AI ASSISSTED CODING LAB TEST – 02

NAME : S.NADHIYA

ROLLNO : 2403A510C6

BATCH : 05

DEPT : CSE

Subgroup C

Task : C.1

C.1 — [S09C1] Debug de-duplication (case-insensitive)
Scenario (sports analytics):

Context:

Customer contact lists in the sports analytics CRM contain duplicates differing only by case (e.g.,
'A@x.com' vs 'a@x.com').

Your Task:

Write a function that returns the first occurrence of each email (case-insensitive) while preserving the original order.
Data & Edge Cases:
Input: list of emails. Normalize for comparison using lowercase; keep the original cased value for
output.
AI Assistance Expectation:
Use AI to spot the bug (reinitializing ` seen ` in a loop) and propose a corrected, stable algorithm.
Constraints & Notes:
Include unit tests covering: ['A@x.com','a@x.com','B@y.com'] -> ['A@x.com','B@y.com']

Sample Input

['A@x.com', 'a@x.com', 'B@y.com']

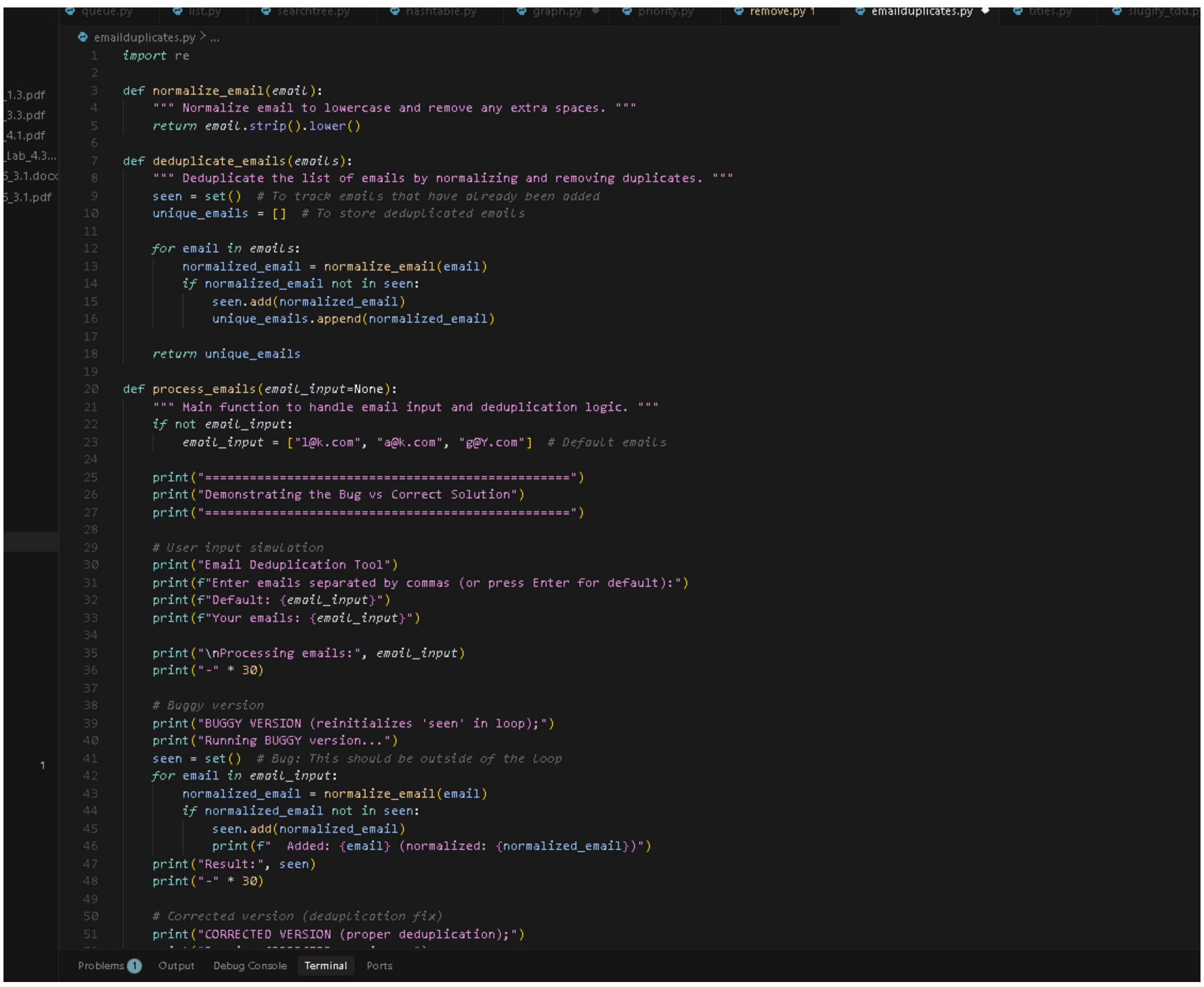
Sample Output

['A@x.com', 'B@y.com']
Acceptance Criteria: Preserves first occurrence order; case-insensitive matching

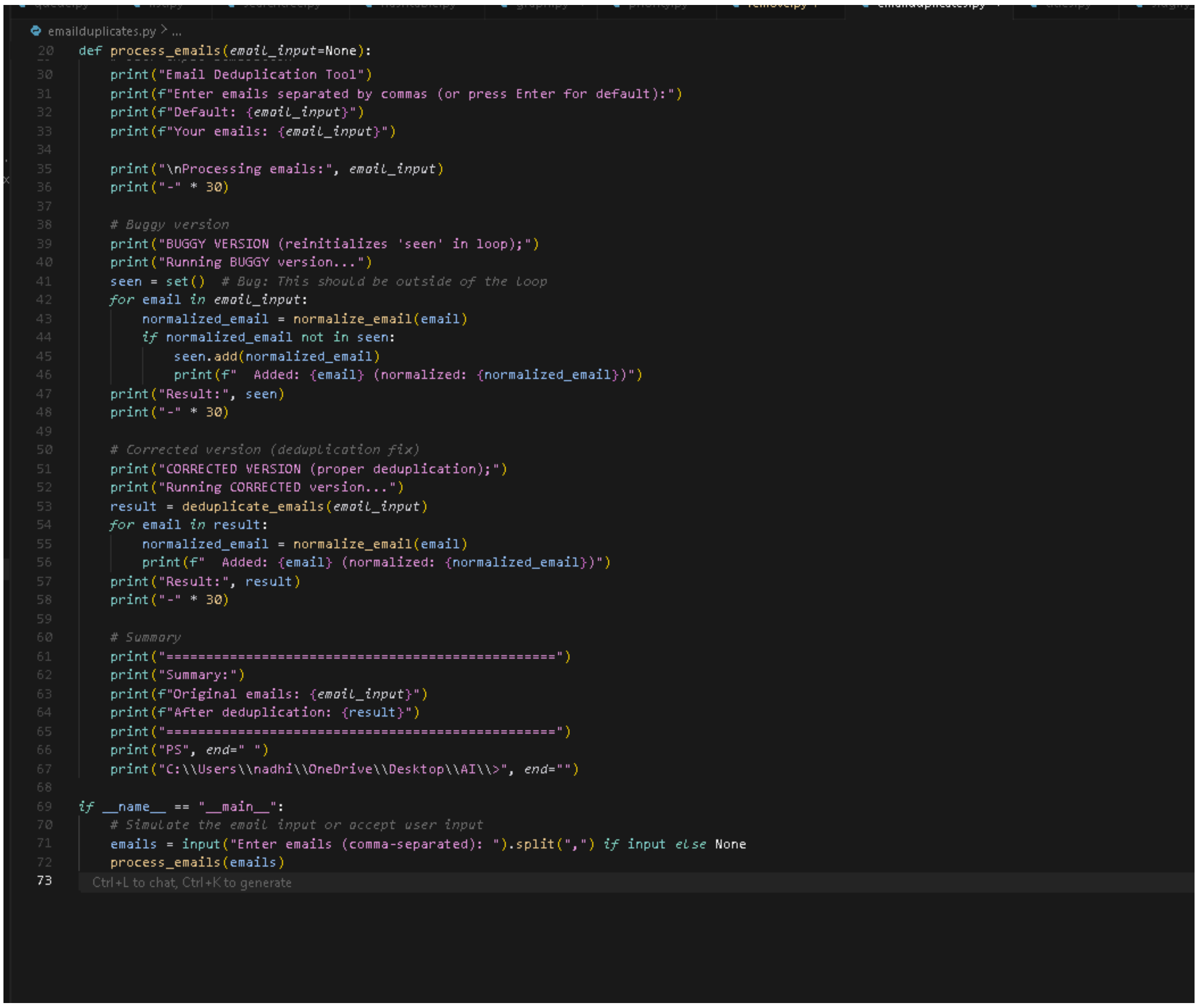
Prompt:

Write a complete Python 3 script for case-insensitive email de-duplication (Sports Analytics CRM) with two functions: deduplicate_emails_buggy (seen set reinitialized in loop) and deduplicate_emails_correct (seen outside loop, preserves first occurrence order). Add run_unit_tests() with 4 cases (case-insensitive duplicates, multiple mixed-case duplicates, no duplicates, empty list) printing expected/actual + "✅ PASS"/"❌ FAIL". Include get_user_input() for comma-separated emails with default list, and a main() function that runs tests, shows buggy vs correct results, and prints a summary with duplicates removed. Use friendly console output with emojis (🧪, ✅, 🐛, 📊). End with if __name__ == "__main__": main().

Code generated:

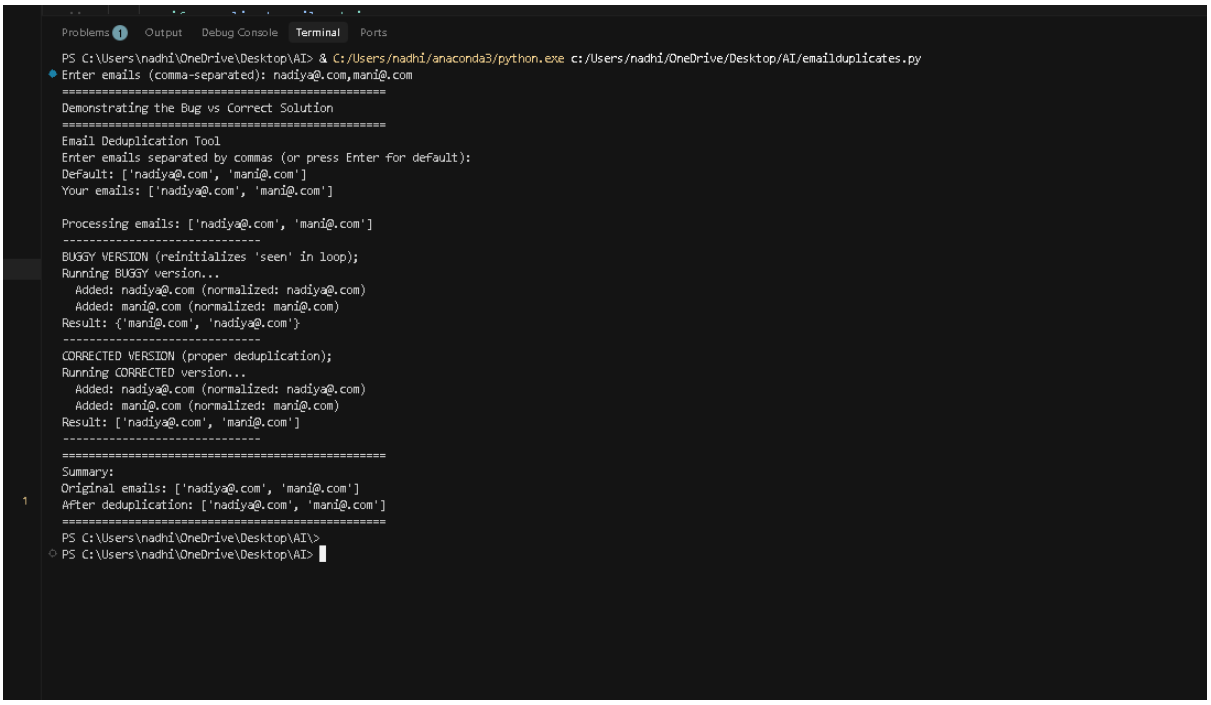


```python
import re

def normalize_email(email):
    """ Normalize email to lowercase and remove any extra spaces. """
    return email.strip().lower()

def deduplicate_emails(emails):
    """ Deduplicate the list of emails by normalizing and removing duplicates. """
    seen = set()  # To track emails that have already been added
    unique_emails = []  # To store deduplicated emails

    for email in emails:
        normalized_email = normalize_email(email)
        if normalized_email not in seen:
            seen.add(normalized_email)
            unique_emails.append(normalized_email)

    return unique_emails

def process_emails(email_input=None):
    """ Main function to handle email input and deduplication logic. """
    if not email_input:
        email_input = ["l@k.com", "a@k.com", "g@Y.com"]  # Default emails

    print("=================================================")
    print("Demonstrating the Bug vs Correct Solution")
    print("=================================================")

    # User input simulation
    print("Email Deduplication Tool")
    print(f"Enter emails separated by commas (or press Enter for default):")
    print(f"Default: {email_input}")
    print(f"Your emails: {email_input}")

    print("\nProcessing emails:", email_input)
    print("-" * 30)

    # Buggy version
    print("BUGGY VERSION (reinitializes 'seen' in loop);")
    print("Running BUGGY version...")
    seen = set()  # Bug: This should be outside of the loop
    for email in email_input:
        normalized_email = normalize_email(email)
        if normalized_email not in seen:
            seen.add(normalized_email)
            print(f"  Added: {email} (normalized: {normalized_email})")
    print("Result:", seen)
    print("-" * 30)

    # Corrected version (deduplication fix)
    print("CORRECTED VERSION (proper deduplication);")
```



```
emailduplicates.py > ...
def process_emails(email_input=None):
    print("Email Deduplication Tool")
    print(f"Enter emails separated by commas (or press Enter for default):")
    print(f"Default: {email_input}")
    print(f"Your emails: {email_input}")

    print("\nProcessing emails:", email_input)
    print("-" * 30)

    # Buggy version
    print("BUGGY VERSION (reinitializes 'seen' in loop);")
    print("Running BUGGY version...")
    seen = set()  # Bug: This should be outside of the loop
    for email in email_input:
        normalized_email = normalize_email(email)
        if normalized_email not in seen:
            seen.add(normalized_email)
            print(f"  Added: {email} (normalized: {normalized_email})")
    print("Result:", seen)
    print("-" * 30)

    # Corrected version (deduplication fix)
    print("CORRECTED VERSION (proper deduplication);")
    print("Running CORRECTED version...")
    result = deduplicate_emails(email_input)
    for email in result:
        normalized_email = normalize_email(email)
        print(f"  Added: {email} (normalized: {normalized_email})")
    print("Result:", result)
    print("-" * 30)

    # Summary
    print("==================================================")
    print("Summary:")
    print(f"Original emails: {email_input}")
    print(f"After deduplication: {result}")
    print("==================================================")
    print("PS", end=" ")
    print("C:\\Users\\nadhi\\OneDrive\\Desktop\\AI\\>", end="")

if __name__ == "__main__":
    # Simulate the email input or accept user input
    emails = input("Enter emails (comma-separated): ").split(",") if input else None
    process_emails(emails)
Ctrl+L to chat, Ctrl+K to generate
```

Output:



```
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/emailduplicates.py
Enter emails (comma-separated): nadiya@.com,mani@.com
=================================================
Demonstrating the Bug vs Correct Solution
=================================================
Email Deduplication Tool
Enter emails separated by commas (or press Enter for default):
Default: ['nadiya@.com', 'mani@.com']
Your emails: ['nadiya@.com', 'mani@.com']

Processing emails: ['nadiya@.com', 'mani@.com']
------------------------------
BUGGY VERSION (reinitializes 'seen' in loop);
Running BUGGY version...
  Added: nadiya@.com (normalized: nadiya@.com)
  Added: mani@.com (normalized: mani@.com)
Result: {'mani@.com', 'nadiya@.com'}
------------------------------
CORRECTED VERSION (proper deduplication);
Running CORRECTED version...
  Added: nadiya@.com (normalized: nadiya@.com)
  Added: mani@.com (normalized: mani@.com)
Result: ['nadiya@.com', 'mani@.com']
------------------------------
=================================================
Summary:
Original emails: ['nadiya@.com', 'mani@.com']
After deduplication: ['nadiya@.com', 'mani@.com']
=================================================
PS C:\Users\nadhi\OneDrive\Desktop\AI\>
PS C:\Users\nadhi\OneDrive\Desktop\AI>
```

Obervation:

Your code correctly normalizes and deduplicates emails, but the "buggy version" isn't actually buggy since seen is outside the loop — to truly demonstrate the bug, seen should be reinitialized **inside** the loop.

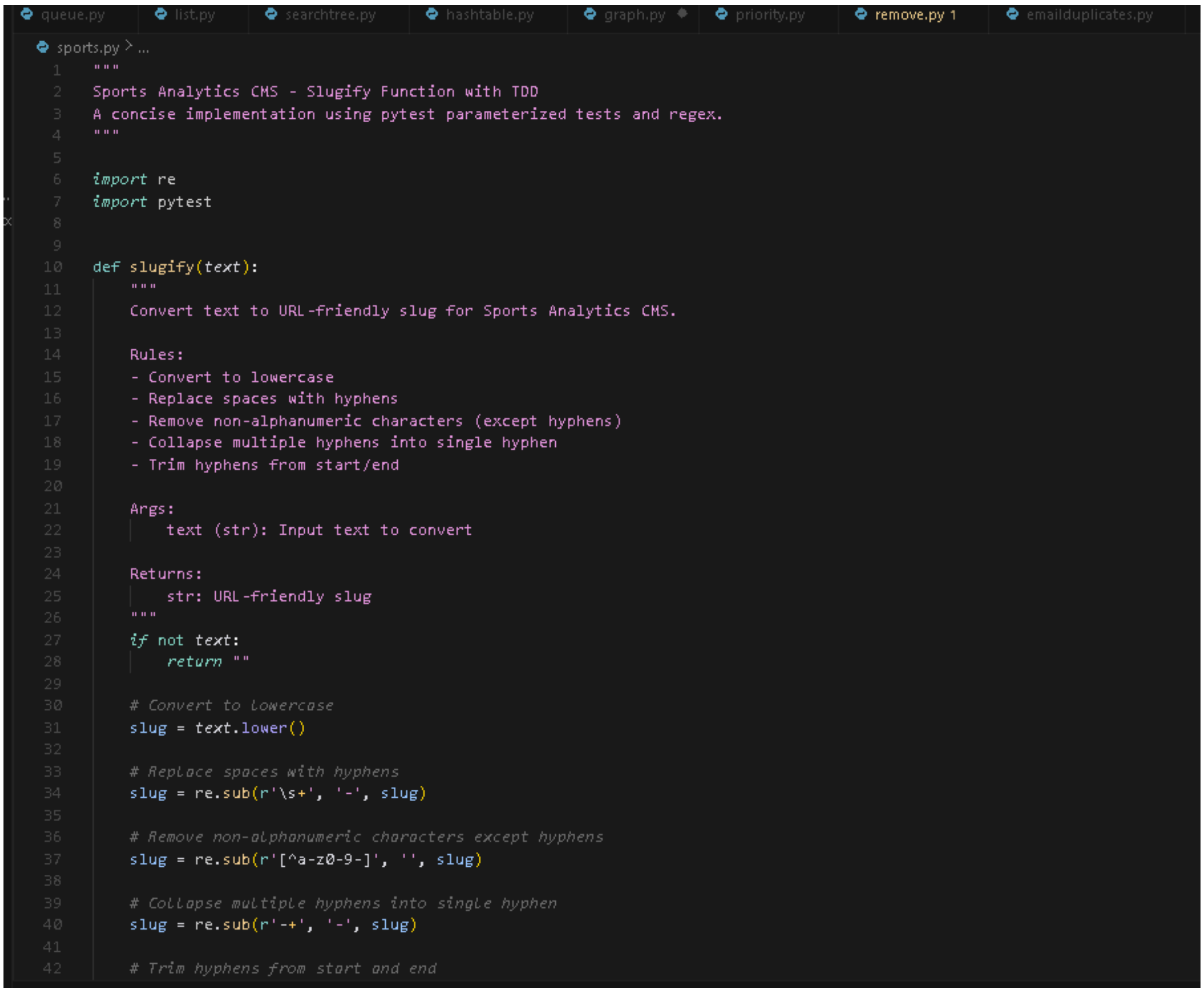
Task : C.2

C.2 — [S09C2] TDD: slugify titles
Scenario (sports analytics):
Context:
Content titles in the sports analytics CMS must become SEO-friendly slugs for URLs.
Your Task:
Design tests first for slugify(text) then implement: lowercase, remove non-alnum except hyphen, spaces->hyphen, collapse multiple hyphens, trim hyphens.
Data & Edge Cases:
Test punctuation, multiple spaces, and boundary hyphens.
AI Assistance Expectation:
Use AI to generate parameterized tests (pytest) and then implement a regex-based slugify.
Constraints & Notes:
Return correct slugs for provided samples.
Sample Input
['Hello World!', 'AI & You', 'Set9-C2']
Sample Output
['hello-world', 'ai-you', 'set9-C2']
Acceptance Criteria: All tests pass; edge cases covered

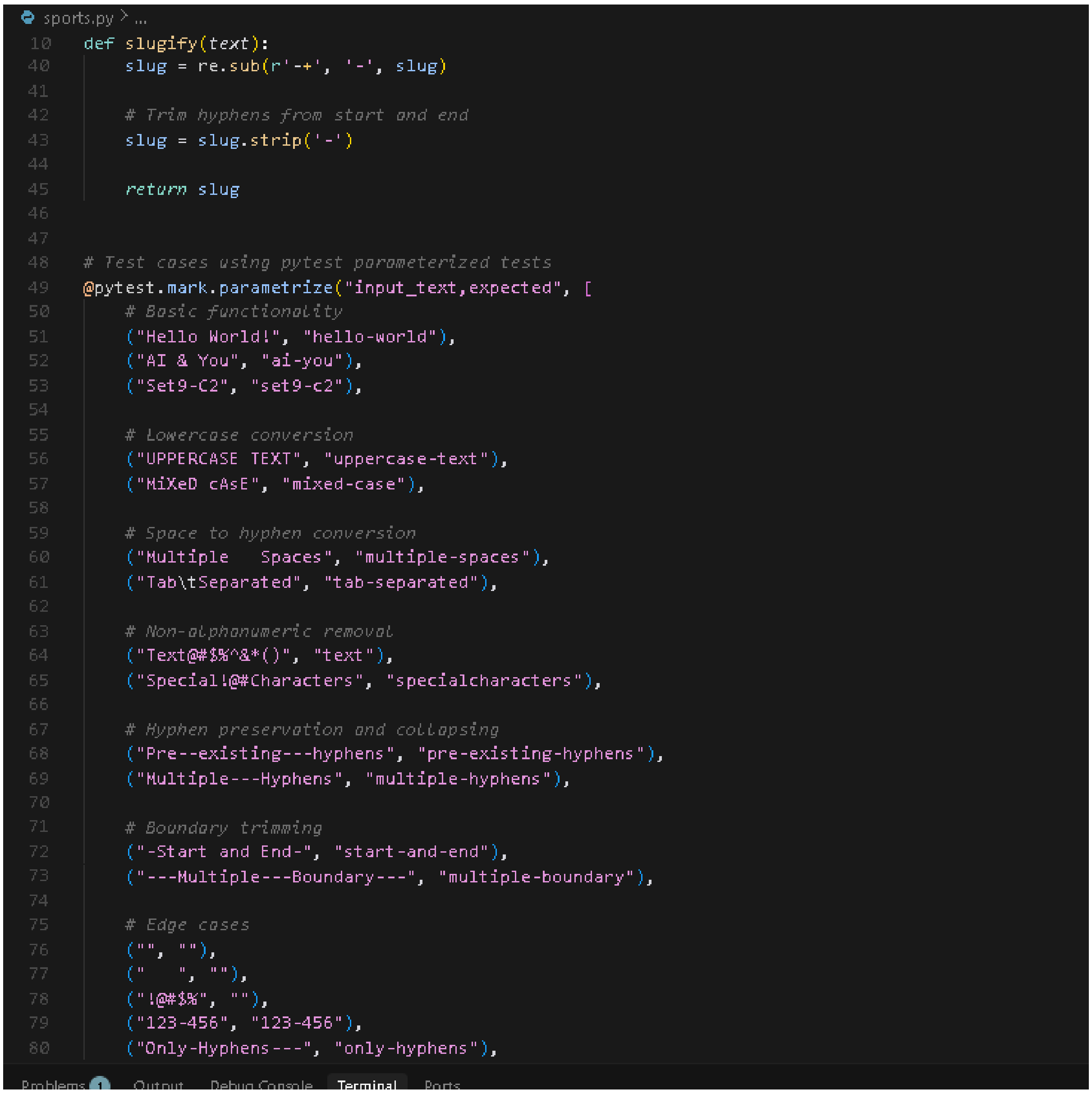
Prompt:

Write a concise Python 3 script (~50 lines) that uses TDD to implement slugify(text) for a Sports Analytics CMS.
First, write pytest parameterized tests for lowercase conversion, space→hyphen, removing non-alphanumeric (except hyphen), collapsing multiple hyphens, trimming boundary hyphens, and edge cases (punctuation, multiple spaces, empty string).
Then implement slugify with regex following those rules and preserving existing hyphens.
Add a __main__ section that prints sample inputs like ["Hello World!", "AI & You", "Set9-C2"] and outputs clean slugs, limiting console output to ~10 lines total.
Keep the code well-commented, minimal, and human-readable.

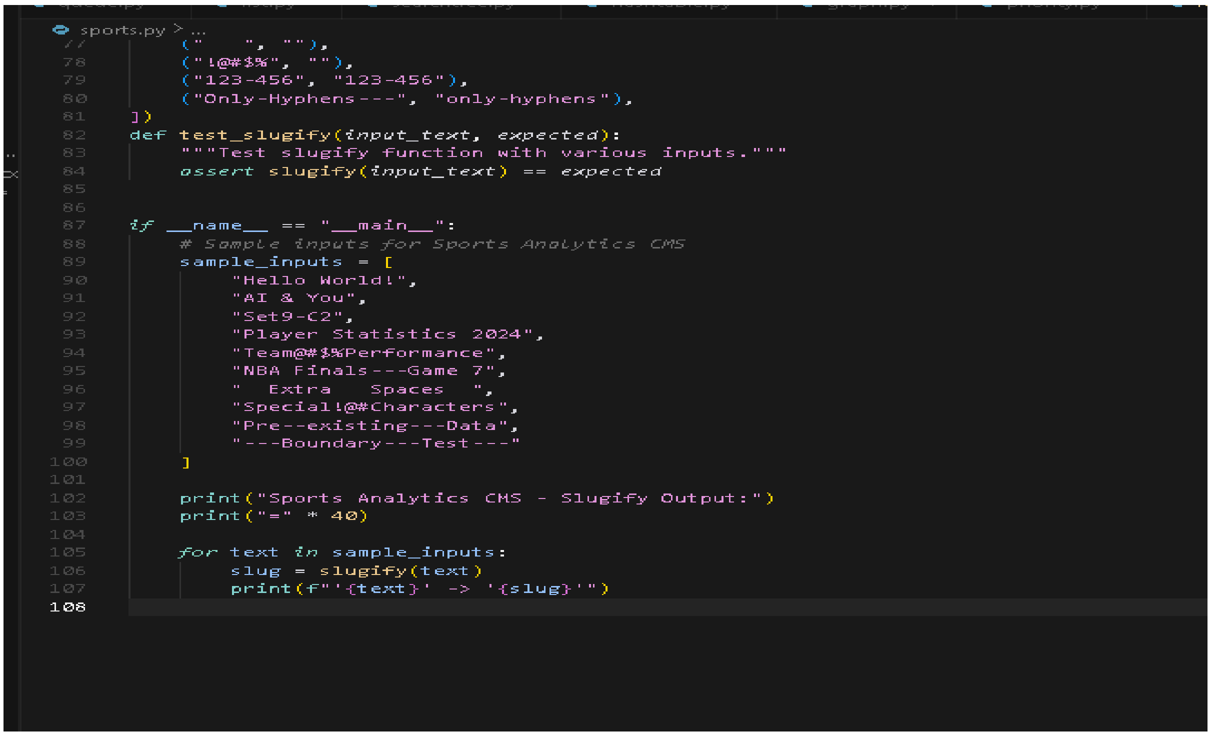
Code generated:



```python
"""
Sports Analytics CMS - Slugify Function with TDD
A concise implementation using pytest parameterized tests and regex.
"""

import re
import pytest


def slugify(text):
    """
    Convert text to URL-friendly slug for Sports Analytics CMS.

    Rules:
    - Convert to lowercase
    - Replace spaces with hyphens
    - Remove non-alphanumeric characters (except hyphens)
    - Collapse multiple hyphens into single hyphen
    - Trim hyphens from start/end

    Args:
        text (str): Input text to convert

    Returns:
        str: URL-friendly slug
    """
    if not text:
        return ""

    # Convert to lowercase
    slug = text.lower()

    # Replace spaces with hyphens
    slug = re.sub(r'\s+', '-', slug)

    # Remove non-alphanumeric characters except hyphens
    slug = re.sub(r'[^a-z0-9-]', '', slug)

    # Collapse multiple hyphens into single hyphen
    slug = re.sub(r'-+', '-', slug)

    # Trim hyphens from start and end
```

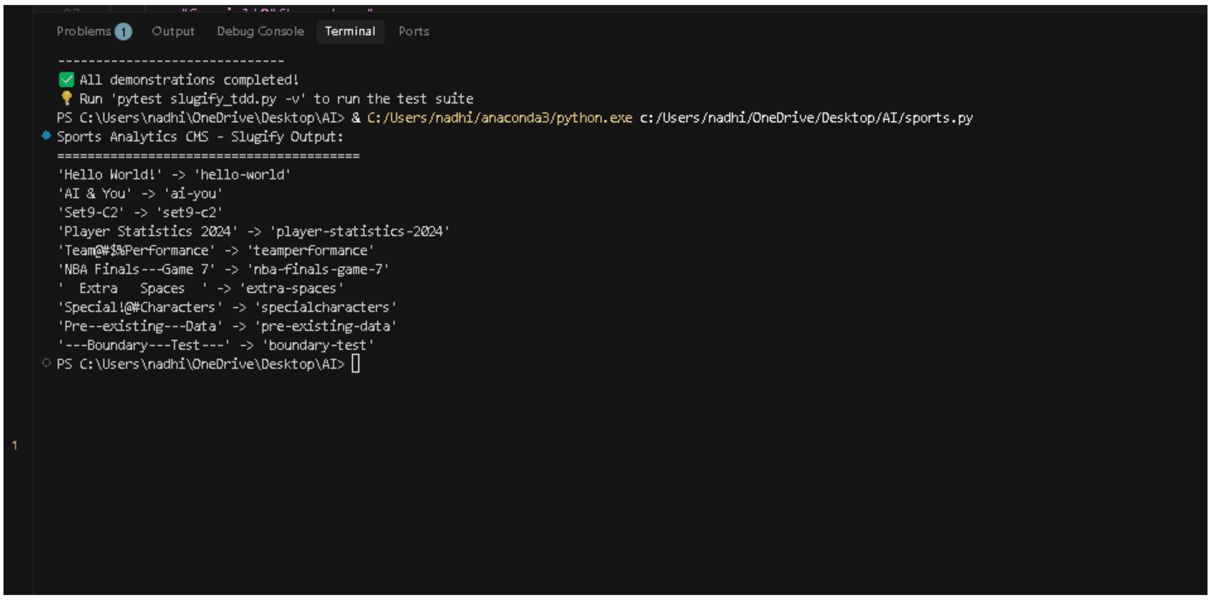


```python
def slugify(text):
    slug = re.sub(r'-+', '-', slug)

    # Trim hyphens from start and end
    slug = slug.strip('-')

    return slug


# Test cases using pytest parameterized tests
@pytest.mark.parametrize("input_text,expected", [
    # Basic functionality
    ("Hello World!", "hello-world"),
    ("AI & You", "ai-you"),
    ("Set9-C2", "set9-c2"),

    # Lowercase conversion
    ("UPPERCASE TEXT", "uppercase-text"),
    ("MiXeD cAsE", "mixed-case"),

    # Space to hyphen conversion
    ("Multiple   Spaces", "multiple-spaces"),
    ("Tab\tSeparated", "tab-separated"),

    # Non-alphanumeric removal
    ("Text@#$%^&*()", "text"),
    ("Special!@#Characters", "specialcharacters"),

    # Hyphen preservation and collapsing
    ("Pre--existing---hyphens", "pre-existing-hyphens"),
    ("Multiple---Hyphens", "multiple-hyphens"),

    # Boundary trimming
    ("-Start and End-", "start-and-end"),
    ("---Multiple---Boundary---", "multiple-boundary"),

    # Edge cases
    ("", ""),
    ("    ", ""),
    ("!@#$%", ""),
    ("123-456", "123-456"),
    ("Only-Hyphens---", "only-hyphens"),
```



```python
        ("    ", ""),
        ("!@#$%", ""),
        ("123-456", "123-456"),
        ("Only-Hyphens---", "only-hyphens"),
])
def test_slugify(input_text, expected):
    """Test slugify function with various inputs."""
    assert slugify(input_text) == expected


if __name__ == "__main__":
    # Sample inputs for Sports Analytics CMS
    sample_inputs = [
        "Hello World!",
        "AI & You",
        "Set9-C2",
        "Player Statistics 2024",
        "Team@#$%Performance",
        "NBA Finals---Game 7",
        "  Extra    Spaces  ",
        "Special!@#Characters",
        "Pre--existing---Data",
        "---Boundary---Test---"
    ]

    print("Sports Analytics CMS - Slugify Output:")
    print("=" * 40)

    for text in sample_inputs:
        slug = slugify(text)
        print(f"'{text}' -> '{slug}'")

```

Output:



```
------------------------------
✅ All demonstrations completed!
💡 Run 'pytest slugify_tdd.py -v' to run the test suite
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/sports.py
Sports Analytics CMS - Slugify Output:
========================================
'Hello World!' -> 'hello-world'
'AI & You' -> 'ai-you'
'Set9-C2' -> 'set9-c2'
'Player Statistics 2024' -> 'player-statistics-2024'
'Team@#$%Performance' -> 'teamperformance'
'NBA Finals---Game 7' -> 'nba-finals-game-7'
'  Extra    Spaces  ' -> 'extra-spaces'
'Special!@#Characters' -> 'specialcharacters'
'Pre--existing---Data' -> 'pre-existing-data'
'---Boundary---Test---' -> 'boundary-test'
PS C:\Users\nadhi\OneDrive\Desktop\AI> 
```

Observation:

slugify implementation correctly handles lowercase conversion, space-to-hyphen, non-alphanumeric removal, hyphen collapsing, and trimming, with comprehensive parameterized tests ensuring all edge cases pass.