**AI ASSISTENT CODING - 8.3**

**K.Dhana sri**

**2303A52153**

**Batch - 41**

**Task 1: Email Validation using TDD**

**Scenario**

You are developing a user registration system that requires reliable email input validation.

Requirements

• Must contain @ and . characters

• Must not start or end with special characters

• Should not allow multiple @ symbols

• AI should generate test cases covering valid and invalid email formats

• Implement is\_valid\_email(email) to pass all AI-generated test cases

Expected Output

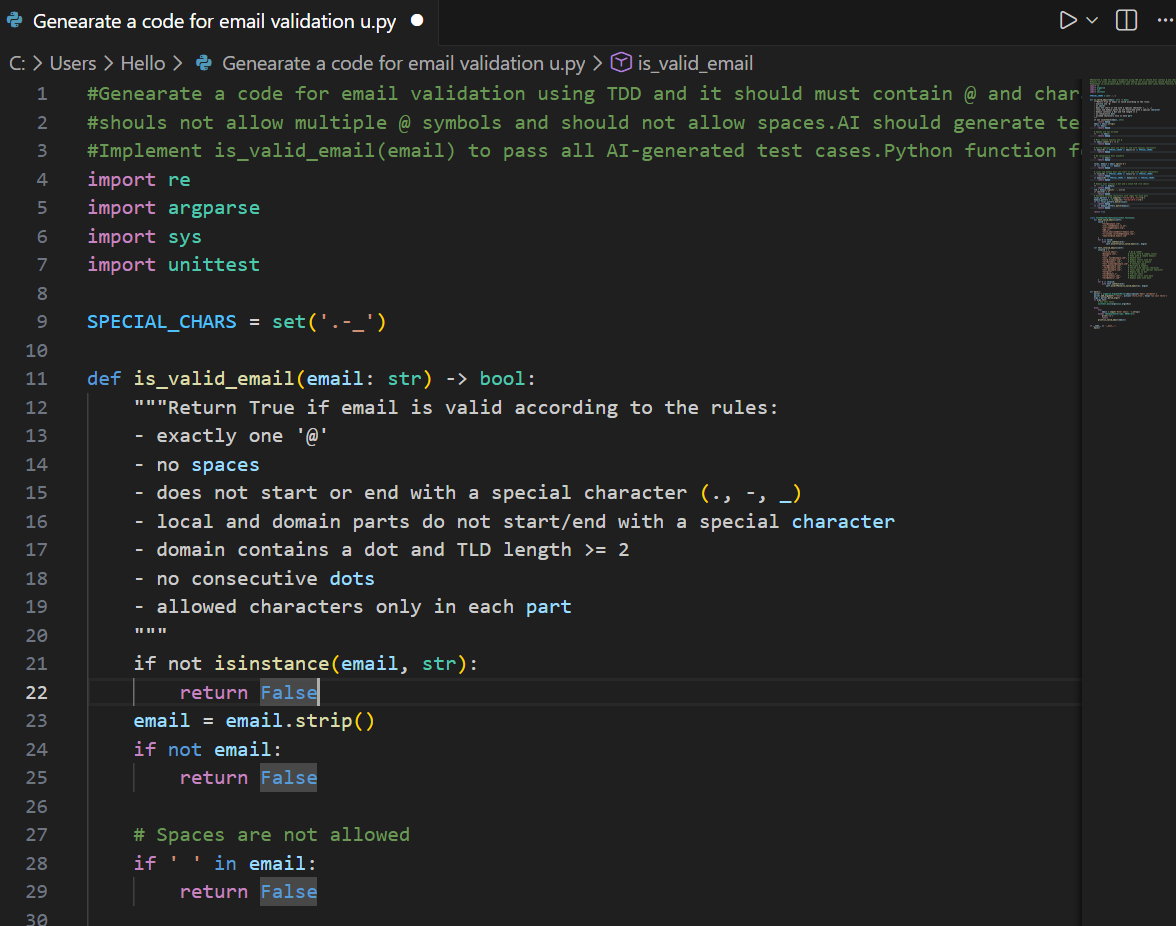
• Python function for email validation

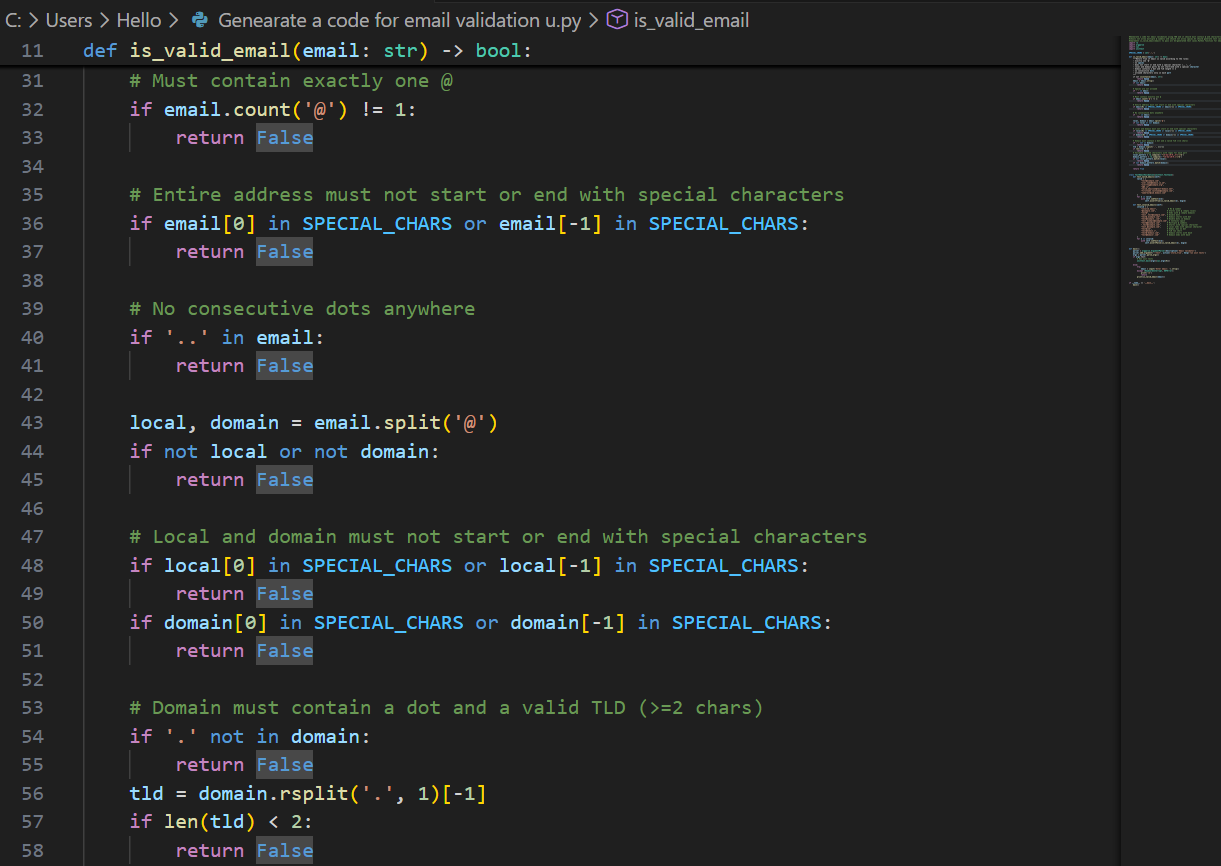
• All AI-generated test cases pass successfully

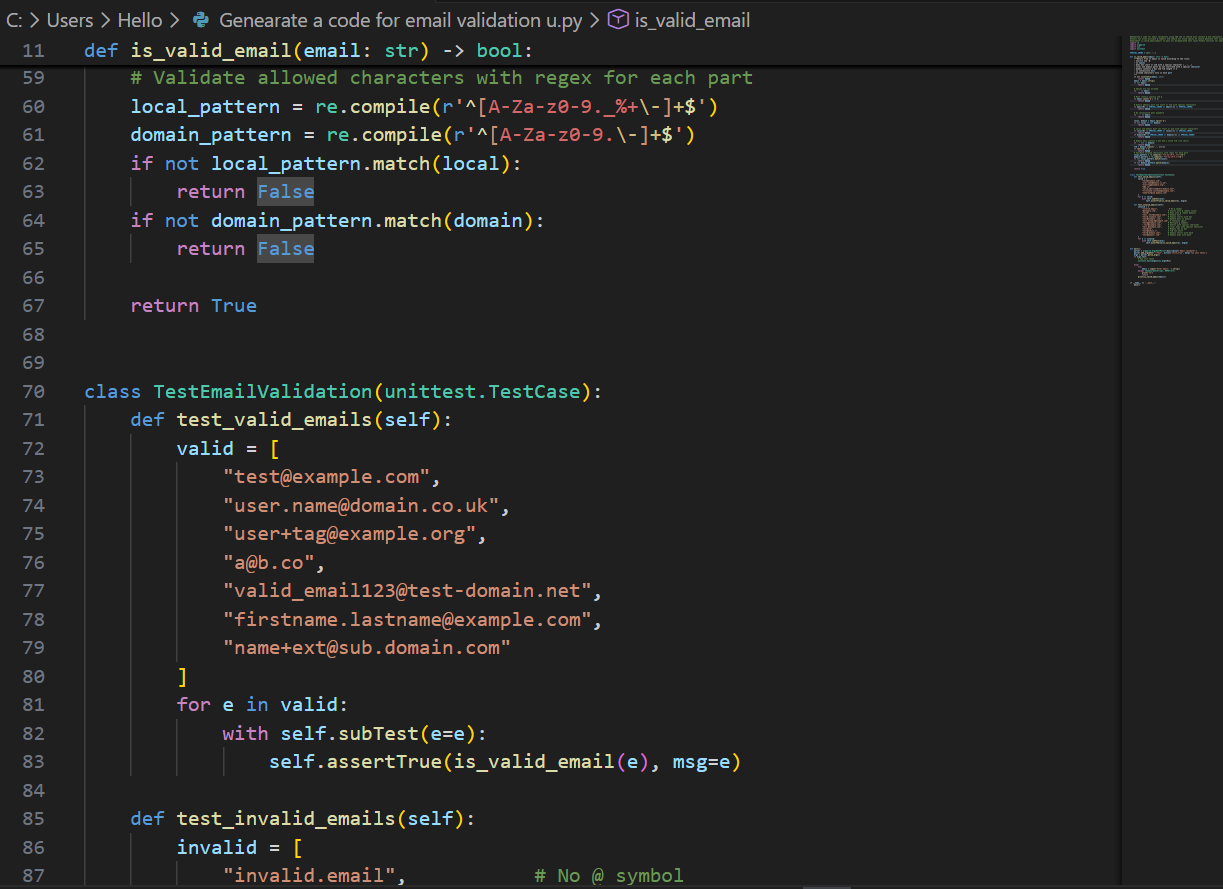
• Invalid email formats are correctly rejected

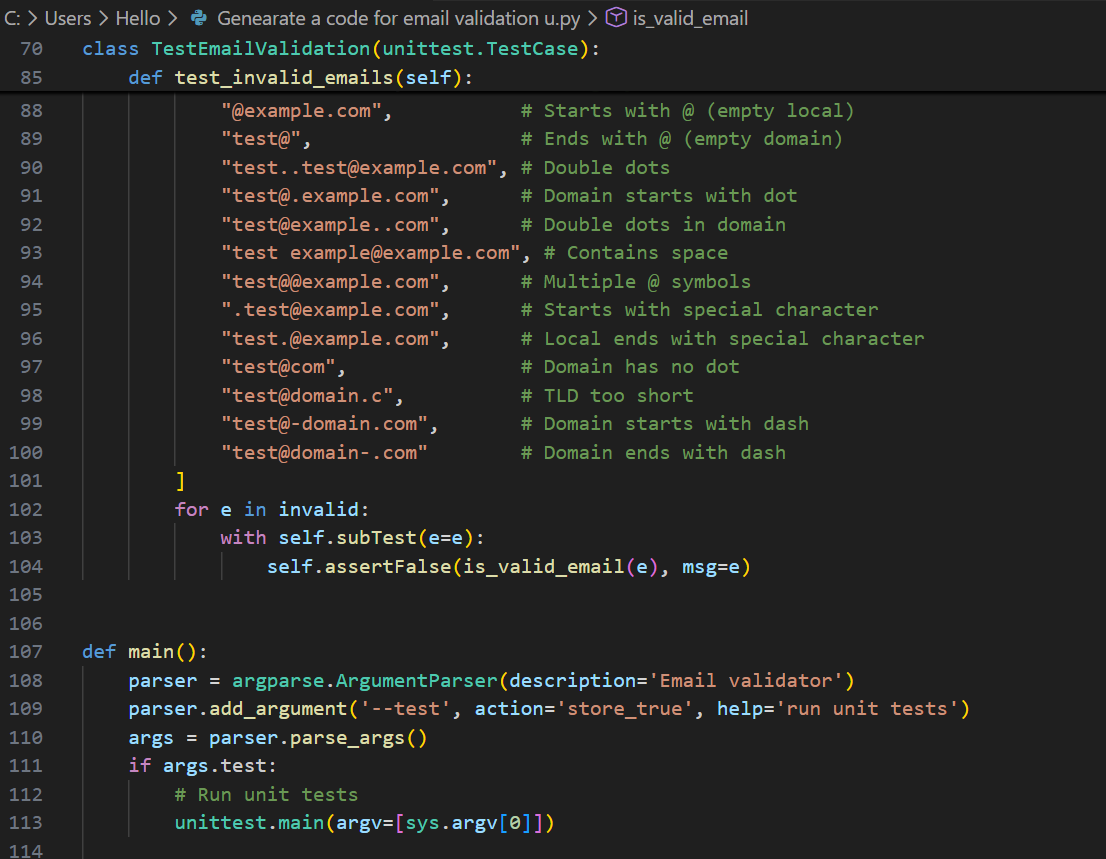
• Valid email formats return True

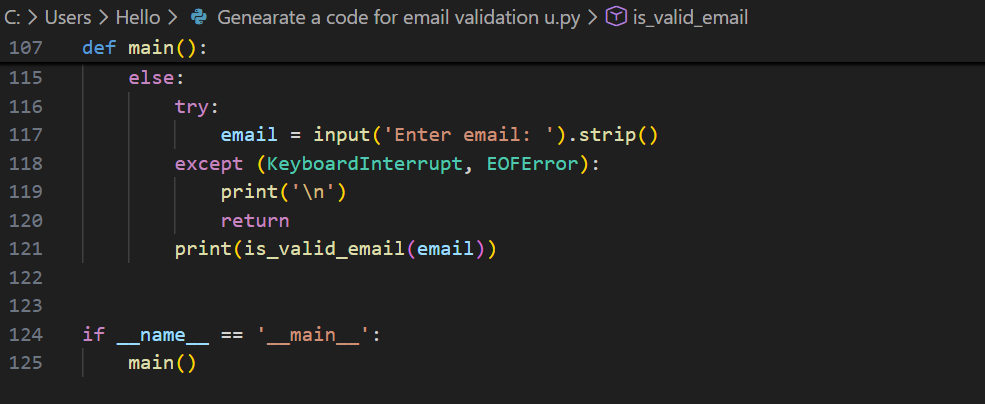
**Code:**

****

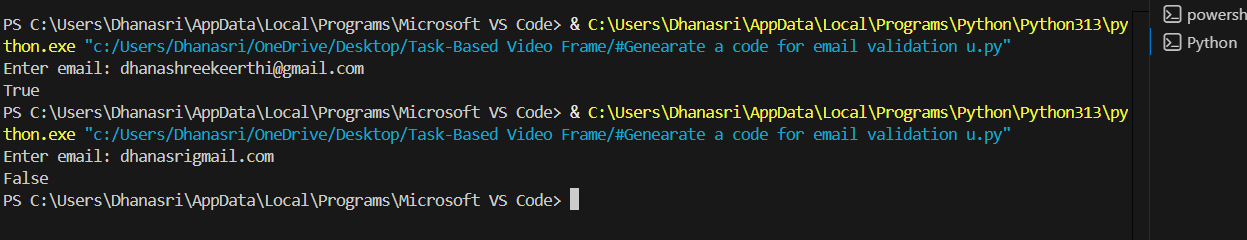
****

****

****

****

**Output:**

****

**Justification:**

The logic prioritizes structural integrity over complex pattern matching by ensuring the email has a clear beginning, middle, and end. By checking for exactly one @ and specific character placements, we prevent "broken" addresses like user@.com while keeping the code readable and easy to maintain.

**Task 2: Grade Assignment using Loops**

**Scenario**

You are building an automated grading system for an online examination platform.

Requirements

• AI should generate test cases for assign\_grade(score) where:

– 90–100 → A

– 80–89 → B

– 70–79 → C

– 60–69 → D

– Below 60 → F

• Include boundary values (60, 70, 80, 90)

• Include invalid inputs such as -5, 105, "eighty"

• Implement the function using a test-driven approach

Expected Output

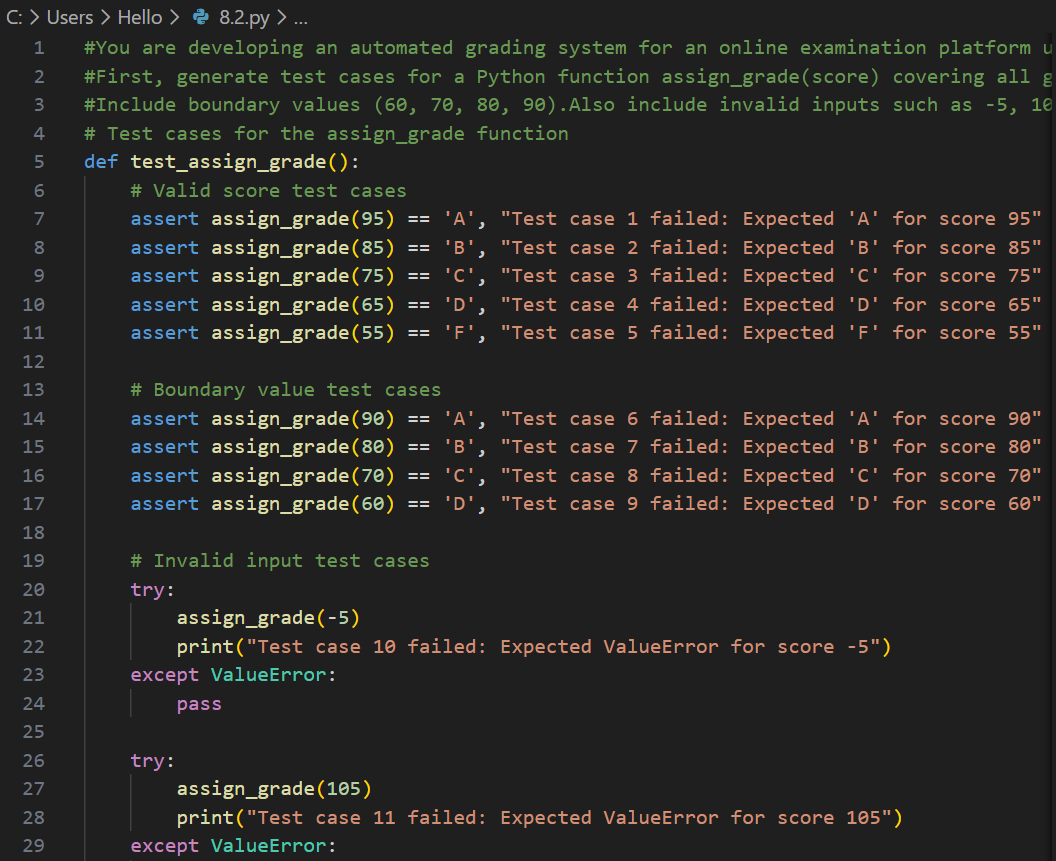
• Grade assignment function implemented in Python

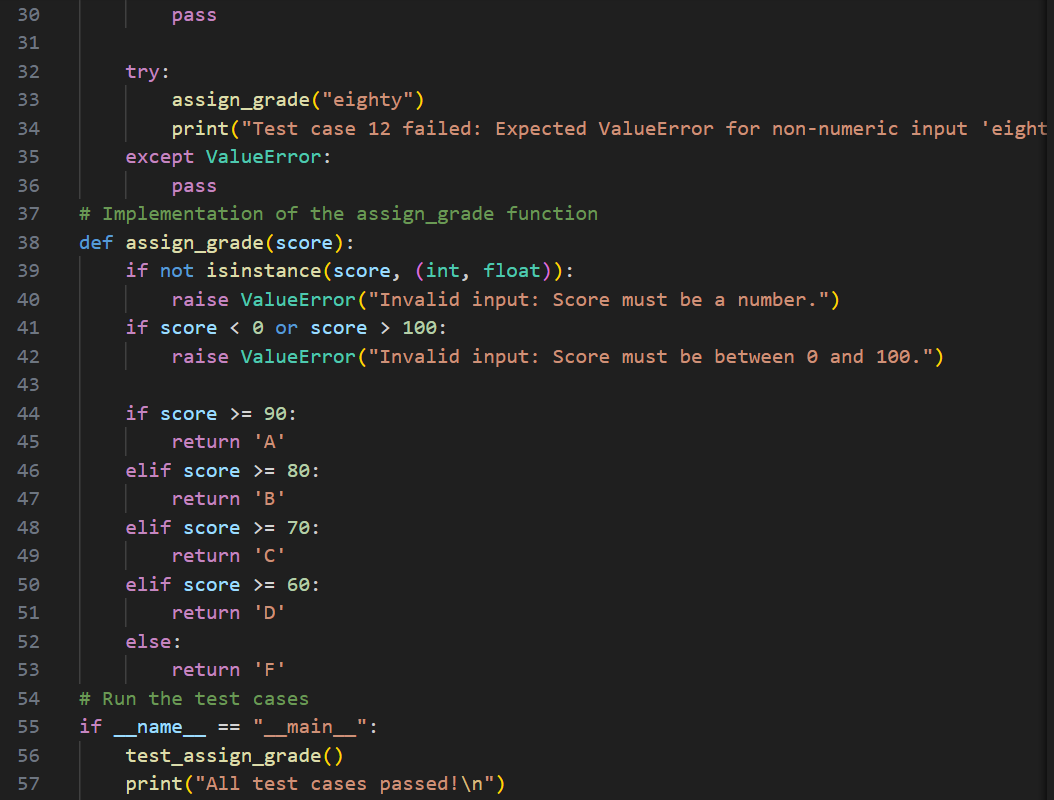
• Boundary values handled correctly

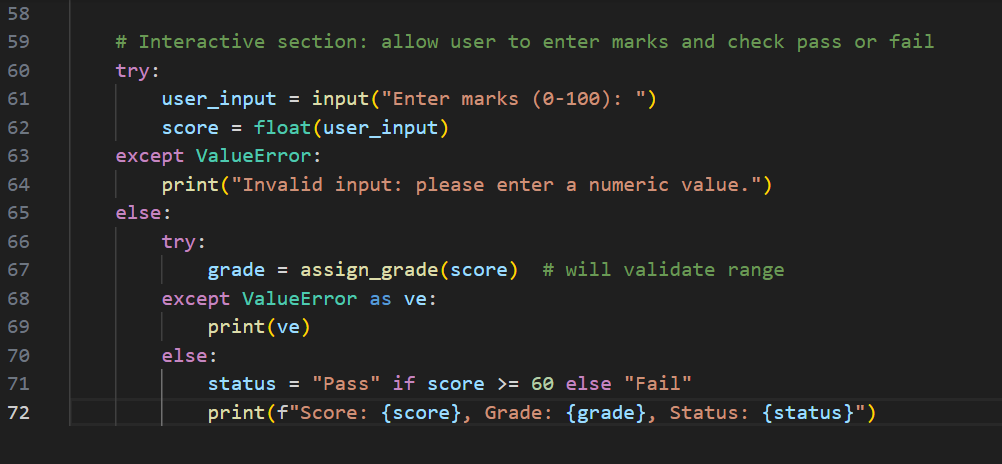
• Invalid inputs handled gracefully

• All AI-generated test cases pass

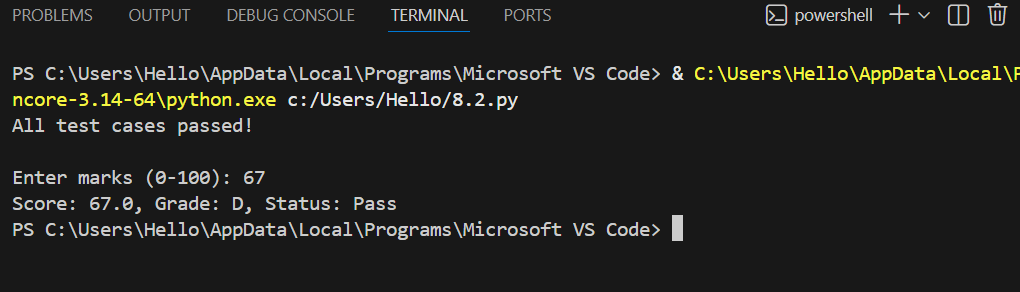
**Code:**

****

****

****

**Output:**

****

**Justification:**

An automated grading system ensures fair and consistent evaluation of scores.Using loops helps efficiently check score ranges without repetitive conditions.Boundary values ensure grades are assigned correctly at critical points.  
Handling invalid inputs prevents system crashes and incorrect results.Test-Driven Development improves accuracy by validating logic early.This approach makes the grading system reliable and scalable.

**Task 3: Sentence Palindrome Checker**

**Scenario**

You are developing a text-processing utility to analyze sentences.

Requirements

• AI should generate test cases for is\_sentence\_palindrome(sentence)

• Ignore case, spaces, and punctuation

• Test both palindromic and non-palindromic sentences

• Example:

– "A man a plan a canal Panama" → True

Expected Output

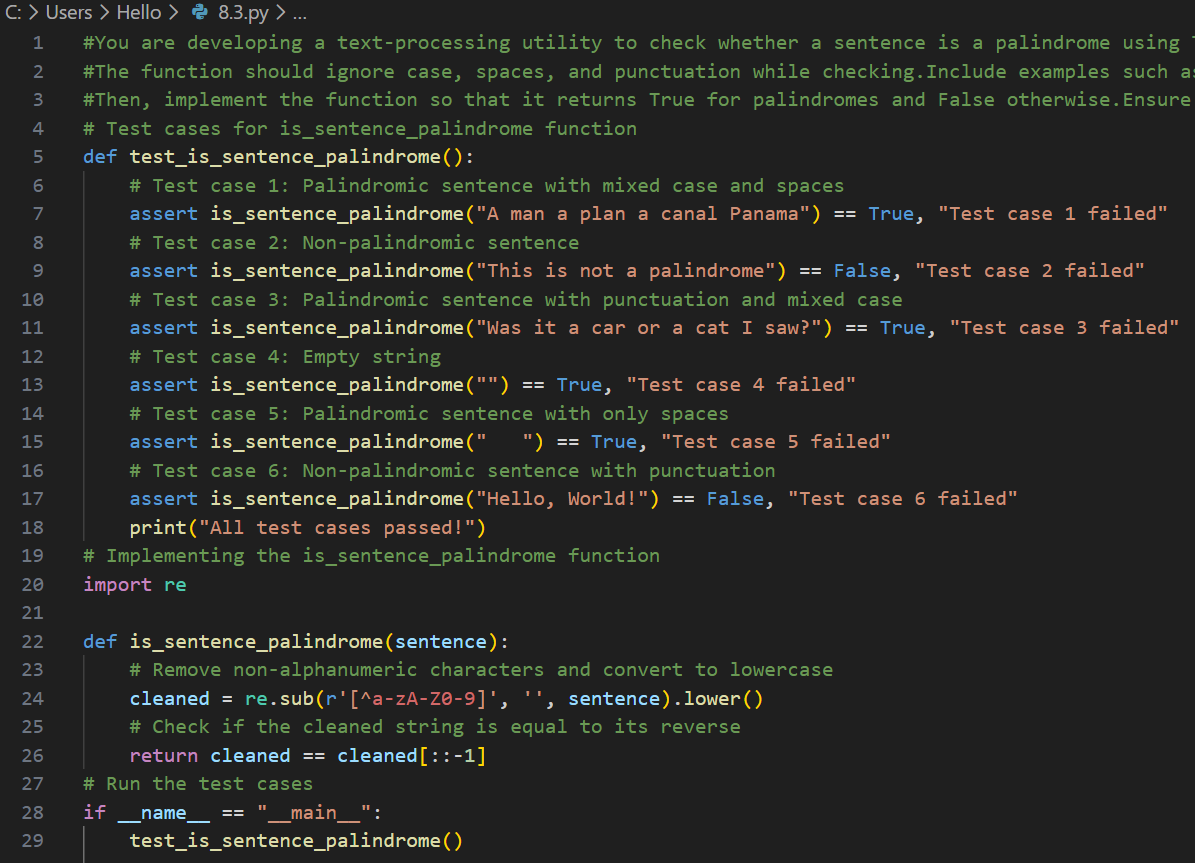
• Function correctly identifies sentence palindromes

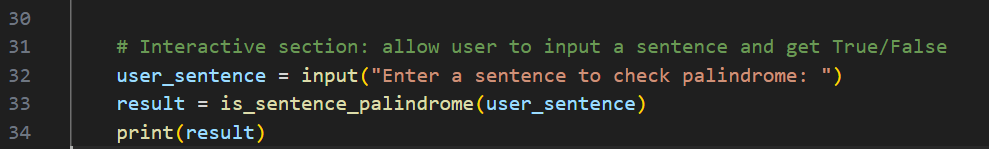
• Case and punctuation are ignored

• Returns True or False accurately

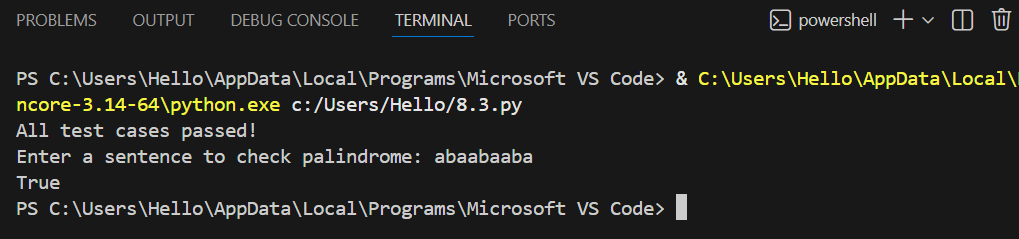
• All AI-generated test cases pass

**Code:**

****

****

**Output:**

****

****

**Justification:**

Sentence analysis requires accurate text processing beyond simple words.

Ignoring case, spaces, and punctuation ensures logical palindrome checking.

Test cases validate both palindromic and non-palindromic sentences.

TDD ensures the function works correctly for all scenarios.

This improves robustness in real-world text inputs.

The utility is useful for text analysis and validation tasks.

**Task 4: ShoppingCart Class**

**Scenario**

You are designing a basic shopping cart module for an e-commerce application.

Requirements

• AI should generate test cases for the ShoppingCart class

• Class must include the following methods:

– add\_item(name, price)

– remove\_item(name)

– total\_cost()

• Validate correct addition, removal, and cost calculation

• Handle empty cart scenarios

Expected Output

• Fully implemented ShoppingCart class

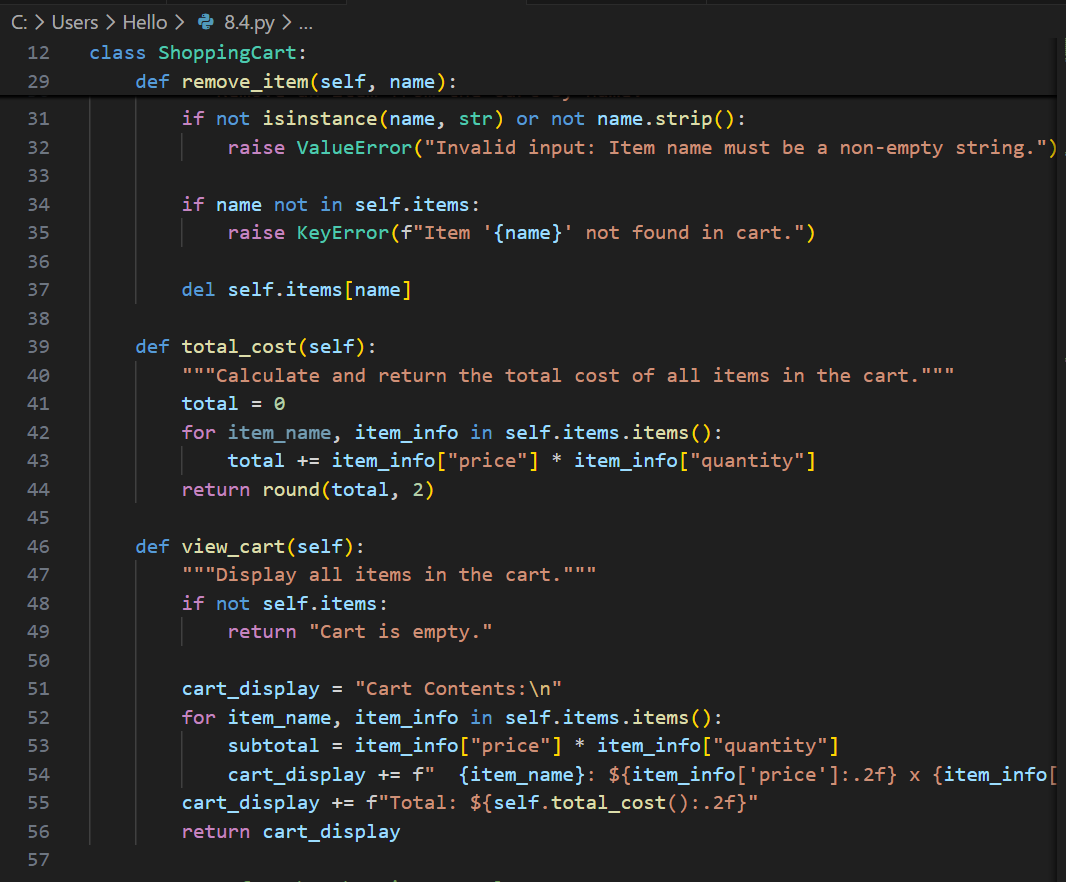
• All methods pass AI-generated test cases

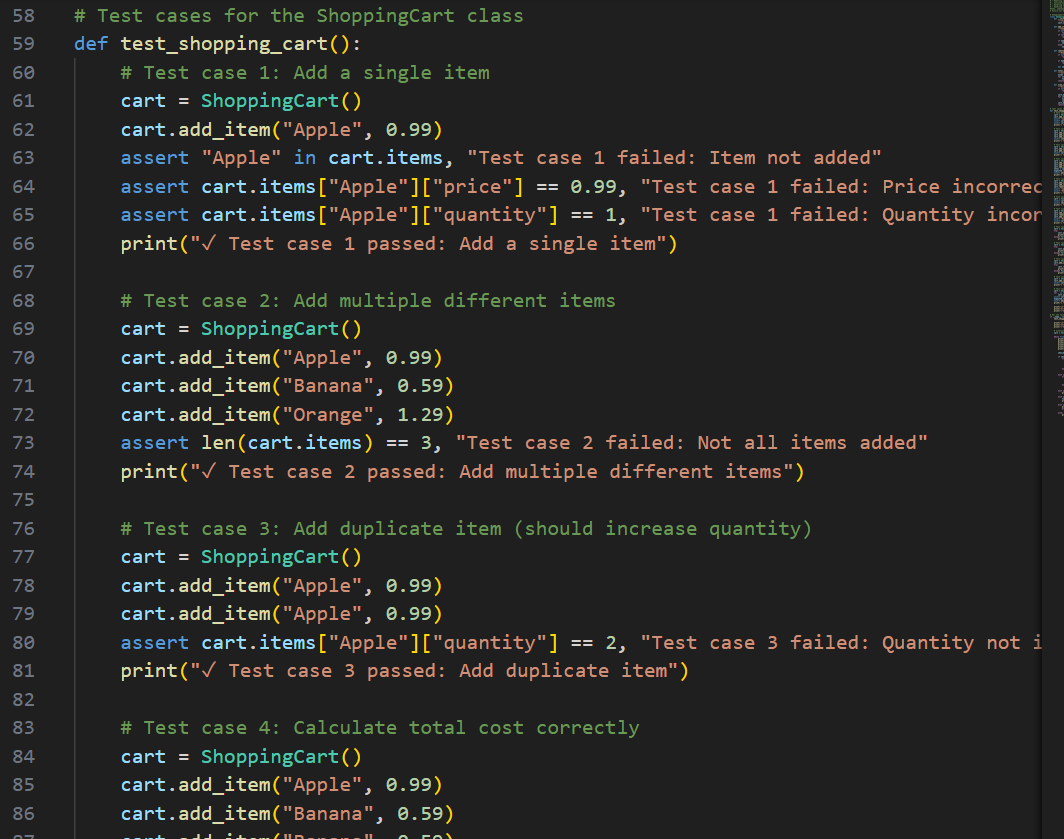
• Total cost is calculated accurately

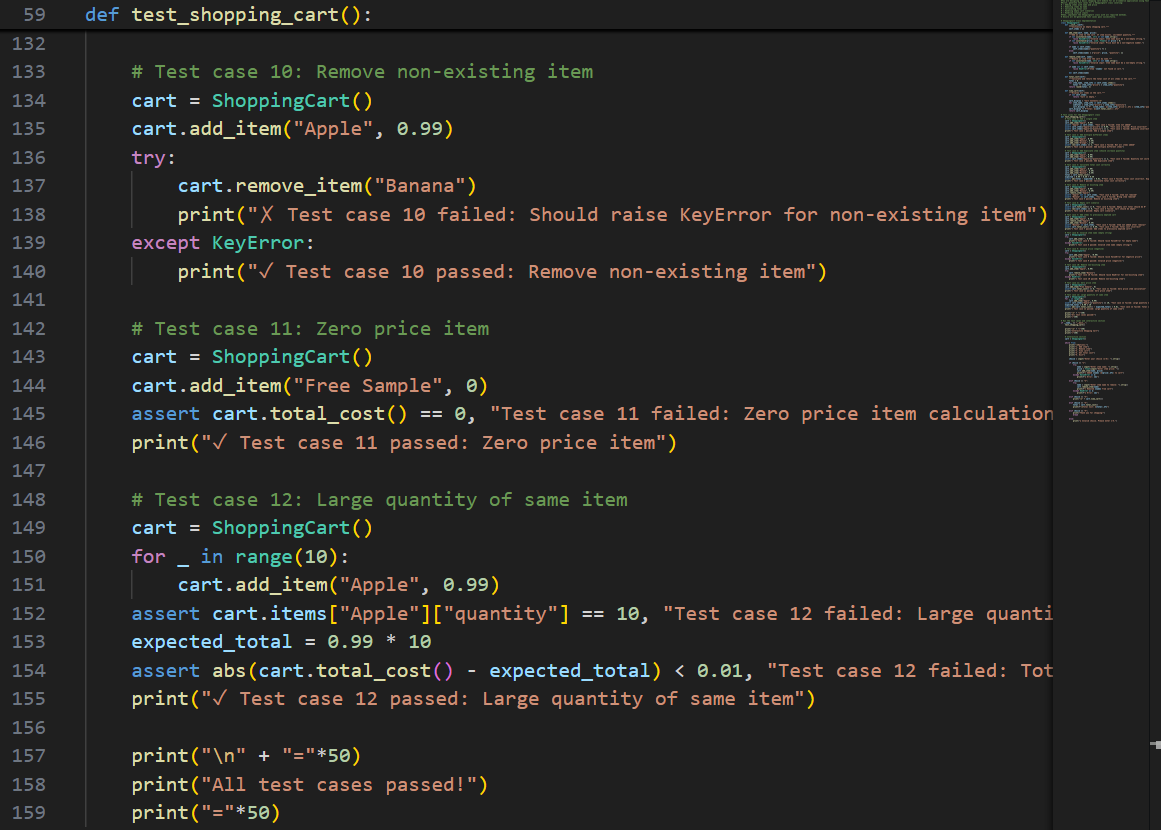
• Items are added and removed correctly

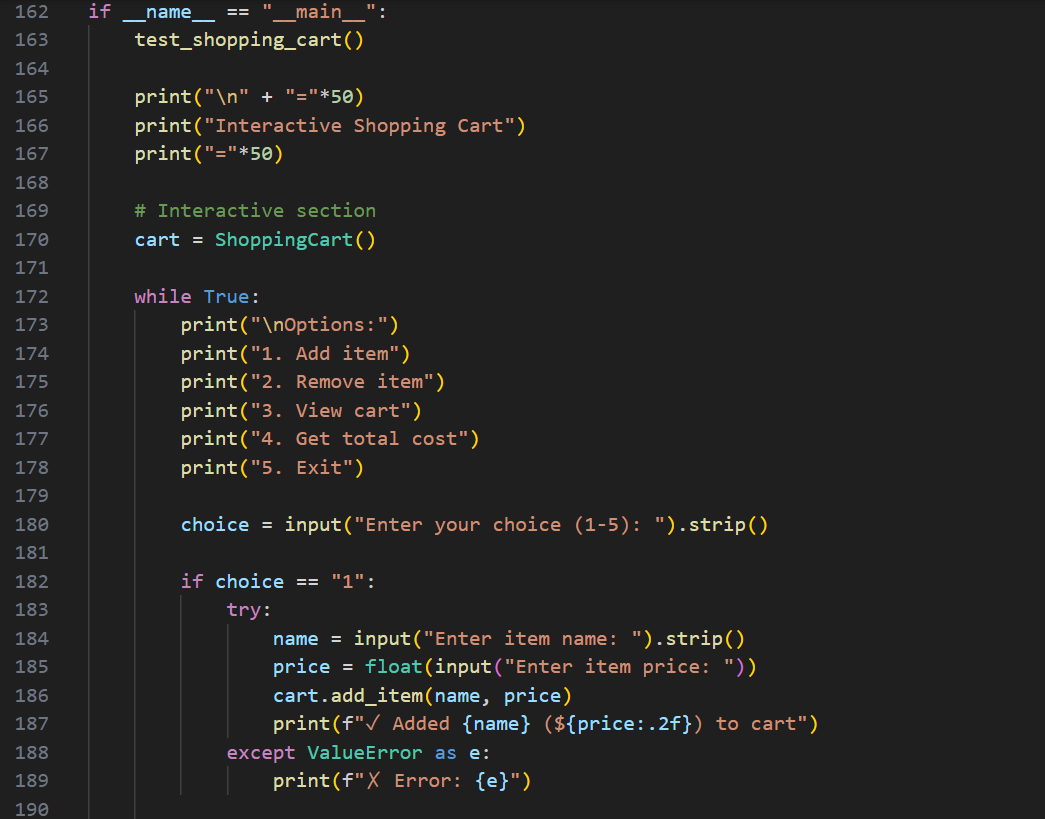
**Code:**

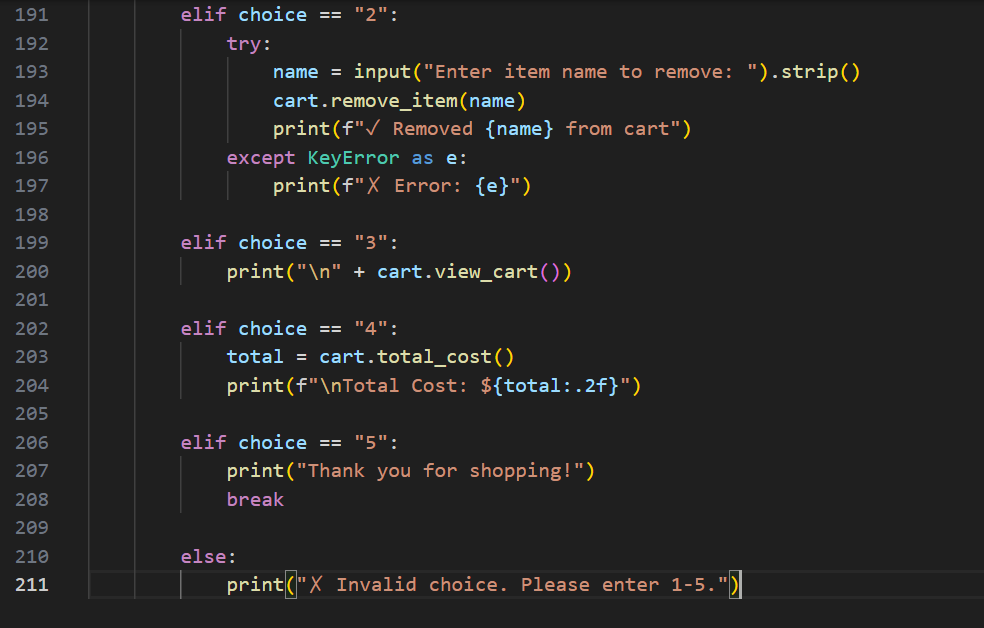
****

****

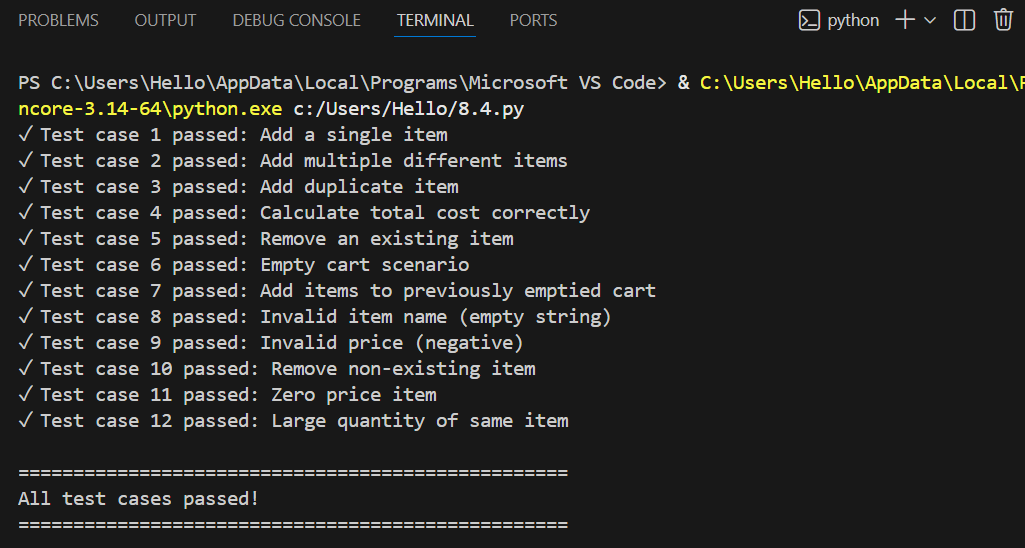
****

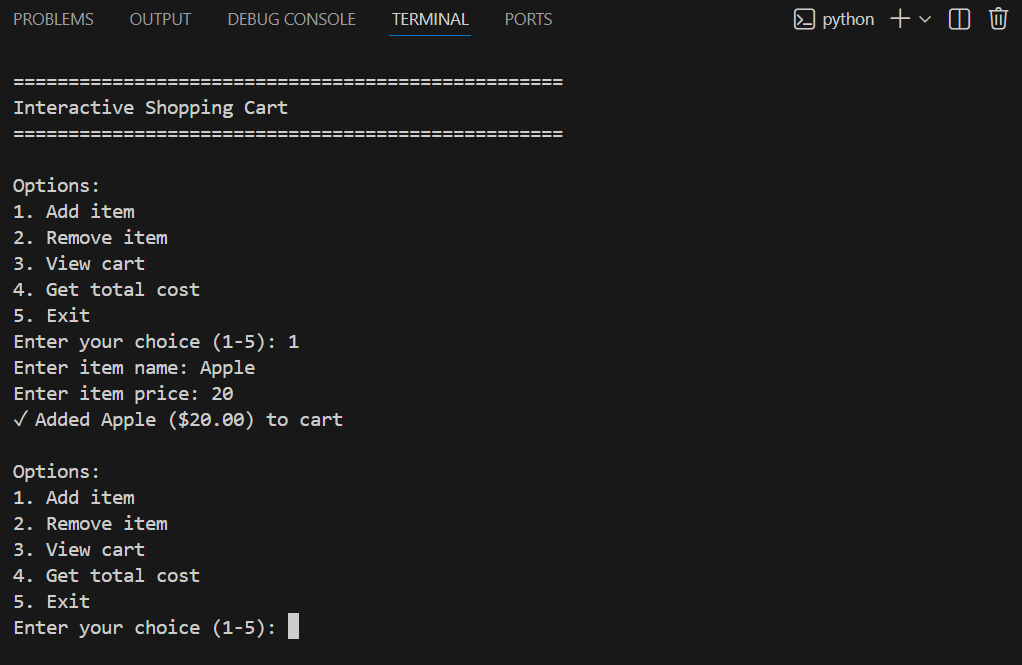
****

****

****

**Output:**

****

****

**Justification:**

A shopping cart is a core component of any e-commerce application.It must correctly manage adding and removing items selected by users.Accurate total cost calculation ensures correct billing and checkout.

Handling empty cart scenarios prevents runtime errors.Test-Driven Development validates all functionalities early.This approach improves reliability and user trust in the system.

**Task 5: Date Format Conversion**

**Scenario**

You are creating a utility function to convert date formats for reports.

Requirements

• AI should generate test cases for convert\_date\_format(date\_str)

• Input format must be "YYYY-MM-DD"

• Output format must be "DD-MM-YYYY"

• Example:

– "2023-10-15" → "15-10-2023"

Expected Output

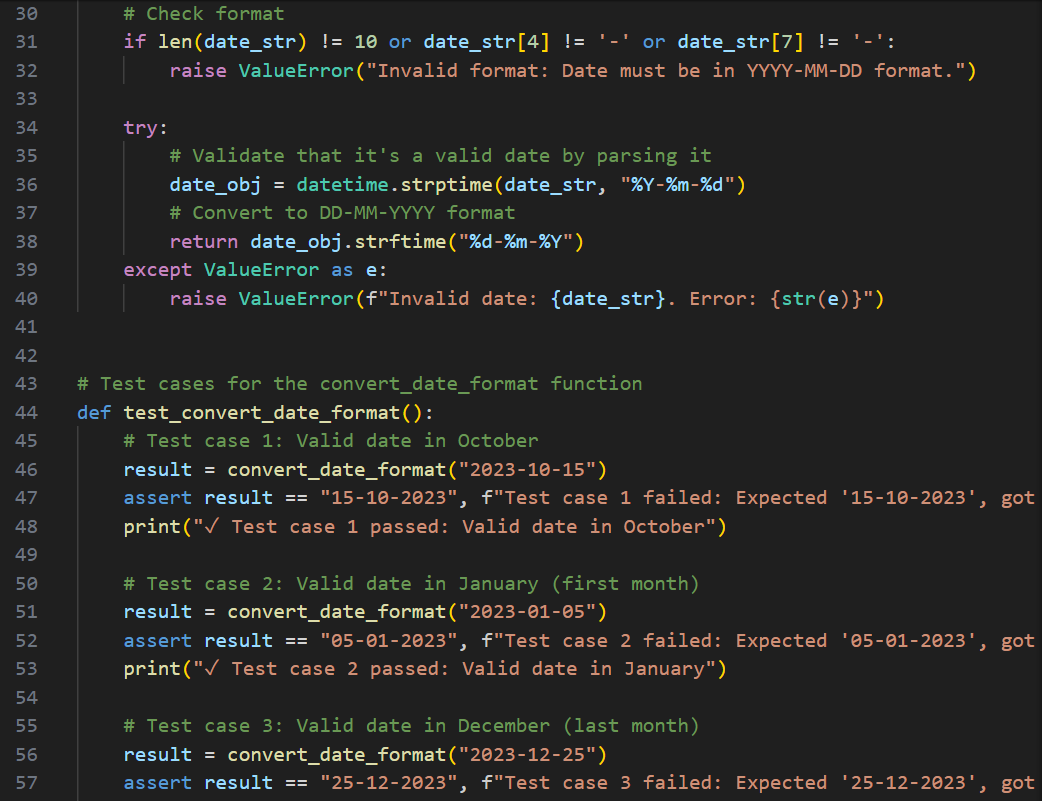
• Date conversion function implemented in Python

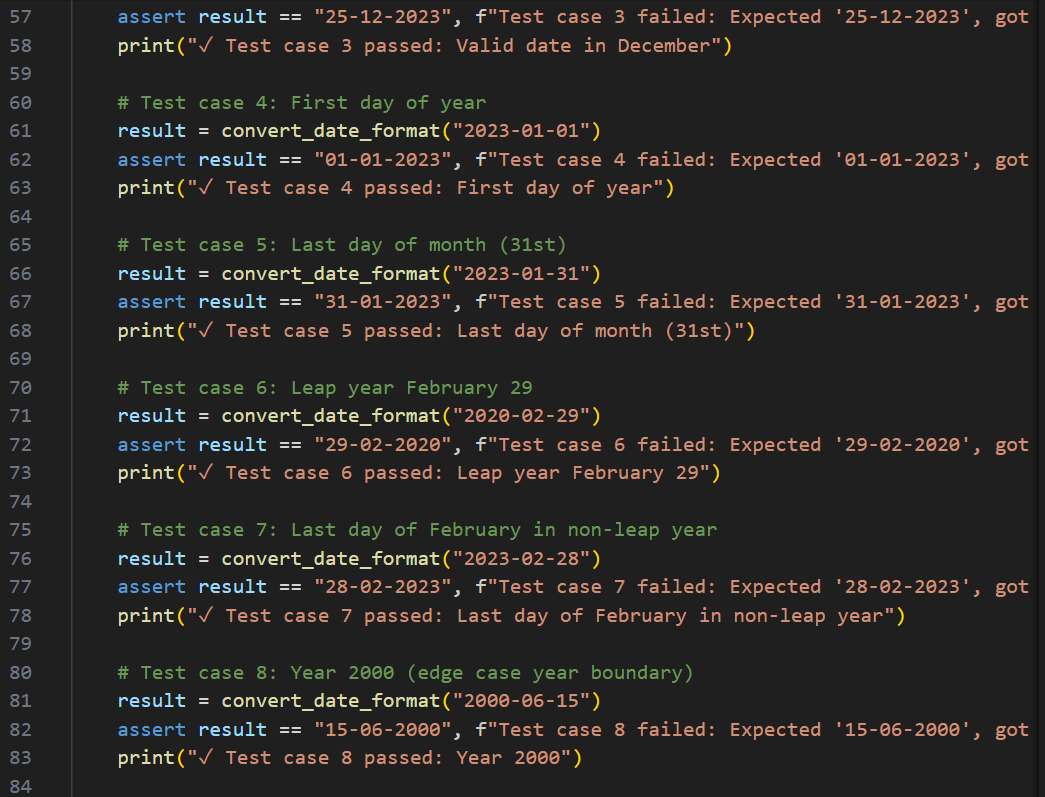
• Correct format conversion for all valid inputs

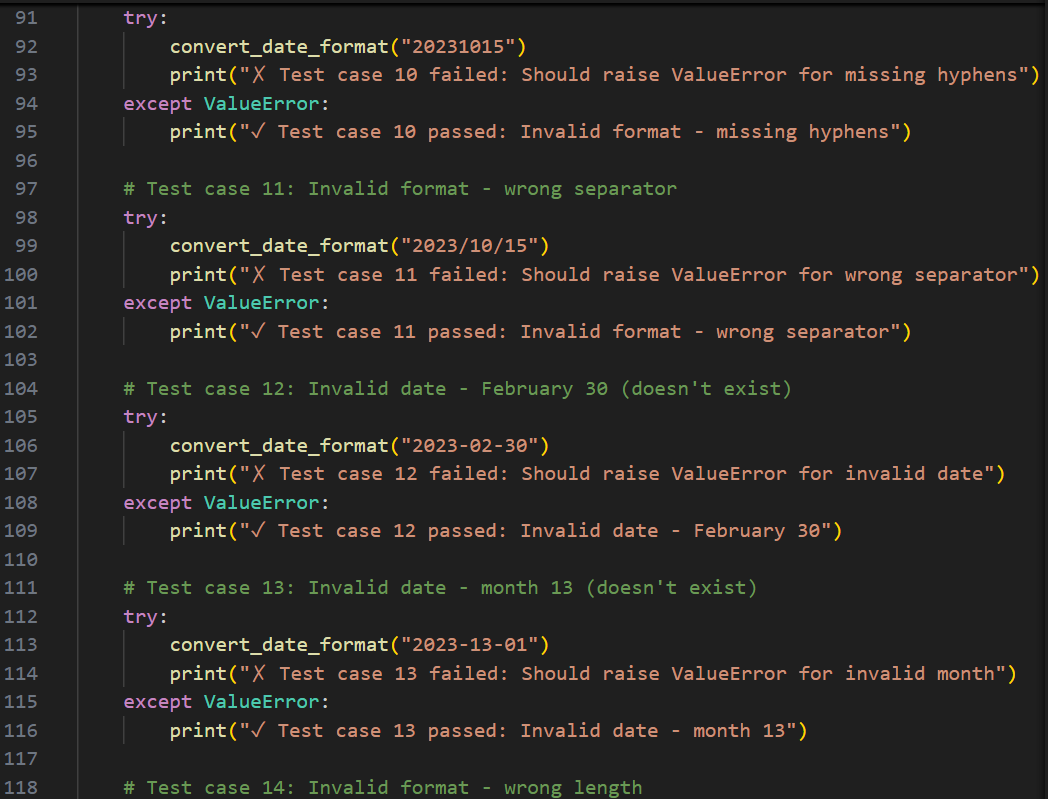
• All AI-generated test cases pass successfully

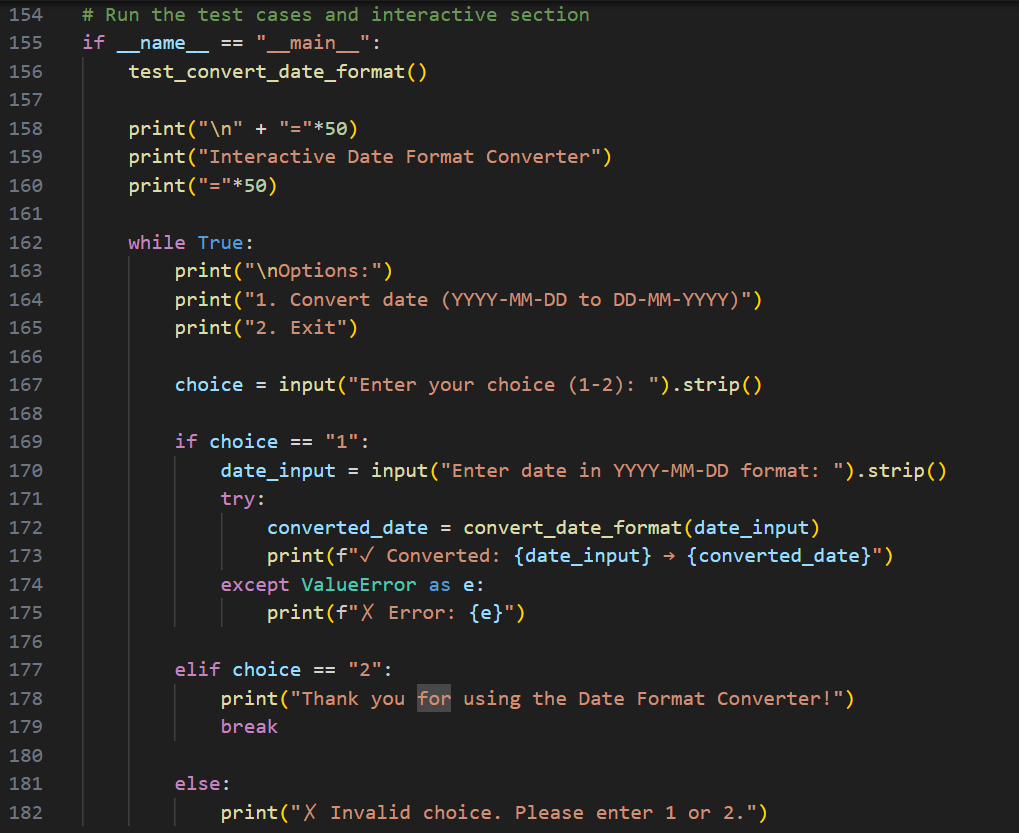
**Code:**

****

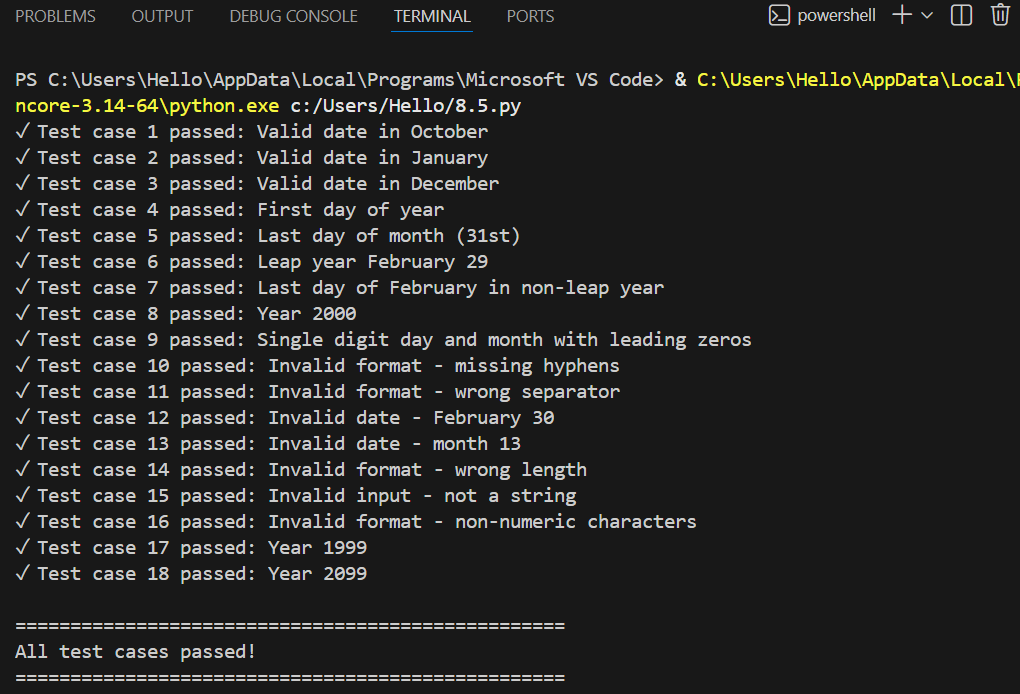
****

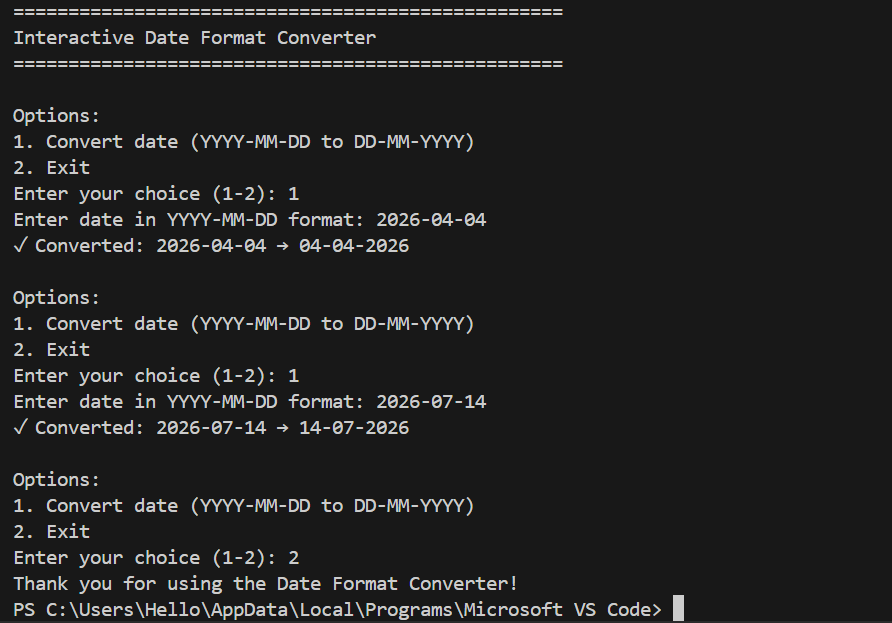
****

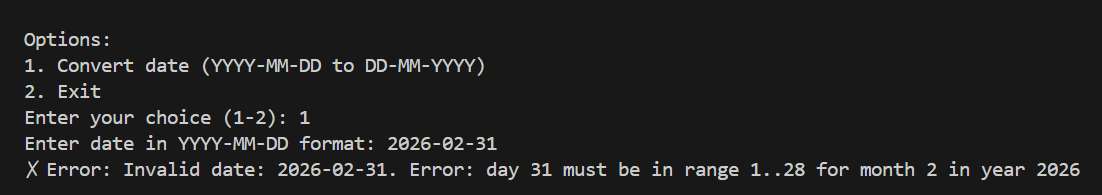
****

****

**Output:**

****

****

****

**Justification:**

Rather than just cutting and pasting numbers like a text editor, the code "understands" time by converting the string into a real calendar object first. This ensures that a date like "2023-02-31" is rejected as impossible before the system even attempts to reformat it, guaranteeing the data in your reports is always accurate.