Ai Assisted Coding

Assignment-8

Name : G.Samshray

Ht.no : 2303A51670

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:

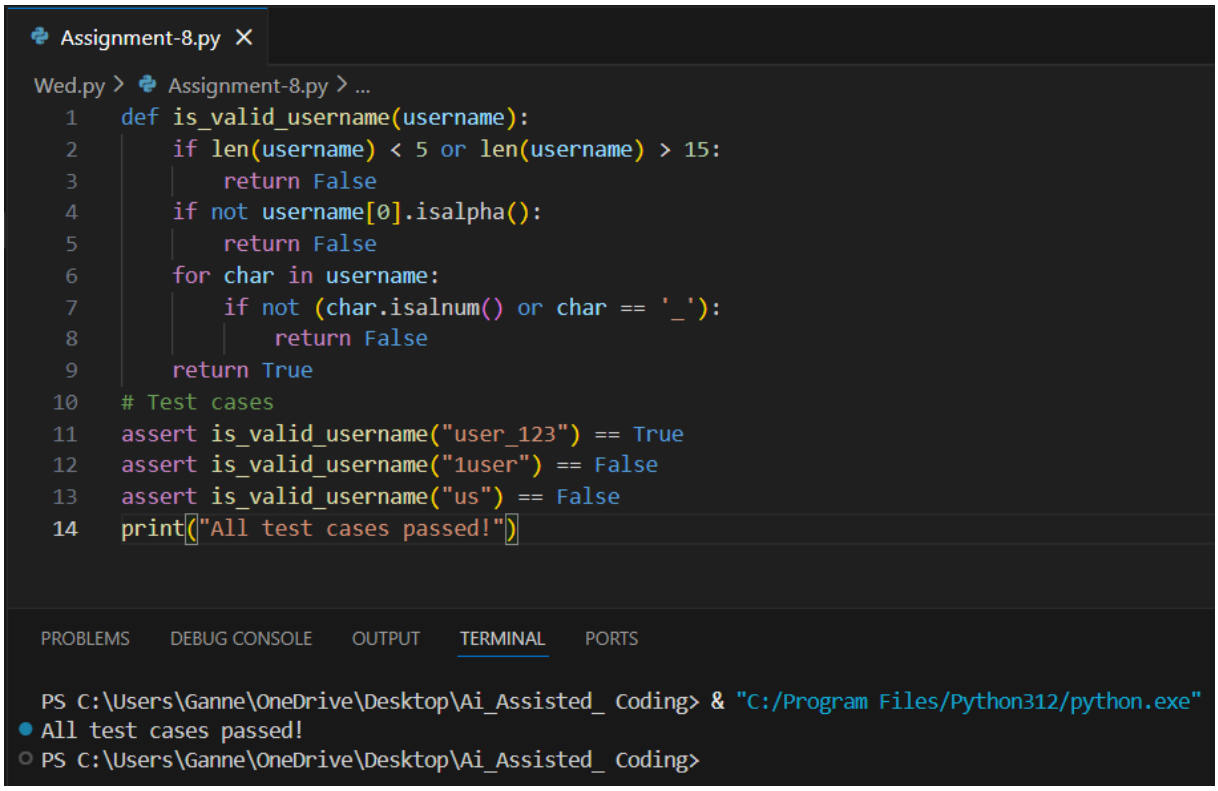
assert is_valid_username("User123") == True

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.



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:

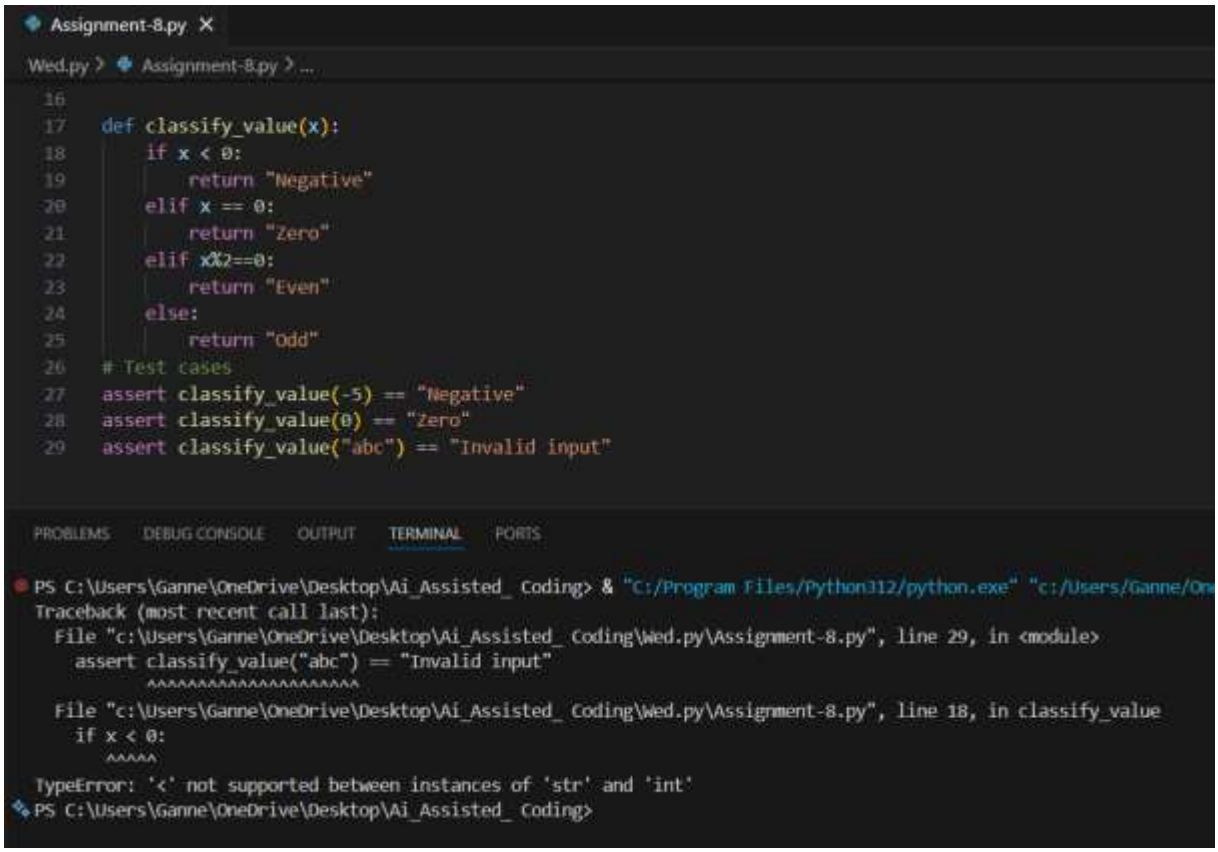
assert classify_value(8) == "Even"

assert classify_value(7) == "Odd"

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

Expected Output #2:

• Function correctly classifying values and passing all test cases.



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.

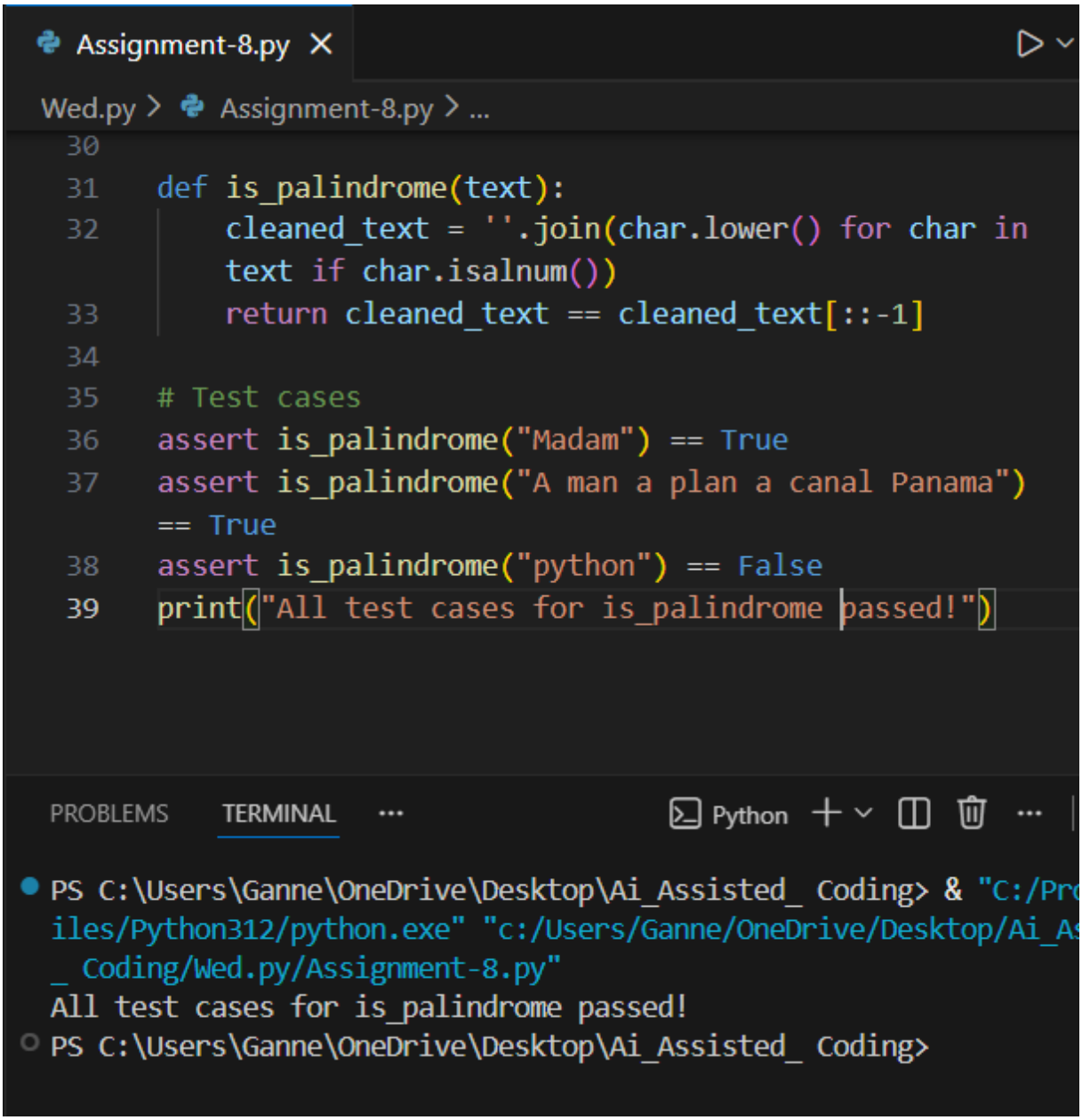
o Handle edge cases such as empty strings and single characters.

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



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:

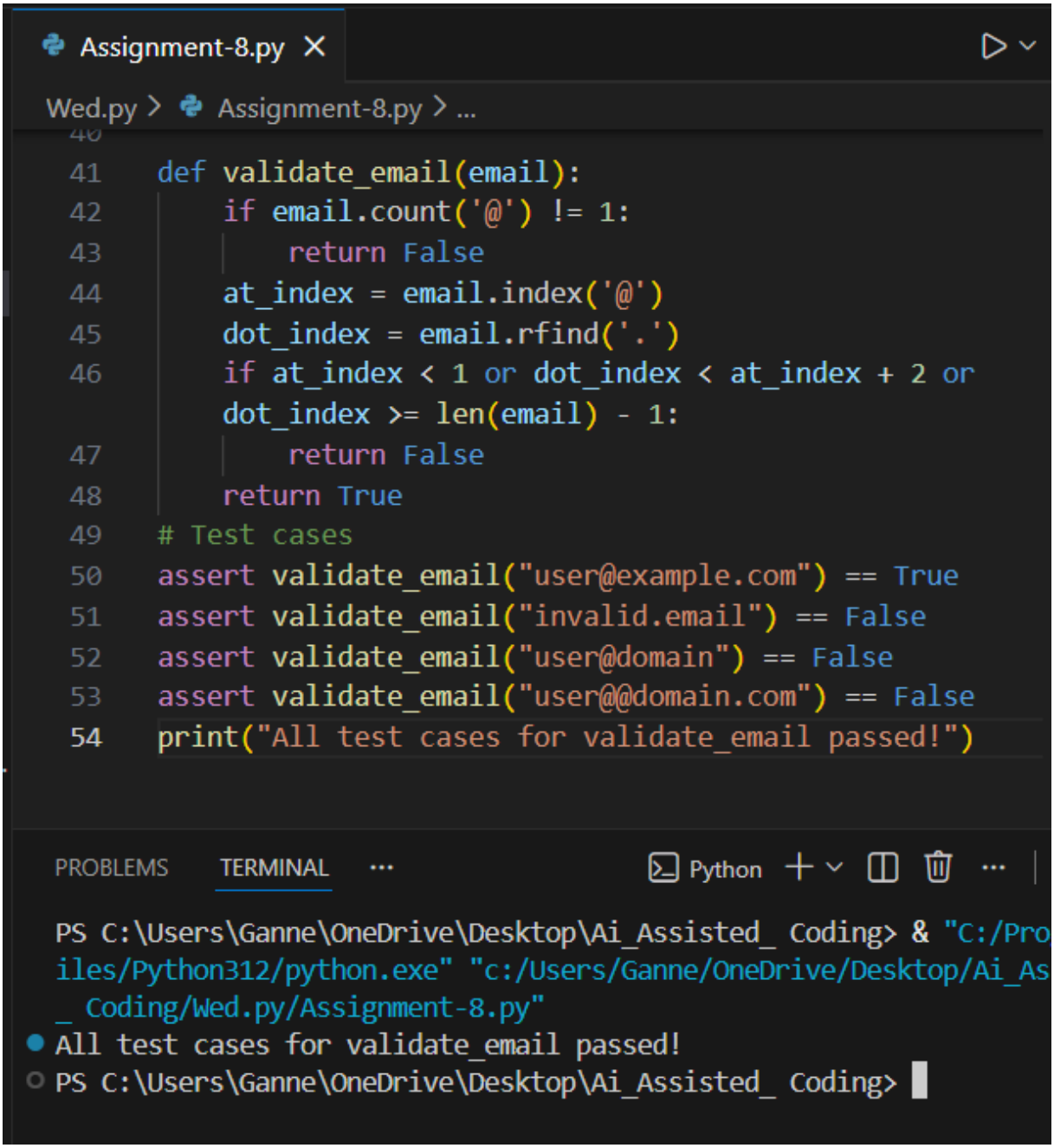
assert validate_email("user@example.com") == True

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.



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.

```python
# generate a python code to display whether the given number is perfect or not.
def is_perfect_number(n):
    if n < 1:
        return False
    sum_of_divisors = sum(i for i in range(1, n) if n % i == 0)
    return sum_of_divisors == n
# Test cases
assert is_perfect_number(6) == True
assert is_perfect_number(28) == True
assert is_perfect_number(12) == False
assert is_perfect_number(0) == False
print("All test cases for is_perfect_number passed!")
```

```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS

PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding> & "C:/Program Files/Python312/python.exe" "c:/Users/Gan
All test cases for is_perfect_number passed!
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding>
```

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.Large case: 945.

Requirement: Validate correctness with unittest

```python
def Abundant_number(n):
    if n < 1:
        return False
    sum_of_divisors = sum(i for i in range(1, n) if n % i == 0)
    return sum_of_divisors > n
import unittest
class TestAbundantNumber(unittest.TestCase):
    def test_abundant_number(self):
        self.assertTrue(Abundant_number(12))
        self.assertTrue(Abundant_number(15))
        self.assertFalse(Abundant_number(1))
        self.assertFalse(Abundant_number(-1))
        self.assertFalse(Abundant_number(987))
if __name__ == '__main__':
    unittest.main()
```

```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS

PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding> & "C:/Program Files/Python312/python.exe" "c:/Users/Ganne/OneDr
F
======================================================================
FAIL: test_abundant_number (__main__.TestAbundantNumber.test_abundant_number)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\Wed.py\Assignment-8.py", line 78, in test_abundant_number
    self.assertTrue(Abundant_number(15))
AssertionError: False is not true

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

FAILED (failures=1)
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding>
```

Task 7 (Deficient Number Checker – Test Case Design)

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

• Test Cases to Design:

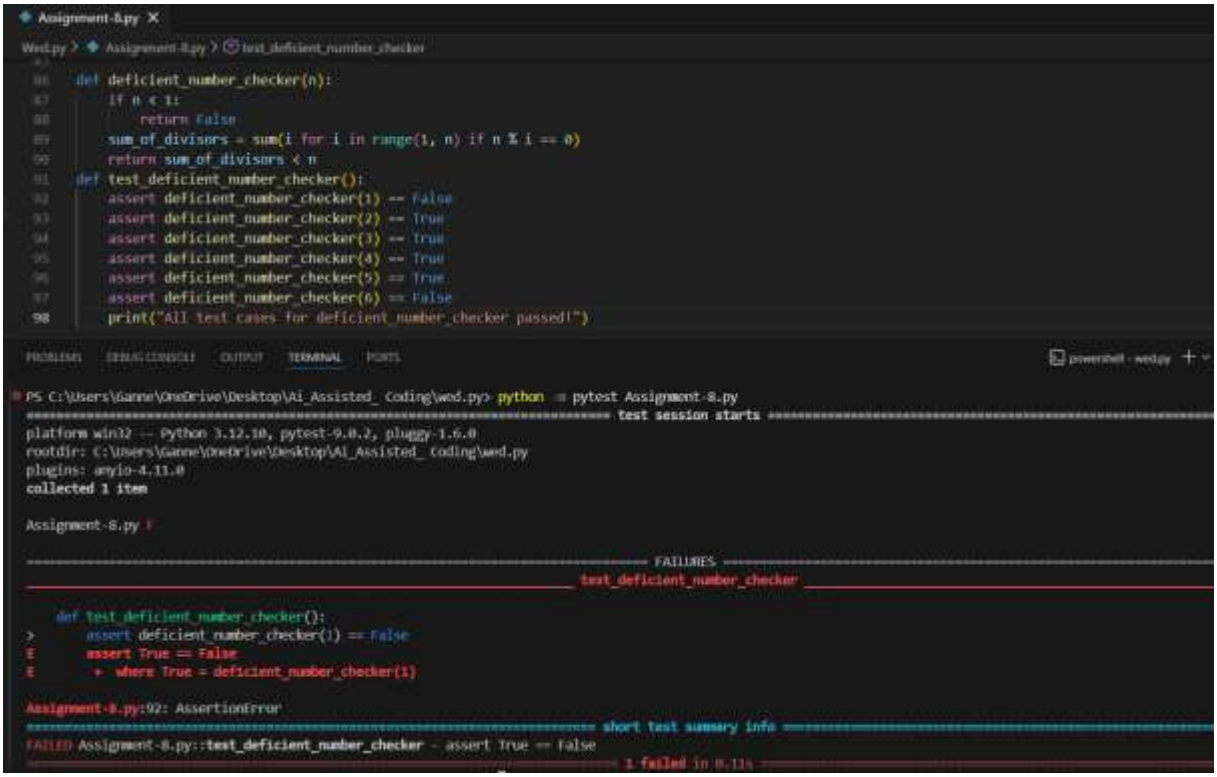
o Normal case: 8 → True, 12 → False.

o Edge case: 1.

o Negative number case.

o Large case: 546.

Requirement: Validate correctness with pytest



Task 8 :

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

Assignment-8.py

Wed.py > Assignment-8.py > test_leap_year_checker

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

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS

```
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding> & "C:/Program Files/Python312/python.exe" "c:/Users/Ganne/OneDrive/D
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding> cd wed.py
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\wed.py> python -m pytest Assignment-8.py
========================= test session starts =========================
platform win32 -- Python 3.12.10, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\wed.py
plugins: anyio-4.11.0
collected 1 item

Assignment-8.py .                                                    [100%]

========================== 1 passed in 0.02s ==========================
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\wed.py>
```

Task 9 :

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

```python
def sum_of_digits(n):
    return sum(int(digit) for digit in str(abs(n)) if digit.isdigit())
def test_sum_of_digits():
    assert sum_of_digits(123) == 6
    assert sum_of_digits(-456) == 15
    assert sum_of_digits(0) == 0
    assert sum_of_digits(78910) == 25
    print("All test cases for sum_of_digits passed!")
```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS

```
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding> & "C:/Program Files/Python312/python.exe" "c:/Users/Ganne/OneDrive/Desk
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding> cd wed.py
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\wed.py> python -m pytest Assignment-8.py
========================= test session starts =========================
platform win32 -- Python 3.12.10, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\wed.py
plugins: anyio-4.11.0
collected 1 item

Assignment-8.py .                                                    [100%]

========================== 1 passed in 0.02s ==========================
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\wed.py>
```

Task 10 :

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

```
120
121  def sortNumbers(numbers) :
122      n = len(numbers)
123      for i in range(n):
124          for j in range(0, n-i-1):
125              if numbers[j] > numbers[j+1] :
126                  numbers[j], numbers[j+1] = numbers[j+1], numbers[j]
127      return numbers
128  def test_sort_numbers():
129      assert sortNumbers([5, 2, 9, 1, 5, 6]) == [1, 2, 5, 5, 6, 9]
130      assert sortNumbers([]) == []
131      assert sortNumbers([3]) == [3]
132      assert sortNumbers([3, 2]) == [2, 3]
133      assert sortNumbers([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
134      assert sortNumbers([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 8]
135      print("All test cases for sortNumbers passed!")
```

.

```
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\wed.py> python -m pytest Assignment-8.py
============== test session starts ==============
platform win32 -- Python 3.12.10, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\wed.py
plugins: anyio-4.11.0
collected 1 item

Assignment-8.py F                                [100%]
    def test_sort_numbers():
        assert sortNumbers([5, 2, 9, 1, 5, 6]) == [1, 2, 5, 5, 6, 9]
        assert sortNumbers([]) == []
    def 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]
        assert sortNumbers([3, 2]) == [2, 3]
        assert sortNumbers([]) == []
        assert sortNumbers([]) == []
        assert sortNumbers([3]) == [3]
        assert sortNumbers([3, 2]) == [2, 3]
        assert sortNumbers([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
>       assert sortNumbers([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 8]
E       assert [1, 2, 3, 4, 5] == [1, 2, 3, 4, 8]
E
E         At index 4 diff: 5 != 8
E         Use -v to get more diff

Assignment-8.py:134: AssertionError
============== short test summary info ==============
FAILED Assignment-8.py::test_sort_numbers - assert [1, 2, 3, 4, 5] == [1, 2, 3, 4, 8]
============== 1 failed in 0.11s ==============
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\wed.py>
```

Task 11 :

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

```
def Reverse_string(s):
    return s[::-1]
import unittest
class TestReverseString(unittest.TestCase):
    def test_reverse_string(self):
        self.assertEqual(Reverse_string("hello"), "olleh")
        self.assertEqual(Reverse_string("Python"), "nohtyP")
        self.assertEqual(Reverse_string(""), "")
        self.assertEqual(Reverse_string("a"), "a")
if __name__ == '__main__':
    unittest.main()
```

```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS

PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding> & "C:/Program Files/Python312/python.exe" "c:/Users/Ganne/O
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding>
```

Task 12 :

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

```

def Anagram_checker(str1, str2):
    return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
import unittest
class TestAnagramChecker(unittest.TestCase):
    def test_anagram_checker(self):
        self.assertTrue(Anagram_checker("listen", "silent"))
        self.assertTrue(Anagram_checker("Triangle", "Integral"))
        self.assertFalse(Anagram_checker("hello", "world"))
        self.assertFalse(Anagram_checker("Python", "Java"))
        self.assertTrue(Anagram_checker("Dormitory", "Dirty Room"))
if __name__ == '__main__':
    unittest.main()

```

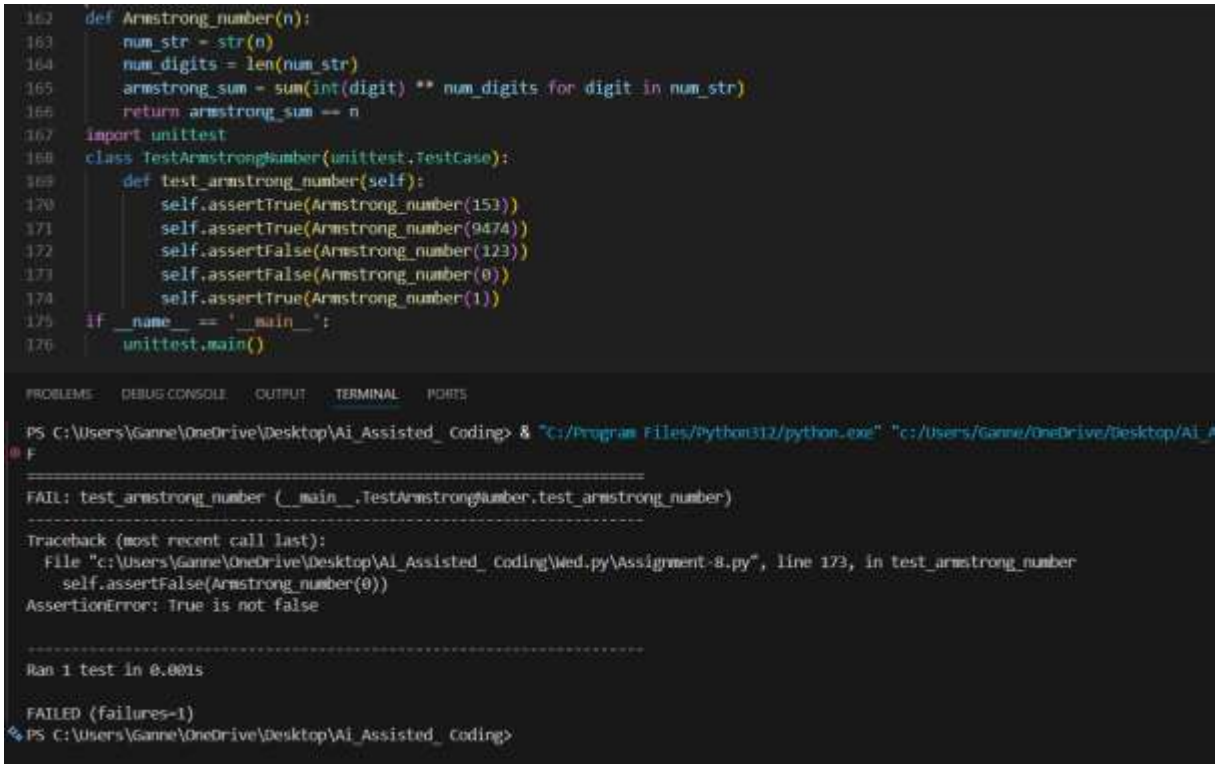
```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS

PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding> & "C:/Program Files/Python312/python.exe" "c:/Users/Ganne/OneDrive/Desktop/Ai
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding>
```

Task 13 :

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



```
def Armstrong_number(n):
    num_str = str(n)
    num_digits = len(num_str)
    armstrong_sum = sum(int(digit) ** num_digits for digit in num_str)
    return armstrong_sum == n
import unittest
class TestArmstrongNumber(unittest.TestCase):
    def test_armstrong_number(self):
        self.assertTrue(Armstrong_number(153))
        self.assertTrue(Armstrong_number(9474))
        self.assertFalse(Armstrong_number(123))
        self.assertFalse(Armstrong_number(0))
        self.assertTrue(Armstrong_number(1))
if __name__ == '__main__':
    unittest.main()
```

```
PROBLEMS  DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS

PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding> & "C:/Program Files/Python312/python.exe" "c:/Users/Ganne/OneDrive/Desktop/Ai_A
F
======================================================================
FAIL: test_armstrong_number (__main__.TestArmstrongNumber.test_armstrong_number)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\Wed.py\Assignment-8.py", line 173, in test_armstrong_number
    self.assertFalse(Armstrong_number(0))
AssertionError: True is not false

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

FAILED (failures=1)
PS C:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding>
```