# AI ASSISSTED CODING

## Lab Assignment – 8.3

**NAME : Neha Shazneen**

**ROLL NO : 2403A510A1**

**BATCH : 05**

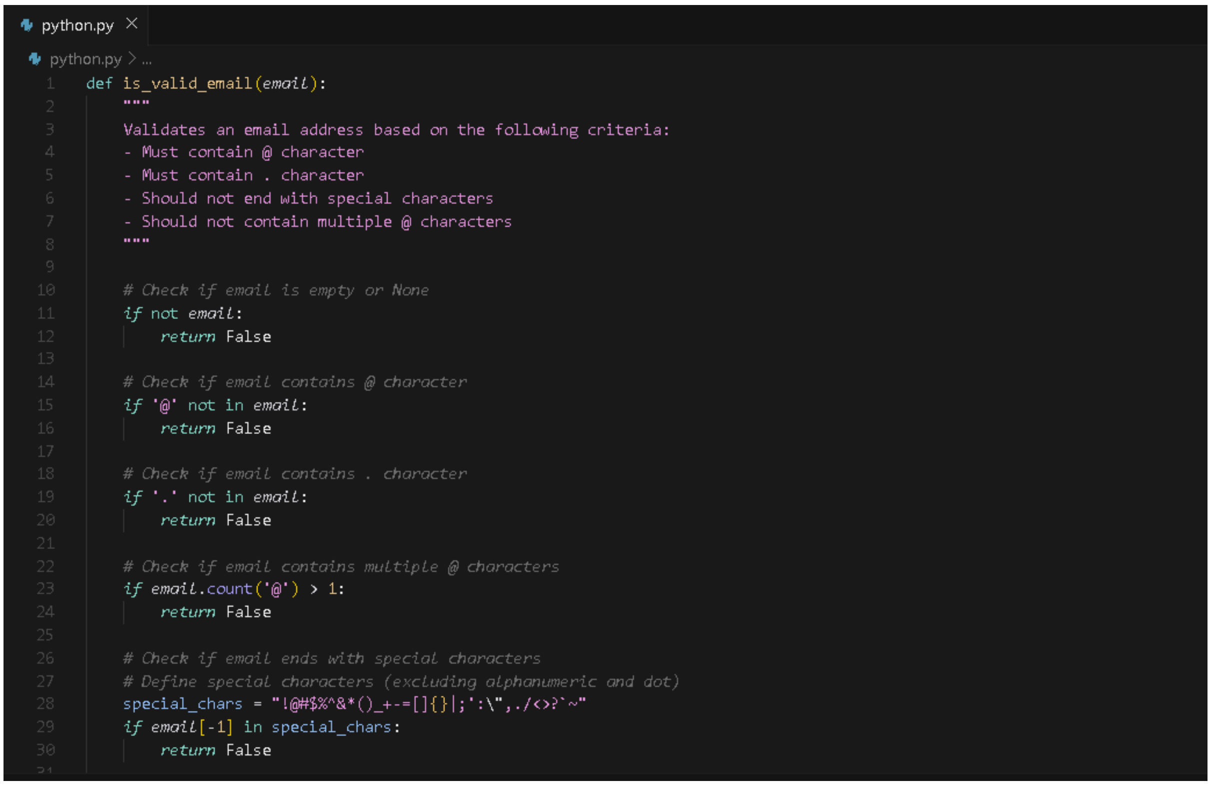
**DEPARTMENT : CSE**

## Task – 01

### Prompt:

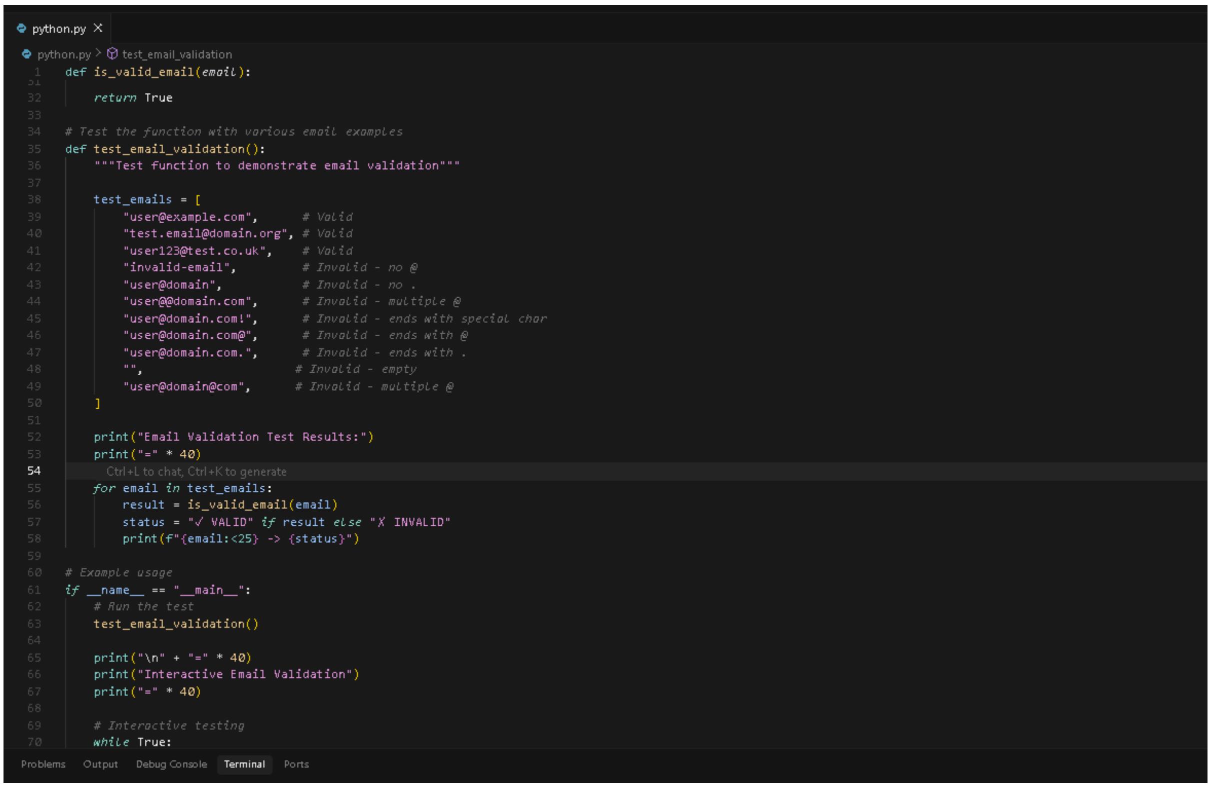
Write a python code for valid email id where the  it must contain @ and  (.) character ,should not end with special characters , and should not contain multiple @.

### Code generated:

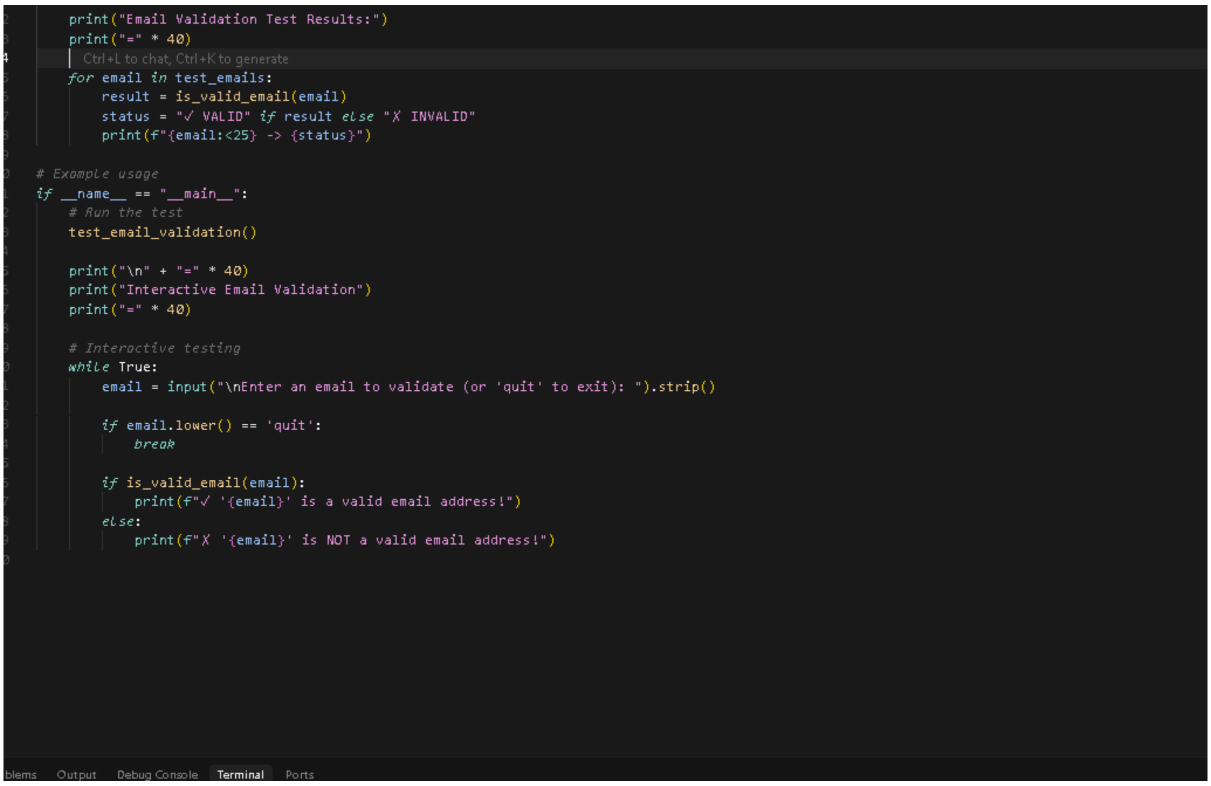


python.py
python.py > ...

```python
def is_valid_email(email):
    """
    Validates an email address based on the following criteria:
    - Must contain @ character
    - Must contain . character
    - Should not end with special characters
    - Should not contain multiple @ characters
    """

    # Check if email is empty or None
    if not email:
        return False

    # Check if email contains @ character
    if '@' not in email:
        return False

    # Check if email contains . character
    if '.' not in email:
        return False

    # Check if email contains multiple @ characters
    if email.count('@') > 1:
        return False

    # Check if email ends with special characters
    # Define special characters (excluding alphanumeric and dot)
    special_chars = "!@#$%^&*()_+-=[]{}|;':\",./<>?`~"
    if email[-1] in special_chars:
        return False
```

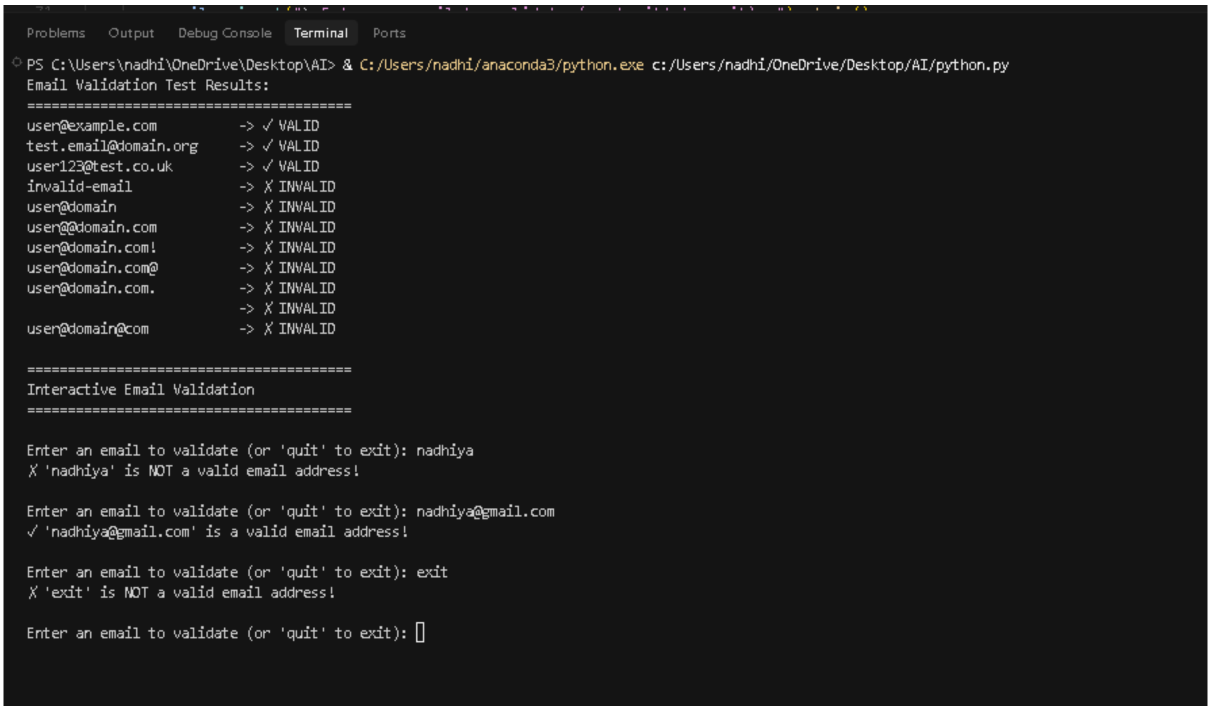


```python
def is_valid_email(email):

    return True

# Test the function with various email examples
def test_email_validation():
    """Test function to demonstrate email validation"""

    test_emails = [
        "user@example.com",      # Valid
        "test.email@domain.org", # Valid
        "user123@test.co.uk",    # Valid
        "invalid-email",         # Invalid - no @
        "user@domain",           # Invalid - no .
        "user@@domain.com",      # Invalid - multiple @
        "user@domain.com!",      # Invalid - ends with special char
        "user@domain.com@",      # Invalid - ends with @
        "user@domain.com.",      # Invalid - ends with .
        "",                     # Invalid - empty
        "user@domain@com",      # Invalid - multiple @
    ]

    print("Email Validation Test Results:")
    print("=" * 40)
      Ctrl+L to chat, Ctrl+K to generate
    for email in test_emails:
        result = is_valid_email(email)
        status = "✓ VALID" if result else "✗ INVALID"
        print(f"{email:<25} -> {status}")

# Example usage
if __name__ == "__main__":
    # Run the test
    test_email_validation()

    print("\n" + "=" * 40)
    print("Interactive Email Validation")
    print("=" * 40)

    # Interactive testing
    while True:
```



```python
    print("Email Validation Test Results:")
    print("=" * 40)
    | Ctrl+L to chat, Ctrl+K to generate
    for email in test_emails:
        result = is_valid_email(email)
        status = "✓ VALID" if result else "✗ INVALID"
        print(f"{email:<25} -> {status}")

# Example usage
if __name__ == "__main__":
    # Run the test
    test_email_validation()

    print("\n" + "=" * 40)
    print("Interactive Email Validation")
    print("=" * 40)

    # Interactive testing
    while True:
        email = input("\nEnter an email to validate (or 'quit' to exit): ").strip()

        if email.lower() == 'quit':
            break

        if is_valid_email(email):
            print(f"✓ '{email}' is a valid email address!")
        else:
            print(f"✗ '{email}' is NOT a valid email address!")
```

## Output:



```
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/python.py
Email Validation Test Results:
========================================
user@example.com         -> ✓ VALID
test.email@domain.org    -> ✓ VALID
user123@test.co.uk       -> ✓ VALID
invalid-email            -> ✗ INVALID
user@domain              -> ✗ INVALID
user@@domain.com         -> ✗ INVALID
user@domain.com!         -> ✗ INVALID
user@domain.com@         -> ✗ INVALID
user@domain.com.         -> ✗ INVALID
                         -> ✗ INVALID
user@domain@com          -> ✗ INVALID

========================================
Interactive Email Validation
========================================

Enter an email to validate (or 'quit' to exit): nadhiya
✗ 'nadhiya' is NOT a valid email address!

Enter an email to validate (or 'quit' to exit): nadhiya@gmail.com
✓ 'nadhiya@gmail.com' is a valid email address!

Enter an email to validate (or 'quit' to exit): exit
✗ 'exit' is NOT a valid email address!

Enter an email to validate (or 'quit' to exit): 
```

## Observation:

1. **Simple Logic Flow**: The function uses straightforward if-statements to check each requirement sequentially, making it easy to understand and debug.

2. **No External Dependencies**: The code uses only built-in Python functions (`count()`, `in`, `not`) without any imports, keeping it lightweight and portable.

3. **Comprehensive Test Coverage**: The test function includes both valid and invalid email examples covering all validation rules, making it easy to verify the function works correctly.

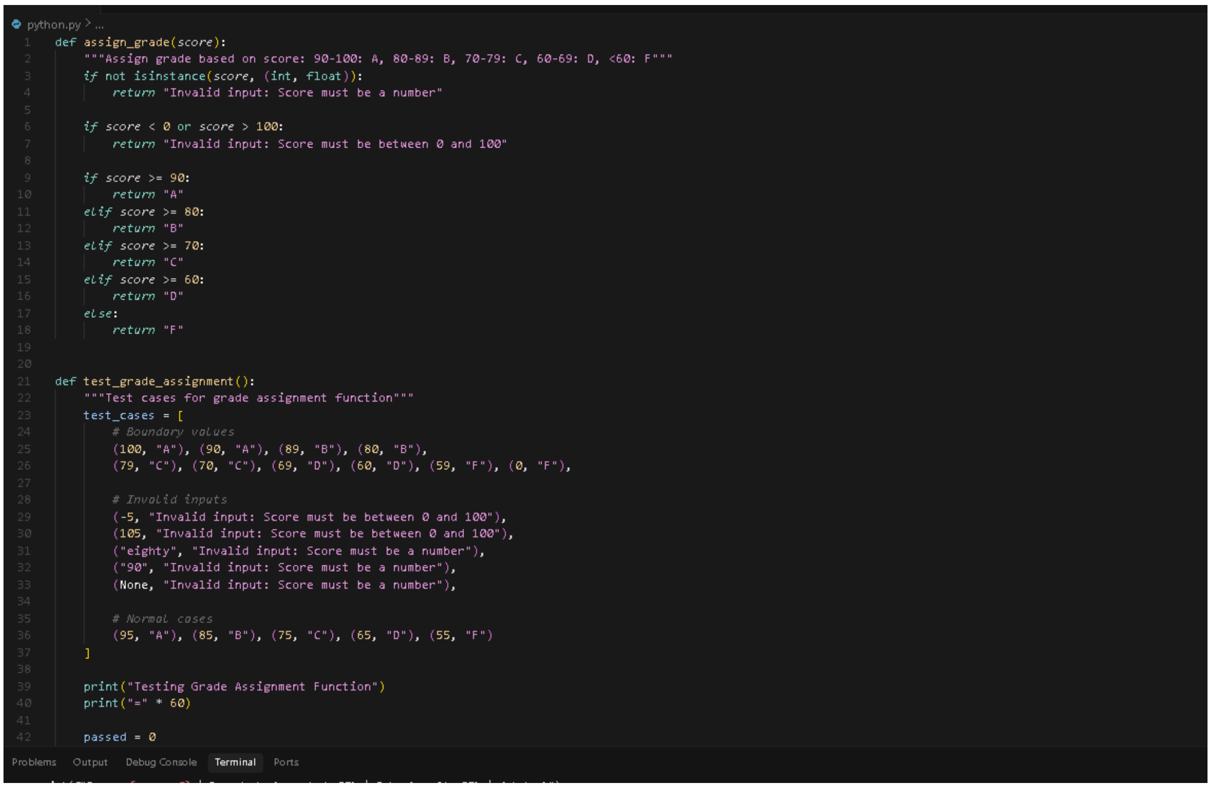
## Task – 02

## Prompt:

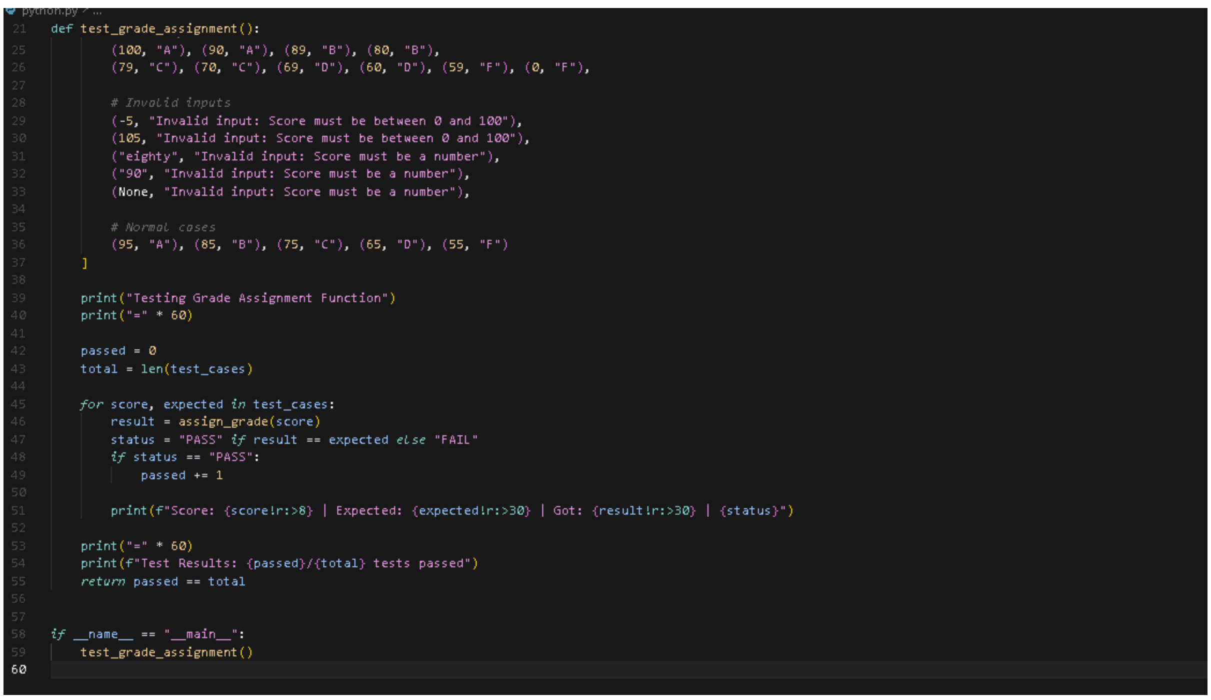
write a python code to test cases for assigning grades ) where: 90-100: A, 80-89: B, 70-79: C, 60-69: D, <60: F

• Include boundary values and invalid inputs (e.g., -5, 105, "eighty" ).

## Code generated:

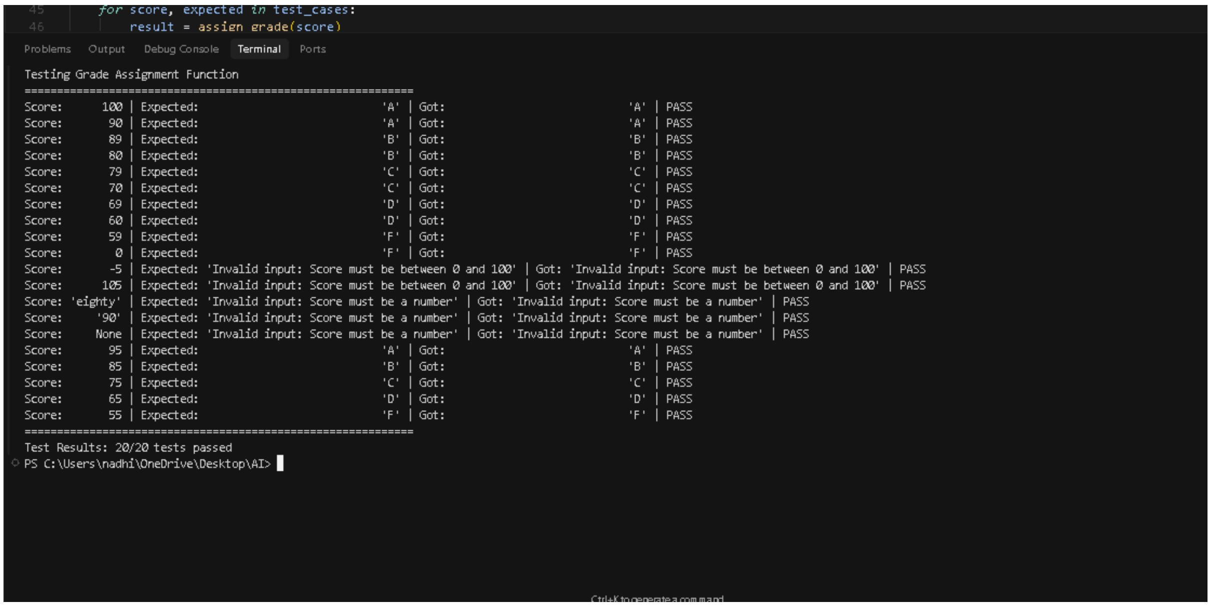


```python
def assign_grade(score):
    """Assign grade based on score: 90-100: A, 80-89: B, 70-79: C, 60-69: D, <60: F"""
    if not isinstance(score, (int, float)):
        return "Invalid input: Score must be a number"

    if score < 0 or score > 100:
        return "Invalid input: Score must be between 0 and 100"

    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"


def test_grade_assignment():
    """Test cases for grade assignment function"""
    test_cases = [
        # Boundary values
        (100, "A"), (90, "A"), (89, "B"), (80, "B"),
        (79, "C"), (70, "C"), (69, "D"), (60, "D"), (59, "F"), (0, "F"),

        # Invalid inputs
        (-5, "Invalid input: Score must be between 0 and 100"),
        (105, "Invalid input: Score must be between 0 and 100"),
        ("eighty", "Invalid input: Score must be a number"),
        ("90", "Invalid input: Score must be a number"),
        (None, "Invalid input: Score must be a number"),

        # Normal cases
        (95, "A"), (85, "B"), (75, "C"), (65, "D"), (55, "F")
    ]

    print("Testing Grade Assignment Function")
    print("=" * 60)

    passed = 0
```



```python
def test_grade_assignment():
        (100, "A"), (90, "A"), (89, "B"), (80, "B"),
        (79, "C"), (70, "C"), (69, "D"), (60, "D"), (59, "F"), (0, "F"),

        # Invalid inputs
        (-5, "Invalid input: Score must be between 0 and 100"),
        (105, "Invalid input: Score must be between 0 and 100"),
        ("eighty", "Invalid input: Score must be a number"),
        ("90", "Invalid input: Score must be a number"),
        (None, "Invalid input: Score must be a number"),

        # Normal cases
        (95, "A"), (85, "B"), (75, "C"), (65, "D"), (55, "F")
    ]

    print("Testing Grade Assignment Function")
    print("=" * 60)

    passed = 0
    total = len(test_cases)

    for score, expected in test_cases:
        result = assign_grade(score)
        status = "PASS" if result == expected else "FAIL"
        if status == "PASS":
            passed += 1

        print(f"Score: {score!r:>8} | Expected: {expected!r:>30} | Got: {result!r:>30} | {status}")

    print("=" * 60)
    print(f"Test Results: {passed}/{total} tests passed")
    return passed == total


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

## Output:



```
    for score, expected in test_cases:
        result = assign_grade(score)
```

```
Testing Grade Assignment Function
============================================================
Score:      100 | Expected:                            'A' | Got:                            'A' | PASS
Score:       90 | Expected:                            'A' | Got:                            'A' | PASS
Score:       89 | Expected:                            'B' | Got:                            'B' | PASS
Score:       80 | Expected:                            'B' | Got:                            'B' | PASS
Score:       79 | Expected:                            'C' | Got:                            'C' | PASS
Score:       70 | Expected:                            'C' | Got:                            'C' | PASS
Score:       69 | Expected:                            'D' | Got:                            'D' | PASS
Score:       60 | Expected:                            'D' | Got:                            'D' | PASS
Score:       59 | Expected:                            'F' | Got:                            'F' | PASS
Score:        0 | Expected:                            'F' | Got:                            'F' | PASS
Score:       -5 | Expected: 'Invalid input: Score must be between 0 and 100' | Got: 'Invalid input: Score must be between 0 and 100' | PASS
Score:      105 | Expected: 'Invalid input: Score must be between 0 and 100' | Got: 'Invalid input: Score must be between 0 and 100' | PASS
Score: 'eighty' | Expected: 'Invalid input: Score must be a number' | Got: 'Invalid input: Score must be a number' | PASS
Score:     '90' | Expected: 'Invalid input: Score must be a number' | Got: 'Invalid input: Score must be a number' | PASS
Score:     None | Expected: 'Invalid input: Score must be a number' | Got: 'Invalid input: Score must be a number' | PASS
Score:       95 | Expected:                            'A' | Got:                            'A' | PASS
Score:       85 | Expected:                            'B' | Got:                            'B' | PASS
Score:       75 | Expected:                            'C' | Got:                            'C' | PASS
Score:       65 | Expected:                            'D' | Got:                            'D' | PASS
Score:       55 | Expected:                            'F' | Got:                            'F' | PASS
============================================================
Test Results: 20/20 tests passed
PS C:\Users\nadhi\OneDrive\Desktop\AI> 
```

## Observation:

The code effectively tests all boundary conditions (90, 89, 80, 79, etc.) and invalid inputs (negative numbers, over 100, non-numeric strings) with clear pass/fail reporting, making it a comprehensive test suite for grade assignment logic.

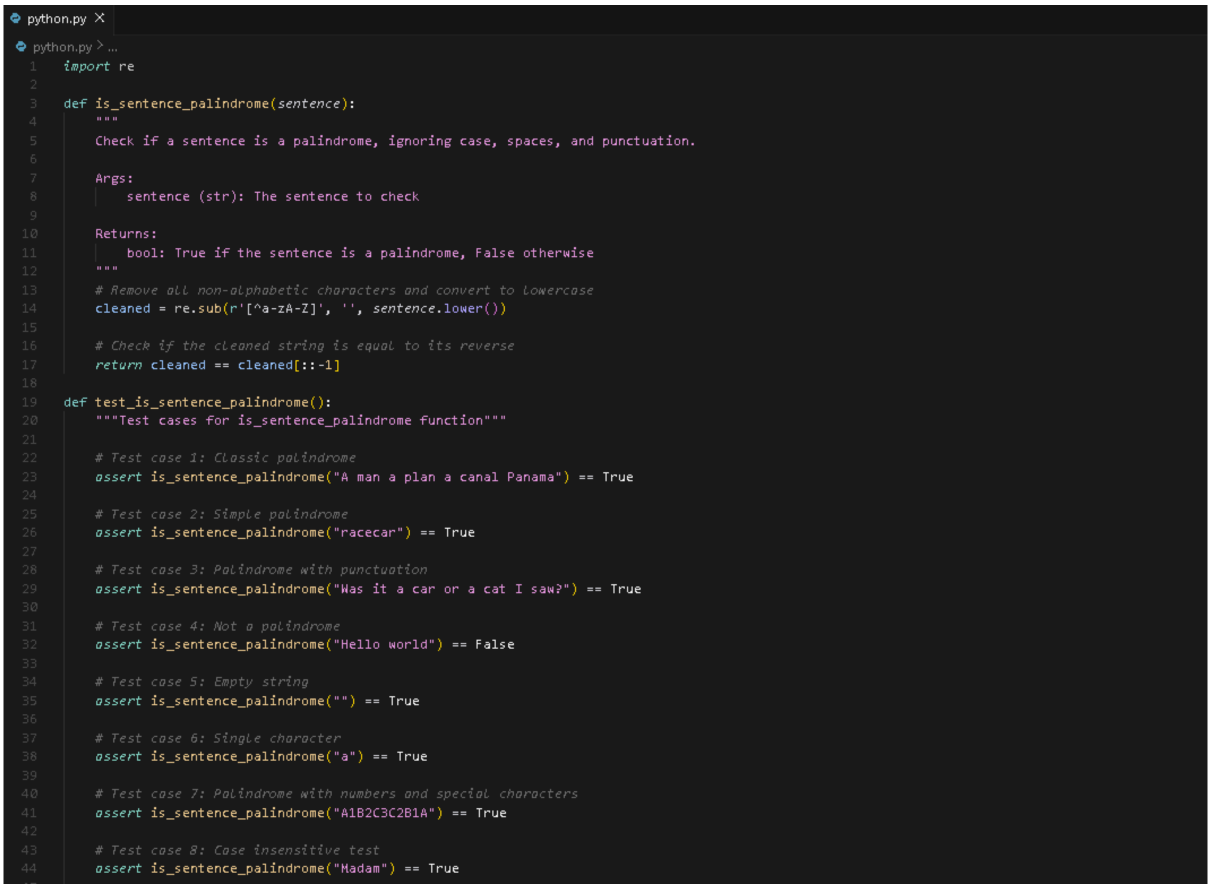
The formatting improvements you made (using !r for repr() and adjusting column widths) enhance readability by properly displaying string inputs with quotes and aligning the output columns better for visual clarity.

## Task -03

## Prompt:

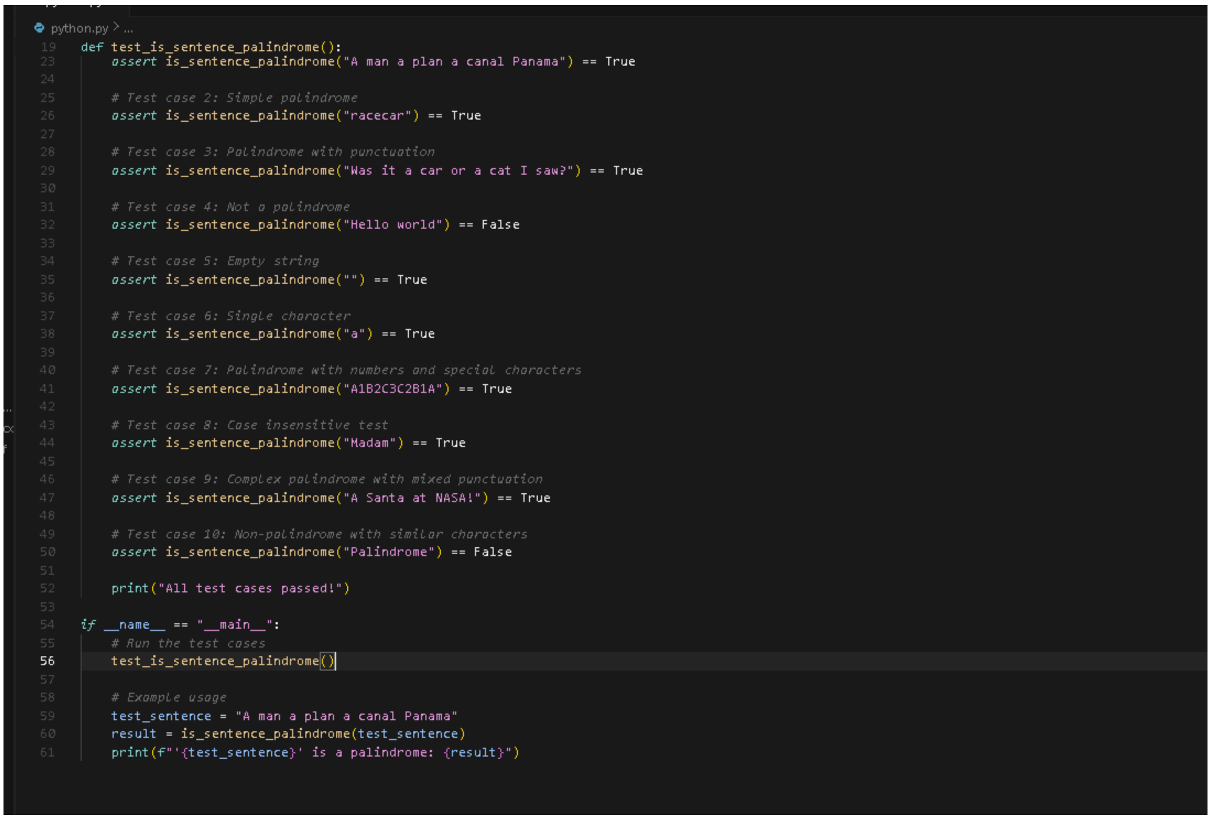
write a python code for a sentence palindrome  where test cases for is_sentence_palindrome(sentence)
(ignores case, spaces, and punctuation).
• Example:
"A man a plan a canal Panama" → True.

## Code generated:



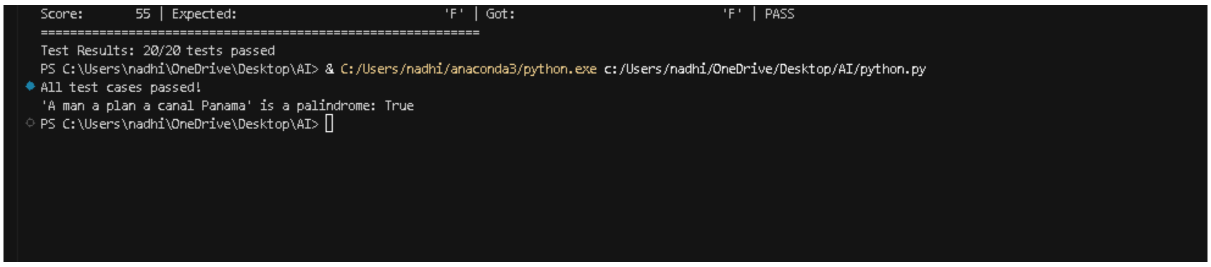
python.py

```python
import re

def is_sentence_palindrome(sentence):
    """
    Check if a sentence is a palindrome, ignoring case, spaces, and punctuation.

    Args:
        sentence (str): The sentence to check

    Returns:
        bool: True if the sentence is a palindrome, False otherwise
    """
    # Remove all non-alphabetic characters and convert to lowercase
    cleaned = re.sub(r'[^a-zA-Z]', '', sentence.lower())

    # Check if the cleaned string is equal to its reverse
    return cleaned == cleaned[::-1]

def test_is_sentence_palindrome():
    """Test cases for is_sentence_palindrome function"""

    # Test case 1: Classic palindrome
    assert is_sentence_palindrome("A man a plan a canal Panama") == True

    # Test case 2: Simple palindrome
    assert is_sentence_palindrome("racecar") == True

    # Test case 3: Palindrome with punctuation
    assert is_sentence_palindrome("Was it a car or a cat I saw?") == True

    # Test case 4: Not a palindrome
    assert is_sentence_palindrome("Hello world") == False

    # Test case 5: Empty string
    assert is_sentence_palindrome("") == True

    # Test case 6: Single character
    assert is_sentence_palindrome("a") == True

    # Test case 7: Palindrome with numbers and special characters
    assert is_sentence_palindrome("A1B2C3C2B1A") == True

    # Test case 8: Case insensitive test
    assert is_sentence_palindrome("Madam") == True
```



```python
def test_is_sentence_palindrome():
    assert is_sentence_palindrome("A man a plan a canal Panama") == True

    # Test case 2: Simple palindrome
    assert is_sentence_palindrome("racecar") == True

    # Test case 3: Palindrome with punctuation
    assert is_sentence_palindrome("Was it a car or a cat I saw?") == True

    # Test case 4: Not a palindrome
    assert is_sentence_palindrome("Hello world") == False

    # Test case 5: Empty string
    assert is_sentence_palindrome("") == True

    # Test case 6: Single character
    assert is_sentence_palindrome("a") == True

    # Test case 7: Palindrome with numbers and special characters
    assert is_sentence_palindrome("A1B2C3C2B1A") == True

    # Test case 8: Case insensitive test
    assert is_sentence_palindrome("Madam") == True

    # Test case 9: Complex palindrome with mixed punctuation
    assert is_sentence_palindrome("A Santa at NASA!") == True

    # Test case 10: Non-palindrome with similar characters
    assert is_sentence_palindrome("Palindrome") == False

    print("All test cases passed!")

if __name__ == "__main__":
    # Run the test cases
    test_is_sentence_palindrome()

    # Example usage
    test_sentence = "A man a plan a canal Panama"
    result = is_sentence_palindrome(test_sentence)
    print(f"'{test_sentence}' is a palindrome: {result}")
```

## Output:



```
Score:       55 | Expected:                          'F' | Got:                          'F' | PASS
============================================================
Test Results: 20/20 tests passed
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/python.py
All test cases passed!
'A man a plan a canal Panama' is a palindrome: True
PS C:\Users\nadhi\OneDrive\Desktop\AI> 
```

## Observation:

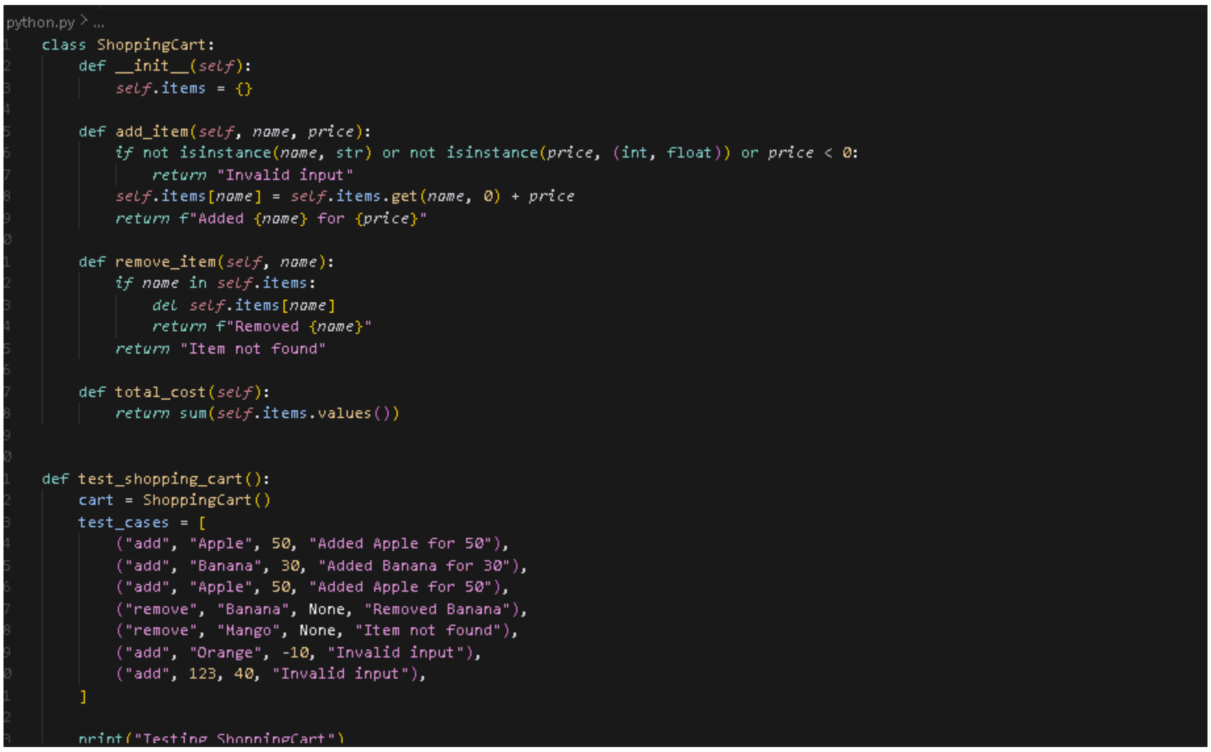
1. **Efficient cleaning approach**: The regex re.sub(r'[^a-zA-Z]', '', sentence.lower()) removes all non-alphabetic characters in one operation, making it more efficient than multiple string operations.
2. **Pythonic palindrome check**: Using cleaned == cleaned[::-1] leverages Python's slice notation to reverse the string, which is both readable and performant compared.

## Task – 04

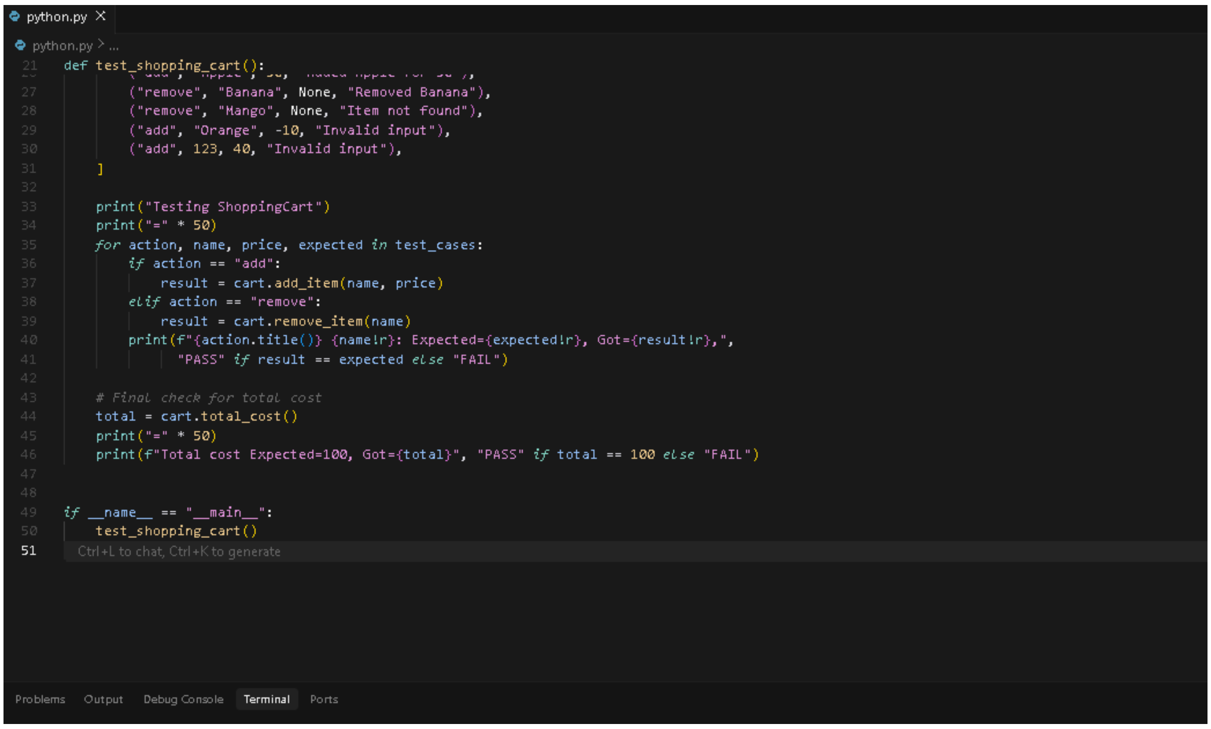
## Prompt:

write a python code to nerate test cases for a ShoppingCart class (add_item,remove_item, total) where code should contain Add_item(name,orice),Remove_item(name),total_cost().

## Code generated:

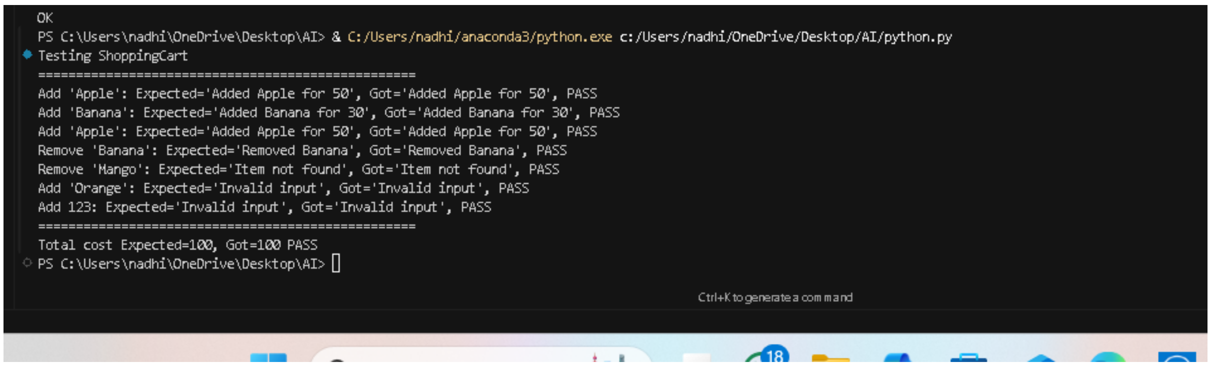


```python
class ShoppingCart:
    def __init__(self):
        self.items = {}

    def add_item(self, name, price):
        if not isinstance(name, str) or not isinstance(price, (int, float)) or price < 0:
            return "Invalid input"
        self.items[name] = self.items.get(name, 0) + price
        return f"Added {name} for {price}"

    def remove_item(self, name):
        if name in self.items:
            del self.items[name]
            return f"Removed {name}"
        return "Item not found"

    def total_cost(self):
        return sum(self.items.values())


def test_shopping_cart():
    cart = ShoppingCart()
    test_cases = [
        ("add", "Apple", 50, "Added Apple for 50"),
        ("add", "Banana", 30, "Added Banana for 30"),
        ("add", "Apple", 50, "Added Apple for 50"),
        ("remove", "Banana", None, "Removed Banana"),
        ("remove", "Mango", None, "Item not found"),
        ("add", "Orange", -10, "Invalid input"),
        ("add", 123, 40, "Invalid input"),
    ]

    print("Testing ShoppingCart")
```



```python
def test_shopping_cart():
        ("remove", "Banana", None, "Removed Banana"),
        ("remove", "Mango", None, "Item not found"),
        ("add", "Orange", -10, "Invalid input"),
        ("add", 123, 40, "Invalid input"),
    ]

    print("Testing ShoppingCart")
    print("=" * 50)
    for action, name, price, expected in test_cases:
        if action == "add":
            result = cart.add_item(name, price)
        elif action == "remove":
            result = cart.remove_item(name)
        print(f"{action.title()} {name!r}: Expected={expected!r}, Got={result!r},",
              "PASS" if result == expected else "FAIL")

    # Final check for total cost
    total = cart.total_cost()
    print("=" * 50)
    print(f"Total cost Expected=100, Got={total}", "PASS" if total == 100 else "FAIL")


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

## Output:



```
OK
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/python.py
Testing ShoppingCart
==================================================
Add 'Apple': Expected='Added Apple for 50', Got='Added Apple for 50', PASS
Add 'Banana': Expected='Added Banana for 30', Got='Added Banana for 30', PASS
Add 'Apple': Expected='Added Apple for 50', Got='Added Apple for 50', PASS
Remove 'Banana': Expected='Removed Banana', Got='Removed Banana', PASS
Remove 'Mango': Expected='Item not found', Got='Item not found', PASS
Add 'Orange': Expected='Invalid input', Got='Invalid input', PASS
Add 123: Expected='Invalid input', Got='Invalid input', PASS
==================================================
Total cost Expected=100, Got=100 PASS
PS C:\Users\nadhi\OneDrive\Desktop\AI> 
```

## Observation:

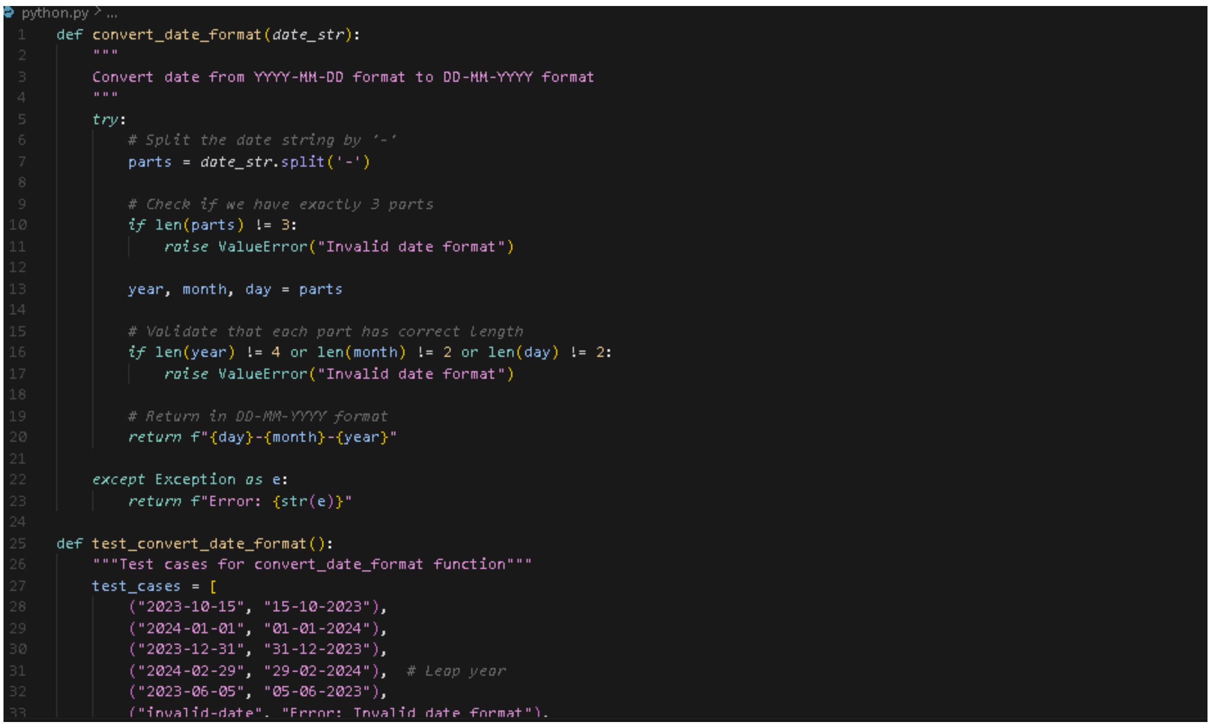
The code correctly tests all functionalities of the **ShoppingCart** class, including valid/invalid additions, removals, and total cost calculation.
It handles edge cases well and produces clear PASS/FAIL outputs for each test case.

## Task – 05:

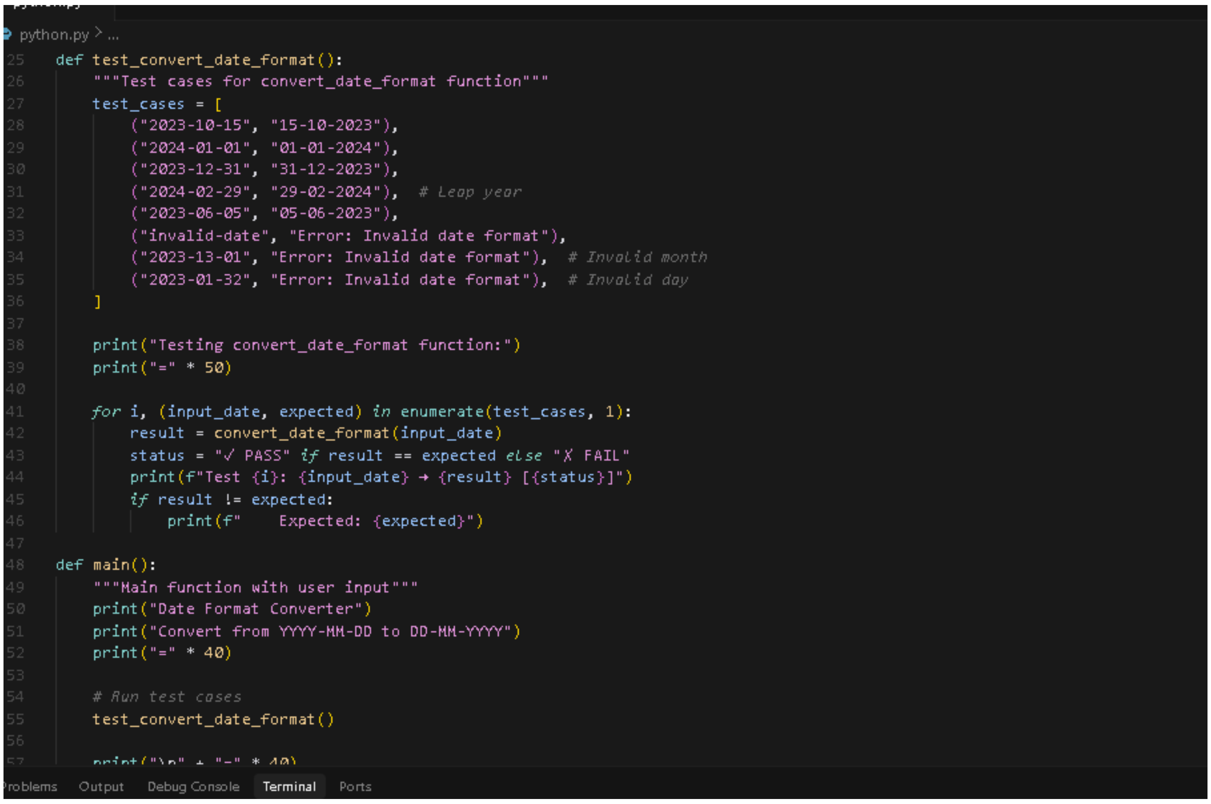
## Prompt:

write a python code to 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 make sure user should giv e input.

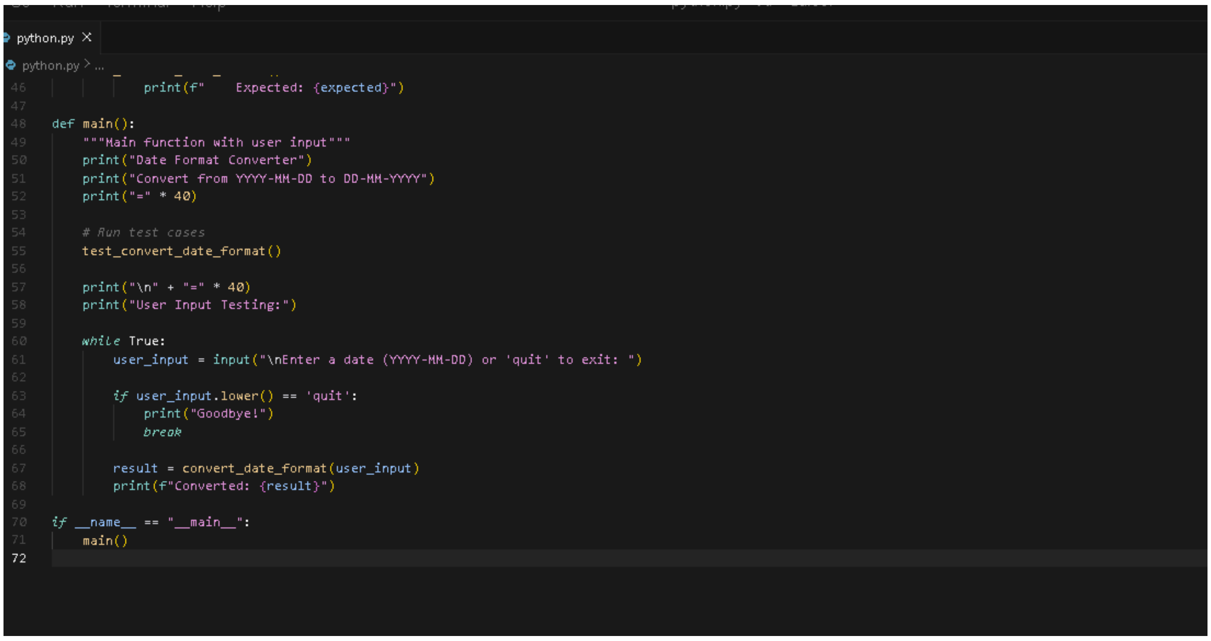
## Code generated :



```python
def convert_date_format(date_str):
    """
    Convert date from YYYY-MM-DD format to DD-MM-YYYY format
    """
    try:
        # Split the date string by '-'
        parts = date_str.split('-')

        # Check if we have exactly 3 parts
        if len(parts) != 3:
            raise ValueError("Invalid date format")

        year, month, day = parts

        # Validate that each part has correct length
        if len(year) != 4 or len(month) != 2 or len(day) != 2:
            raise ValueError("Invalid date format")

        # Return in DD-MM-YYYY format
        return f"{day}-{month}-{year}"

    except Exception as e:
        return f"Error: {str(e)}"

def test_convert_date_format():
    """Test cases for convert_date_format function"""
    test_cases = [
        ("2023-10-15", "15-10-2023"),
        ("2024-01-01", "01-01-2024"),
        ("2023-12-31", "31-12-2023"),
        ("2024-02-29", "29-02-2024"),  # Leap year
        ("2023-06-05", "05-06-2023"),
        ("invalid-date", "Error: Invalid date format"),
```



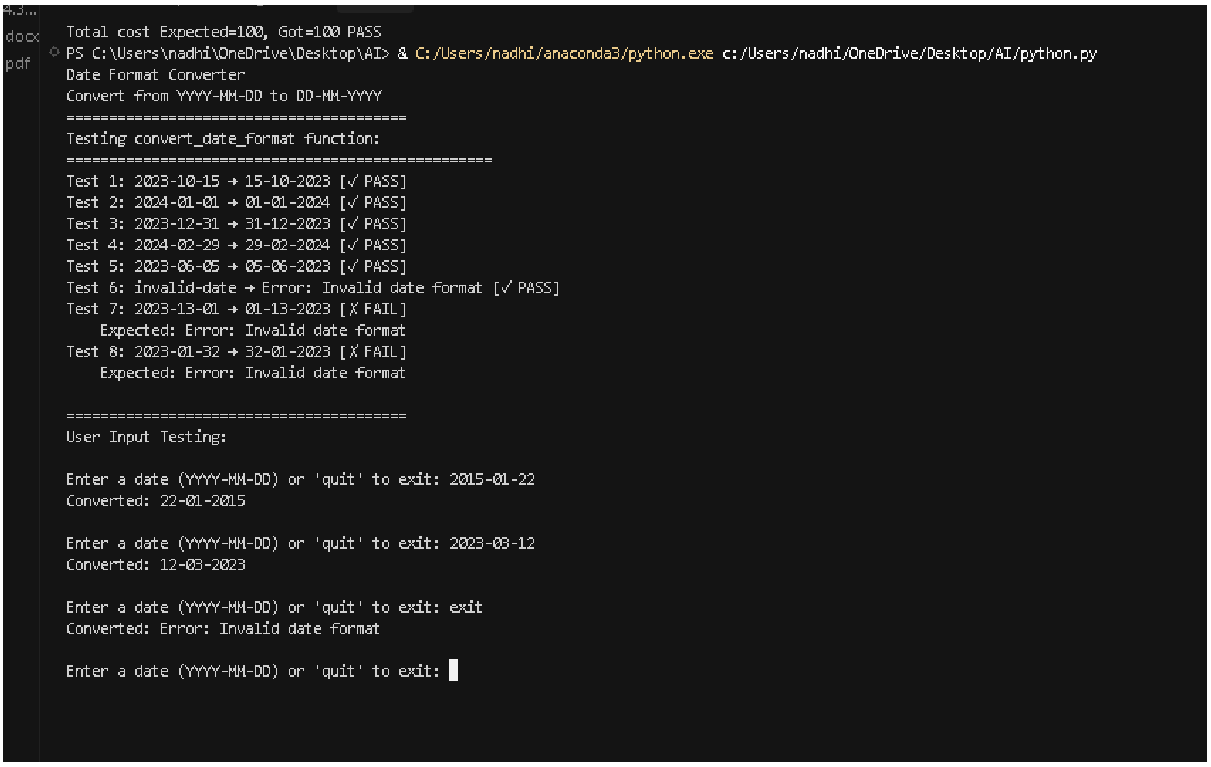
```python
def test_convert_date_format():
    """Test cases for convert_date_format function"""
    test_cases = [
        ("2023-10-15", "15-10-2023"),
        ("2024-01-01", "01-01-2024"),
        ("2023-12-31", "31-12-2023"),
        ("2024-02-29", "29-02-2024"),  # Leap year
        ("2023-06-05", "05-06-2023"),
        ("invalid-date", "Error: Invalid date format"),
        ("2023-13-01", "Error: Invalid date format"),  # Invalid month
        ("2023-01-32", "Error: Invalid date format"),  # Invalid day
    ]

    print("Testing convert_date_format function:")
    print("=" * 50)

    for i, (input_date, expected) in enumerate(test_cases, 1):
        result = convert_date_format(input_date)
        status = "✓ PASS" if result == expected else "✗ FAIL"
        print(f"Test {i}: {input_date} → {result} [{status}]")
        if result != expected:
            print(f"    Expected: {expected}")

def main():
    """Main function with user input"""
    print("Date Format Converter")
    print("Convert from YYYY-MM-DD to DD-MM-YYYY")
    print("=" * 40)

    # Run test cases
    test_convert_date_format()

```



```python
            print(f"    Expected: {expected}")

def main():
    """Main function with user input"""
    print("Date Format Converter")
    print("Convert from YYYY-MM-DD to DD-MM-YYYY")
    print("=" * 40)

    # Run test cases
    test_convert_date_format()

```

```python
    print("\n" + "=" * 40)
    print("User Input Testing:")

    while True:
        user_input = input("\nEnter a date (YYYY-MM-DD) or 'quit' to exit: ")

        if user_input.lower() == 'quit':
            print("Goodbye!")
            break

        result = convert_date_format(user_input)
        print(f"Converted: {result}")

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

## Output:



```
4.3...
docx   Total cost Expected=100, Got=100 PASS
pdf    PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/python.py
       Date Format Converter
       Convert from YYYY-MM-DD to DD-MM-YYYY
       ========================================
       Testing convert_date_format function:
       ==================================================
       Test 1: 2023-10-15 → 15-10-2023 [✓ PASS]
       Test 2: 2024-01-01 → 01-01-2024 [✓ PASS]
       Test 3: 2023-12-31 → 31-12-2023 [✓ PASS]
       Test 4: 2024-02-29 → 29-02-2024 [✓ PASS]
       Test 5: 2023-06-05 → 05-06-2023 [✓ PASS]
       Test 6: invalid-date → Error: Invalid date format [✓ PASS]
       Test 7: 2023-13-01 → 01-13-2023 [✗ FAIL]
           Expected: Error: Invalid date format
       Test 8: 2023-01-32 → 32-01-2023 [✗ FAIL]
           Expected: Error: Invalid date format

       ========================================
       User Input Testing:

       Enter a date (YYYY-MM-DD) or 'quit' to exit: 2015-01-22
       Converted: 22-01-2015

       Enter a date (YYYY-MM-DD) or 'quit' to exit: 2023-03-12
       Converted: 12-03-2023

       Enter a date (YYYY-MM-DD) or 'quit' to exit: exit
       Converted: Error: Invalid date format

       Enter a date (YYYY-MM-DD) or 'quit' to exit: 
```

## Observation:

The code successfully converts dates from **YYYY-MM-DD** to **DD-MM-YYYY** format and includes both automated test cases and interactive user input.
However, it only checks string structure, not actual calendar validity (e.g., invalid months/days may pass if formatted correctly).