):
        # Method to remove an item from the shopping cart
        # This method should prompt the user for the name of the item to remove
        if not self.items:
            print("The cart is empty. Nothing to remove.")
            return

        print("Current items in cart:")
        for i, item in enumerate(self.items):
            print(f"{i+1}. {item['name']} (Quantity: {item['quantity']})")

        item_to_remove_name = input("Enter the name of the item to remove: ")
        item_found = False
        # Iterate through the items to find and remove the specified item
        # Use a list comprehension to create a new list without the removed item(s)
        initial_item_count = len(self.items)
        self.items = [item for item in self.items if item['name'].lower() !=
item_to_remove_name.lower()]

        if len(self.items) < initial_item_count:
            item_found = True
```

```
            print(f"'{item_to_remove_name}' removed from cart.")
        else:
            print(f"'{item_to_remove_name}' not found in cart.")

    def calculate_total(self):
        # Method to calculate the total bill for all items in the cart
        # It should iterate through the items and sum (price * quantity)
        total_bill = 0
        for item in self.items:
            total_bill += item["price"] * item["quantity"]
        return total_bill

    def apply_discount(self):
        # Method to apply conditional discounts to the total bill
        # It should ask the user for a discount percentage if a condition is met
(e.g., total exceeds a certain amount)
        current_total = self.calculate_total()
        print(f"Current total before discount: ${current_total:.2f}")

        if current_total > 100: # Example condition for applying a discount
            print("Your total exceeds $100! You are eligible for a discount.")
            try:
                discount_percentage = float(input("Enter discount percentage (e.g.,
10 for 10%): "))
                if 0 <= discount_percentage <= 100:
                    discount_amount = current_total * (discount_percentage / 100)
                    discounted_total = current_total - discount_amount
                    print(f"Applied {discount_percentage}% discount
(${discount_amount:.2f}).")
                    print(f"New total after discount: ${discounted_total:.2f}")
                    return discounted_total
                else:
                    print("Invalid discount percentage. No discount applied.")
            except ValueError:
                print("Invalid input. Please enter a number for the discount
percentage.")
        else:
            print("No discount applicable at this time (total must exceed $100).")
        return current_total # Return original total if no discount applied or
invalid input

    def view_cart(self):
        # Method to display all items currently in the cart
        if not self.items:
            print("Your shopping cart is empty.")
            return
        print("\n--- Your Shopping Cart ---")
        for item in self.items:
            print(f"Name: {item['name']}, Price: ${item['price']:.2f}, Quantity:
{item['quantity']}")
        print("--------------------------")
        print(f"Subtotal: ${self.calculate_total():.2f}")

# Example usage of the ShoppingCart class:
```

```
if __name__ == '__main__':
    cart = ShoppingCart()

    while True:
        print("\n1. Add item")
        print("2. Remove item")
        print("3. View cart")
        print("4. Calculate total bill")
        print("5. Apply discount")
        print("6. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            cart.add_item()
        elif choice == '2':
            cart.remove_item()
        elif choice == '3':
            cart.view_cart()
        elif choice == '4':
            total = cart.calculate_total()
            print(f"Total bill: ${total:.2f}")
        elif choice == '5':
            cart.apply_discount()
        elif choice == '6':
            print("Exiting shopping cart. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")
```

Output:

Explanation:

The prompt guides Copilot to generate a ShoppingCart class that manages items using a list structure.
It automates item addition and removal, calculates the total cost with a loop, and applies discounts based on conditional logic.