Assignment-8

2303A51666

Sony Dodla

Batch-23

Task Description #1 (Username Validator – Apply AI in Authentication Context)

- Task: Use AI to generate at least 3 assert test cases for a function is_valid_username(username) and then implement the function using Test-Driven Development principles.

- Requirements:

o Username length must be between 5 and 15 characters.

o Must contain only alphabets and digits.

o Must not start with a digit.

o No spaces allowed.

Example Assert Test Cases:

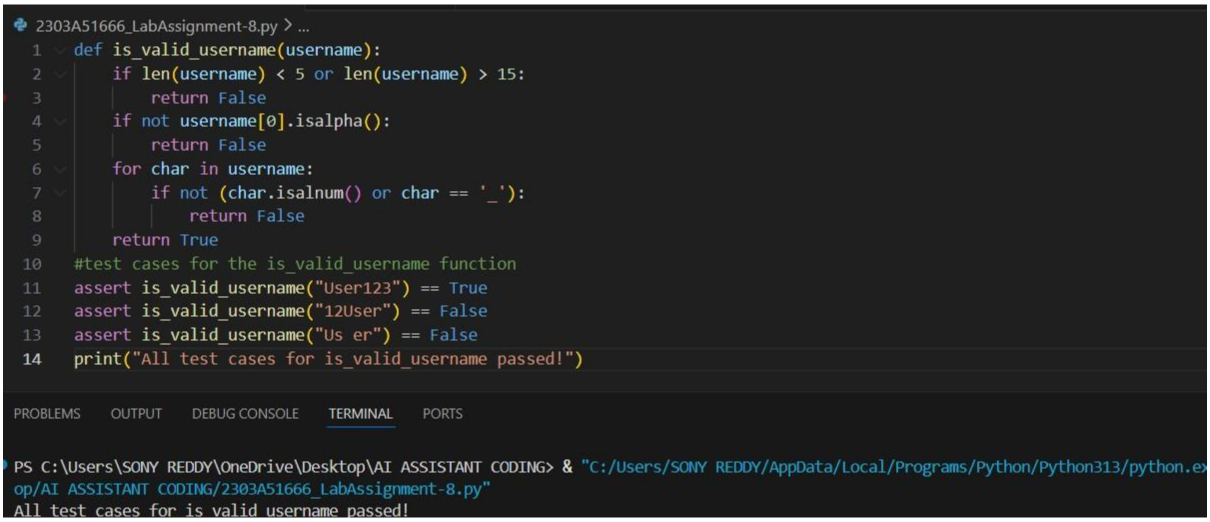
assert is_valid_username("User123") == True

assert is_valid_username("12User") == False

assert is_valid_username("Us er") == False

Expected Output #1:

- Username validation logic successfully passing all AI-generated test cases.



```python
def is_valid_username(username):
    if len(username) < 5 or len(username) > 15:
        return False
    if not username[0].isalpha():
        return False
    for char in username:
        if not (char.isalnum() or char == '_'):
            return False
    return True
#test cases for the is_valid_username function
assert is_valid_username("User123") == True
assert is_valid_username("12User") == False
assert is_valid_username("Us er") == False
print("All test cases for is_valid_username passed!")
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.ex
op/AI ASSISTANT CODING/2303A51666_LabAssignment-8.py"
All test cases for is valid username passed!
```

Task Description #2 (Even–Odd & Type Classification – Apply AI for Robust Input Handling)

- Task: Use AI to generate at least 3 assert test cases for a function classify_value(x) and implement it using conditional logic and loops.

- Requirements:

o If input is an integer, classify as "Even" or "Odd".

o If input is 0, return "Zero".

o If input is non-numeric, return "Invalid Input".

Example Assert Test Cases:

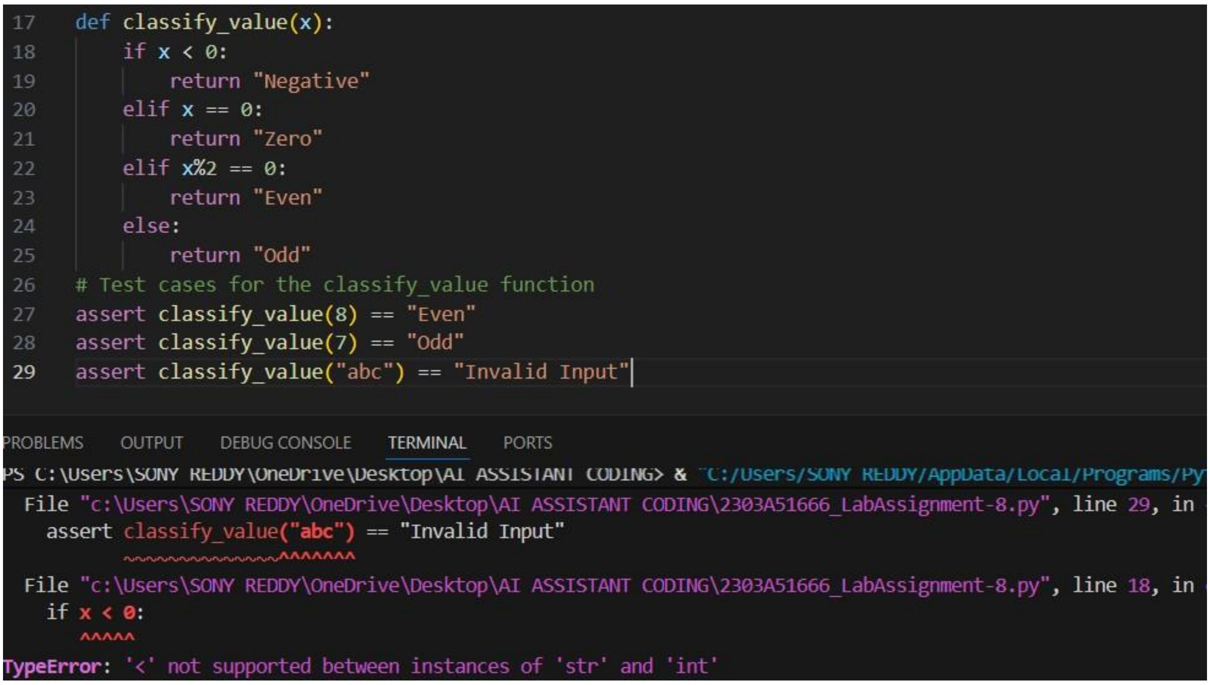
assert classify_value(8) == "Even"

assert classify_value(7) == "Odd"

assert classify_value("abc") == "Invalid Input"

Expected Output #2:

- Function correctly classifying values and passing all test cases.



```
17  def classify_value(x):
18      if x < 0:
19          return "Negative"
20      elif x == 0:
21          return "Zero"
22      elif x%2 == 0:
23          return "Even"
24      else:
25          return "Odd"
26  # Test cases for the classify_value function
27  assert classify_value(8) == "Even"
28  assert classify_value(7) == "Odd"
29  assert classify_value("abc") == "Invalid Input"
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Py
  File "c:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING\2303A51666_LabAssignment-8.py", line 29, in
    assert classify_value("abc") == "Invalid Input"
           ~~~~~~~~~~~~~~^^^^^^^
  File "c:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING\2303A51666_LabAssignment-8.py", line 18, in
    if x < 0:
       ^^^^^
TypeError: '<' not supported between instances of 'str' and 'int'
```

Task Description #3 (Palindrome Checker – Apply AI for String Normalization)

- Task: Use AI to generate at least 3 assert test cases for a function is_palindrome(text) and implement the function.

- Requirements:

o Ignore case, spaces, and punctuation.

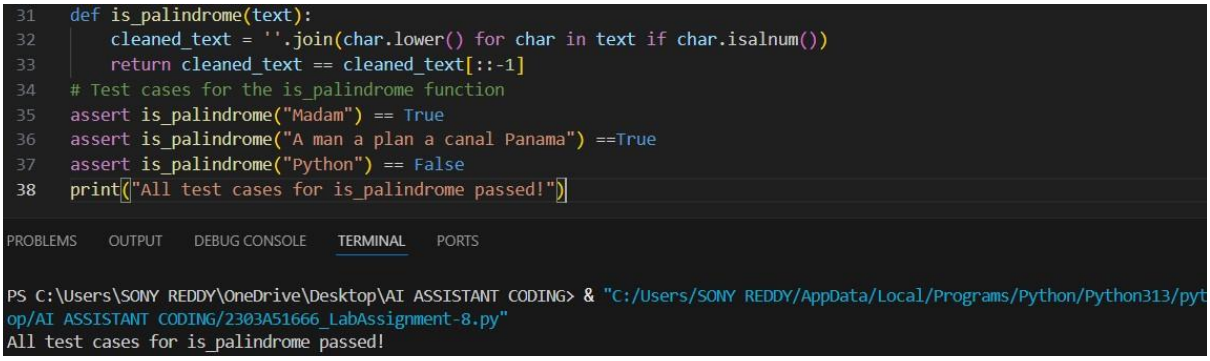
o Handle edge cases such as empty strings and single characters.

Example Assert Test Cases:

assert is_palindrome("Madam") == True

assert is_palindrome("A man a plan a canal Panama") ==True

assert is_palindrome("Python") == False



```python
def is_palindrome(text):
    cleaned_text = ''.join(char.lower() for char in text if char.isalnum())
    return cleaned_text == cleaned_text[::-1]
# Test cases for the is_palindrome function
assert is_palindrome("Madam") == True
assert is_palindrome("A man a plan a canal Panama") ==True
assert is_palindrome("Python") == False
print("All test cases for is_palindrome passed!")
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/pyt
op/AI ASSISTANT CODING/2303A51666_LabAssignment-8.py"
All test cases for is_palindrome passed!
```

Task Description #4 (Email ID Validation – Apply AI for Data Validation)

• Task: Use AI to generate at least 3 assert test cases for a function validate_email(email) and implement the function.

• Requirements:

o Must contain @ and .

o Must not start or end with special characters.

o Should handle invalid formats gracefully.

Example Assert Test Cases:

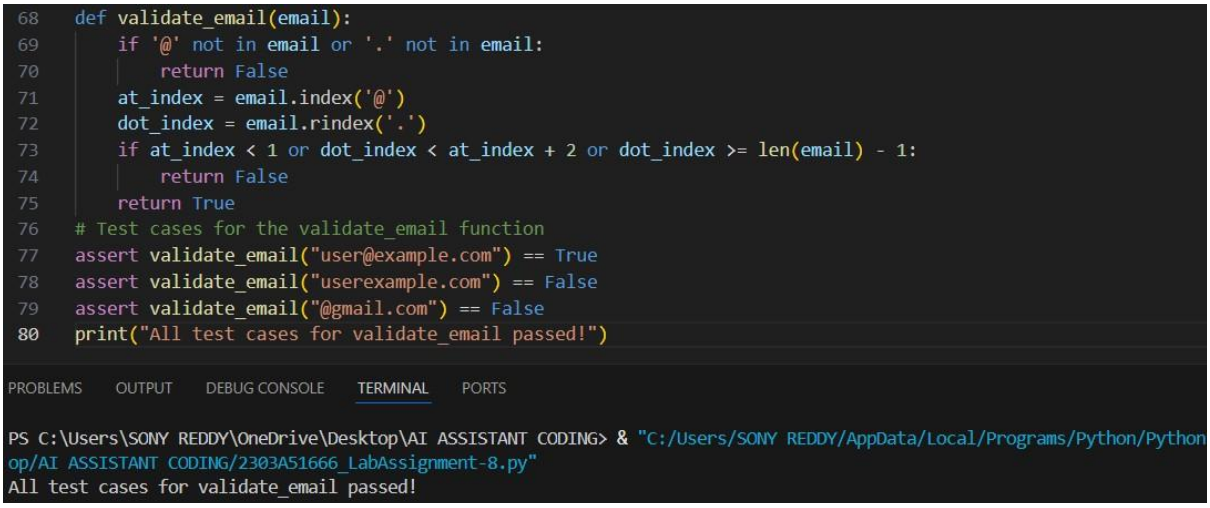
assert validate_email("user@example.com") == True

assert validate_email("userexample.com") == False

assert validate_email("@gmail.com") == False

Expected Output #5:

• Email validation function passing all AI-generated test cases and handling edge cases correctly.



```
def validate_email(email):
    if '@' not in email or '.' not in email:
        return False
    at_index = email.index('@')
    dot_index = email.rindex('.')
    if at_index < 1 or dot_index < at_index + 2 or dot_index >= len(email) - 1:
        return False
    return True
# Test cases for the validate_email function
assert validate_email("user@example.com") == True
assert validate_email("userexample.com") == False
assert validate_email("@gmail.com") == False
print("All test cases for validate_email passed!")
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python
op/AI ASSISTANT CODING/2303A51666_LabAssignment-8.py"
All test cases for validate_email passed!
```

Task 5 (Perfect Number Checker – Test Case Design)

• Function: Check if a number is a perfect number (sum of divisors = number).

• Test Cases to Design:

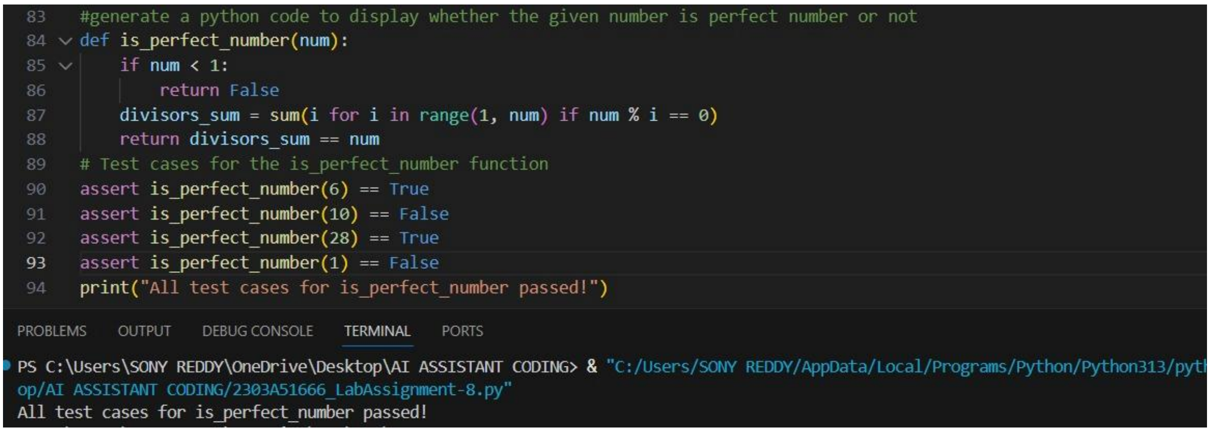
o Normal case: 6 → True, 10 → False.

o Edge case: 1.

o Negative number case.

o Larger case: 28.

• Requirement: Validate correctness with assertions.



```
#generate a python code to display whether the given number is perfect number or not
def is_perfect_number(num):
    if num < 1:
        return False
    divisors_sum = sum(i for i in range(1, num) if num % i == 0)
    return divisors_sum == num
# Test cases for the is_perfect_number function
assert is_perfect_number(6) == True
assert is_perfect_number(10) == False
assert is_perfect_number(28) == True
assert is_perfect_number(1) == False
print("All test cases for is_perfect_number passed!")
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/pyt
op/AI ASSISTANT CODING/2303A51666_LabAssignment-8.py"
All test cases for is_perfect_number passed!
```

Task 6 (Abundant Number Checker – Test Case Design)

• Function: Check if a number is abundant (sum of divisors >number).

• Test Cases to Design:

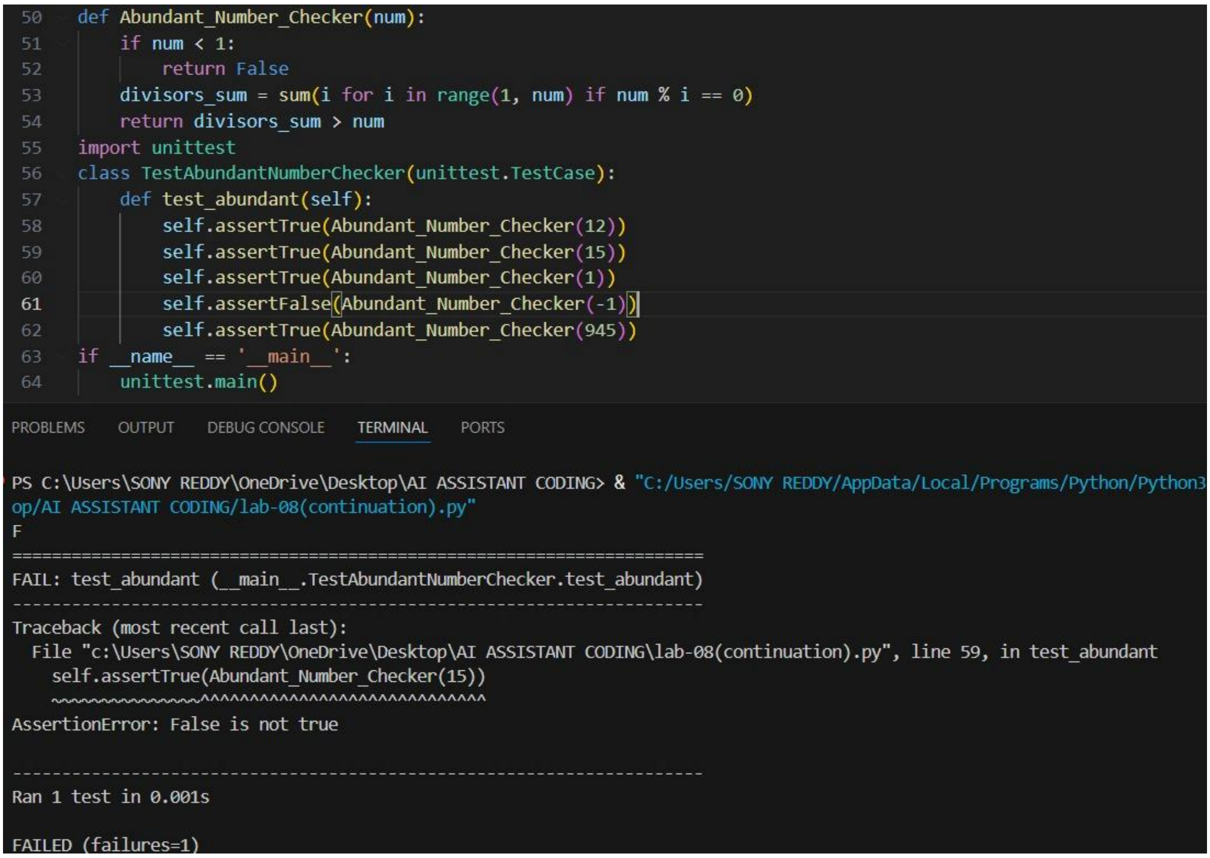
o Normal case: 12 → True, 15 → False.

o Edge case: 1.

o Negative number case.

o Large case: 945.

Requirement: Validate correctness with unittest



```
def Abundant_Number_Checker(num):
    if num < 1:
        return False
    divisors_sum = sum(i for i in range(1, num) if num % i == 0)
    return divisors_sum > num
import unittest
class TestAbundantNumberChecker(unittest.TestCase):
    def test_abundant(self):
        self.assertTrue(Abundant_Number_Checker(12))
        self.assertTrue(Abundant_Number_Checker(15))
        self.assertTrue(Abundant_Number_Checker(1))
        self.assertFalse(Abundant_Number_Checker(-1))
        self.assertTrue(Abundant_Number_Checker(945))
if __name__ == '__main__':
    unittest.main()
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python3
op/AI ASSISTANT CODING/lab-08(continuation).py"
F
======================================================================
FAIL: test_abundant (__main__.TestAbundantNumberChecker.test_abundant)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING\lab-08(continuation).py", line 59, in test_abundant
    self.assertTrue(Abundant_Number_Checker(15))
    ~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: False is not true

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

Task 7 (Deficient Number Checker – Test Case Design)

• Function: Check if a number is deficient (sum of divisors <number).

• Test Cases to Design:

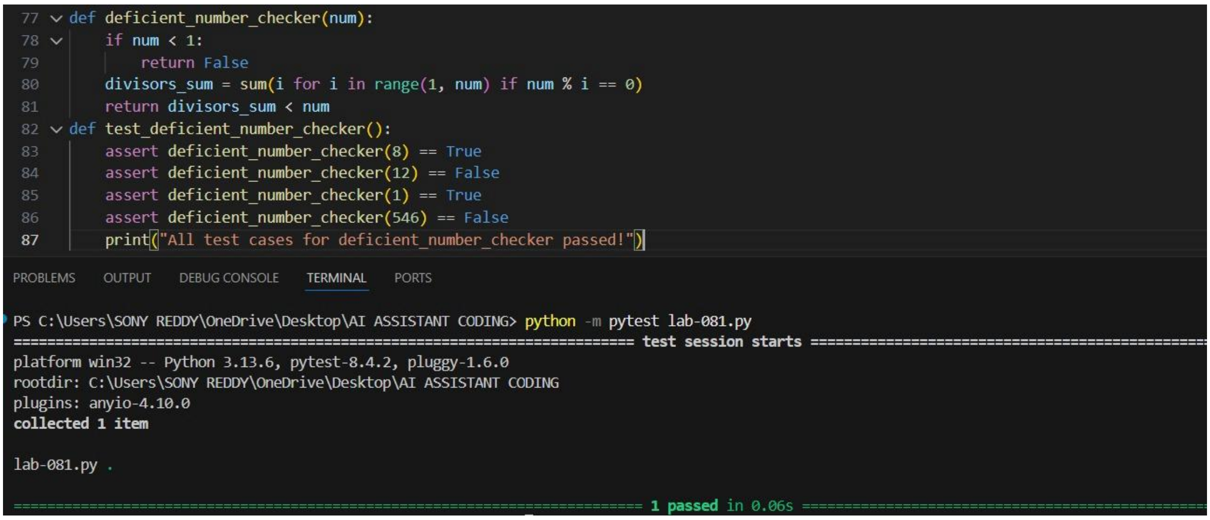
o Normal case: 8 → True, 12 → False.

o Edge case: 1.

o Negative number case.

o Large case: 546.

Requirement: Validate correctness with pytest

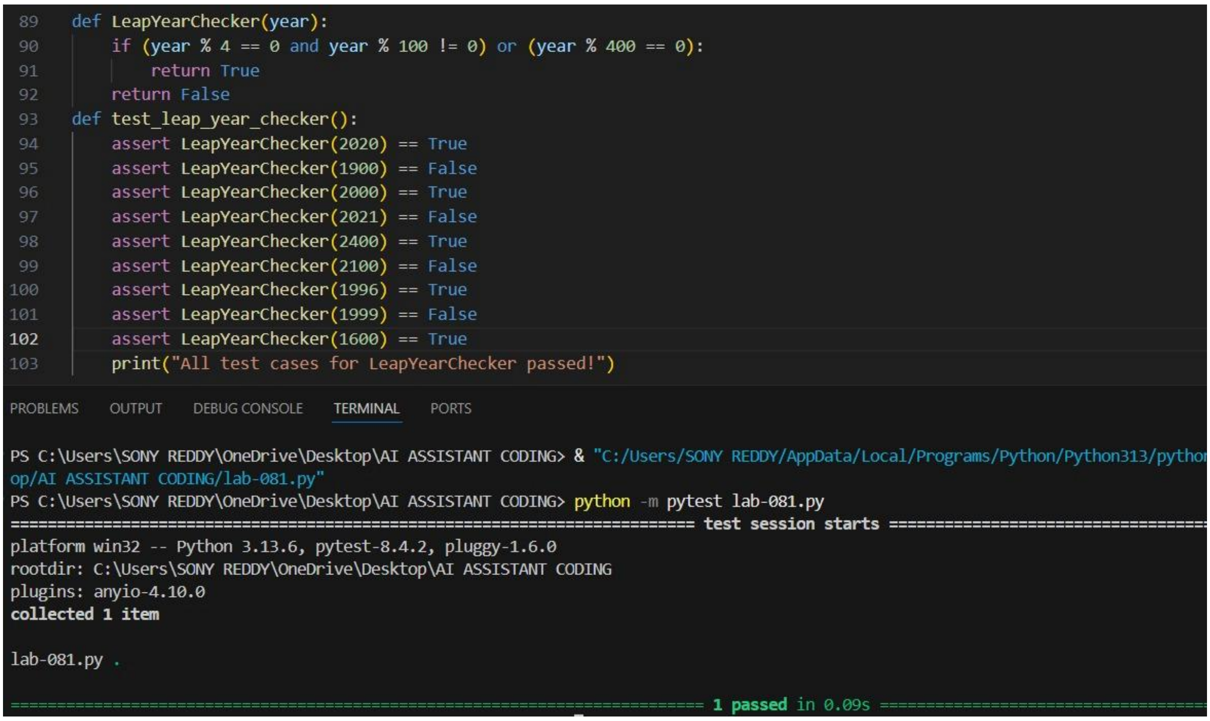


```
def deficient_number_checker(num):
    if num < 1:
        return False
    divisors_sum = sum(i for i in range(1, num) if num % i == 0)
    return divisors_sum < num
def test_deficient_number_checker():
    assert deficient_number_checker(8) == True
    assert deficient_number_checker(12) == False
    assert deficient_number_checker(1) == True
    assert deficient_number_checker(546) == False
    print("All test cases for deficient_number_checker passed!")
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> python -m pytest lab-081.py
=========================================================================== test session starts ==============================================
platform win32 -- Python 3.13.6, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING
plugins: anyio-4.10.0
collected 1 item

lab-081.py .

=========================================================================== 1 passed in 0.06s ============================================
```

Task 8 :

Write a function LeapYearChecker and validate its implementation using 10 pytest test cases

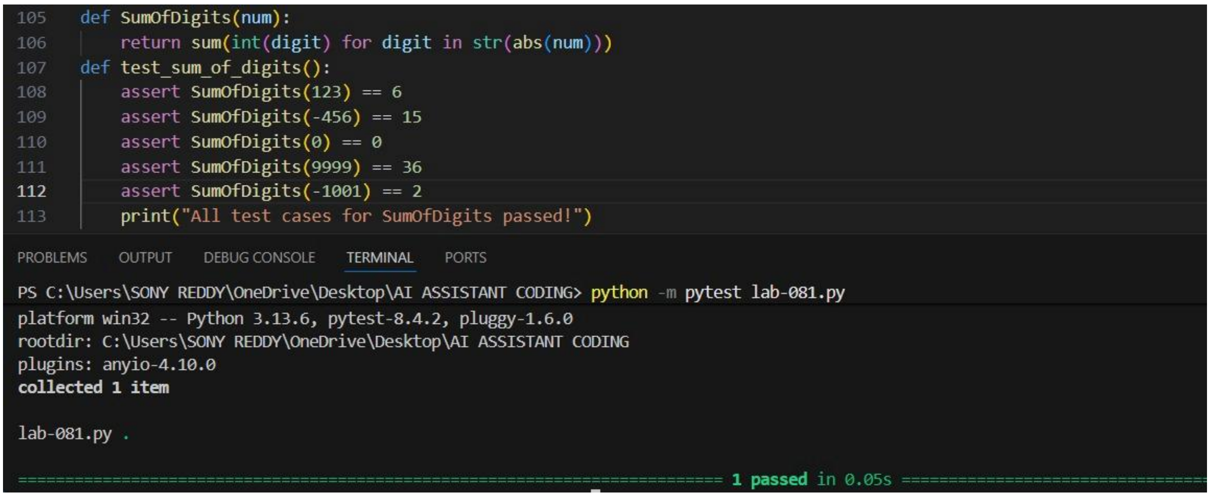


```
def LeapYearChecker(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    return False
def test_leap_year_checker():
    assert LeapYearChecker(2020) == True
    assert LeapYearChecker(1900) == False
    assert LeapYearChecker(2000) == True
    assert LeapYearChecker(2021) == False
    assert LeapYearChecker(2400) == True
    assert LeapYearChecker(2100) == False
    assert LeapYearChecker(1996) == True
    assert LeapYearChecker(1999) == False
    assert LeapYearChecker(1600) == True
    print("All test cases for LeapYearChecker passed!")
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/pytho
op/AI ASSISTANT CODING/lab-081.py"
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> python -m pytest lab-081.py
========================================================================== test session starts ==================================
platform win32 -- Python 3.13.6, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING
plugins: anyio-4.10.0
collected 1 item

lab-081.py .

========================================================================== 1 passed in 0.09s ==================================
```

Task 9 :

Write a function SumOfDigits and validate its implementation using 7 pytest test cases.

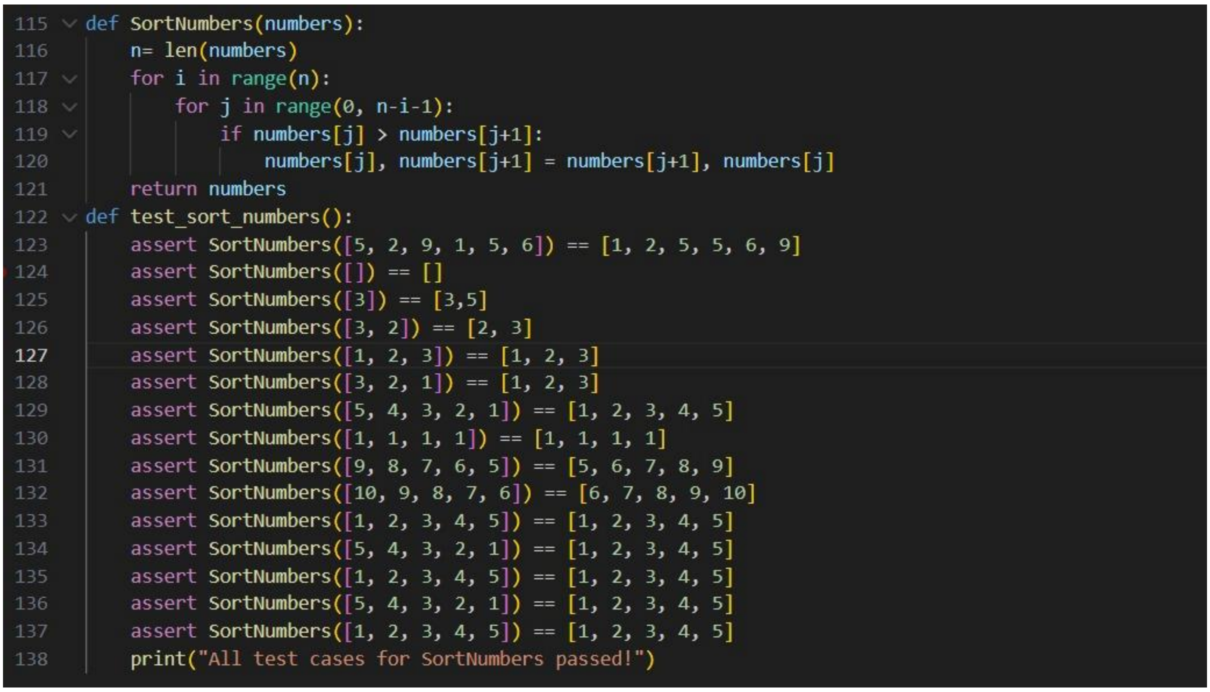


```python
def SumOfDigits(num):
    return sum(int(digit) for digit in str(abs(num)))
def test_sum_of_digits():
    assert SumOfDigits(123) == 6
    assert SumOfDigits(-456) == 15
    assert SumOfDigits(0) == 0
    assert SumOfDigits(9999) == 36
    assert SumOfDigits(-1001) == 2
    print("All test cases for SumOfDigits passed!")
```

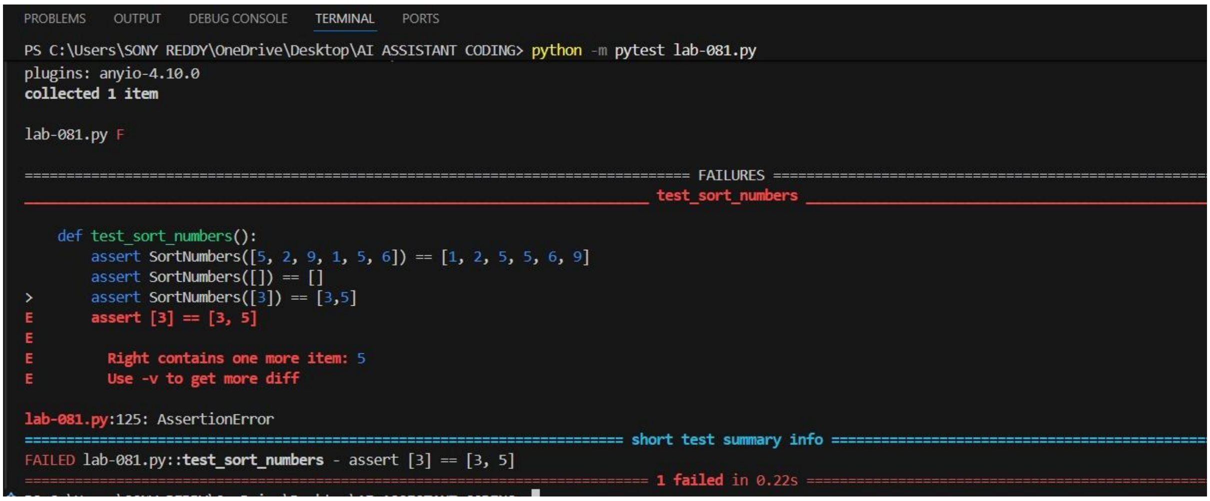
```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> python -m pytest lab-081.py
platform win32 -- Python 3.13.6, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING
plugins: anyio-4.10.0
collected 1 item

lab-081.py .

========================================================================== 1 passed in 0.05s ==================================
```

Task 10 :

Write a function SortNumbers (implement bubble sort) and validate its implementation using 25 pytest test cases.



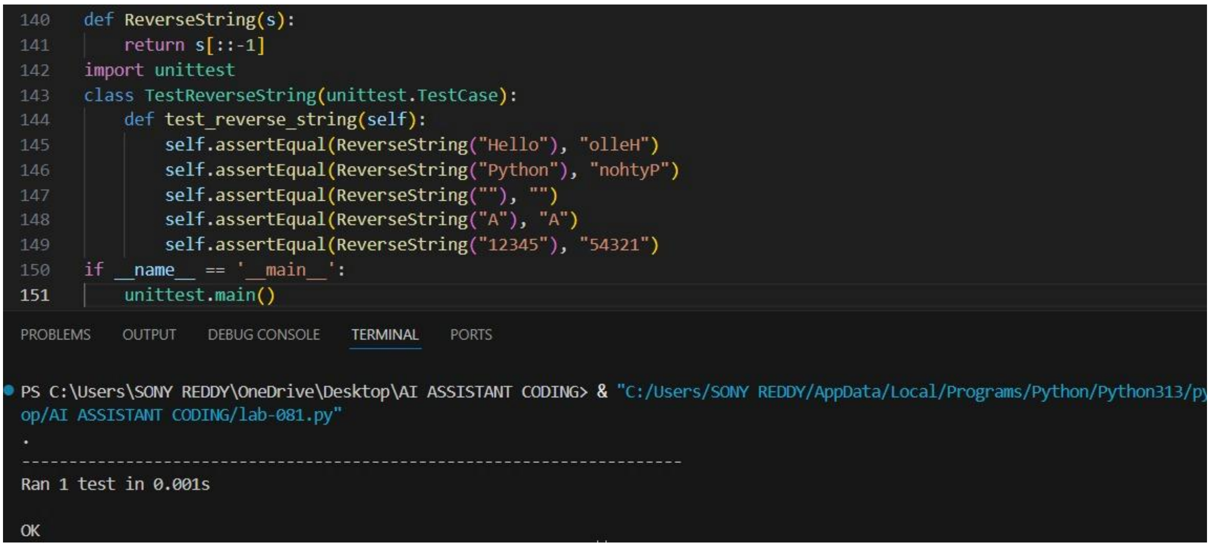
```python
def SortNumbers(numbers):
    n= len(numbers)
    for i in range(n):
        for j in range(0, n-i-1):
            if numbers[j] > numbers[j+1]:
                numbers[j], numbers[j+1] = numbers[j+1], numbers[j]
    return numbers
def test_sort_numbers():
    assert SortNumbers([5, 2, 9, 1, 5, 6]) == [1, 2, 5, 5, 6, 9]
    assert SortNumbers([]) == []
    assert SortNumbers([3]) == [3,5]
    assert SortNumbers([3, 2]) == [2, 3]
    assert SortNumbers([1, 2, 3]) == [1, 2, 3]
    assert SortNumbers([3, 2, 1]) == [1, 2, 3]
    assert SortNumbers([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5]
    assert SortNumbers([1, 1, 1, 1]) == [1, 1, 1, 1]
    assert SortNumbers([9, 8, 7, 6, 5]) == [5, 6, 7, 8, 9]
    assert SortNumbers([10, 9, 8, 7, 6]) == [6, 7, 8, 9, 10]
    assert SortNumbers([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
    assert SortNumbers([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5]
    assert SortNumbers([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
    assert SortNumbers([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5]
    assert SortNumbers([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
    print("All test cases for SortNumbers passed!")
```



```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> python -m pytest lab-081.py
plugins: anyio-4.10.0
collected 1 item

lab-081.py F

================================================================================ FAILURES ====================================================
___________________________________________________________________________ test_sort_numbers ________________________________________________

    def test_sort_numbers():
        assert SortNumbers([5, 2, 9, 1, 5, 6]) == [1, 2, 5, 5, 6, 9]
        assert SortNumbers([]) == []
>       assert SortNumbers([3]) == [3,5]
E       assert [3] == [3, 5]
E
E         Right contains one more item: 5
E         Use -v to get more diff

lab-081.py:125: AssertionError
======================================================================= short test summary info ==============================================
FAILED lab-081.py::test_sort_numbers - assert [3] == [3, 5]
=========================================================================== 1 failed in 0.22s ================================================
```

Task 11 :

Write a function ReverseString and validate its implementation using 5 unittest test cases

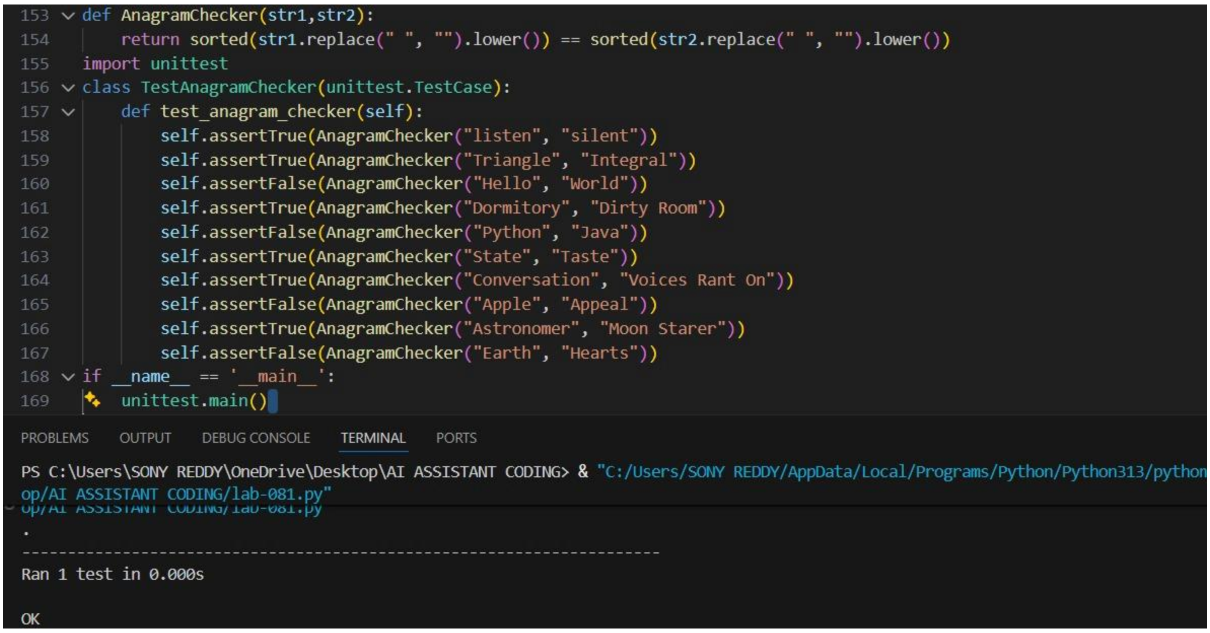


```python
def ReverseString(s):
    return s[::-1]
import unittest
class TestReverseString(unittest.TestCase):
    def test_reverse_string(self):
        self.assertEqual(ReverseString("Hello"), "olleH")
        self.assertEqual(ReverseString("Python"), "nohtyP")
        self.assertEqual(ReverseString(""), "")
        self.assertEqual(ReverseString("A"), "A")
        self.assertEqual(ReverseString("12345"), "54321")
if __name__ == '__main__':
    unittest.main()
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/py
op/AI ASSISTANT CODING/lab-081.py"
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Task 12 :

Write a function AnagramChecker and validate its implementation using 10 unittest test cases.

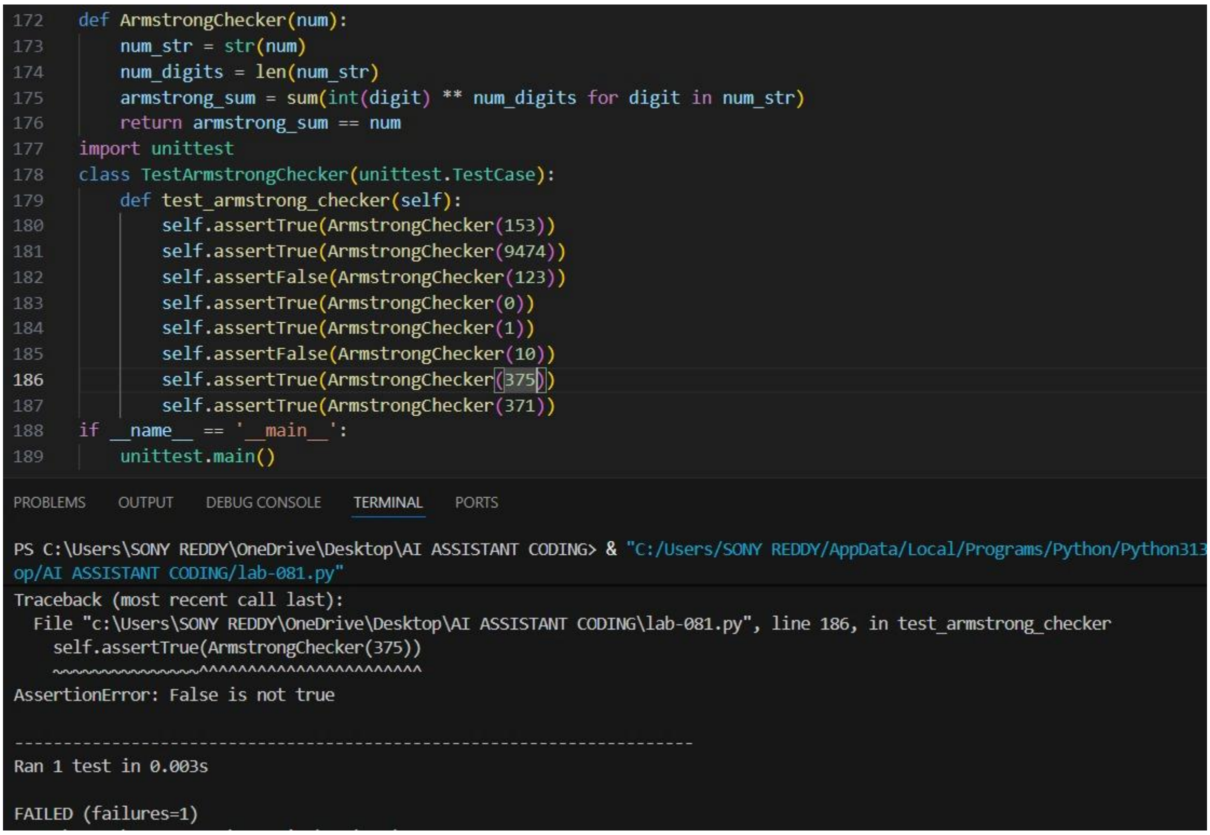


```python
def AnagramChecker(str1,str2):
    return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
import unittest
class TestAnagramChecker(unittest.TestCase):
    def test_anagram_checker(self):
        self.assertTrue(AnagramChecker("listen", "silent"))
        self.assertTrue(AnagramChecker("Triangle", "Integral"))
        self.assertFalse(AnagramChecker("Hello", "World"))
        self.assertTrue(AnagramChecker("Dormitory", "Dirty Room"))
        self.assertFalse(AnagramChecker("Python", "Java"))
        self.assertTrue(AnagramChecker("State", "Taste"))
        self.assertTrue(AnagramChecker("Conversation", "Voices Rant On"))
        self.assertFalse(AnagramChecker("Apple", "Appeal"))
        self.assertTrue(AnagramChecker("Astronomer", "Moon Starer"))
        self.assertFalse(AnagramChecker("Earth", "Hearts"))
if __name__ == '__main__':
    unittest.main()
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python
op/AI ASSISTANT CODING/lab-081.py"
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Task 13 :

Write a function ArmstrongChecker and validate its implementation using 8 unittest test cases.



```python
def ArmstrongChecker(num):
    num_str = str(num)
    num_digits = len(num_str)
    armstrong_sum = sum(int(digit) ** num_digits for digit in num_str)
    return armstrong_sum == num
import unittest
class TestArmstrongChecker(unittest.TestCase):
    def test_armstrong_checker(self):
        self.assertTrue(ArmstrongChecker(153))
        self.assertTrue(ArmstrongChecker(9474))
        self.assertFalse(ArmstrongChecker(123))
        self.assertTrue(ArmstrongChecker(0))
        self.assertTrue(ArmstrongChecker(1))
        self.assertFalse(ArmstrongChecker(10))
        self.assertTrue(ArmstrongChecker(375))
        self.assertTrue(ArmstrongChecker(371))
if __name__ == '__main__':
    unittest.main()
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313
op/AI ASSISTANT CODING/lab-081.py"
Traceback (most recent call last):
  File "c:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING\lab-081.py", line 186, in test_armstrong_checker
    self.assertTrue(ArmstrongChecker(375))
    ~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: False is not true

----------------------------------------------------------------------
Ran 1 test in 0.003s

FAILED (failures=1)
```