Name: K.Likitha

Hall Ticket: 2303A52144

Batch: 41

Task 1: Email Validation using TDD

Prompt:

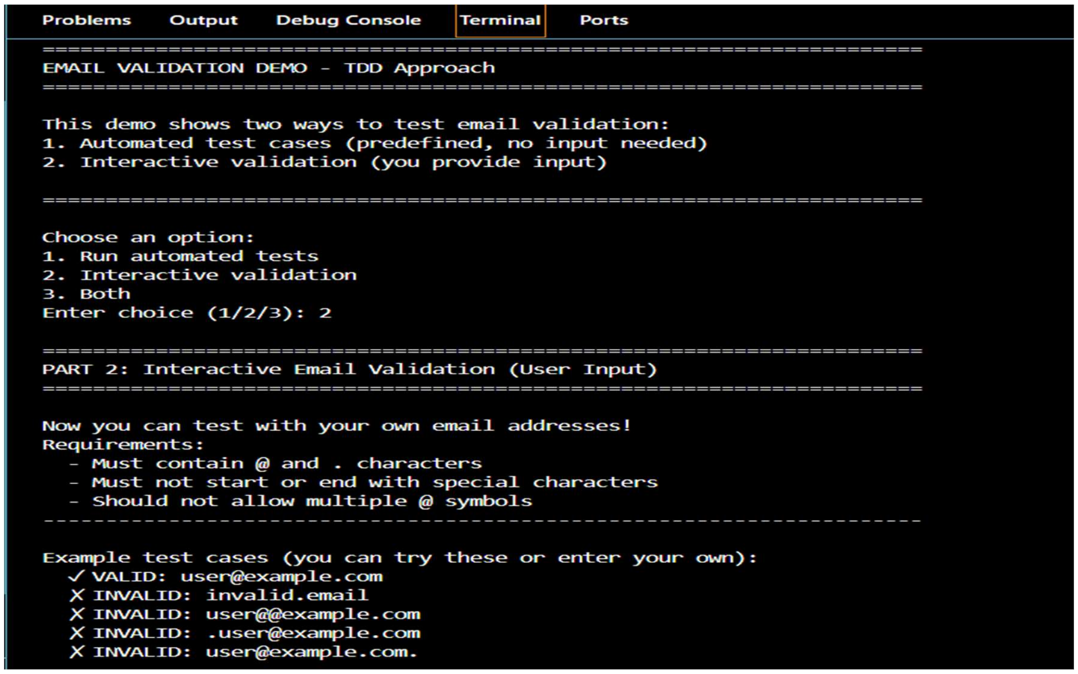
Generate comprehensive unit test cases (valid and invalid) for email validation using TDD based on standard email rules. Then implement a Python function is_valid_email(email) that passes all the generated test cases.

Code:

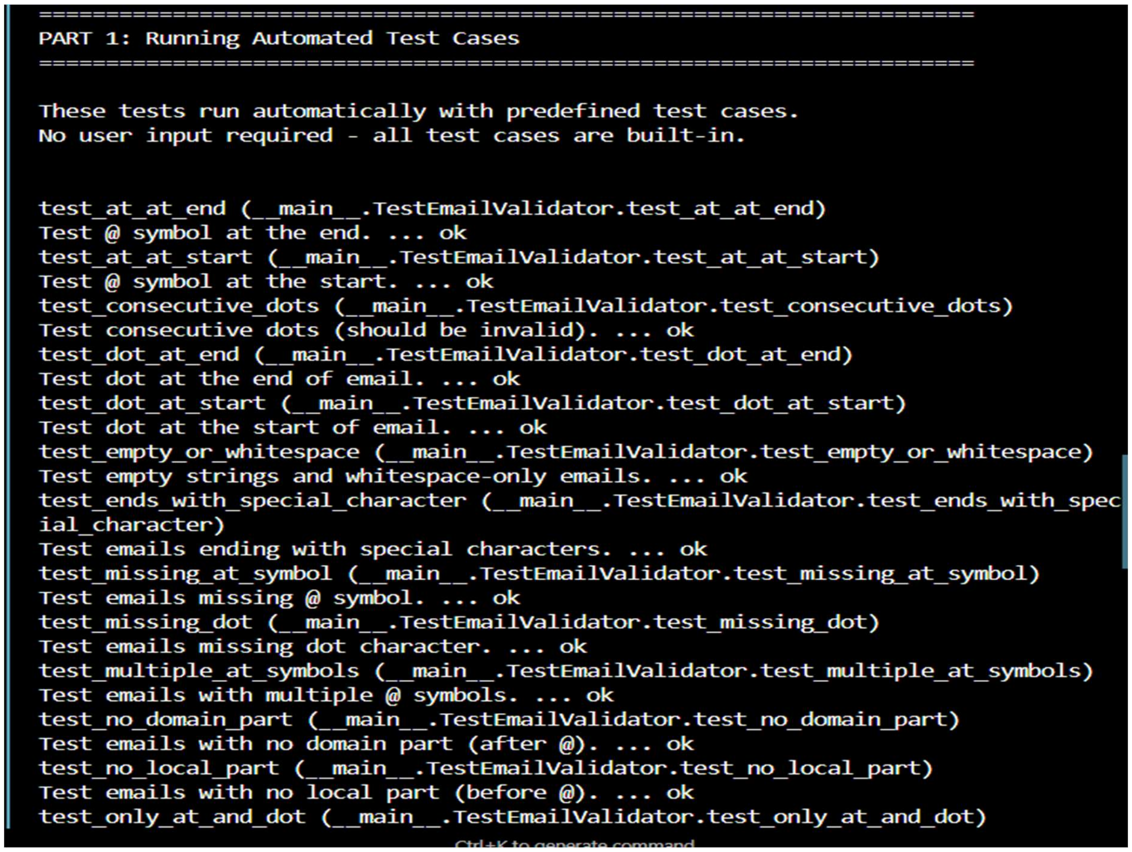
```python
def is_valid_email(email):
    """
    Validate an email address according to specified requirements.

    Requirements:
    - Must contain @ and . characters
    - Must not start or end with special characters
    - Should not allow multiple @ symbols

    Args:
        email (str): Email address to validate

    Returns:
        bool: True if email is valid, False otherwise
    """
    # Check if email is empty or None
    if not email or not isinstance(email, str):
        return False

    # Remove leading/trailing whitespace for validation
    email = email.strip()

    # Check if email is empty after stripping
    if not email:
        return False

    # Check for spaces (not allowed in email addresses)
    if ' ' in email:
        return False

    # Requirement 1: Must contain @ symbol
    if '@' not in email:
        return False

    # Requirement 2: Must contain . (dot) character
    if '.' not in email:
        return False

    # Requirement 3: Should not allow multiple @ symbols
    if email.count('@') != 1:
        return False
    # Split email into local part (before @) and domain part (after @)
    parts = email.split('@')
    if len(parts) != 2:
        return False

    local_part = parts[0]
    domain_part = parts[1]

    # Check if local part or domain part is empty
    if not local_part or not domain_part:
        return False

    # Requirement 4: Must not start with special characters
    # Special characters that shouldn't be at the start: . _ - + @
    if local_part[0] in ['.', '_', '-', '+', '@']:
        return False

    # Requirement 5: Must not end with special characters
    # Check local part doesn't end with special characters
    if local_part[-1] in ['.', '_', '-', '+']:
        return False

    # Check domain part doesn't end with special characters
    if domain_part[-1] in ['.', '_', '-', '+']:
        return False

    # Check for consecutive dots (invalid)
    if '..' in local_part or '..' in domain_part:
        return False

    # Check that domain part contains at least one dot (for TLD)
    if '.' not in domain_part:
        return False

    # Check that domain part doesn't start with dot
    if domain_part[0] == '.':
        return False

    # All requirements passed
    return True


def get_user_email():
    """
    Interactive function to get email input from user and validate it.

    Returns:
        tuple: (email, is_valid) - The email entered and validation result
    """
    email = input("Enter an email address to validate: ").strip()
    is_valid = is_valid_email(email)
    return email, is_valid
```

```python
def main():
    """Main function for interactive email validation."""
    print("=" * 60)
    print("Email Validation System")
    print("=" * 60)
    print("\nRequirements:")
    print("- Must contain @ and . characters")
    print("- Must not start or end with special characters")
    print("- Should not allow multiple @ symbols")
    print("-" * 60)

    while True:
        print()
        email, is_valid = get_user_email()

        if is_valid:
            print(f"✓ '{email}' is a VALID email address.")
        else:
            print(f"✗ '{email}' is an INVALID email address.")

        # Ask if user wants to continue
        print()
        continue_choice = input("Do you want to validate another email? (yes/no): 
").strip().lower()
        if continue_choice not in ['yes', 'y']:
            print("\nThank you for using Email Validation System!")
            break


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

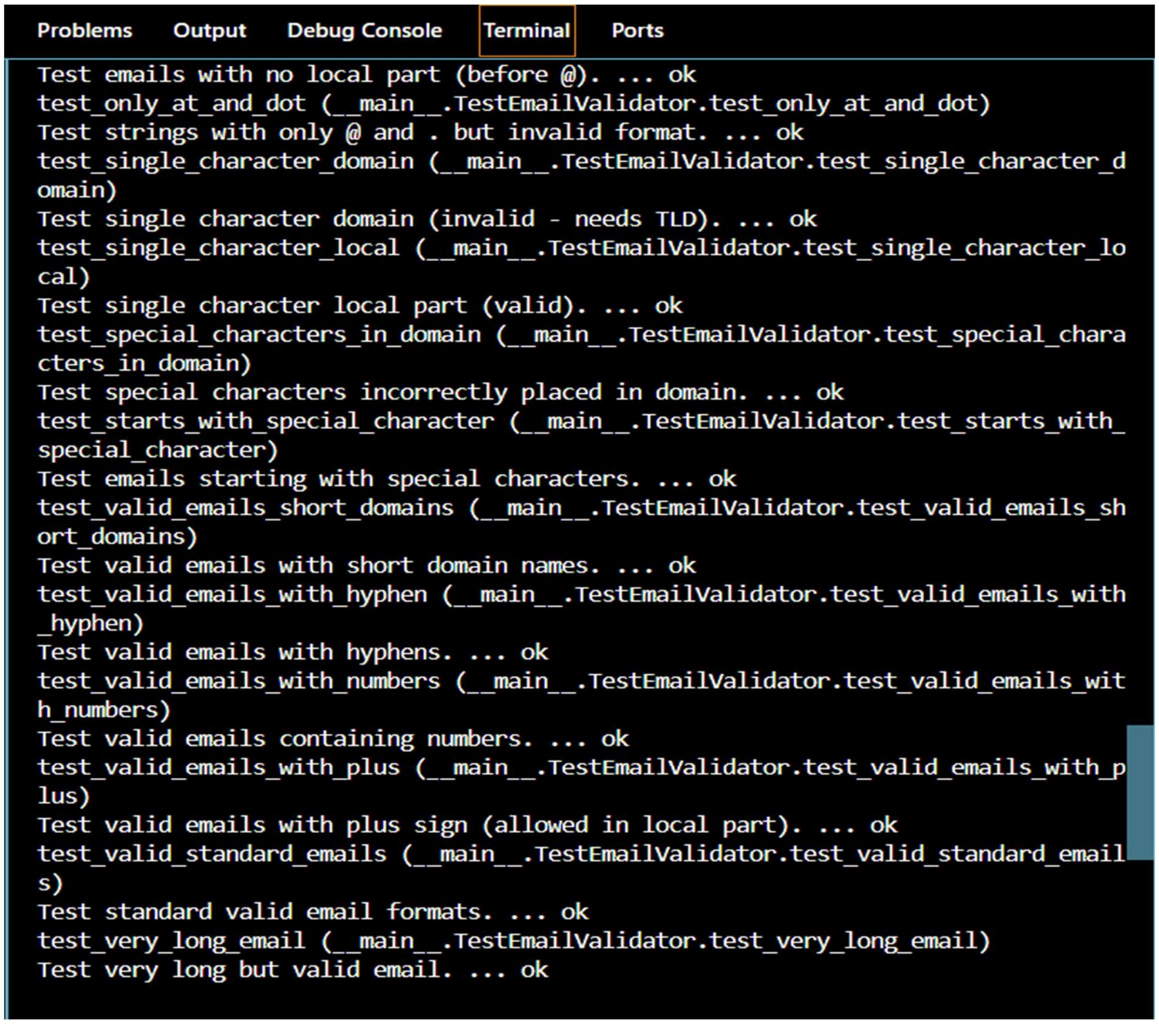
Output:



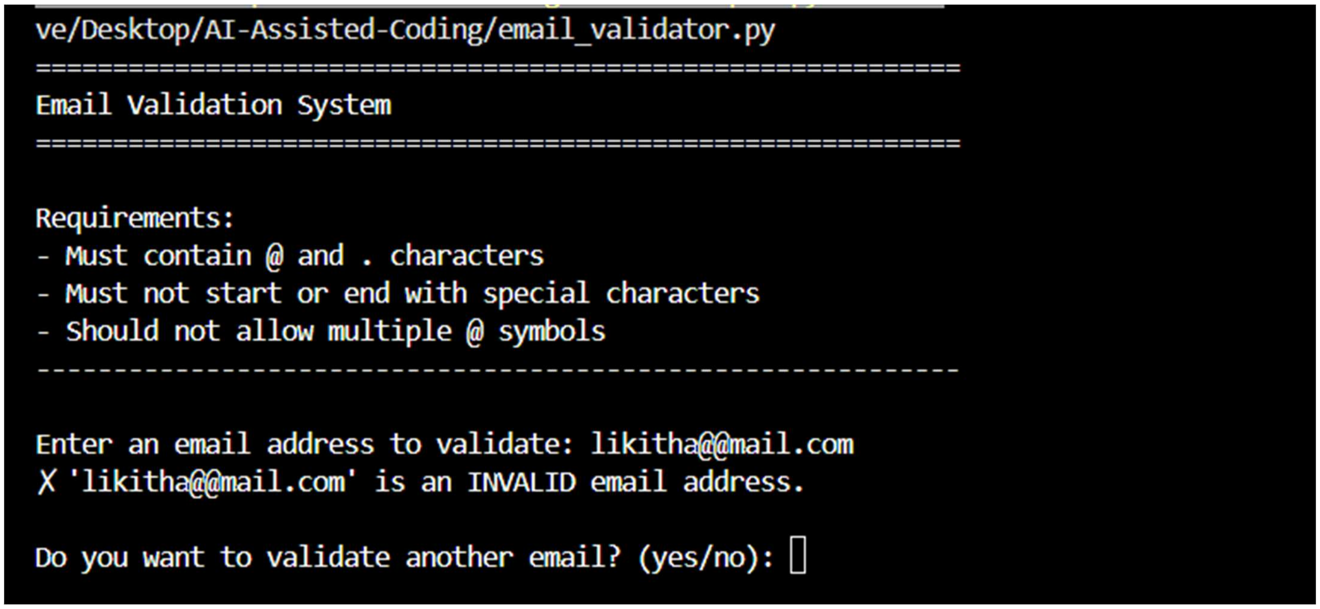
```
======================================================================
EMAIL VALIDATION DEMO - TDD Approach
======================================================================

This demo shows two ways to test email validation:
1. Automated test cases (predefined, no input needed)
2. Interactive validation (you provide input)

======================================================================

Choose an option:
1. Run automated tests
2. Interactive validation
3. Both
Enter choice (1/2/3): 2

======================================================================
PART 2: Interactive Email Validation (User Input)
======================================================================

Now you can test with your own email addresses!
Requirements:
  - Must contain @ and . characters
  - Must not start or end with special characters
  - Should not allow multiple @ symbols
----------------------------------------------------------------------

Example test cases (you can try these or enter your own):
  ✓ VALID: user@example.com
  ✗ INVALID: invalid.email
  ✗ INVALID: user@@example.com
  ✗ INVALID: .user@example.com
  ✗ INVALID: user@example.com.
```



```
======================================================================
PART 1: Running Automated Test Cases
======================================================================

These tests run automatically with predefined test cases.
No user input required - all test cases are built-in.


test_at_at_end (__main__.TestEmailValidator.test_at_at_end)
Test @ symbol at the end. ... ok
test_at_at_start (__main__.TestEmailValidator.test_at_at_start)
Test @ symbol at the start. ... ok
test_consecutive_dots (__main__.TestEmailValidator.test_consecutive_dots)
Test consecutive dots (should be invalid). ... ok
test_dot_at_end (__main__.TestEmailValidator.test_dot_at_end)
Test dot at the end of email. ... ok
test_dot_at_start (__main__.TestEmailValidator.test_dot_at_start)
Test dot at the start of email. ... ok
test_empty_or_whitespace (__main__.TestEmailValidator.test_empty_or_whitespace)
Test empty strings and whitespace-only emails. ... ok
test_ends_with_special_character (__main__.TestEmailValidator.test_ends_with_spec
ial_character)
Test emails ending with special characters. ... ok
test_missing_at_symbol (__main__.TestEmailValidator.test_missing_at_symbol)
Test emails missing @ symbol. ... ok
test_missing_dot (__main__.TestEmailValidator.test_missing_dot)
Test emails missing dot character. ... ok
test_multiple_at_symbols (__main__.TestEmailValidator.test_multiple_at_symbols)
Test emails with multiple @ symbols. ... ok
test_no_domain_part (__main__.TestEmailValidator.test_no_domain_part)
Test emails with no domain part (after @). ... ok
test_no_local_part (__main__.TestEmailValidator.test_no_local_part)
Test emails with no local part (before @). ... ok
test_only_at_and_dot (__main__.TestEmailValidator.test_only_at_and_dot)
```



```
Test emails with no local part (before @). ... ok
test_only_at_and_dot (__main__.TestEmailValidator.test_only_at_and_dot)
Test strings with only @ and . but invalid format. ... ok
test_single_character_domain (__main__.TestEmailValidator.test_single_character_d
omain)
Test single character domain (invalid - needs TLD). ... ok
test_single_character_local (__main__.TestEmailValidator.test_single_character_lo
cal)
Test single character local part (valid). ... ok
test_special_characters_in_domain (__main__.TestEmailValidator.test_special_chara
cters_in_domain)
Test special characters incorrectly placed in domain. ... ok
test_starts_with_special_character (__main__.TestEmailValidator.test_starts_with_
special_character)
Test emails starting with special characters. ... ok
test_valid_emails_short_domains (__main__.TestEmailValidator.test_valid_emails_sh
ort_domains)
Test valid emails with short domain names. ... ok
test_valid_emails_with_hyphen (__main__.TestEmailValidator.test_valid_emails_with
_hyphen)
Test valid emails with hyphens. ... ok
test_valid_emails_with_numbers (__main__.TestEmailValidator.test_valid_emails_wit
h_numbers)
Test valid emails containing numbers. ... ok
test_valid_emails_with_plus (__main__.TestEmailValidator.test_valid_emails_with_p
lus)
Test valid emails with plus sign (allowed in local part). ... ok
test_valid_standard_emails (__main__.TestEmailValidator.test_valid_standard_email
s)
Test standard valid email formats. ... ok
test_very_long_email (__main__.TestEmailValidator.test_very_long_email)
Test very long but valid email. ... ok
```



```
ve/Desktop/AI-Assisted-Coding/email_validator.py
============================================================
Email Validation System
============================================================

Requirements:
- Must contain @ and . characters
- Must not start or end with special characters
- Should not allow multiple @ symbols
------------------------------------------------------------

Enter an email address to validate: likitha@@mail.com
X 'likitha@@mail.com' is an INVALID email address.

Do you want to validate another email? (yes/no): 
```

Explanation:

The code validates an email by checking required symbols, ensuring correct placement of special characters, and preventing multiple @ symbols. It is written to satisfy all test cases first, following the Test-Driven Development (TDD) approach.

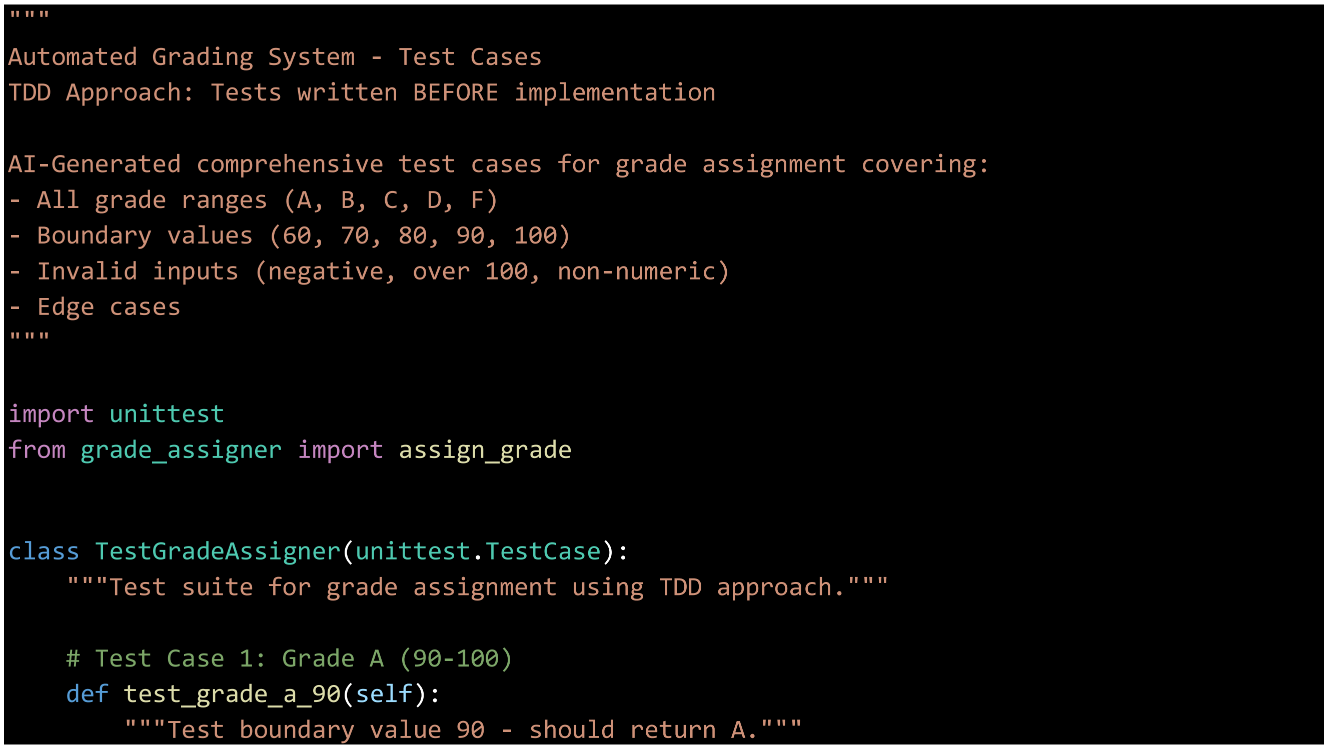
Task 2: Grade Assignment using Loops

Python Code for Bubble Sort:

Prompt:

Generate unit test cases using TDD for assign_grade(score) where 90–100 → A, 80–89 → B, 70–79 → C, 60–69 → D, below 60 → F, including boundary and invalid inputs. Then implement the Python function assign_grade(score) to pass all generated test cases.

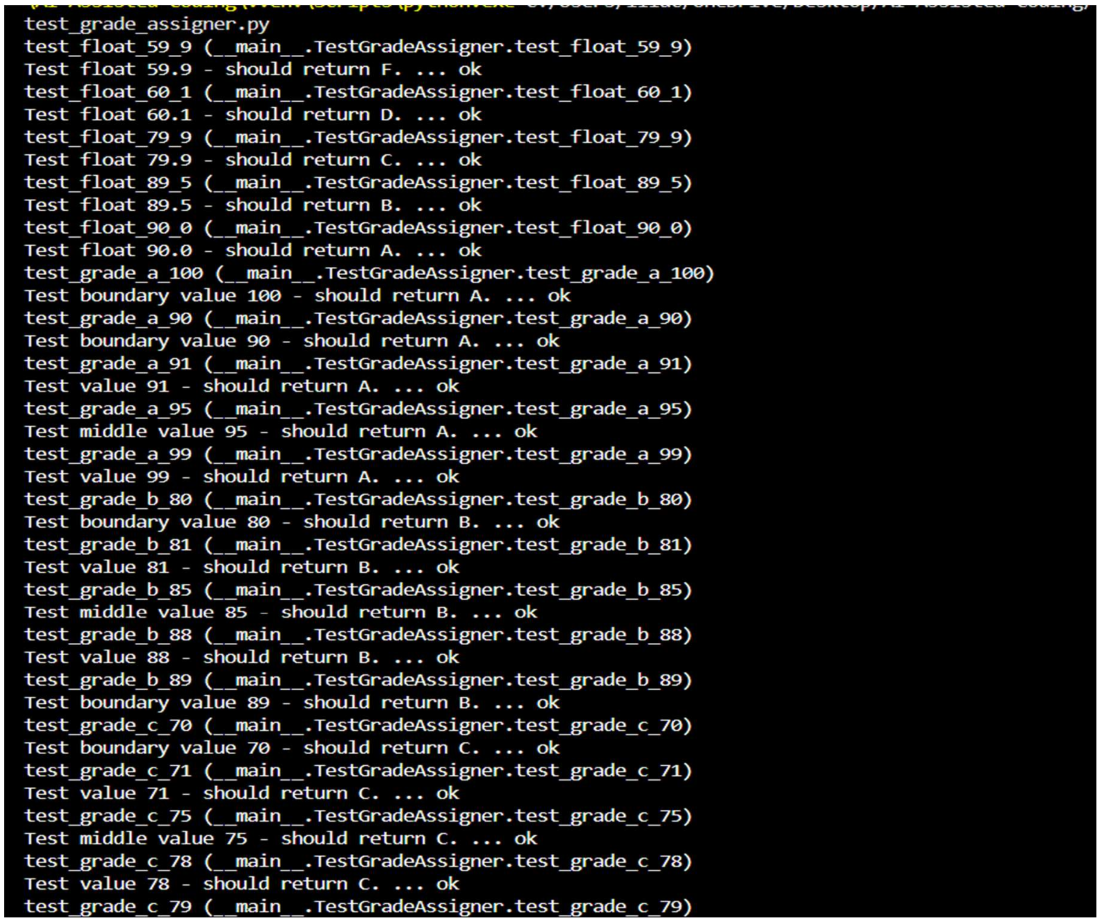
Code:



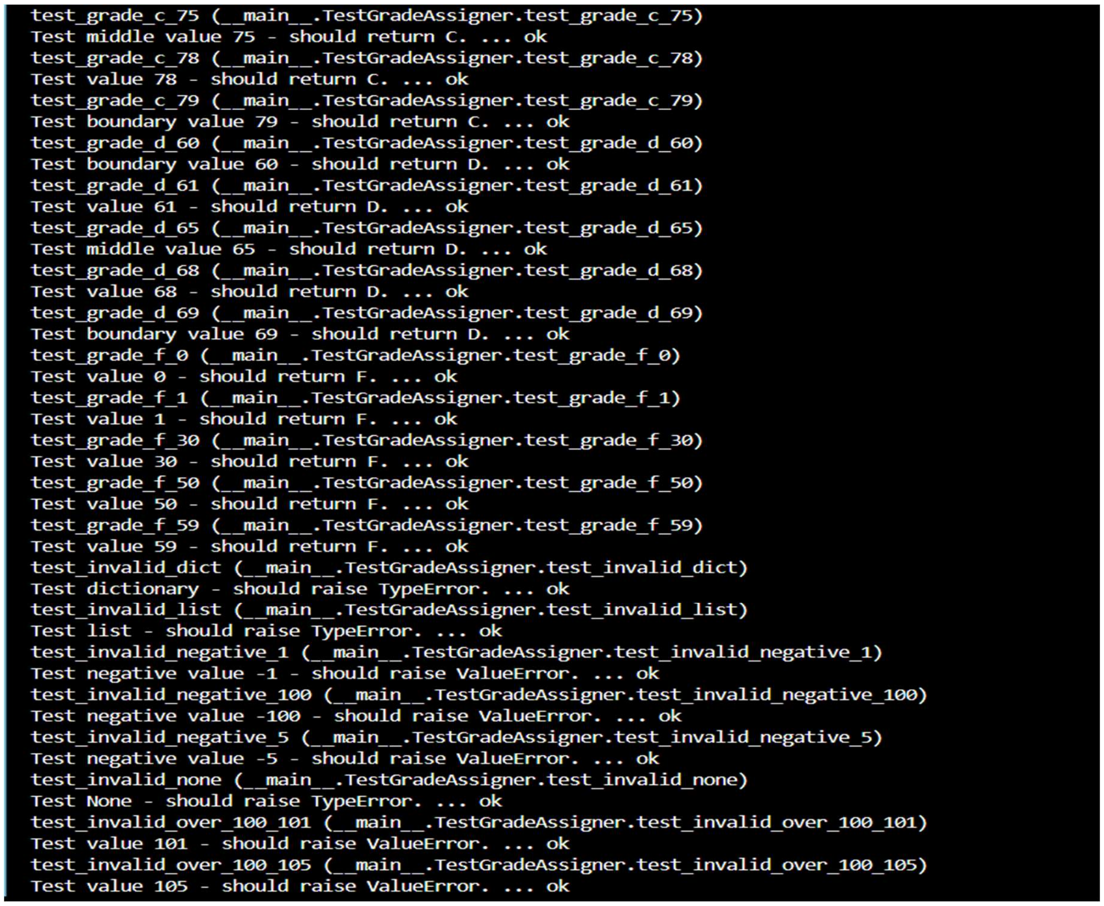
```python
"""
Automated Grading System - Test Cases
TDD Approach: Tests written BEFORE implementation

AI-Generated comprehensive test cases for grade assignment covering:
- All grade ranges (A, B, C, D, F)
- Boundary values (60, 70, 80, 90, 100)
- Invalid inputs (negative, over 100, non-numeric)
- Edge cases
"""

import unittest
from grade_assigner import assign_grade


class TestGradeAssigner(unittest.TestCase):
    """Test suite for grade assignment using TDD approach."""

    # Test Case 1: Grade A (90-100)
    def test_grade_a_90(self):
        """Test boundary value 90 - should return A."""
        self.assertEqual(assign_grade(90), 'A')

    def test_grade_a_95(self):
        """Test middle value 95 - should return A."""
        self.assertEqual(assign_grade(95), 'A')

    def test_grade_a_100(self):
        """Test boundary value 100 - should return A."""
        self.assertEqual(assign_grade(100), 'A')

    def test_grade_a_91(self):
        """Test value 91 - should return A."""
        self.assertEqual(assign_grade(91), 'A')

    def test_grade_a_99(self):
        """Test value 99 - should return A."""
        self.assertEqual(assign_grade(99), 'A')

    # Test Case 2: Grade B (80-89)
    def test_grade_b_80(self):
        """Test boundary value 80 - should return B."""
        self.assertEqual(assign_grade(80), 'B')

    def test_grade_b_85(self):
        """Test middle value 85 - should return B."""
        self.assertEqual(assign_grade(85), 'B')

    def test_grade_b_89(self):
        """Test boundary value 89 - should return B."""
        self.assertEqual(assign_grade(89), 'B')

    def test_grade_b_81(self):
        """Test value 81 - should return B."""
        self.assertEqual(assign_grade(81), 'B')

    def test_grade_b_88(self):
        """Test value 88 - should return B."""
        self.assertEqual(assign_grade(88), 'B')

    # Test Case 3: Grade C (70-79)
    def test_grade_c_70(self):
        """Test boundary value 70 - should return C."""
        self.assertEqual(assign_grade(70), 'C')

    def test_grade_c_75(self):
        """Test middle value 75 - should return C."""
        self.assertEqual(assign_grade(75), 'C')

    def test_grade_c_79(self):
        """Test boundary value 79 - should return C."""
        self.assertEqual(assign_grade(79), 'C')

    def test_grade_c_71(self):
        """Test value 71 - should return C."""
        self.assertEqual(assign_grade(71), 'C')

    def test_grade_c_78(self):
        """Test value 78 - should return C."""
        self.assertEqual(assign_grade(78), 'C')

    # Test Case 4: Grade D (60-69)
    def test_grade_d_60(self):
        """Test boundary value 60 - should return D."""
        self.assertEqual(assign_grade(60), 'D')

    def test_grade_d_65(self):
        """Test middle value 65 - should return D."""
        self.assertEqual(assign_grade(65), 'D')

    def test_grade_d_69(self):
        """Test boundary value 69 - should return D."""
        self.assertEqual(assign_grade(69), 'D')

    def test_grade_d_61(self):
        """Test value 61 - should return D."""
        self.assertEqual(assign_grade(61), 'D')

    def test_grade_d_68(self):
        """Test value 68 - should return D."""
        self.assertEqual(assign_grade(68), 'D')

    # Test Case 5: Grade F (Below 60)
    def test_grade_f_59(self):
        """Test value 59 - should return F."""
        self.assertEqual(assign_grade(59), 'F')

    def test_grade_f_50(self):
        """Test value 50 - should return F."""
        self.assertEqual(assign_grade(50), 'F')

    def test_grade_f_0(self):
        """Test value 0 - should return F."""
        self.assertEqual(assign_grade(0), 'F')

    def test_grade_f_30(self):
        """Test value 30 - should return F."""
        self.assertEqual(assign_grade(30), 'F')

    def test_grade_f_1(self):
        """Test value 1 - should return F."""
        self.assertEqual(assign_grade(1), 'F')

    # Test Case 6: Invalid Inputs - Negative Numbers
    def test_invalid_negative_5(self):
        """Test negative value -5 - should raise ValueError."""
        with self.assertRaises(ValueError):
            assign_grade(-5)
```

```python
    def test_invalid_negative_1(self):
        """Test negative value -1 - should raise ValueError."""
        with self.assertRaises(ValueError):
            assign_grade(-1)

    def test_invalid_negative_100(self):
        """Test negative value -100 - should raise ValueError."""
        with self.assertRaises(ValueError):
            assign_grade(-100)

    # Test Case 7: Invalid Inputs - Over 100
    def test_invalid_over_100_105(self):
        """Test value 105 - should raise ValueError."""
        with self.assertRaises(ValueError):
            assign_grade(105)

    def test_invalid_over_100_101(self):
        """Test value 101 - should raise ValueError."""
        with self.assertRaises(ValueError):
            assign_grade(101)

    def test_invalid_over_100_200(self):
        """Test value 200 - should raise ValueError."""
        with self.assertRaises(ValueError):
            assign_grade(200)

    # Test Case 8: Invalid Inputs - Non-Numeric Strings
    def test_invalid_string_eighty(self):
        """Test string 'eighty' - should raise TypeError."""
        with self.assertRaises(TypeError):
            assign_grade("eighty")

    def test_invalid_string_abc(self):
        """Test string 'abc' - should raise TypeError."""
        with self.assertRaises(TypeError):
            assign_grade("abc")

    def test_invalid_string_empty(self):
        """Test empty string - should raise TypeError."""
        with self.assertRaises(TypeError):
            assign_grade("")

    def test_invalid_string_ninety(self):
        """Test string 'ninety' - should raise TypeError."""
        with self.assertRaises(TypeError):
            assign_grade("ninety")

    # Test Case 9: Invalid Inputs - Other Types
    def test_invalid_none(self):
        """Test None - should raise TypeError."""
        with self.assertRaises(TypeError):
            assign_grade(None)

    def test_invalid_list(self):
        """Test list - should raise TypeError."""
        with self.assertRaises(TypeError):
            assign_grade([90])

    def test_invalid_dict(self):
        """Test dictionary - should raise TypeError."""
        with self.assertRaises(TypeError):
            assign_grade({"score": 90})

    # Test Case 10: Edge Cases - Float Values
    def test_float_89_5(self):
        """Test float 89.5 - should return B."""
        self.assertEqual(assign_grade(89.5), 'B')

    def test_float_79_9(self):
        """Test float 79.9 - should return C."""
        self.assertEqual(assign_grade(79.9), 'C')

    def test_float_90_0(self):
        """Test float 90.0 - should return A."""
        self.assertEqual(assign_grade(90.0), 'A')

    def test_float_59_9(self):
        """Test float 59.9 - should return F."""
        self.assertEqual(assign_grade(59.9), 'F')

    def test_float_60_1(self):
        """Test float 60.1 - should return D."""
        self.assertEqual(assign_grade(60.1), 'D')


if __name__ == '__main__':
    # Run the tests with detailed output
    unittest.main(verbosity=2)
```

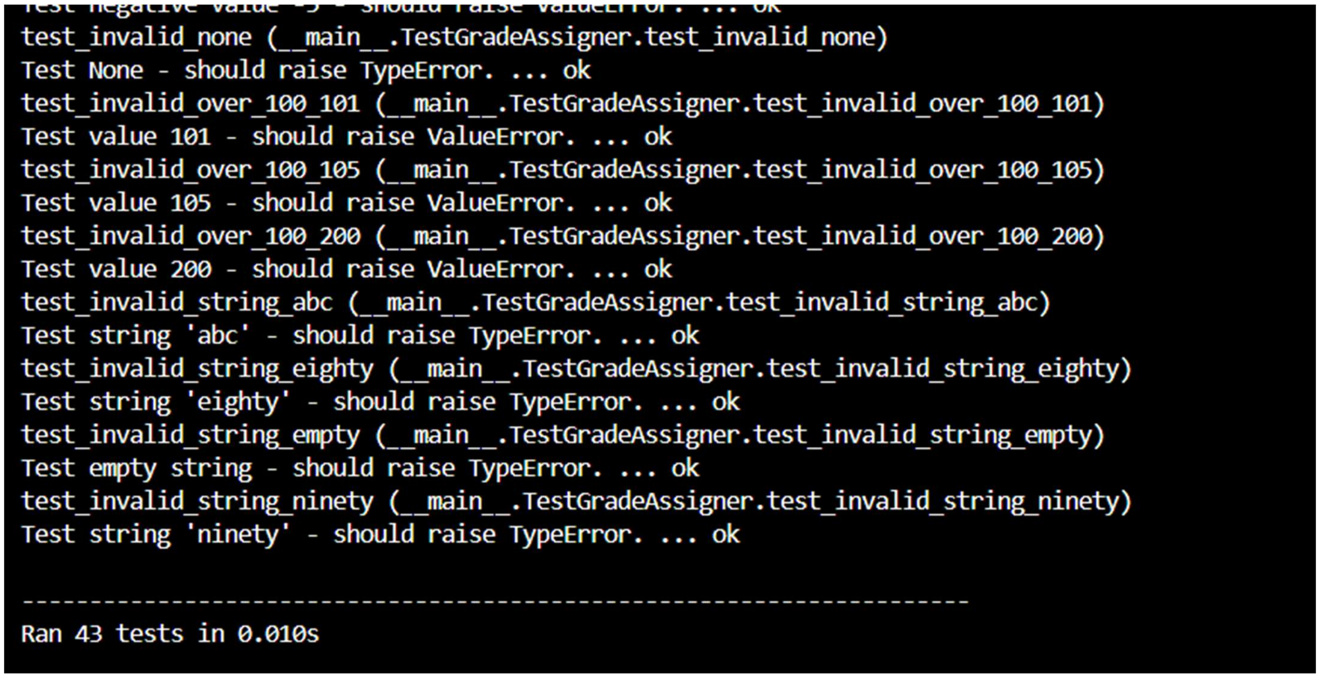
Output:



```
test_grade_assigner.py
test_float_59_9 (__main__.TestGradeAssigner.test_float_59_9)
Test float 59.9 - should return F. ... ok
test_float_60_1 (__main__.TestGradeAssigner.test_float_60_1)
Test float 60.1 - should return D. ... ok
test_float_79_9 (__main__.TestGradeAssigner.test_float_79_9)
Test float 79.9 - should return C. ... ok
test_float_89_5 (__main__.TestGradeAssigner.test_float_89_5)
Test float 89.5 - should return B. ... ok
test_float_90_0 (__main__.TestGradeAssigner.test_float_90_0)
Test float 90.0 - should return A. ... ok
test_grade_a_100 (__main__.TestGradeAssigner.test_grade_a_100)
Test boundary value 100 - should return A. ... ok
test_grade_a_90 (__main__.TestGradeAssigner.test_grade_a_90)
Test boundary value 90 - should return A. ... ok
test_grade_a_91 (__main__.TestGradeAssigner.test_grade_a_91)
Test value 91 - should return A. ... ok
test_grade_a_95 (__main__.TestGradeAssigner.test_grade_a_95)
Test middle value 95 - should return A. ... ok
test_grade_a_99 (__main__.TestGradeAssigner.test_grade_a_99)
Test value 99 - should return A. ... ok
test_grade_b_80 (__main__.TestGradeAssigner.test_grade_b_80)
Test boundary value 80 - should return B. ... ok
test_grade_b_81 (__main__.TestGradeAssigner.test_grade_b_81)
Test value 81 - should return B. ... ok
test_grade_b_85 (__main__.TestGradeAssigner.test_grade_b_85)
Test middle value 85 - should return B. ... ok
test_grade_b_88 (__main__.TestGradeAssigner.test_grade_b_88)
Test value 88 - should return B. ... ok
test_grade_b_89 (__main__.TestGradeAssigner.test_grade_b_89)
Test boundary value 89 - should return B. ... ok
test_grade_c_70 (__main__.TestGradeAssigner.test_grade_c_70)
Test boundary value 70 - should return C. ... ok
test_grade_c_71 (__main__.TestGradeAssigner.test_grade_c_71)
Test value 71 - should return C. ... ok
test_grade_c_75 (__main__.TestGradeAssigner.test_grade_c_75)
Test middle value 75 - should return C. ... ok
test_grade_c_78 (__main__.TestGradeAssigner.test_grade_c_78)
Test value 78 - should return C. ... ok
test_grade_c_79 (__main__.TestGradeAssigner.test_grade_c_79)
```



```
test_grade_c_75 (__main__.TestGradeAssigner.test_grade_c_75)
Test middle value 75 - should return C. ... ok
test_grade_c_78 (__main__.TestGradeAssigner.test_grade_c_78)
Test value 78 - should return C. ... ok
test_grade_c_79 (__main__.TestGradeAssigner.test_grade_c_79)
Test boundary value 79 - should return C. ... ok
test_grade_d_60 (__main__.TestGradeAssigner.test_grade_d_60)
Test boundary value 60 - should return D. ... ok
test_grade_d_61 (__main__.TestGradeAssigner.test_grade_d_61)
Test value 61 - should return D. ... ok
test_grade_d_65 (__main__.TestGradeAssigner.test_grade_d_65)
Test middle value 65 - should return D. ... ok
test_grade_d_68 (__main__.TestGradeAssigner.test_grade_d_68)
Test value 68 - should return D. ... ok
test_grade_d_69 (__main__.TestGradeAssigner.test_grade_d_69)
Test boundary value 69 - should return D. ... ok
test_grade_f_0 (__main__.TestGradeAssigner.test_grade_f_0)
Test value 0 - should return F. ... ok
test_grade_f_1 (__main__.TestGradeAssigner.test_grade_f_1)
Test value 1 - should return F. ... ok
test_grade_f_30 (__main__.TestGradeAssigner.test_grade_f_30)
Test value 30 - should return F. ... ok
test_grade_f_50 (__main__.TestGradeAssigner.test_grade_f_50)
Test value 50 - should return F. ... ok
test_grade_f_59 (__main__.TestGradeAssigner.test_grade_f_59)
Test value 59 - should return F. ... ok
test_invalid_dict (__main__.TestGradeAssigner.test_invalid_dict)
Test dictionary - should raise TypeError. ... ok
test_invalid_list (__main__.TestGradeAssigner.test_invalid_list)
Test list - should raise TypeError. ... ok
test_invalid_negative_1 (__main__.TestGradeAssigner.test_invalid_negative_1)
Test negative value -1 - should raise ValueError. ... ok
test_invalid_negative_100 (__main__.TestGradeAssigner.test_invalid_negative_100)
Test negative value -100 - should raise ValueError. ... ok
test_invalid_negative_5 (__main__.TestGradeAssigner.test_invalid_negative_5)
Test negative value -5 - should raise ValueError. ... ok
test_invalid_none (__main__.TestGradeAssigner.test_invalid_none)
Test None - should raise TypeError. ... ok
test_invalid_over_100_101 (__main__.TestGradeAssigner.test_invalid_over_100_101)
Test value 101 - should raise ValueError. ... ok
test_invalid_over_100_105 (__main__.TestGradeAssigner.test_invalid_over_100_105)
Test value 105 - should raise ValueError. ... ok
```



```
Test negative value -5 - should raise ValueError. ... ok
test_invalid_none (__main__.TestGradeAssigner.test_invalid_none)
Test None - should raise TypeError. ... ok
test_invalid_over_100_101 (__main__.TestGradeAssigner.test_invalid_over_100_101)
Test value 101 - should raise ValueError. ... ok
test_invalid_over_100_105 (__main__.TestGradeAssigner.test_invalid_over_100_105)
Test value 105 - should raise ValueError. ... ok
test_invalid_over_100_200 (__main__.TestGradeAssigner.test_invalid_over_100_200)
Test value 200 - should raise ValueError. ... ok
test_invalid_string_abc (__main__.TestGradeAssigner.test_invalid_string_abc)
Test string 'abc' - should raise TypeError. ... ok
test_invalid_string_eighty (__main__.TestGradeAssigner.test_invalid_string_eighty)
Test string 'eighty' - should raise TypeError. ... ok
test_invalid_string_empty (__main__.TestGradeAssigner.test_invalid_string_empty)
Test empty string - should raise TypeError. ... ok
test_invalid_string_ninety (__main__.TestGradeAssigner.test_invalid_string_ninety)
Test string 'ninety' - should raise TypeError. ... ok

----------------------------------------------------------------------
Ran 43 tests in 0.010s
```

Explanation:

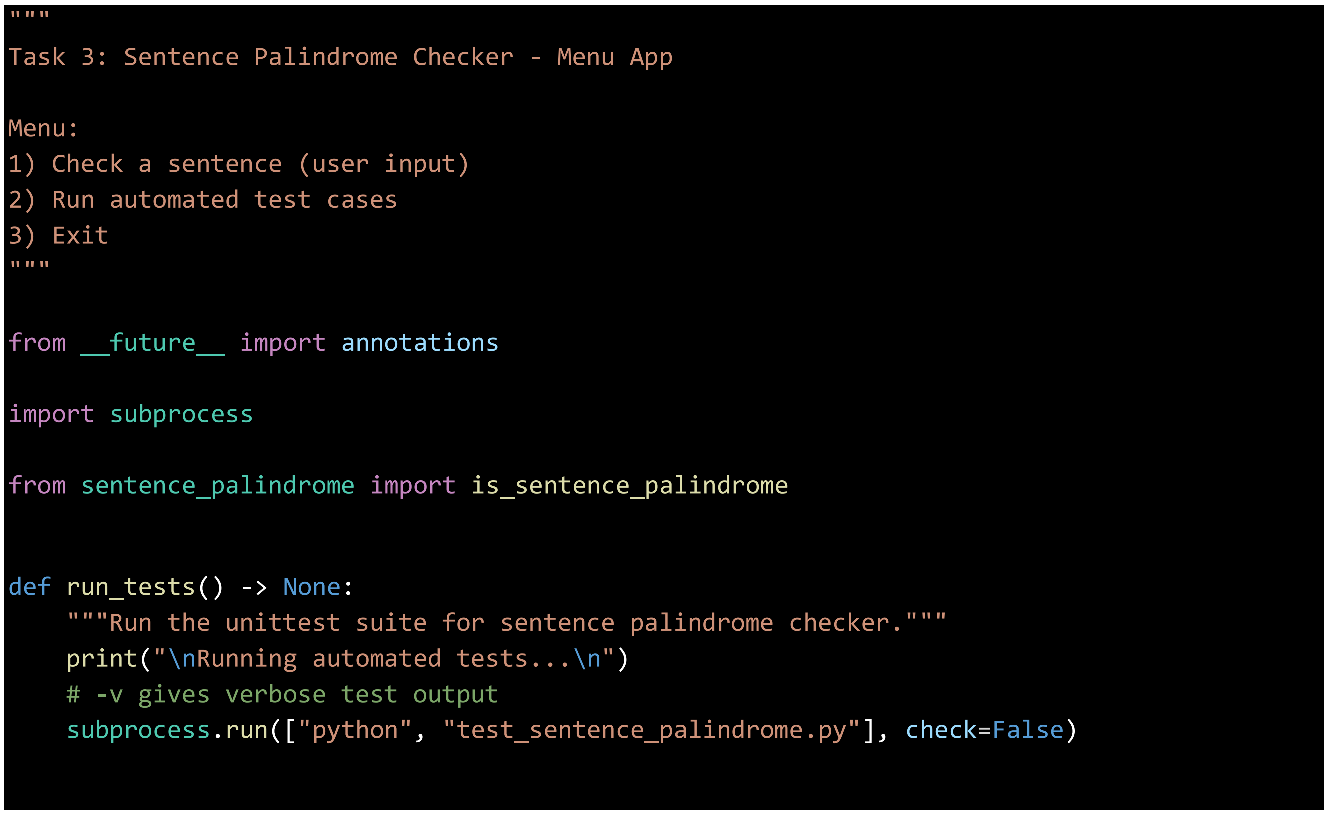
The function assigns letter grades based on score ranges while correctly handling boundary values.
Invalid inputs are detected and handled to ensure reliable and safe grading.

Task 3: Sentence Palindrome Checker

Prompt:

Generate comprehensive unit test cases for is_sentence_palindrome(sentence) that ignore case, spaces, and punctuation, covering both palindromic and non-palindromic sentences. Then implement the Python function is_sentence_palindrome(sentence) so it passes all the generated test cases.

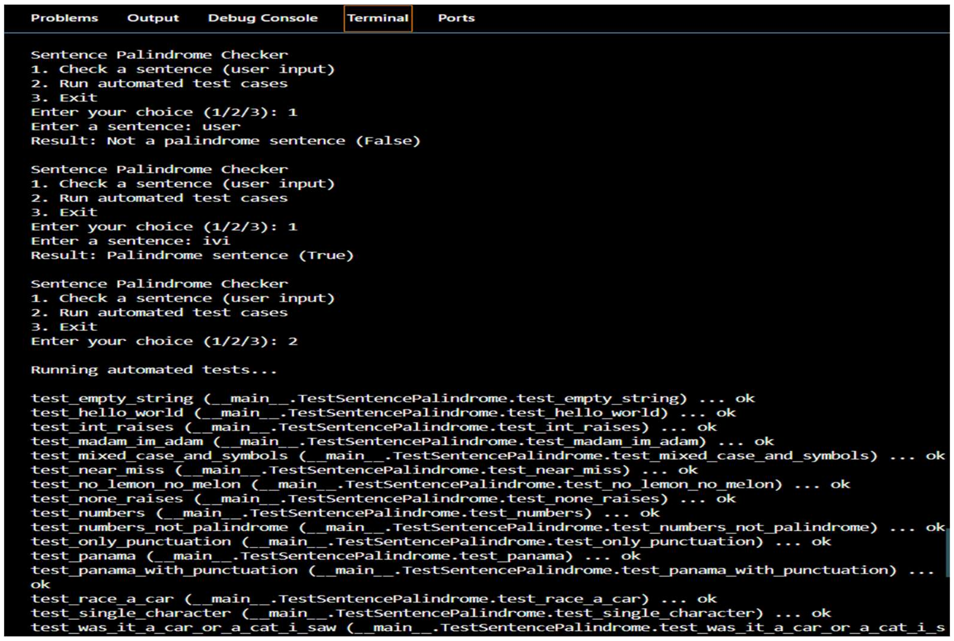
Code:



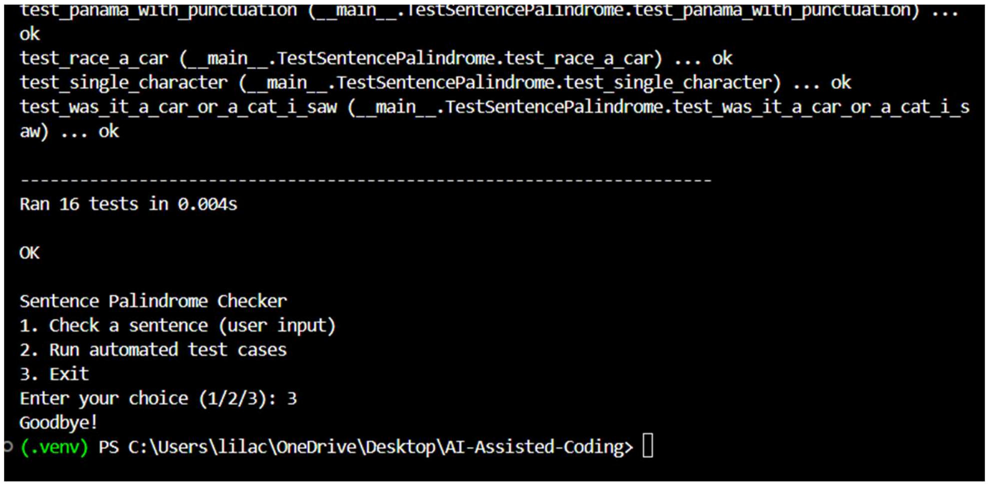
```python
"""
Task 3: Sentence Palindrome Checker - Menu App

Menu:
1) Check a sentence (user input)
2) Run automated test cases
3) Exit
"""

from __future__ import annotations

import subprocess

from sentence_palindrome import is_sentence_palindrome


def run_tests() -> None:
    """Run the unittest suite for sentence palindrome checker."""
    print("\nRunning automated tests...\n")
    # -v gives verbose test output
    subprocess.run(["python", "test_sentence_palindrome.py"], check=False)
```

```python
def check_user_sentence() -> None:
    """Prompt the user for a sentence and print palindrome result."""
    sentence = input("Enter a sentence: ")
    result = is_sentence_palindrome(sentence)
    if result:
        print("Result: Palindrome sentence (True)")
    else:
        print("Result: Not a palindrome sentence (False)")


def main() -> None:
    while True:
        print("\nSentence Palindrome Checker")
        print("1. Check a sentence (user input)")
        print("2. Run automated test cases")
        print("3. Exit")

        choice = input("Enter your choice (1/2/3): ").strip()

        if choice == "1":
            check_user_sentence()
        elif choice == "2":
            run_tests()
        elif choice == "3":
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Please enter 1, 2, or 3.")


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

Output:



```
Sentence Palindrome Checker
1. Check a sentence (user input)
2. Run automated test cases
3. Exit
Enter your choice (1/2/3): 1
Enter a sentence: user
Result: Not a palindrome sentence (False)

Sentence Palindrome Checker
1. Check a sentence (user input)
2. Run automated test cases
3. Exit
Enter your choice (1/2/3): 1
Enter a sentence: ivi
Result: Palindrome sentence (True)

Sentence Palindrome Checker
1. Check a sentence (user input)
2. Run automated test cases
3. Exit
Enter your choice (1/2/3): 2

Running automated tests...

test_empty_string (__main__.TestSentencePalindrome.test_empty_string) ... ok
test_hello_world (__main__.TestSentencePalindrome.test_hello_world) ... ok
test_int_raises (__main__.TestSentencePalindrome.test_int_raises) ... ok
test_madam_im_adam (__main__.TestSentencePalindrome.test_madam_im_adam) ... ok
test_mixed_case_and_symbols (__main__.TestSentencePalindrome.test_mixed_case_and_symbols) ... ok
test_near_miss (__main__.TestSentencePalindrome.test_near_miss) ... ok
test_no_lemon_no_melon (__main__.TestSentencePalindrome.test_no_lemon_no_melon) ... ok
test_none_raises (__main__.TestSentencePalindrome.test_none_raises) ... ok
test_numbers (__main__.TestSentencePalindrome.test_numbers) ... ok
test_numbers_not_palindrome (__main__.TestSentencePalindrome.test_numbers_not_palindrome) ... ok
test_only_punctuation (__main__.TestSentencePalindrome.test_only_punctuation) ... ok
test_panama (__main__.TestSentencePalindrome.test_panama) ... ok
test_panama_with_punctuation (__main__.TestSentencePalindrome.test_panama_with_punctuation) ...
ok
test_race_a_car (__main__.TestSentencePalindrome.test_race_a_car) ... ok
test_single_character (__main__.TestSentencePalindrome.test_single_character) ... ok
test_was_it_a_car_or_a_cat_i_saw (__main__.TestSentencePalindrome.test_was_it_a_car_or_a_cat_i_s
```



```
test_panama_with_punctuation (__main__.TestSentencePalindrome.test_panama_with_punctuation) ...
ok
test_race_a_car (__main__.TestSentencePalindrome.test_race_a_car) ... ok
test_single_character (__main__.TestSentencePalindrome.test_single_character) ... ok
test_was_it_a_car_or_a_cat_i_saw (__main__.TestSentencePalindrome.test_was_it_a_car_or_a_cat_i_s
aw) ... ok

----------------------------------------------------------------------
Ran 16 tests in 0.004s

OK

Sentence Palindrome Checker
1. Check a sentence (user input)
2. Run automated test cases
3. Exit
Enter your choice (1/2/3): 3
Goodbye!
(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding>
```

Explanation:

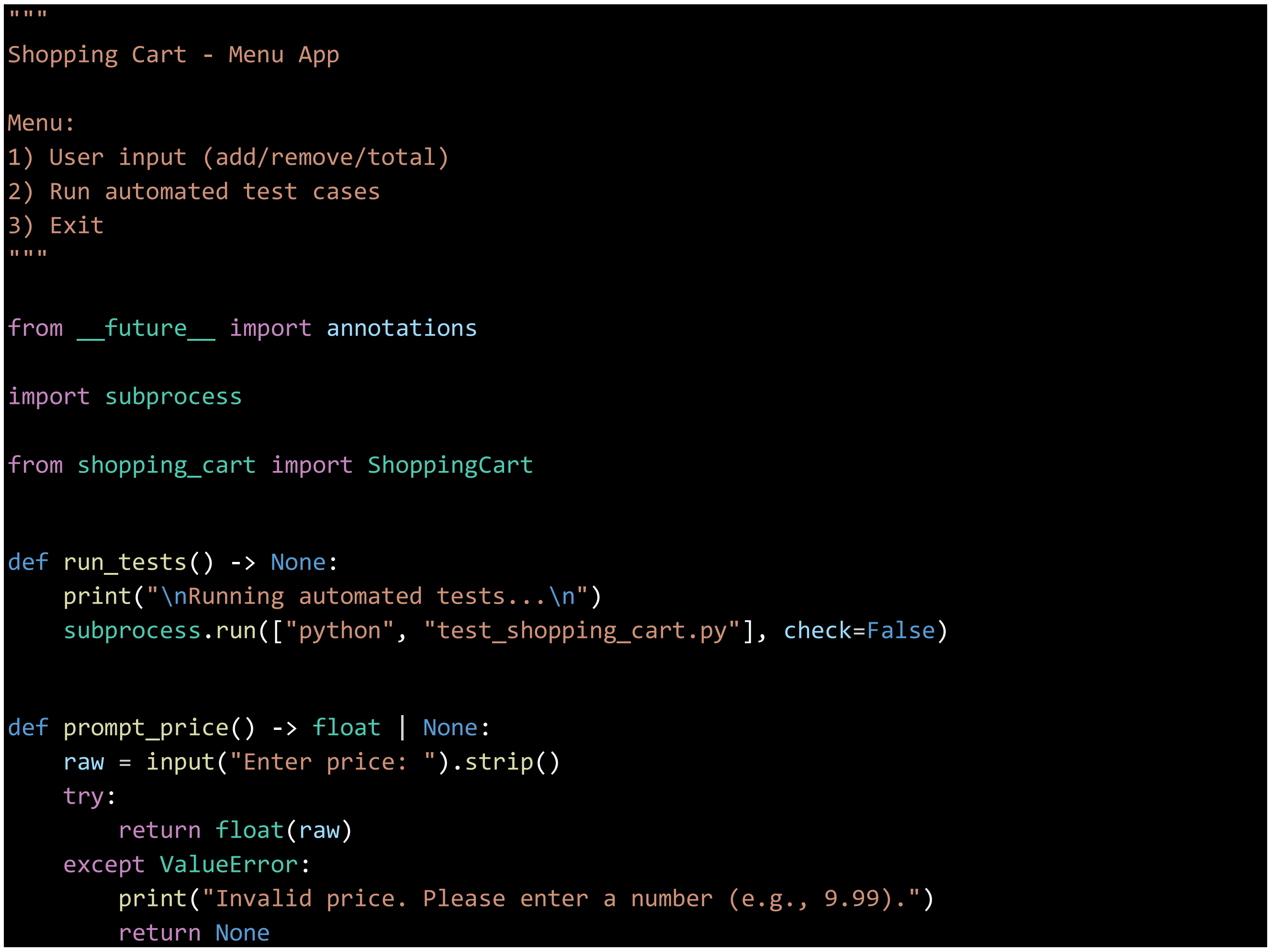
The function checks whether a sentence reads the same forward and backward after removing spaces, punctuation, and case differences.

Task 4: ShoppingCart Class

Prompt:

Generate TDD-based unit test cases for a ShoppingCart class with add_item, remove_item, and total_cost methods, covering correct operations, cost calculation, and empty cart scenarios, then implement the class to pass all tests.

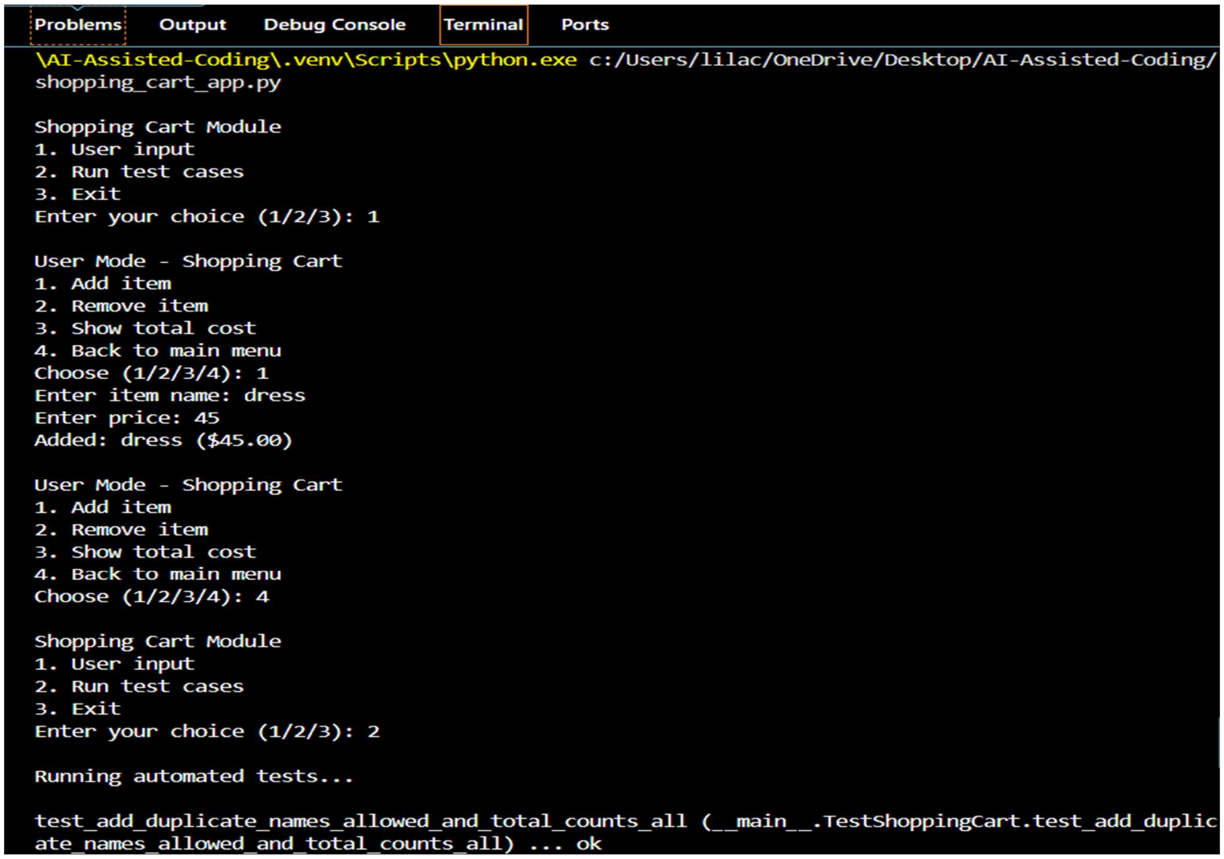
Code:



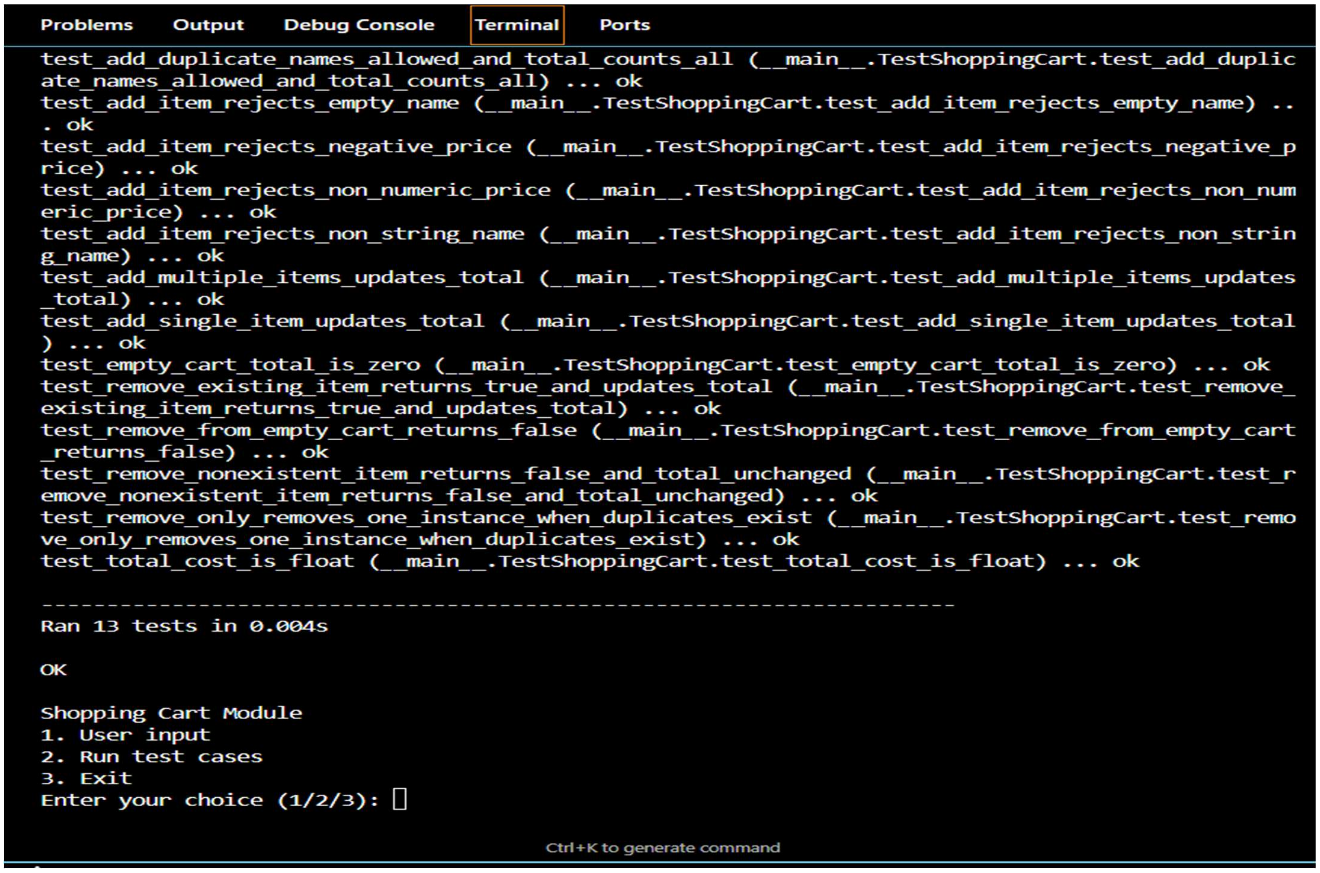
```python
"""
Shopping Cart - Menu App

Menu:
1) User input (add/remove/total)
2) Run automated test cases
3) Exit
"""

from __future__ import annotations

import subprocess

from shopping_cart import ShoppingCart


def run_tests() -> None:
    print("\nRunning automated tests...\n")
    subprocess.run(["python", "test_shopping_cart.py"], check=False)


def prompt_price() -> float | None:
    raw = input("Enter price: ").strip()
    try:
        return float(raw)
    except ValueError:
        print("Invalid price. Please enter a number (e.g., 9.99).")
        return None
```

```python
def user_mode() -> None:
    cart = ShoppingCart()
    while True:
        print("\nUser Mode - Shopping Cart")
        print("1. Add item")
        print("2. Remove item")
        print("3. Show total cost")
        print("4. Back to main menu")

        choice = input("Choose (1/2/3/4): ").strip()

        if choice == "1":
            name = input("Enter item name: ").strip()
            price = prompt_price()
            if price is None:
                continue
            try:
                cart.add_item(name, price)
                print(f"Added: {name} (${price:.2f})")
            except (TypeError, ValueError) as e:
                print(f"Error: {e}")

        elif choice == "2":
            name = input("Enter item name to remove: ").strip()
            try:
                removed = cart.remove_item(name)
                if removed:
                    print(f"Removed one '{name}'.")
                else:
                    print(f"Item '{name}' not found.")
            except TypeError as e:
                print(f"Error: {e}")

        elif choice == "3":
            print(f"Total cost: ${cart.total_cost():.2f}")

        elif choice == "4":
            break
        else:
            print("Invalid choice.")


def main() -> None:
    while True:
        print("\nShopping Cart Module")
        print("1. User input")
        print("2. Run test cases")
        print("3. Exit")

        choice = input("Enter your choice (1/2/3): ").strip()

        if choice == "1":
            user_mode()
        elif choice == "2":
            run_tests()
        elif choice == "3":
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Please enter 1, 2, or 3.")


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

Output:



```
\AI-Assisted-Coding\.venv\Scripts\python.exe c:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/
shopping_cart_app.py

Shopping Cart Module
1. User input
2. Run test cases
3. Exit
Enter your choice (1/2/3): 1

User Mode - Shopping Cart
1. Add item
2. Remove item
3. Show total cost
4. Back to main menu
Choose (1/2/3/4): 1
Enter item name: dress
Enter price: 45
Added: dress ($45.00)

User Mode - Shopping Cart
1. Add item
2. Remove item
3. Show total cost
4. Back to main menu
Choose (1/2/3/4): 4

Shopping Cart Module
1. User input
2. Run test cases
3. Exit
Enter your choice (1/2/3): 2

Running automated tests...

test_add_duplicate_names_allowed_and_total_counts_all (__main__.TestShoppingCart.test_add_duplic
ate_names_allowed_and_total_counts_all) ... ok
```



```
test_add_duplicate_names_allowed_and_total_counts_all (__main__.TestShoppingCart.test_add_duplic
ate_names_allowed_and_total_counts_all) ... ok
test_add_item_rejects_empty_name (__main__.TestShoppingCart.test_add_item_rejects_empty_name) ..
. ok
test_add_item_rejects_negative_price (__main__.TestShoppingCart.test_add_item_rejects_negative_p
rice) ... ok
test_add_item_rejects_non_numeric_price (__main__.TestShoppingCart.test_add_item_rejects_non_num
eric_price) ... ok
test_add_item_rejects_non_string_name (__main__.TestShoppingCart.test_add_item_rejects_non_strin
g_name) ... ok
test_add_multiple_items_updates_total (__main__.TestShoppingCart.test_add_multiple_items_updates
_total) ... ok
test_add_single_item_updates_total (__main__.TestShoppingCart.test_add_single_item_updates_total
) ... ok
test_empty_cart_total_is_zero (__main__.TestShoppingCart.test_empty_cart_total_is_zero) ... ok
test_remove_existing_item_returns_true_and_updates_total (__main__.TestShoppingCart.test_remove_
existing_item_returns_true_and_updates_total) ... ok
test_remove_from_empty_cart_returns_false (__main__.TestShoppingCart.test_remove_from_empty_cart
_returns_false) ... ok
test_remove_nonexistent_item_returns_false_and_total_unchanged (__main__.TestShoppingCart.test_r
emove_nonexistent_item_returns_false_and_total_unchanged) ... ok
test_remove_only_removes_one_instance_when_duplicates_exist (__main__.TestShoppingCart.test_remo
ve_only_removes_one_instance_when_duplicates_exist) ... ok
test_total_cost_is_float (__main__.TestShoppingCart.test_total_cost_is_float) ... ok

----------------------------------------------------------------------
Ran 13 tests in 0.004s

OK

Shopping Cart Module
1. User input
2. Run test cases
3. Exit
Enter your choice (1/2/3):
```

Explanation:

The module manages cart items and accurately calculates the total cost while safely handling additions, removals, and empty cart cases.

Task 5: Date Format Conversion

Prompt:

Generate TDD-based unit test cases for convert_date_format(date_str) that converts dates from "YYYY-MM-DD" to "DD-MM-YYYY", then implement the Python function to pass all generated tests.

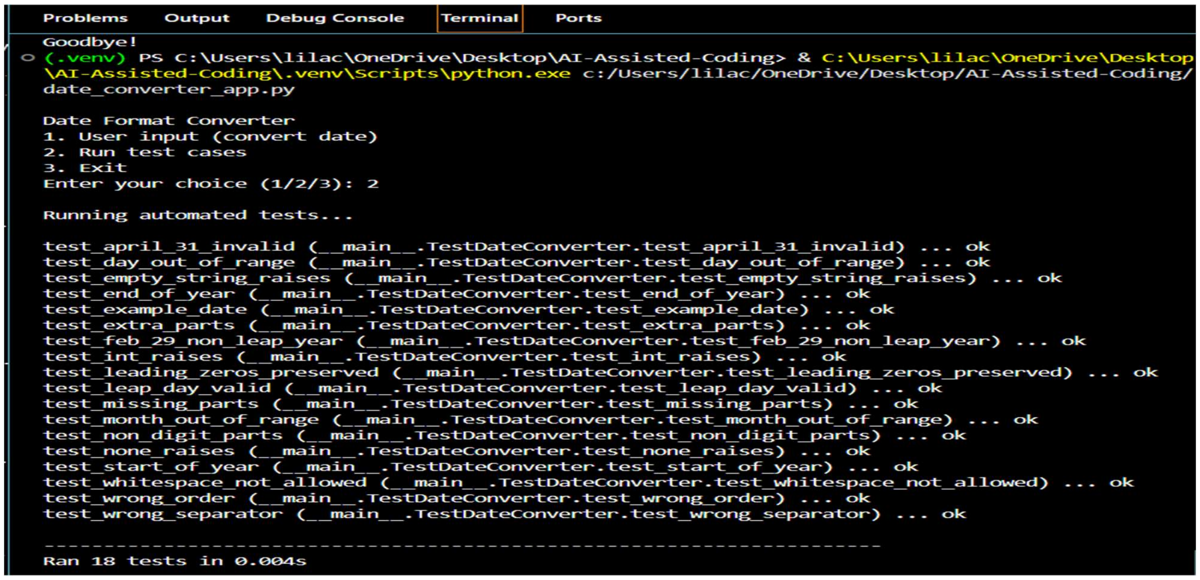
Code:

```python
"""
Task 5: Date Format Conversion - Menu App

Menu:
1) Convert a date (user input)
2) Run automated test cases
3) Exit
"""

from __future__ import annotations

import subprocess

from date_converter import convert_date_format
```

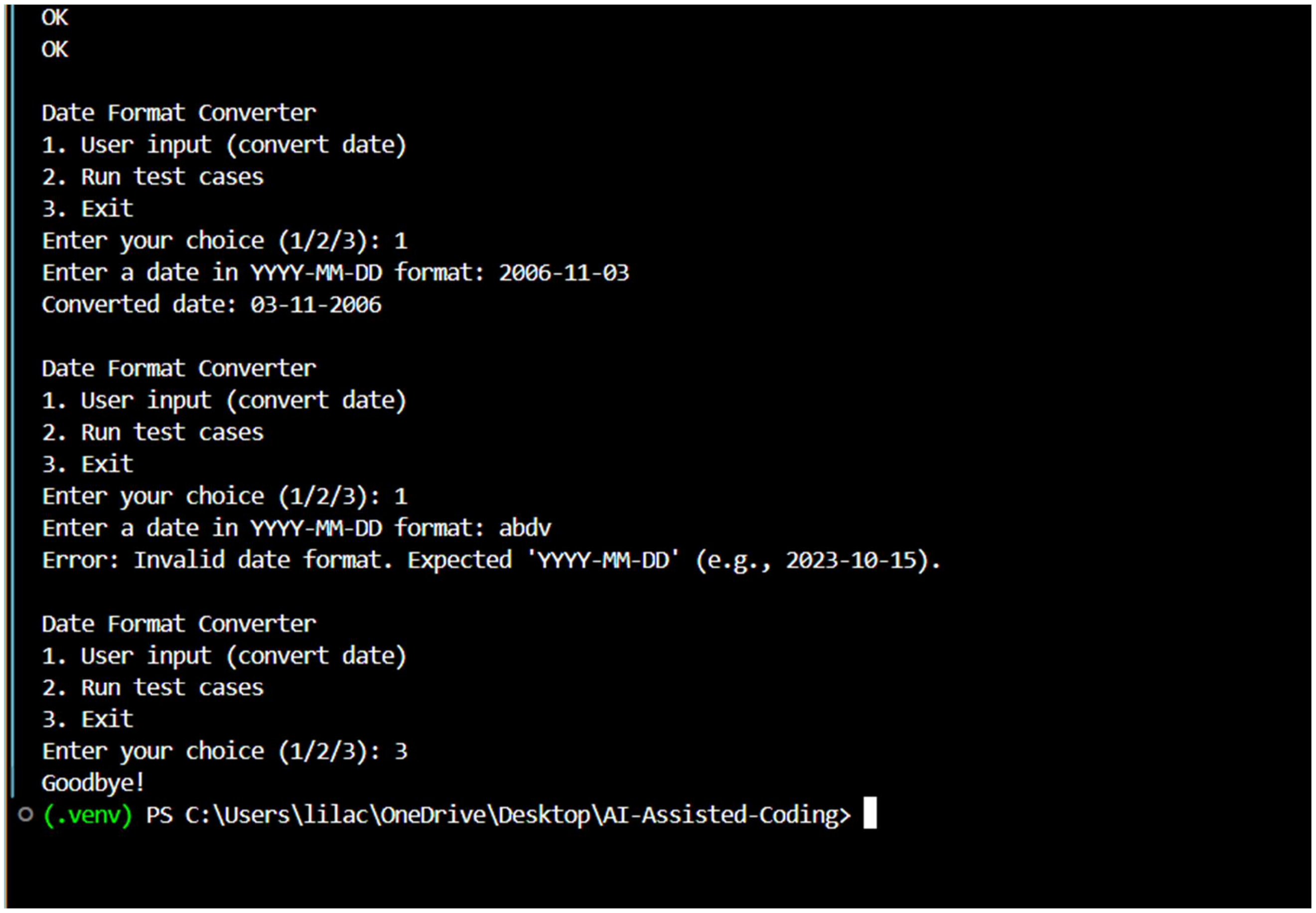
```python
def run_tests() -> None:
    print("\nRunning automated tests...\n")
    subprocess.run(["python", "test_date_converter.py"], check=False)


def user_convert() -> None:
    date_str = input("Enter a date in YYYY-MM-DD format: ")
    try:
        converted = convert_date_format(date_str)
        print(f"Converted date: {converted}")
    except (TypeError, ValueError) as e:
        print(f"Error: {e}")


def main() -> None:
    while True:
        print("\nDate Format Converter")
        print("1. User input (convert date)")
        print("2. Run test cases")
        print("3. Exit")

        choice = input("Enter your choice (1/2/3): ").strip()

        if choice == "1":
            user_convert()
        elif choice == "2":
            run_tests()
        elif choice == "3":
            print("Goodbye!")
            break
        else:
            print("Invalid choice. Please enter 1, 2, or 3.")


if __name__ == "__main__":
    main()


"""
Task 5: Date Format Conversion (TDD)

convert_date_format(date_str):
- Input:  "YYYY-MM-DD"
- Output: "DD-MM-YYYY"
"""

from __future__ import annotations

import re
from datetime import date
from typing import Final


_DATE_RE: Final[re.Pattern[str]] = re.compile(r"^(?P<y>\d{4})-(?P<m>\d{2})-(?P<d>\d{2})$")
```

```python
def convert_date_format(date_str: str) -> str:
    """
    Convert a date string from 'YYYY-MM-DD' to 'DD-MM-YYYY'.

    Args:
        date_str: Date string in the input format.

    Returns:
        Converted date string in the output format.

    Raises:
        TypeError: If date_str is not a string.
        ValueError: If date_str is not exactly 'YYYY-MM-DD' or is not a real calendar
date.
    """
    if not isinstance(date_str, str):
        raise TypeError(f"date_str must be str, got {type(date_str).__name__}")

    match = _DATE_RE.match(date_str)
    if not match:
        raise ValueError("Invalid date format. Expected 'YYYY-MM-DD' (e.g., 2023-10-15).")

    y = int(match.group("y"))
    m = int(match.group("m"))
    d = int(match.group("d"))

    # Validate actual calendar date (handles leap years, month lengths, etc.)
    try:
        date(y, m, d)
    except ValueError as e:
        raise ValueError(f"Invalid date: {e}") from None

    return f"{d:02d}-{m:02d}-{y:04d}"
```

Output:



```
Goodbye!
(.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/
date_converter_app.py

Date Format Converter
1. User input (convert date)
2. Run test cases
3. Exit
Enter your choice (1/2/3): 2

Running automated tests...

test_april_31_invalid (__main__.TestDateConverter.test_april_31_invalid) ... ok
test_day_out_of_range (__main__.TestDateConverter.test_day_out_of_range) ... ok
test_empty_string_raises (__main__.TestDateConverter.test_empty_string_raises) ... ok
test_end_of_year (__main__.TestDateConverter.test_end_of_year) ... ok
test_example_date (__main__.TestDateConverter.test_example_date) ... ok
test_extra_parts (__main__.TestDateConverter.test_extra_parts) ... ok
test_feb_29_non_leap_year (__main__.TestDateConverter.test_feb_29_non_leap_year) ... ok
test_int_raises (__main__.TestDateConverter.test_int_raises) ... ok
test_leading_zeros_preserved (__main__.TestDateConverter.test_leading_zeros_preserved) ... ok
test_leap_day_valid (__main__.TestDateConverter.test_leap_day_valid) ... ok
test_missing_parts (__main__.TestDateConverter.test_missing_parts) ... ok
test_month_out_of_range (__main__.TestDateConverter.test_month_out_of_range) ... ok
test_non_digit_parts (__main__.TestDateConverter.test_non_digit_parts) ... ok
test_none_raises (__main__.TestDateConverter.test_none_raises) ... ok
test_start_of_year (__main__.TestDateConverter.test_start_of_year) ... ok
test_whitespace_not_allowed (__main__.TestDateConverter.test_whitespace_not_allowed) ... ok
test_wrong_order (__main__.TestDateConverter.test_wrong_order) ... ok
test_wrong_separator (__main__.TestDateConverter.test_wrong_separator) ... ok

----------------------------------------------------------------------
Ran 18 tests in 0.004s
```



```
OK
OK

Date Format Converter
1. User input (convert date)
2. Run test cases
3. Exit
Enter your choice (1/2/3): 1
Enter a date in YYYY-MM-DD format: 2006-11-03
Converted date: 03-11-2006

Date Format Converter
1. User input (convert date)
2. Run test cases
3. Exit
Enter your choice (1/2/3): 1
Enter a date in YYYY-MM-DD format: abdv
Error: Invalid date format. Expected 'YYYY-MM-DD' (e.g., 2023-10-15).

Date Format Converter
1. User input (convert date)
2. Run test cases
3. Exit
Enter your choice (1/2/3): 3
Goodbye!
(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding>
```

Explanation:

The function rearranges a valid date string from year–month–day format into day–month–year format while preserving correctness.