Name:K.Rajavrdhan Reddy

2403A51108

Lab:8.4

Task1:

Use AI to generate test cases for is\_valid\_email(email) and then implement the validator function.  
Requirements:  
• Must contain @ and . characters.  
• Must not start or end with special characters.  
• Should not allow multiple @

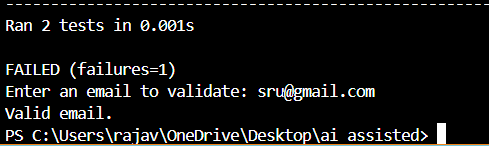
Prompt:

generate test cases for is\_valid\_email(email) and then implement the  
validator function

code:



Output:



Explanation:

* **Imports**:
  + re: For regex-based email validation.
  + unittest: For automated testing.
* **Function**:  
  is\_valid\_email(email) uses a regex pattern to check if the email format is valid (e.g., has @, domain, and TLD).
* **Tests**:
  + TestIsValidEmail: Contains two test methods.
    - test\_valid\_email: Checks known good emails.
    - test\_invalid\_email: Checks common bad formats (missing @, domain, etc.).
* **Main Block**:
  + Runs the tests.
  + Prompts user to enter an email.
  + Prints whether it’s valid or not.

Task2:

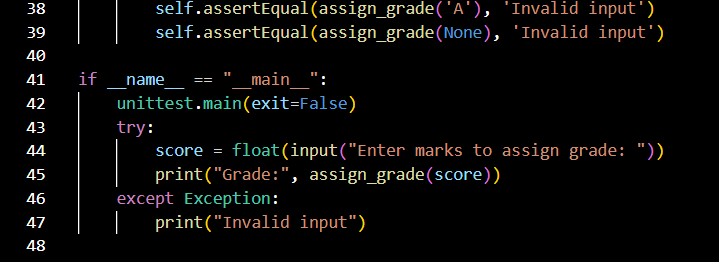
Ask AI to generate test cases for assign\_grade(score) function. Handle boundary and invalid inputs.  
Requirements  
• AI should generate test cases for assign\_grade(score) where: 90-100: A, 80-89: B, 70-  
79: C, 60-69: D, <60: F  
• Include boundary values and invalid inputs (e.g., -5, 105, "eighty").

Prompt:

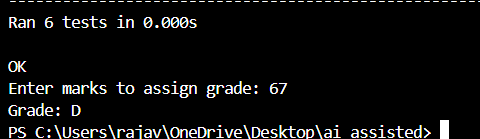
generate test cases for assign\_grade(score) function. Handle boundary and invalid inputs.

Code:





Output:



Explanation:

**🎓 Function: assign\_grade(score)**

This function maps a numeric score to a letter grade:

* **Input validation**:
  + Must be an int or float.
  + Must be between 0 and 100.
  + Otherwise, returns 'Invalid input'.
* **Grading logic**:
  + 90–100 → 'A'
  + 80–89 → 'B'
  + 70–79 → 'C'
  + 60–69 → 'D'
  + <60 → 'F'

**🧪 Unit Tests: TestAssignGrade**

Using unittest to verify correctness:

* **test\_A\_grade to test\_F\_grade**: Confirms correct letter grade for valid scores.
* **test\_invalid\_input**: Handles out-of-range values and wrong types like strings or None.

**🖥️ Main Execution**

* Runs the test suite.
* Prompts the user to enter a score.
* Converts input to float and prints the corresponding grade.
* Catches exceptions (e.g., non-numeric input) and prints "Invalid input".

Task3:

Generate test cases using AI for is\_sentence\_palindrome(sentence). Ignore case,punctuation, and spaces  
Requirement  
• Ask AI to create test cases for is\_sentence\_palindrome(sentence)  
(ignores case, spaces, and punctuation).

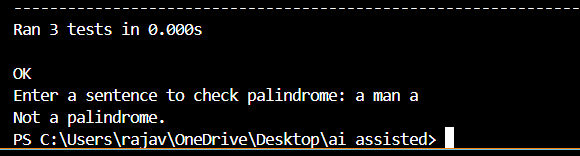
Prompt:

Generate test cases using AI for is\_sentence\_palindrome(sentence). Ignore case,punctuation, and spaces.

Code:



Output:



Explanation:

**🔍 Function: is\_sentence\_palindrome(sentence)**

* **Goal**: Check if a sentence reads the same forward and backward, ignoring spaces, punctuation, and case.
* **Steps**:
  + Removes all non-alphanumeric characters.
  + Converts everything to lowercase.
  + Compares the cleaned string to its reverse.

**🧪 Unit Tests: TestIsSentencePalindrome**

* **test\_true\_palindromes**: Validates famous palindromic phrases.
* **test\_false\_palindromes**: Ensures non-palindromes return False.
* **test\_empty\_and\_single**: Handles edge cases like empty strings and single characters.

**🖥️ Main Execution**

* Runs the tests.
* Prompts the user to enter a sentence.
* Prints whether it’s a palindrome or not.

Task4:

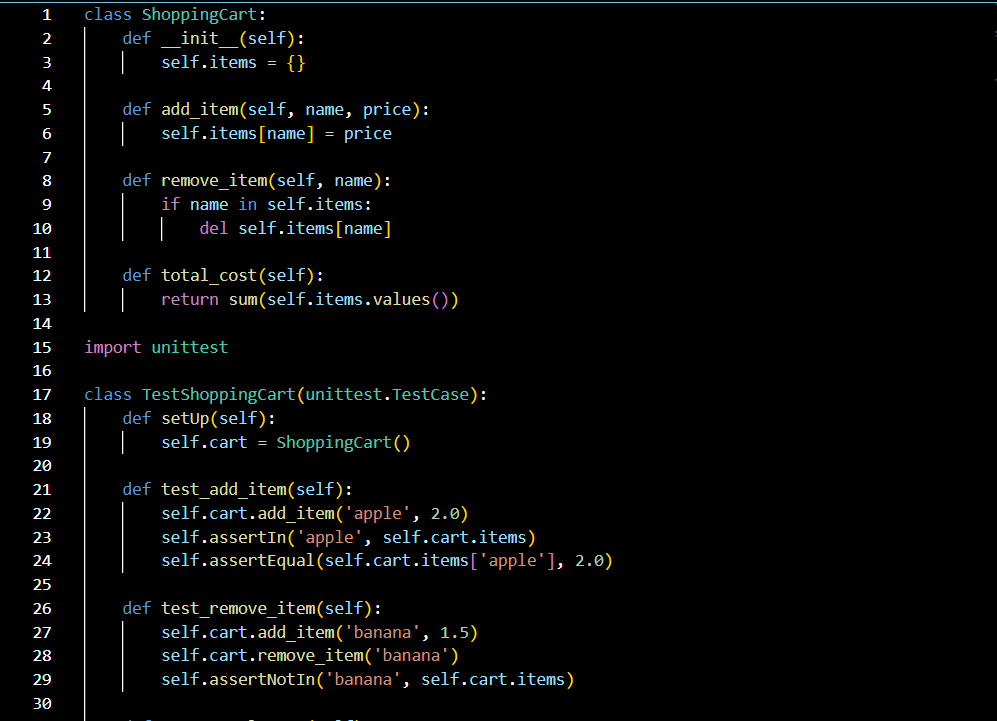
• Let AI fix it Prompt AI to generate test cases for a ShoppingCart class (add\_item, remove\_item, total\_cost).  
Methods:  
Add\_item(name,orice)  
Remove\_item(name)

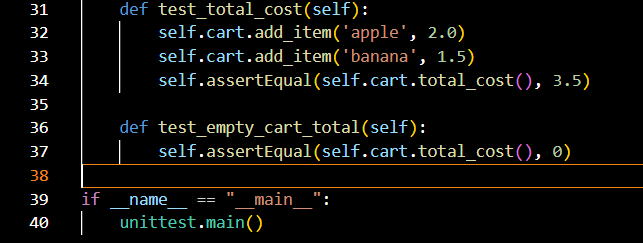
Total\_cost()

Prompt:

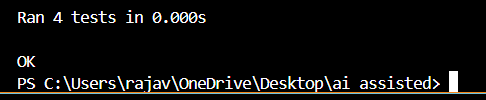
generate test cases for a ShoppingCart class (add\_item,remove\_item,total\_cost).

Code:





Output:



Explanation:

**🛒 Class: ShoppingCart**

This class models a simple shopping cart with basic operations:

* **\_\_init\_\_**: Initializes an empty dictionary items to store item names and their prices.
* **add\_item(name, price)**: Adds or updates an item in the cart.
* **remove\_item(name)**: Removes an item if it exists.
* **total\_cost()**: Returns the sum of all item prices in the cart.

**🧪 Unit Tests: TestShoppingCart**

Using Python’s unittest framework to validate the cart’s behavior:

* **setUp()**: Creates a fresh cart before each test.
* **test\_add\_item()**: Verifies that adding an item stores it correctly.
* **test\_remove\_item()**: Checks that an item is removed properly.
* **test\_total\_cost()**: Confirms the total cost calculation with multiple items.
* **test\_empty\_cart\_total()**: Ensures the total cost is zero when the cart is empty.

**✅ Execution**

When run directly, it executes all tests to confirm the class behaves as expected.

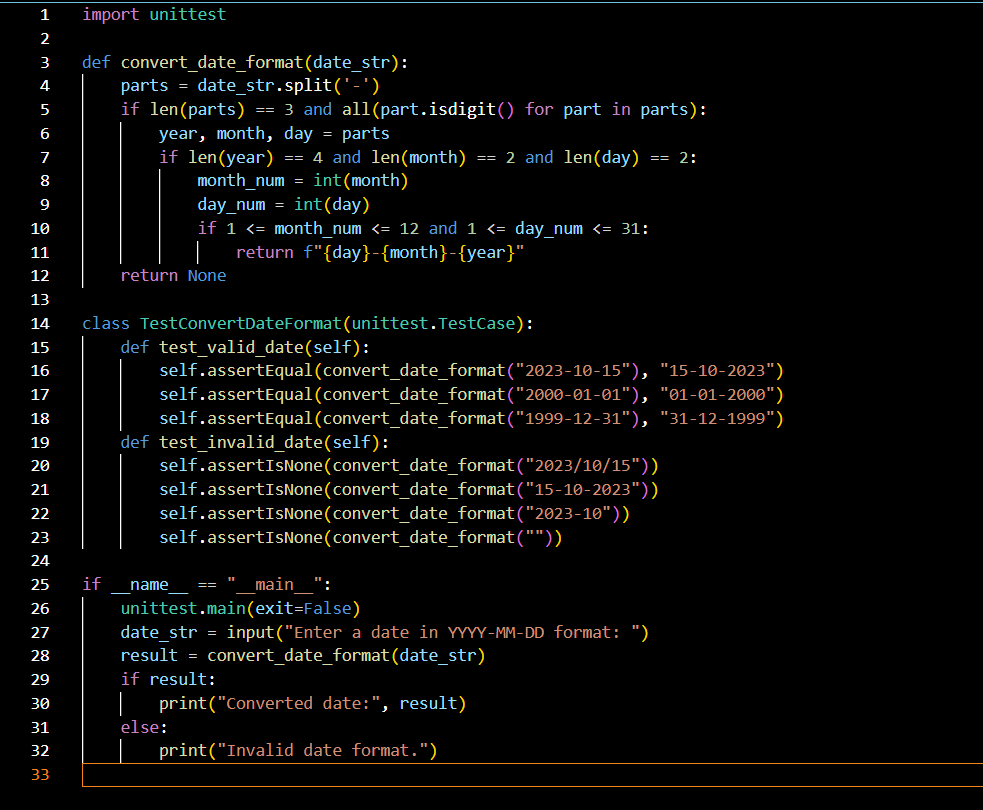
Task5:

Use AI to write test cases for convert\_date\_format(date\_str) to switch from "YYYY-  
MM-DD" to "DD-MM-YYYY".  
Example: "2023-10-15" → "15-10-2023"

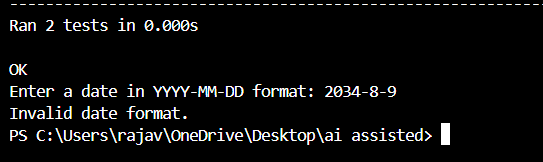
Prompt:

write test cases for convert\_date\_format(date\_str) to switch from "YYYY-MM-DD" to "DD-MM-YYYY".

Code:



Output:



Explanation:

**🧠 Purpose**

It converts a date from YYYY-MM-DD format to DD-MM-YYYY format, and includes unit tests to validate the logic.

**🔧 Function: convert\_date\_format(date\_str)**

* Splits the input string by -.
* Checks if:
  + There are exactly 3 parts.
  + All parts are digits.
  + Year has 4 digits, month and day have 2 digits each.
  + Month is between 1–12 and day is between 1–31.
* If valid, returns the date in DD-MM-YYYY format.
* Otherwise, returns None.

**✅ Unit Tests**

* **test\_valid\_date**: Confirms correct conversion for valid inputs.
* **test\_invalid\_date**: Ensures invalid formats return None.

**🖥️ Main Execution**

* Runs the tests.
* Prompts user for a date.
* Converts and prints the result or shows an error message.