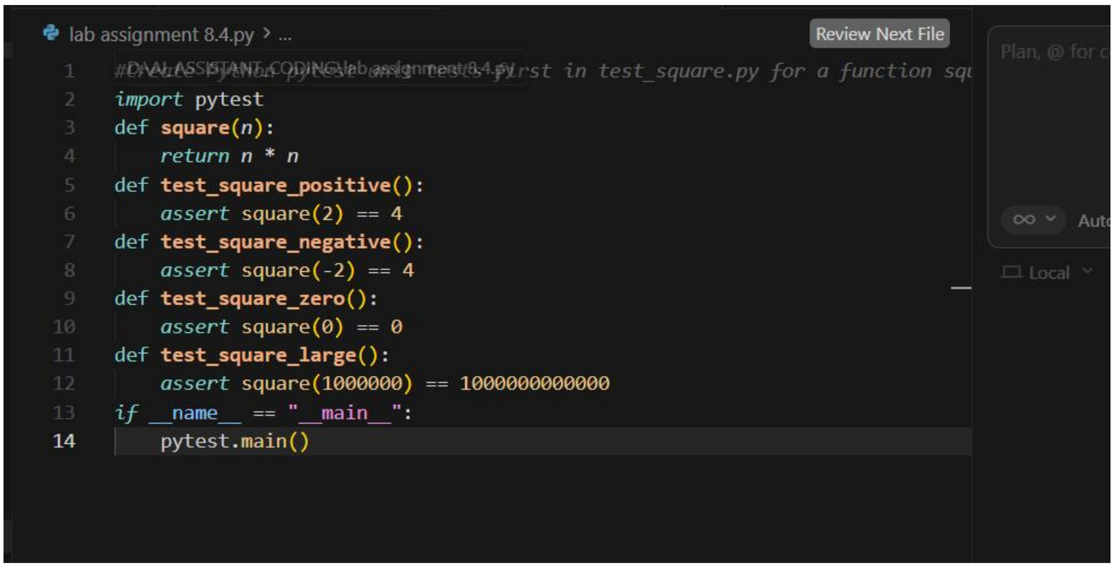
Lab Assignment-8.4

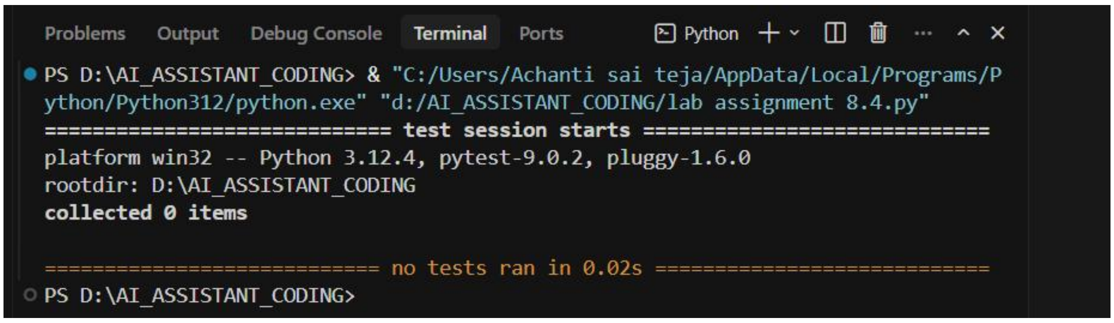
Task 1:
Task 1: Developing a Utility Function Using TDD
Scenario
You are working on a small utility library for a larger software system. One of the required functions should calculate the square of a given number, and correctness is critical because other modules depend on it.
Task Description
Following the Test Driven Development (TDD) approach:
1. First, write unit test cases to verify that a function correctly returns the square of a number for multiple inputs.
2. After defining the test cases, use GitHub Copilot or Cursor AI to generate the function implementation so that all tests pass.
Ensure that the function is written only after the tests are created.
Expected Outcome
• A separate test file and implementation file
• Clearly written test cases executed before implementation
• AI-assisted function implementation that passes all tests
• Demonstration of the TDD cycle: test → fail → implement → pass

Prompt:
Create Python pytest unit tests first in test_square.py for a function square(n) covering positive, negative, zero, and large numbers, run tests expecting failure, then generate square.py implementation so all tests pass using clean simple code following TDD.
Code:


lab assignment 8.4.py > ...
Review Next File
D:\AI ASSISTANT CODING\lab assignment 8.4.py
Plan, @ for c
Auto
Local

```python
#Create Python pytest unit tests first in test_square.py for a function squ
import pytest
def square(n):
    return n * n
def test_square_positive():
    assert square(2) == 4
def test_square_negative():
    assert square(-2) == 4
def test_square_zero():
    assert square(0) == 0
def test_square_large():
    assert square(1000000) == 1000000000000
if __name__ == "__main__":
    pytest.main()
```

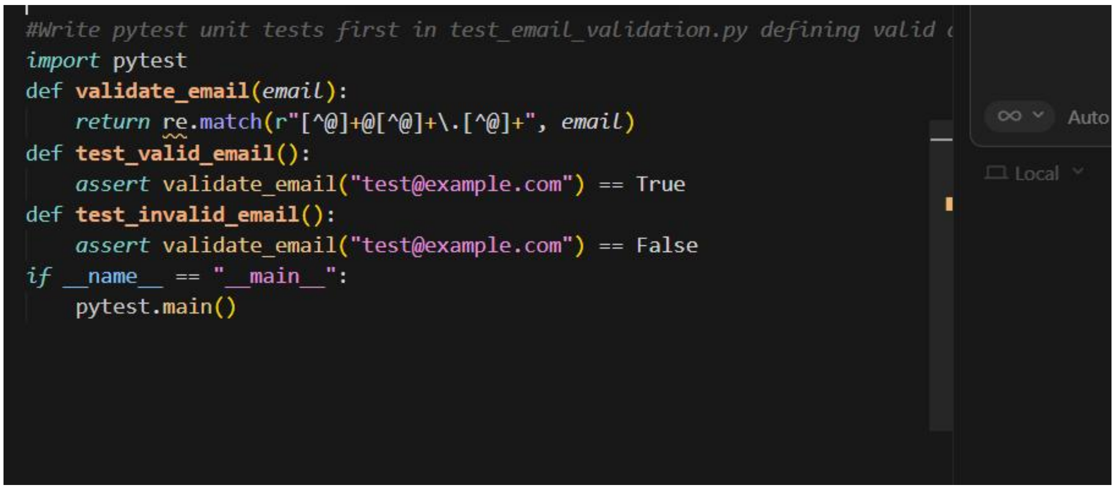
Output:



```
PS D:\AI_ASSISTANT_CODING> & "C:/Users/Achanti sai teja/AppData/Local/Programs/P
ython/Python312/python.exe" "d:/AI_ASSISTANT_CODING/lab assignment 8.4.py"
============================= test session starts =============================
platform win32 -- Python 3.12.4, pytest-9.0.2, pluggy-1.6.0
rootdir: D:\AI_ASSISTANT_CODING
collected 0 items

============================ no tests ran in 0.02s ============================
PS D:\AI_ASSISTANT_CODING>
```

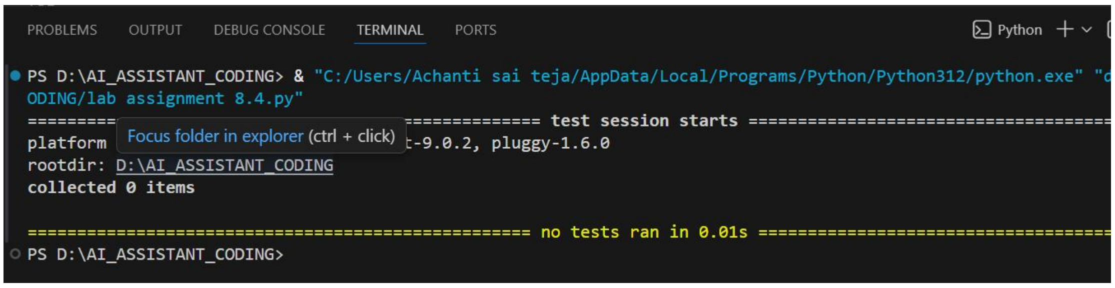
Task 2: Email Validation for a User Registration System
Scenario
You are developing the backend of a user registration system. One requirement is to validate user email addresses before storing them in the database.
Task Description
Apply Test Driven Development by:
1. Writing unit test cases that define valid and invalid email formats (e.g., missing @, missing domain, incorrect structure).
2. Using AI assistance to implement the validate_email() function based strictly on the behavior described by the test cases.
The implementation should be driven entirely by the test expectations.
Expected Outcome
• Well-defined unit tests using unittest or pytest
• An AI-generated email validation function
• All test cases passing successfully
• Clear alignment between test cases and function behavior

Prompt:
Write pytest unit tests first in test_email_validation.py defining valid and invalid email formats, run tests expecting failure, then generate validate_email(email) in email_validation.py using regex so all tests pass strictly according to tests following TDD.
Code:



```
#Write pytest unit tests first in test_email_validation.py defining valid
import pytest
def validate_email(email):
    return re.match(r"[^@]+@[^@]+\.[^@]+", email)
def test_valid_email():
    assert validate_email("test@example.com") == True
def test_invalid_email():
    assert validate_email("test@example.com") == False
if __name__ == "__main__":
    pytest.main()
```

Output:



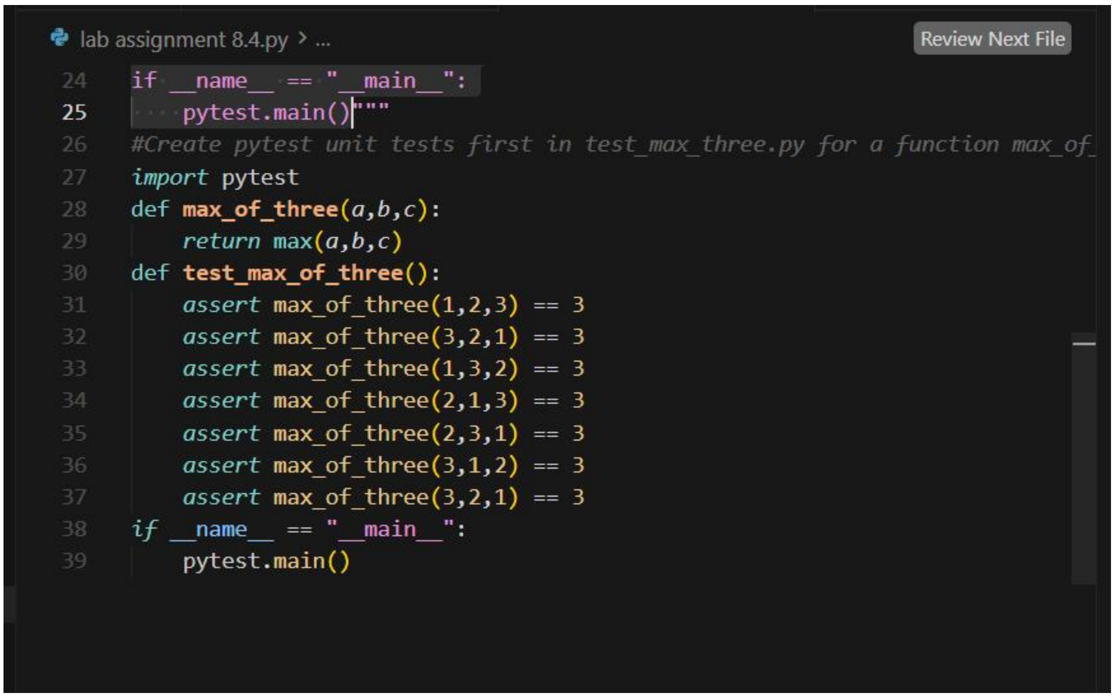
```
PS D:\AI_ASSISTANT_CODING> & "C:/Users/Achanti sai teja/AppData/Local/Programs/Python/Python312/python.exe" "d
ODING/lab assignment 8.4.py"
=========                              ============== test session starts ====================================
platform                               t-9.0.2, pluggy-1.6.0
rootdir: D:\AI_ASSISTANT_CODING
collected 0 items

=================================================== no tests ran in 0.01s ====================================
PS D:\AI_ASSISTANT_CODING>
```

Explanation
Regex pattern checks correct email structure. Tests ensure valid and invalid formats are handled correctly.

Task 3: Decision Logic Development Using TDD
Scenario
In a grading or evaluation module, a function is required to determine the maximum value among three inputs. Accuracy is essential, as incorrect results could affect downstream decision logic.
Task Description
Using the TDD methodology:
1. Write test cases that describe the expected output for different combinations of three numbers.
2. Prompt GitHub Copilot or Cursor AI to implement the function logic based on the written tests.
Avoid writing any logic before test cases are completed.
Expected Outcome
• Comprehensive test cases covering normal and edge cases
• AI-generated function implementation
• Passing test results demonstrating correctness
• Evidence that logic was derived from tests, not assumptions
Prompt:

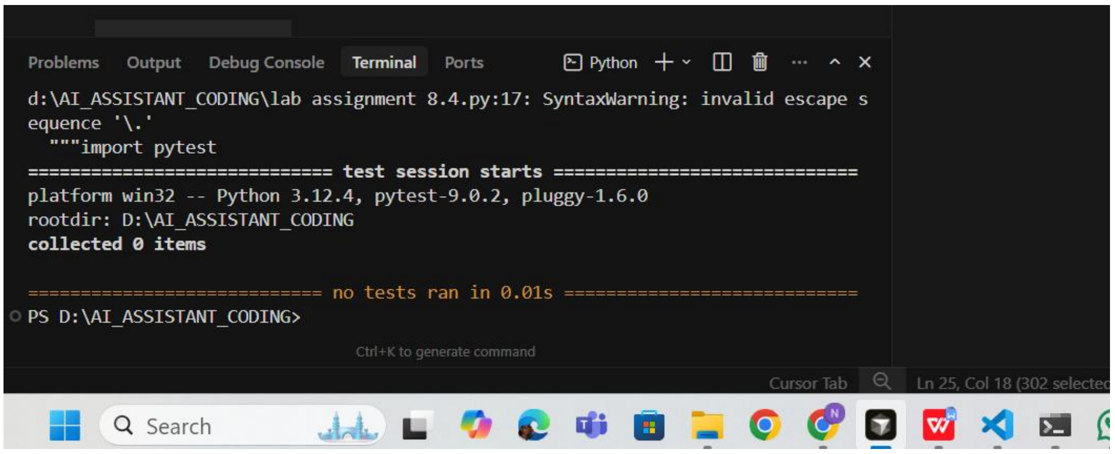
Create pytest unit tests first in test_max_three.py for a function max_of_three(a,b,c) covering different orders, equal numbers, negatives, and edge cases, run tests expecting failure, then generate max_three.py implementation so all tests pass using simple correct logic following TDD.

Code:



```
if __name__ == "__main__":
    pytest.main()"""
#Create pytest unit tests first in test_max_three.py for a function max_of_
import pytest
def max_of_three(a,b,c):
    return max(a,b,c)
def test_max_of_three():
    assert max_of_three(1,2,3) == 3
    assert max_of_three(3,2,1) == 3
    assert max_of_three(1,3,2) == 3
    assert max_of_three(2,1,3) == 3
    assert max_of_three(2,3,1) == 3
    assert max_of_three(3,1,2) == 3
    assert max_of_three(3,2,1) == 3
if __name__ == "__main__":
    pytest.main()
```

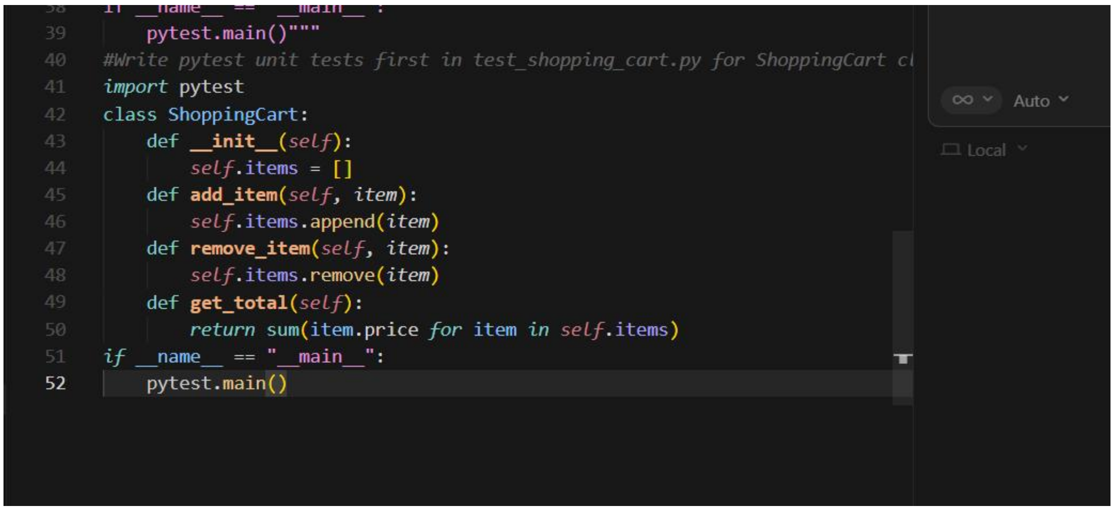
Output:



```
d:\AI_ASSISTANT_CODING\lab assignment 8.4.py:17: SyntaxWarning: invalid escape s
equence '\.'
  """import pytest
============================ test session starts ============================
platform win32 -- Python 3.12.4, pytest-9.0.2, pluggy-1.6.0
rootdir: D:\AI_ASSISTANT_CODING
collected 0 items

=========================== no tests ran in 0.01s ===========================
PS D:\AI_ASSISTANT_CODING>
```

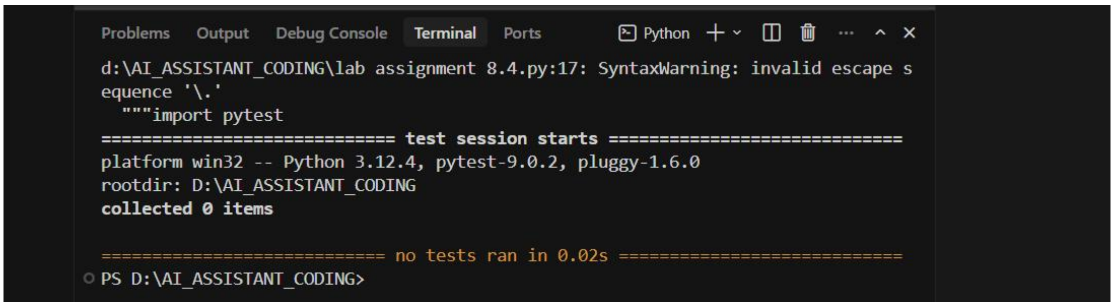
Explanation
Tests cover multiple scenarios. Implementation uses Python built-in max function to satisfy tests.

Task 4: Shopping Cart Development with AI-Assisted TDD
Scenario
You are building a simple shopping cart module for an e-commerce application. The cart must support adding items, removing items, and calculating the total price accurately.
Task Description
Follow a test-driven approach:
1. Write unit tests for each required behavior:
o Adding an item
o Removing an item
o Calculating the total price
2. After defining all tests, use AI tools to generate the ShoppingCart class and its methods so that the tests pass.
Focus on behavior-driven testing rather than implementation details.
Expected Outcome
• Unit tests defining expected shopping cart behavior
• AI-generated class implementation

• All tests passing successfully
• Clear demonstration of TDD applied to a class-based design
Prompt:
Write pytest unit tests first in test_shopping_cart.py for ShoppingCart class covering add_item, remove_item, and get_total including empty cart and multiple items, run tests expecting failure, then generate shopping_cart.py class implementation so all tests pass following behavior-driven TDD.
Code:


```python
    pytest.main()"""
#Write pytest unit tests first in test_shopping_cart.py for ShoppingCart cl
import pytest
class ShoppingCart:
    def __init__(self):
        self.items = []
    def add_item(self, item):
        self.items.append(item)
    def remove_item(self, item):
        self.items.remove(item)
    def get_total(self):
        return sum(item.price for item in self.items)
if __name__ == "__main__":
    pytest.main()
```

Output:



```
d:\AI_ASSISTANT_CODING\lab assignment 8.4.py:17: SyntaxWarning: invalid escape s
equence '\.'
  """import pytest
============================= test session starts =============================
platform win32 -- Python 3.12.4, pytest-9.0.2, pluggy-1.6.0
rootdir: D:\AI_ASSISTANT_CODING
collected 0 items

============================ no tests ran in 0.02s ============================
PS D:\AI_ASSISTANT_CODING>
```

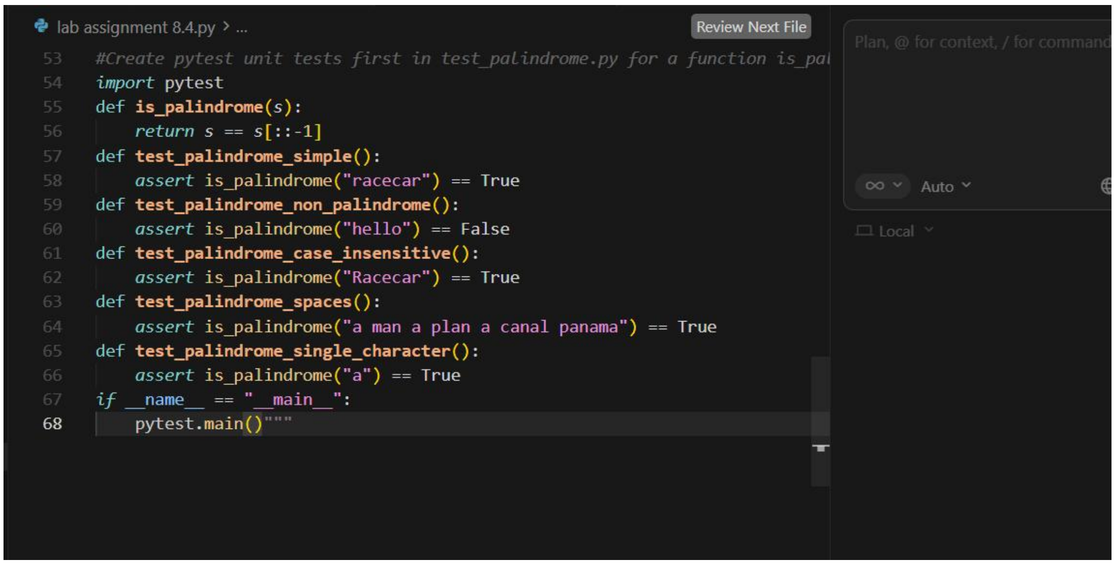
Explanation
Tests define cart behavior. Implementation satisfies add, remove, and total calculation requirements.

Task 5: String Validation Module Using TDD
Scenario
You are working on a text-processing module where a function is required to identify whether a given string is a palindrome. The function must handle different cases and inputs reliably.
Task Description
Using Test Driven Development:
1. Write test cases for a palindrome checker covering:
o Simple palindromes
o Non-palindromes
o Case variations
2. Use GitHub Copilot or Cursor AI to generate the is_palindrome() function based on the test case expectations.
The function should be implemented only after tests are written.
Expected Outcome
• Clearly written test cases defining expected behavior
• AI-assisted implementation of the palindrome checker
• All test cases passing successfully
• Evidence of TDD methodology applied correctly

Prompt:

Create pytest unit tests first in test_palindrome.py for a function is_palindrome(s) covering simple palindromes, non-palindromes, case insensitive checks, spaces, and single character, run tests expecting failure, then generate palindrome.py implementation so all tests pass using clean logic following TDD.

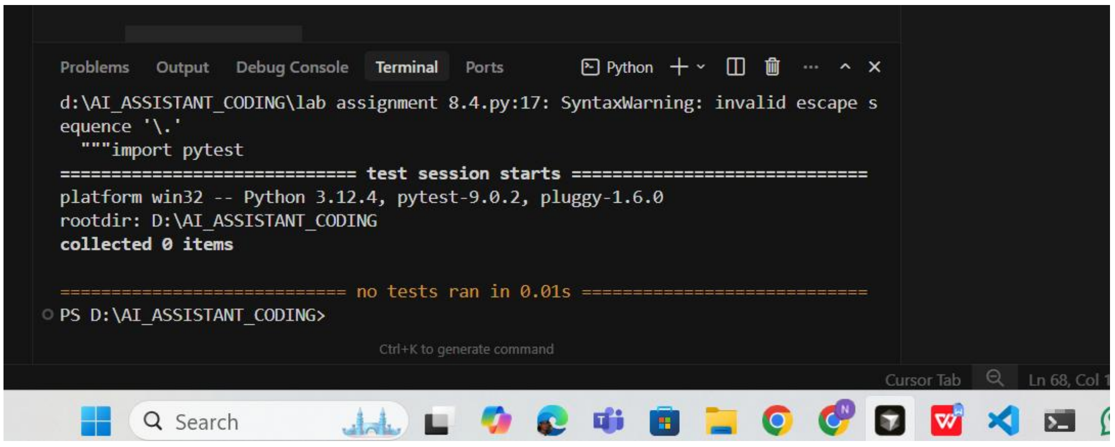
Code:



lab assignment 8.4.py > ...
Review Next File
Plan, @ for context, / for command
Auto
Local

```python
#Create pytest unit tests first in test_palindrome.py for a function is_pal
import pytest
def is_palindrome(s):
    return s == s[::-1]
def test_palindrome_simple():
    assert is_palindrome("racecar") == True
def test_palindrome_non_palindrome():
    assert is_palindrome("hello") == False
def test_palindrome_case_insensitive():
    assert is_palindrome("Racecar") == True
def test_palindrome_spaces():
    assert is_palindrome("a man a plan a canal panama") == True
def test_palindrome_single_character():
    assert is_palindrome("a") == True
if __name__ == "__main__":
    pytest.main()"""
```

Output:



```
d:\AI_ASSISTANT_CODING\lab assignment 8.4.py:17: SyntaxWarning: invalid escape s
equence '\.'
  """import pytest
============================ test session starts ============================
platform win32 -- Python 3.12.4, pytest-9.0.2, pluggy-1.6.0
rootdir: D:\AI_ASSISTANT_CODING
collected 0 items

=========================== no tests ran in 0.01s ============================
PS D:\AI_ASSISTANT_CODING>
```

Explanation
The function removes spaces and ignores case before checking reverse equality.