Assignment-6.4

Task-01: 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.

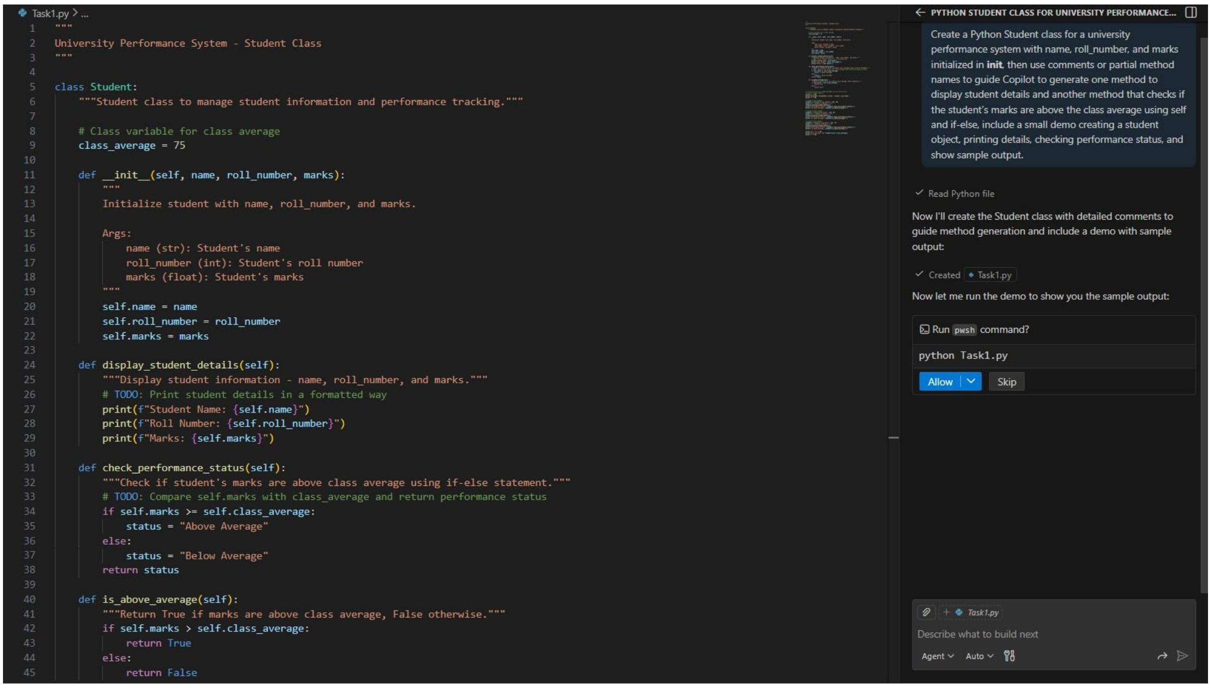
Then, using GitHub Copilot, prompt the tool to complete:

- **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**



```python
"""
University Performance System - Student Class
"""

class Student:
    """Student class to manage student information and performance tracking."""

    # Class variable for class average
    class_average = 75

    def __init__(self, name, roll_number, marks):
        """
        Initialize student with name, roll_number, and marks.

        Args:
            name (str): Student's name
            roll_number (int): Student's roll number
            marks (float): Student's marks
        """
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_student_details(self):
        """Display student information - name, roll_number, and marks."""
        # TODO: Print student details in a formatted way
        print(f"Student Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

    def check_performance_status(self):
        """Check if student's marks are above class average using if-else statement."""
        # TODO: Compare self.marks with class_average and return performance status
        if self.marks >= self.class_average:
            status = "Above Average"
        else:
            status = "Below Average"
        return status

    def is_above_average(self):
        """Return True if marks are above class average, False otherwise."""
        if self.marks > self.class_average:
            return True
        else:
            return False
```

PYTHON STUDENT CLASS FOR UNIVERSITY PERFORMANCE...

Create a Python Student class for a university performance system with name, roll_number, and marks initialized in **init**, then use comments or partial method names to guide Copilot to generate one method to display student details and another method that checks if the student's marks are above the class average using self and if-else, include a small demo creating a student object, printing details, checking performance status, and show sample output.

✓ Read Python file

Now I'll create the Student class with detailed comments to guide method generation and include a demo with sample output:

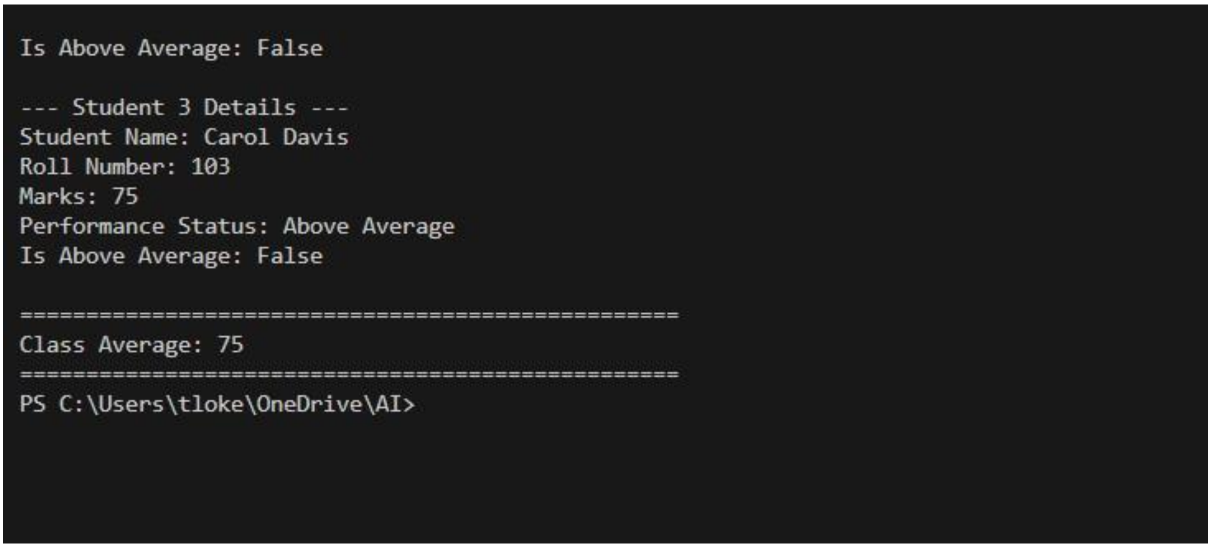
✓ Created Task1.py

Now let me run the demo to show you the sample output:

Run `pwsh` command?

```
python Task1.py
```

OUTPUT:



```
Is Above Average: False

--- Student 3 Details ---
Student Name: Carol Davis
Roll Number: 103
Marks: 75
Performance Status: Above Average
Is Above Average: False

==================================================
Class Average: 75
==================================================
PS C:\Users\tloke\OneDrive\AI>
```

Task-02: 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

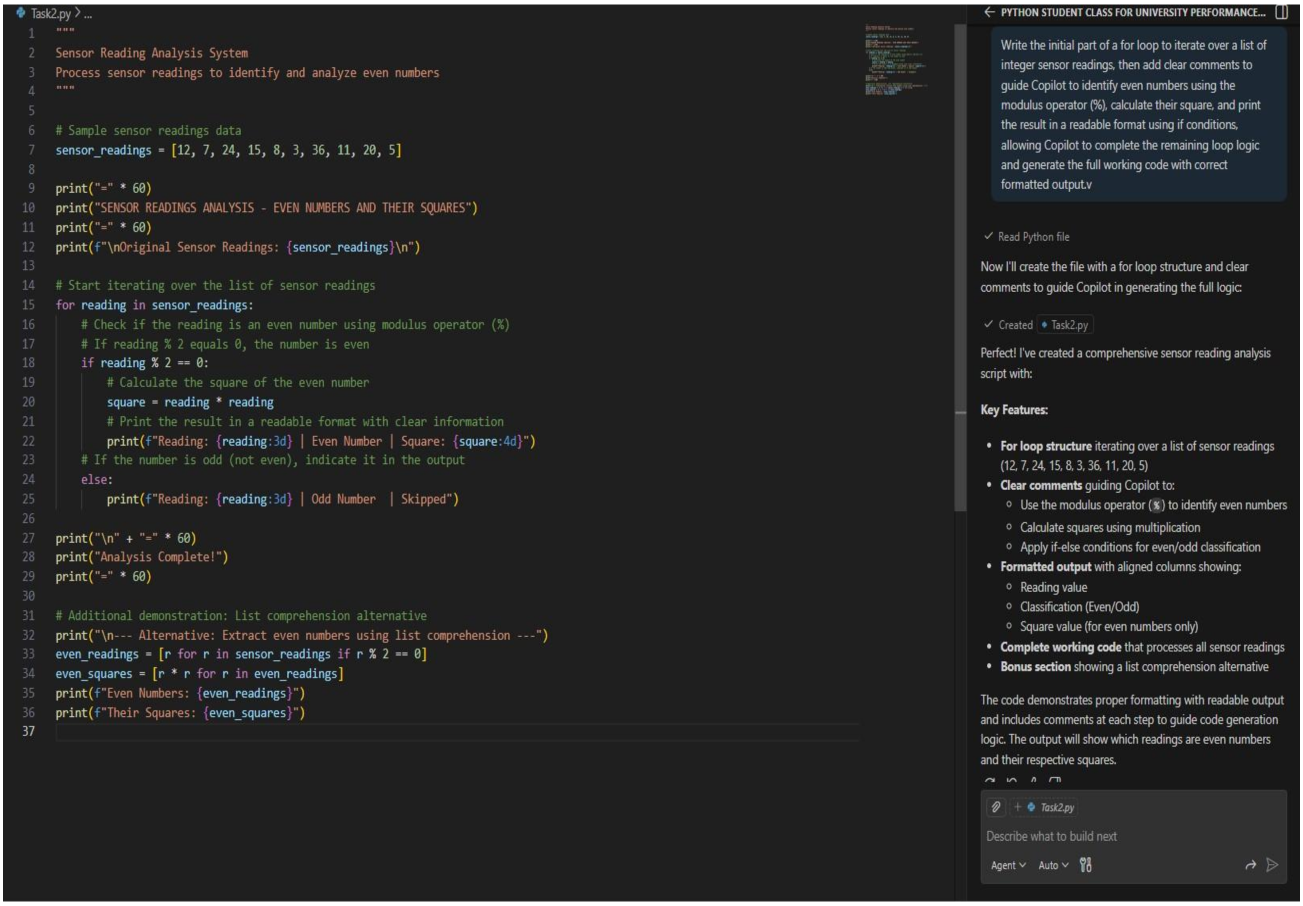
• A complete for loop generated by Copilot

• Use of:

o Modulus operator to identify even numbers o

Conditional statements

• Correct and formatted output for valid inputs


Task2.py > ...
PYTHON STUDENT CLASS FOR UNIVERSITY PERFORMANCE...
Read Python file
Created Task2.py
Task2.py
Describe what to build next
Agent
Auto

```python
"""
Sensor Reading Analysis System
Process sensor readings to identify and analyze even numbers
"""

# Sample sensor readings data
sensor_readings = [12, 7, 24, 15, 8, 3, 36, 11, 20, 5]

print("=" * 60)
print("SENSOR READINGS ANALYSIS - EVEN NUMBERS AND THEIR SQUARES")
print("=" * 60)
print(f"\nOriginal Sensor Readings: {sensor_readings}\n")

# Start iterating over the list of sensor readings
for reading in sensor_readings:
    # Check if the reading is an even number using modulus operator (%)
    # If reading % 2 equals 0, the number is even
    if reading % 2 == 0:
        # Calculate the square of the even number
        square = reading * reading
        # Print the result in a readable format with clear information
        print(f"Reading: {reading:3d} | Even Number | Square: {square:4d}")
    # If the number is odd (not even), indicate it in the output
    else:
        print(f"Reading: {reading:3d} | Odd Number  | Skipped")

print("\n" + "=" * 60)
print("Analysis Complete!")
print("=" * 60)

# Additional demonstration: List comprehension alternative
print("\n--- Alternative: Extract even numbers using list comprehension ---")
even_readings = [r for r in sensor_readings if r % 2 == 0]
even_squares = [r * r for r in even_readings]
print(f"Even Numbers: {even_readings}")
print(f"Their Squares: {even_squares}")
```

Write the initial part of a for loop to iterate over a list of integer sensor readings, then add clear comments to guide Copilot to identify even numbers using the modulus operator (%), calculate their square, and print the result in a readable format using if conditions, allowing Copilot to complete the remaining loop logic and generate the full working code with correct formatted output.v

Now I'll create the file with a for loop structure and clear comments to guide Copilot in generating the full logic:

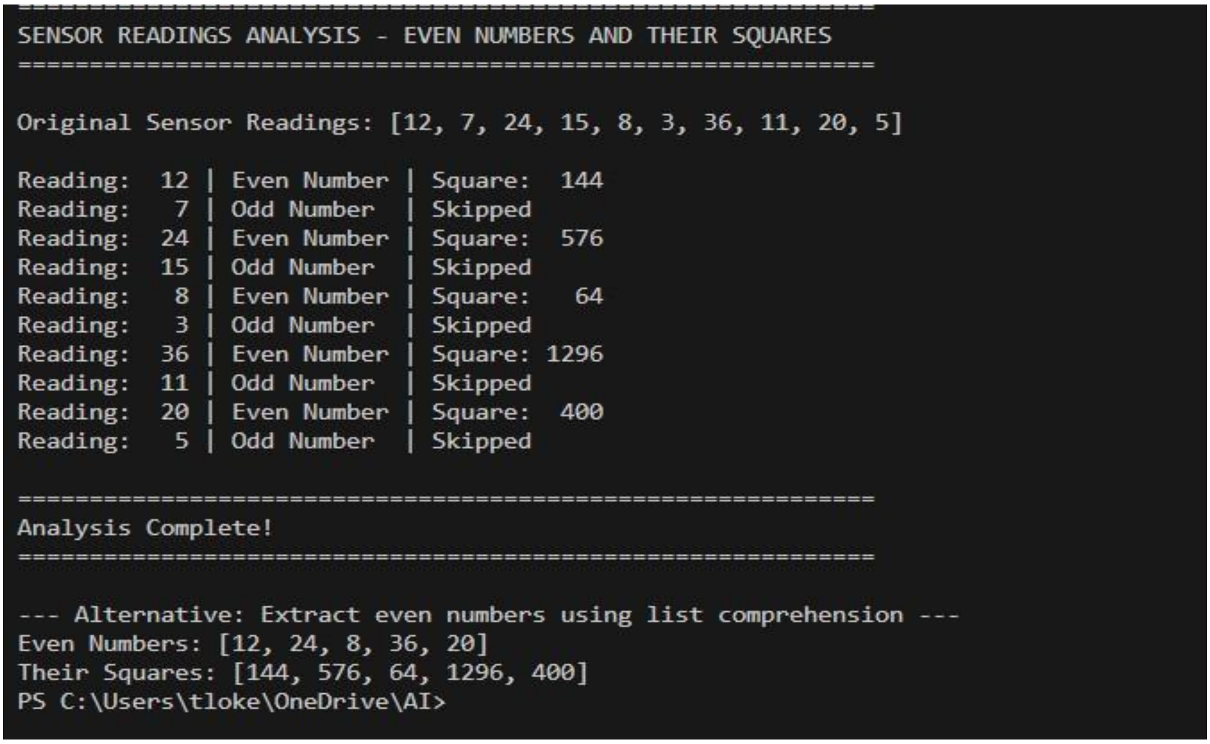
Perfect! I've created a comprehensive sensor reading analysis script with:

Key Features:

- **For loop structure** iterating over a list of sensor readings (12, 7, 24, 15, 8, 3, 36, 11, 20, 5)
- **Clear comments** guiding Copilot to:
  - Use the modulus operator (`%`) to identify even numbers
  - Calculate squares using multiplication
  - Apply if-else conditions for even/odd classification
- **Formatted output** with aligned columns showing:
  - Reading value
  - Classification (Even/Odd)
  - Square value (for even numbers only)
- **Complete working code** that processes all sensor readings
- **Bonus section** showing a list comprehension alternative

The code demonstrates proper formatting with readable output and includes comments at each step to guide code generation logic. The output will show which readings are even numbers and their respective squares.

OUTPUT:



```
============================================================
SENSOR READINGS ANALYSIS - EVEN NUMBERS AND THEIR SQUARES
============================================================

Original Sensor Readings: [12, 7, 24, 15, 8, 3, 36, 11, 20, 5]

Reading:  12 | Even Number | Square:  144
Reading:   7 | Odd Number  | Skipped
Reading:  24 | Even Number | Square:  576
Reading:  15 | Odd Number  | Skipped
Reading:   8 | Even Number | Square:   64
Reading:   3 | Odd Number  | Skipped
Reading:  36 | Even Number | Square: 1296
Reading:  11 | Odd Number  | Skipped
Reading:  20 | Even Number | Square:  400
Reading:   5 | Odd Number  | Skipped

============================================================
Analysis Complete!
============================================================

--- Alternative: Extract even numbers using list comprehension ---
Even Numbers: [12, 24, 8, 36, 20]
Their Squares: [144, 576, 64, 1296, 400]
PS C:\Users\tloke\OneDrive\AI>
```

Task-03: 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

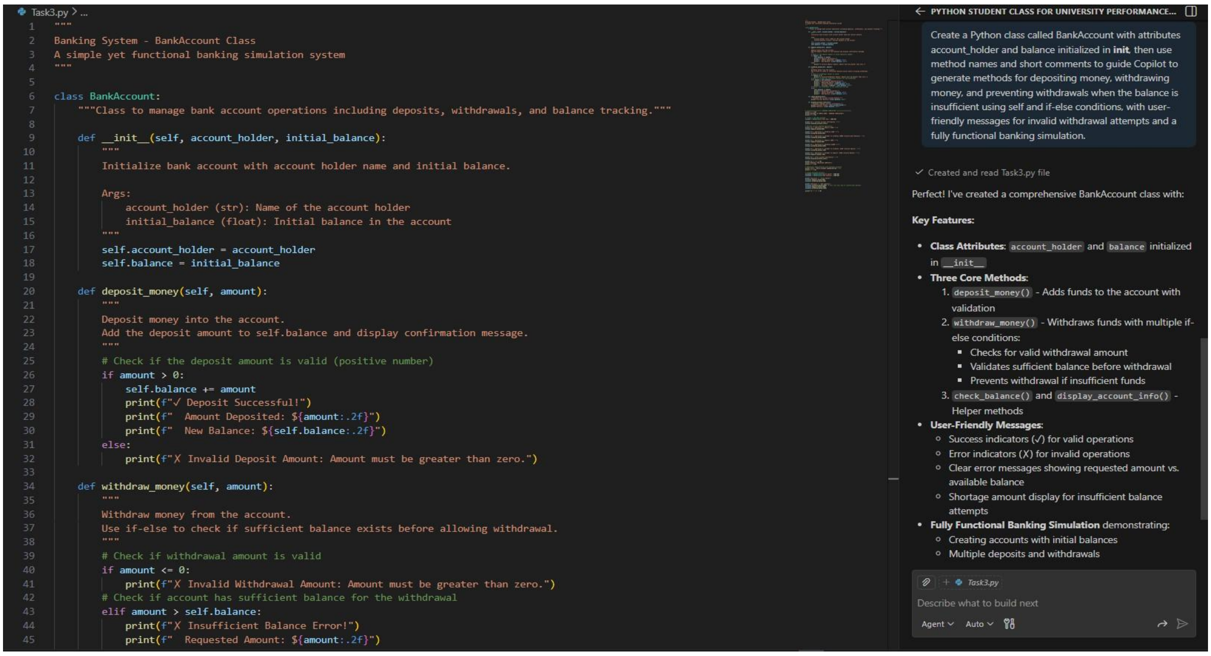
- **A fully functional BankAccount class**
- **Copilot-generated methods using:**

o if-else conditions o

Class attributes via self

- **Proper handling of invalid withdrawal attempts with user-friendly**

Messages



```python
"""
Banking System - BankAccount Class
A simple yet functional banking simulation system
"""

class BankAccount:
    """Class to manage bank account operations including deposits, withdrawals, and balance tracking."""

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

        Args:
            account_holder (str): Name of the account holder
            initial_balance (float): Initial balance in the account
        """
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit_money(self, amount):
        """
        Deposit money into the account.
        Add the deposit amount to self.balance and display confirmation message.
        """
        # Check if the deposit amount is valid (positive number)
        if amount > 0:
            self.balance += amount
            print(f"✓ Deposit Successful!")
            print(f"  Amount Deposited: ${amount:.2f}")
            print(f"  New Balance: ${self.balance:.2f}")
        else:
            print(f"X Invalid Deposit Amount: Amount must be greater than zero.")

    def withdraw_money(self, amount):
        """
        Withdraw money from the account.
        Use if-else to check if sufficient balance exists before allowing withdrawal.
        """
        # Check if withdrawal amount is valid
        if amount <= 0:
            print(f"X Invalid Withdrawal Amount: Amount must be greater than zero.")
        # Check if account has sufficient balance for the withdrawal
        elif amount > self.balance:
            print(f"X Insufficient Balance Error!")
            print(f"  Requested Amount: ${amount:.2f}")
```

Create a Python class called BankAccount with attributes account_holder and balance initialized in **init**, then use method names and short comments to guide Copilot to generate methods for depositing money, withdrawing money, and preventing withdrawals when the balance is insufficient using self and if-else conditions, with user-friendly messages for invalid withdrawal attempts and a fully functional banking simulation.

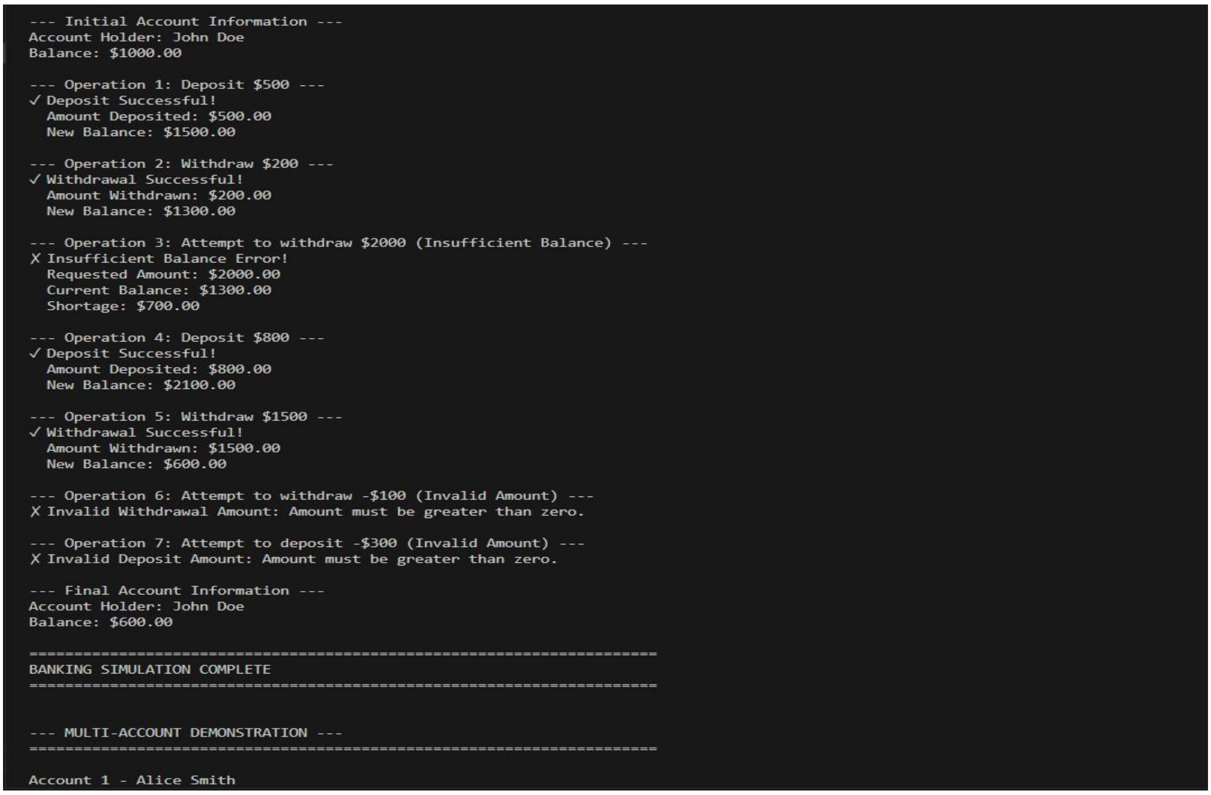
✓ Created and read Task3.py file

Perfect! I've created a comprehensive BankAccount class with:

Key Features:

- **Class Attributes:** `account_holder` and `balance` initialized in `__init__`
- **Three Core Methods:**
  1. `deposit_money()` - Adds funds to the account with validation
  2. `withdraw_money()` - Withdraws funds with multiple if-else conditions:
     - Checks for valid withdrawal amount
     - Validates sufficient balance before withdrawal
     - Prevents withdrawal if insufficient funds
  3. `check_balance()` and `display_account_info()` - Helper methods
- **User-Friendly Messages:**
  - Success indicators (✓) for valid operations
  - Error indicators (X) for invalid operations
  - Clear error messages showing requested amount vs. available balance
  - Shortage amount display for insufficient balance attempts
- **Fully Functional Banking Simulation** demonstrating:
  - Creating accounts with initial balances
  - Multiple deposits and withdrawals

OUTPUT:



```
--- Initial Account Information ---
Account Holder: John Doe
Balance: $1000.00

--- Operation 1: Deposit $500 ---
✓ Deposit Successful!
  Amount Deposited: $500.00
  New Balance: $1500.00

--- Operation 2: Withdraw $200 ---
✓ Withdrawal Successful!
  Amount Withdrawn: $200.00
  New Balance: $1300.00

--- Operation 3: Attempt to withdraw $2000 (Insufficient Balance) ---
X Insufficient Balance Error!
  Requested Amount: $2000.00
  Current Balance: $1300.00
  Shortage: $700.00

--- Operation 4: Deposit $800 ---
✓ Deposit Successful!
  Amount Deposited: $800.00
  New Balance: $2100.00

--- Operation 5: Withdraw $1500 ---
✓ Withdrawal Successful!
  Amount Withdrawn: $1500.00
  New Balance: $600.00

--- Operation 6: Attempt to withdraw -$100 (Invalid Amount) ---
X Invalid Withdrawal Amount: Amount must be greater than zero.

--- Operation 7: Attempt to deposit -$300 (Invalid Amount) ---
X Invalid Deposit Amount: Amount must be greater than zero.

--- Final Account Information ---
Account Holder: John Doe
Balance: $600.00

======================================================================
BANKING SIMULATION COMPLETE
======================================================================


--- MULTI-ACCOUNT DEMONSTRATION ---
======================================================================

Account 1 - Alice Smith
```

Task-04:

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.

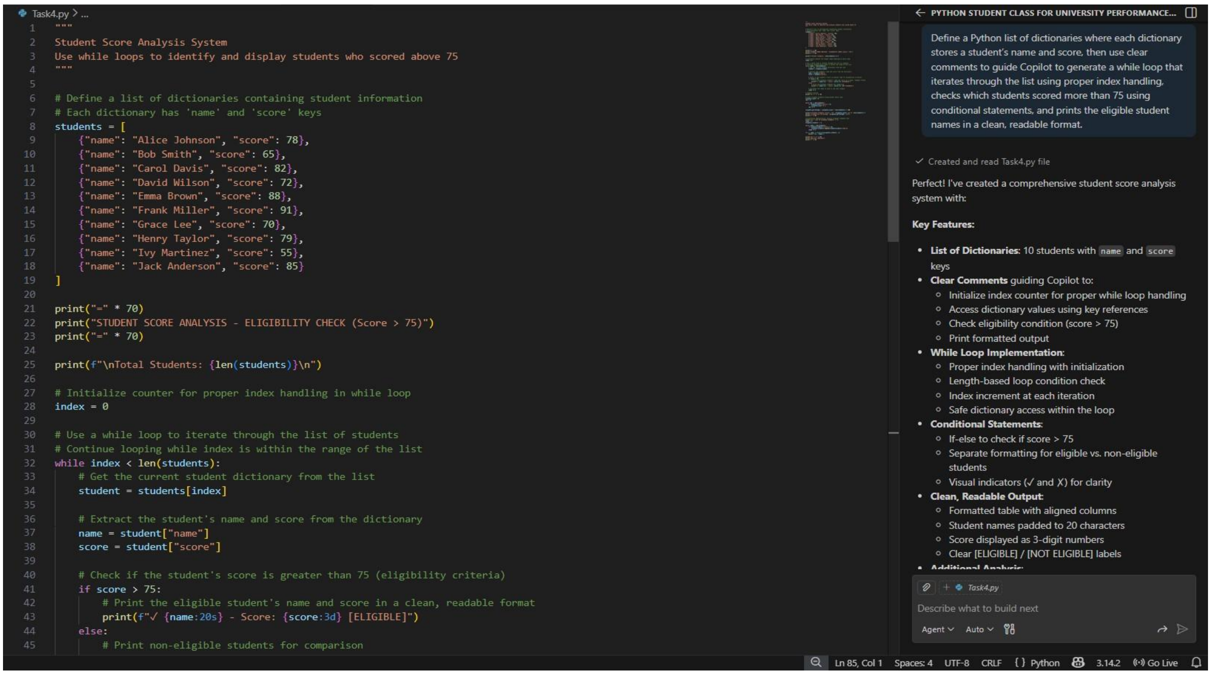
Then, prompt GitHub Copilot to generate a while loop that:

- 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



```python
"""
Student Score Analysis System
Use while loops to identify and display students who scored above 75
"""

# Define a list of dictionaries containing student information
# Each dictionary has 'name' and 'score' keys
students = [
    {"name": "Alice Johnson", "score": 78},
    {"name": "Bob Smith", "score": 65},
    {"name": "Carol Davis", "score": 82},
    {"name": "David Wilson", "score": 72},
    {"name": "Emma Brown", "score": 88},
    {"name": "Frank Miller", "score": 91},
    {"name": "Grace Lee", "score": 70},
    {"name": "Henry Taylor", "score": 79},
    {"name": "Ivy Martinez", "score": 55},
    {"name": "Jack Anderson", "score": 85}
]

print("=" * 70)
print("STUDENT SCORE ANALYSIS - ELIGIBILITY CHECK (Score > 75)")
print("=" * 70)

print(f"\nTotal Students: {len(students)}\n")

# Initialize counter for proper index handling in while loop
index = 0

# Use a while loop to iterate through the list of students
# Continue looping while index is within the range of the list
while index < len(students):
    # Get the current student dictionary from the list
    student = students[index]

    # Extract the student's name and score from the dictionary
    name = student["name"]
    score = student["score"]

    # Check if the student's score is greater than 75 (eligibility criteria)
    if score > 75:
        # Print the eligible student's name and score in a clean, readable format
        print(f"✓ {name:20s} - Score: {score:3d} [ELIGIBLE]")
    else:
        # Print non-eligible students for comparison
```

Define a Python list of dictionaries where each dictionary stores a student's name and score, then use clear comments to guide Copilot to generate a while loop that iterates through the list using proper index handling, checks which students scored more than 75 using conditional statements, and prints the eligible student names in a clean, readable format.

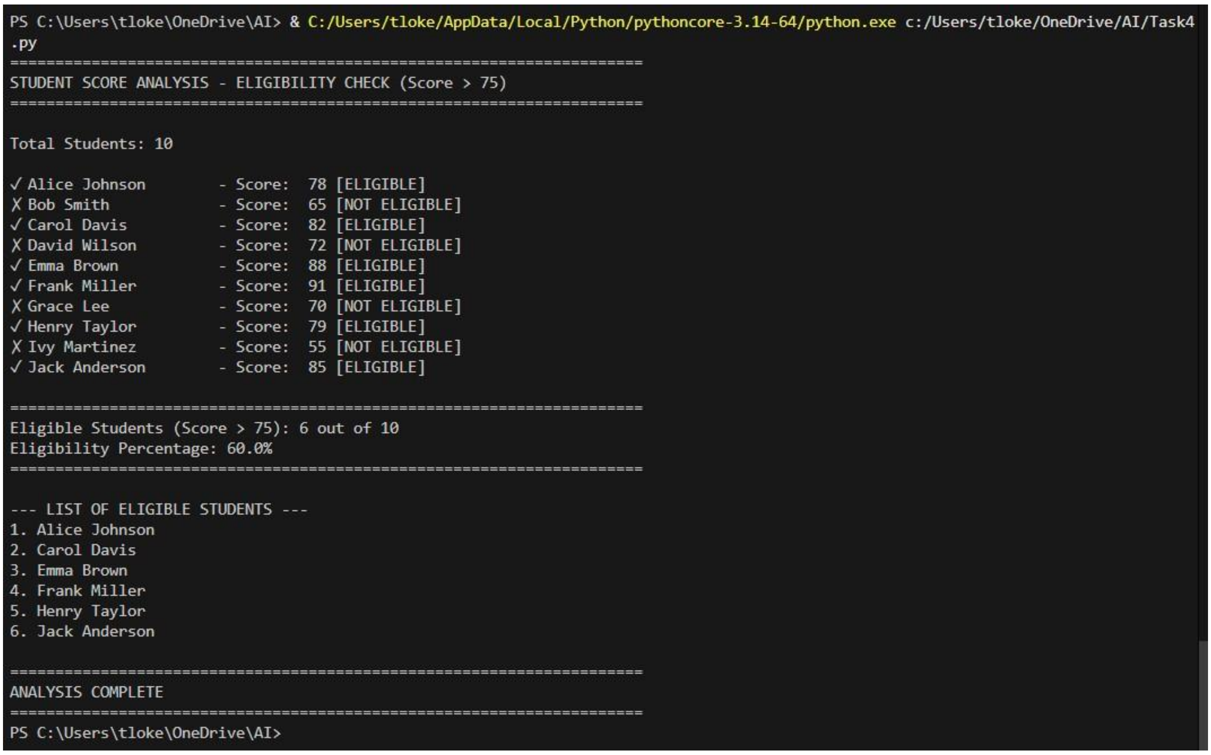
✓ Created and read Task4.py file

Perfect! I've created a comprehensive student score analysis system with:

Key Features:

- **List of Dictionaries:** 10 students with `name` and `score` keys
- **Clear Comments** guiding Copilot to:
  - Initialize index counter for proper while loop handling
  - Access dictionary values using key references
  - Check eligibility condition (score > 75)
  - Print formatted output
- **While Loop Implementation**:
  - Proper index handling with initialization
  - Length-based loop condition check
  - Index increment at each iteration
  - Safe dictionary access within the loop
- **Conditional Statements**:
  - If-else to check if score > 75
  - Separate formatting for eligible vs. non-eligible students
  - Visual indicators (✓ and ✗) for clarity
- **Clean, Readable Output**:
  - Formatted table with aligned columns
  - Student names padded to 20 characters
  - Score displayed as 3-digit numbers
  - Clear [ELIGIBLE] / [NOT ELIGIBLE] labels

OUTPUT:



```
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/Task4
.py
======================================================================
STUDENT SCORE ANALYSIS - ELIGIBILITY CHECK (Score > 75)
======================================================================

Total Students: 10

✓ Alice Johnson        - Score:  78 [ELIGIBLE]
✗ Bob Smith            - Score:  65 [NOT ELIGIBLE]
✓ Carol Davis          - Score:  82 [ELIGIBLE]
✗ David Wilson         - Score:  72 [NOT ELIGIBLE]
✓ Emma Brown           - Score:  88 [ELIGIBLE]
✓ Frank Miller         - Score:  91 [ELIGIBLE]
✗ Grace Lee            - Score:  70 [NOT ELIGIBLE]
✓ Henry Taylor         - Score:  79 [ELIGIBLE]
✗ Ivy Martinez         - Score:  55 [NOT ELIGIBLE]
✓ Jack Anderson        - Score:  85 [ELIGIBLE]

======================================================================
Eligible Students (Score > 75): 6 out of 10
Eligibility Percentage: 60.0%
======================================================================

--- LIST OF ELIGIBLE STUDENTS ---
1. Alice Johnson
2. Carol Davis
3. Emma Brown
4. Frank Miller
5. Henry Taylor
6. Jack Anderson

======================================================================
ANALYSIS COMPLETE
======================================================================
PS C:\Users\tloke\OneDrive\AI>
```

Task-05: 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)**

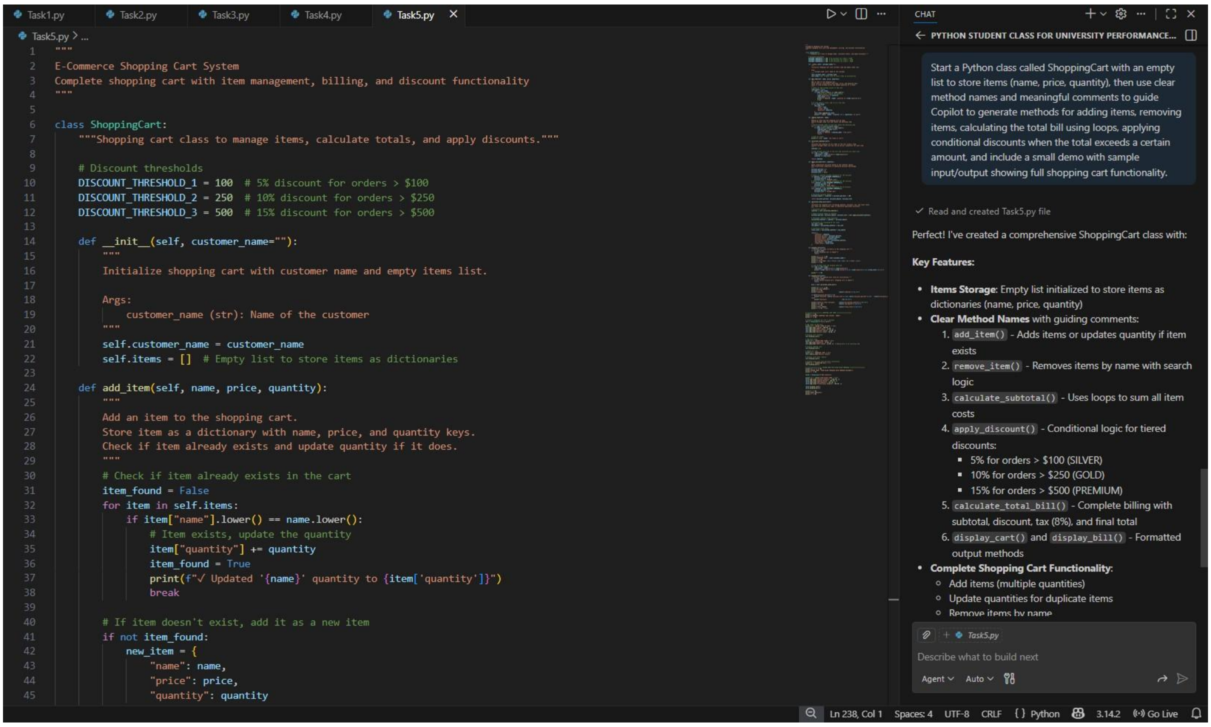
Use GitHub Copilot to generate methods that:

- **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**



```python
"""
E-Commerce Shopping Cart System
Complete shopping cart with item management, billing, and discount functionality
"""

class ShoppingCart:
    """Shopping cart class to manage items, calculate totals, and apply discounts."""

    # Discount thresholds
    DISCOUNT_THRESHOLD_1 = 100  # 5% discount for orders > $100
    DISCOUNT_THRESHOLD_2 = 250  # 10% discount for orders > $250
    DISCOUNT_THRESHOLD_3 = 500  # 15% discount for orders > $500

    def __init__(self, customer_name=""):
        """
        Initialize shopping cart with customer name and empty items list.

        Args:
            customer_name (str): Name of the customer
        """
        self.customer_name = customer_name
        self.items = []  # Empty list to store items as dictionaries

    def add_item(self, name, price, quantity):
        """
        Add an item to the shopping cart.
        Store item as a dictionary with name, price, and quantity keys.
        Check if item already exists and update quantity if it does.
        """
        # Check if item already exists in the cart
        item_found = False
        for item in self.items:
            if item["name"].lower() == name.lower():
                # Item exists, update the quantity
                item["quantity"] += quantity
                item_found = True
                print(f"✓ Updated '{name}' quantity to {item['quantity']}")
                break

        # If item doesn't exist, add it as a new item
        if not item_found:
            new_item = {
                "name": name,
                "price": price,
                "quantity": quantity
```

Start a Python class called ShoppingCart with an empty list to store items (name, price, quantity), then use clear method names and meaningful comments to guide Copilot to generate methods for adding items, removing items, calculating the total bill using loops, applying conditional discounts when the total exceeds a certain amount, and include a small demo with sample input/output showing full shopping cart functionality.

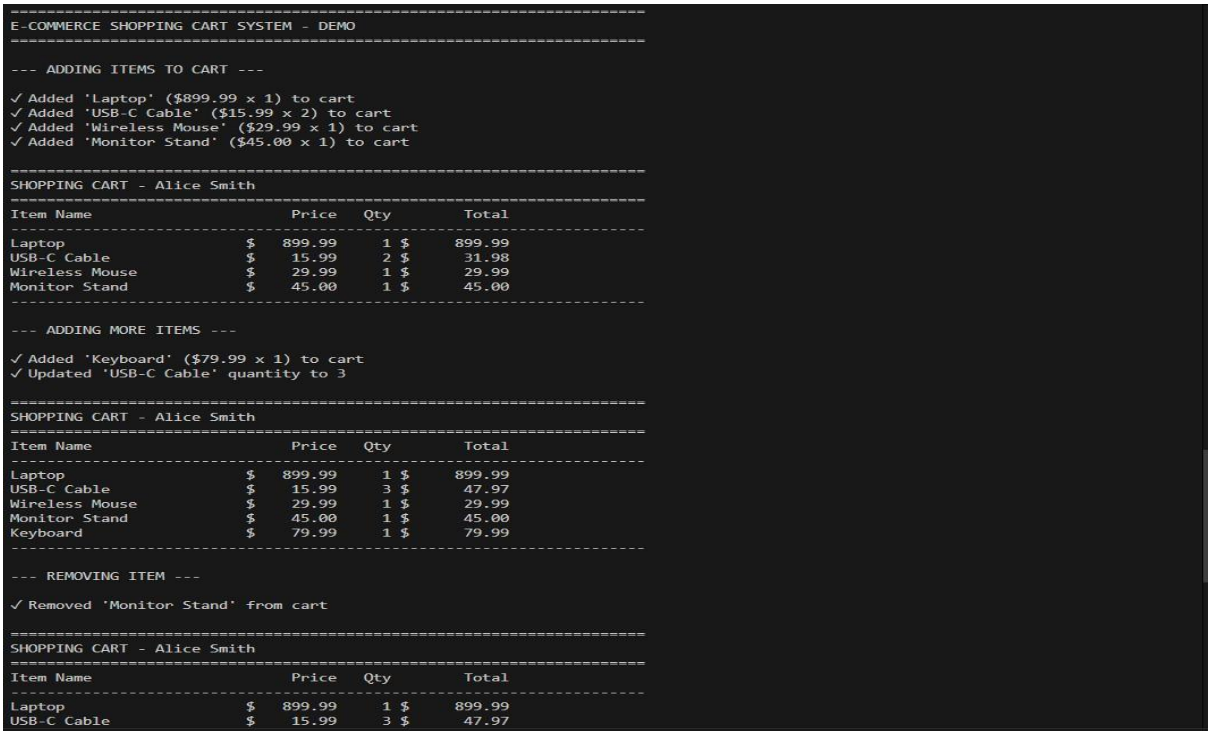
✓ Read and created Task5.py file

Perfect! I've created a comprehensive ShoppingCart class with:

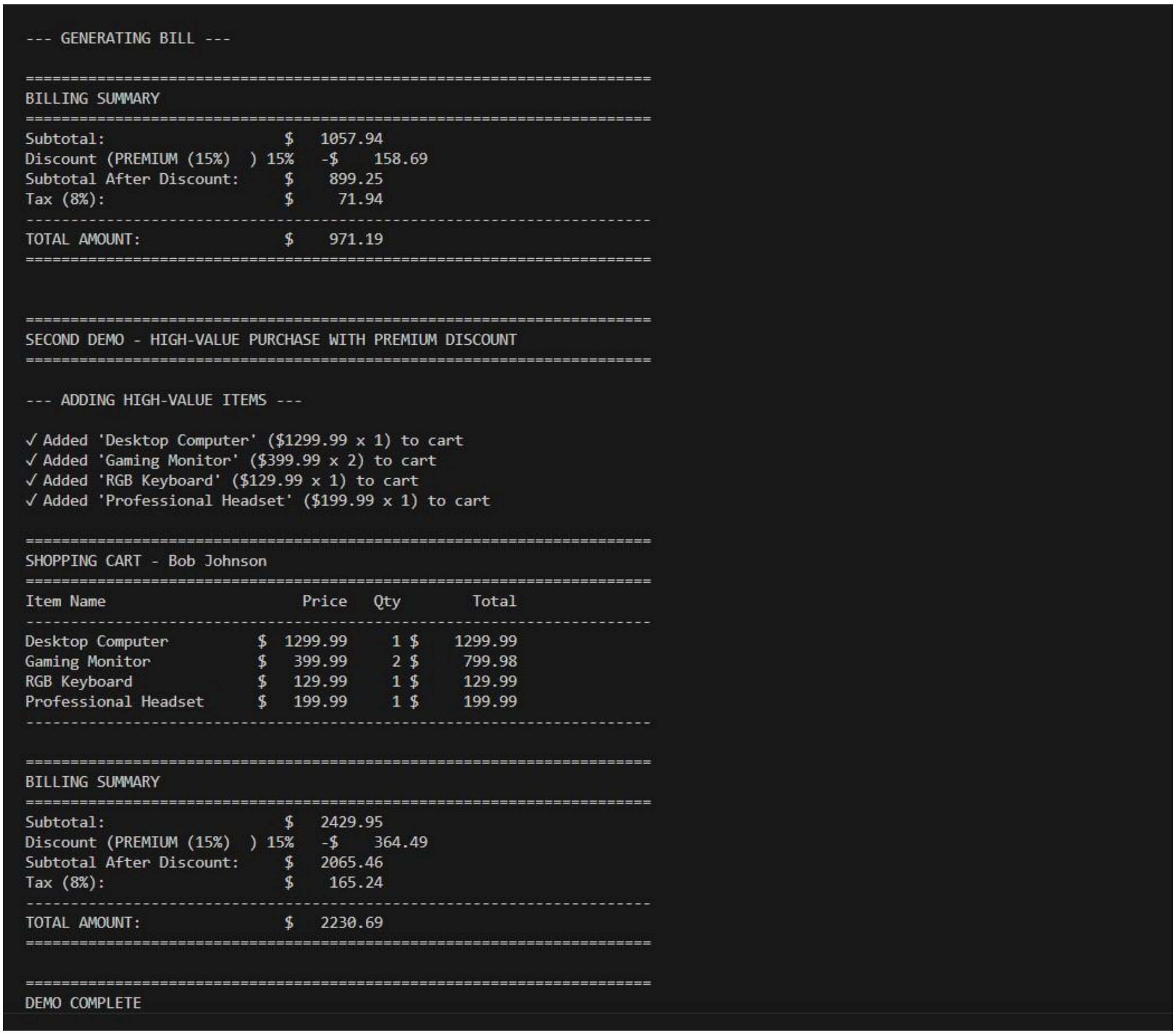
Key Features:

- **Items Storage**: Empty list initialized to store items as dictionaries (name, price, quantity)
- **Clear Method Names** with guiding comments:
  1. `add_item()` - Adds items or updates quantity if item exists
  2. `remove_item()` - Removes items by name with search logic
  3. `calculate_subtotal()` - Uses loops to sum all item costs
  4. `apply_discount()` - Conditional logic for tiered discounts:
     - 5% for orders > $100 (SILVER)
     - 10% for orders > $250 (GOLD)
     - 15% for orders > $500 (PREMIUM)
  5. `calculate_total_bill()` - Complete billing with subtotal, discount, tax (8%), and final total
  6. `display_cart()` and `display_bill()` - Formatted output methods
- **Complete Shopping Cart Functionality**:
  - Add items (multiple quantities)
  - Update quantities for duplicate items
  - Remove items by name

OUTPUT:



```
======================================================================
E-COMMERCE SHOPPING CART SYSTEM - DEMO
======================================================================

--- ADDING ITEMS TO CART ---

✓ Added 'Laptop' ($899.99 x 1) to cart
✓ Added 'USB-C Cable' ($15.99 x 2) to cart
✓ Added 'Wireless Mouse' ($29.99 x 1) to cart
✓ Added 'Monitor Stand' ($45.00 x 1) to cart

======================================================================
SHOPPING CART - Alice Smith
======================================================================
Item Name                      Price   Qty        Total
----------------------------------------------------------------------
Laptop                    $   899.99     1 $     899.99
USB-C Cable               $    15.99     2 $      31.98
Wireless Mouse            $    29.99     1 $      29.99
Monitor Stand             $    45.00     1 $      45.00
----------------------------------------------------------------------

--- ADDING MORE ITEMS ---

✓ Added 'Keyboard' ($79.99 x 1) to cart
✓ Updated 'USB-C Cable' quantity to 3

======================================================================
SHOPPING CART - Alice Smith
======================================================================
Item Name                      Price   Qty        Total
----------------------------------------------------------------------
Laptop                    $   899.99     1 $     899.99
USB-C Cable               $    15.99     3 $      47.97
Wireless Mouse            $    29.99     1 $      29.99
Monitor Stand             $    45.00     1 $      45.00
Keyboard                  $    79.99     1 $      79.99
----------------------------------------------------------------------

--- REMOVING ITEM ---

✓ Removed 'Monitor Stand' from cart

======================================================================
SHOPPING CART - Alice Smith
======================================================================
Item Name                      Price   Qty        Total
----------------------------------------------------------------------
Laptop                    $   899.99     1 $     899.99
USB-C Cable               $    15.99     3 $      47.97
```



```
--- GENERATING BILL ---

======================================================================
BILLING SUMMARY
======================================================================
Subtotal:                    $   1057.94
Discount (PREMIUM (15%)  ) 15%   -$    158.69
Subtotal After Discount:     $    899.25
Tax (8%):                    $     71.94
----------------------------------------------------------------------
TOTAL AMOUNT:                $    971.19
======================================================================


======================================================================
SECOND DEMO - HIGH-VALUE PURCHASE WITH PREMIUM DISCOUNT
======================================================================

--- ADDING HIGH-VALUE ITEMS ---

✓ Added 'Desktop Computer' ($1299.99 x 1) to cart
✓ Added 'Gaming Monitor' ($399.99 x 2) to cart
✓ Added 'RGB Keyboard' ($129.99 x 1) to cart
✓ Added 'Professional Headset' ($199.99 x 1) to cart

======================================================================
SHOPPING CART - Bob Johnson
======================================================================
Item Name                      Price   Qty        Total
----------------------------------------------------------------------
Desktop Computer          $  1299.99     1 $    1299.99
Gaming Monitor            $   399.99     2 $     799.98
RGB Keyboard              $   129.99     1 $     129.99
Professional Headset      $   199.99     1 $     199.99
----------------------------------------------------------------------

======================================================================
BILLING SUMMARY
======================================================================
Subtotal:                    $   2429.95
Discount (PREMIUM (15%)  ) 15%   -$    364.49
Subtotal After Discount:     $   2065.46
Tax (8%):                    $    165.24
----------------------------------------------------------------------
TOTAL AMOUNT:                $   2230.69
======================================================================

======================================================================
DEMO COMPLETE
```