Lab Assignment_08(Monday)

Name:M.Hasini

Hallticket:2303A51109

Batch-02

Task Description #1 (Username Validator – Apply AI in

Authentication Context)

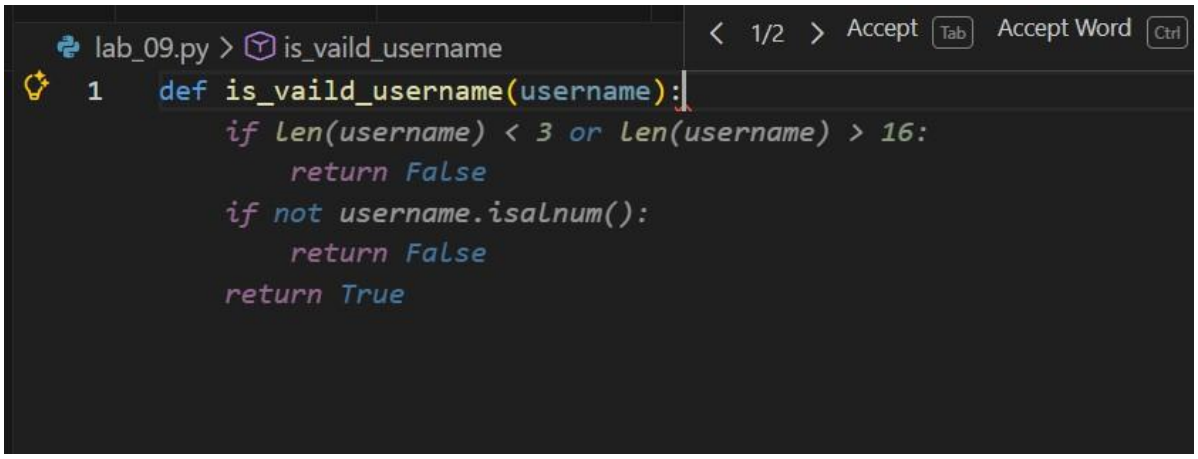
- Task: Use AI to generate at least 3 assert test cases for a function is_valid_username(username) and then implement the function using Test-Driven Development principles.
- Requirements:
  - o Username length must be between 5 and 15 characters.
  - o Must contain only alphabets and digits.
  - o Must not start with a digit.
  - o No spaces allowed.

Example Assert Test Cases:

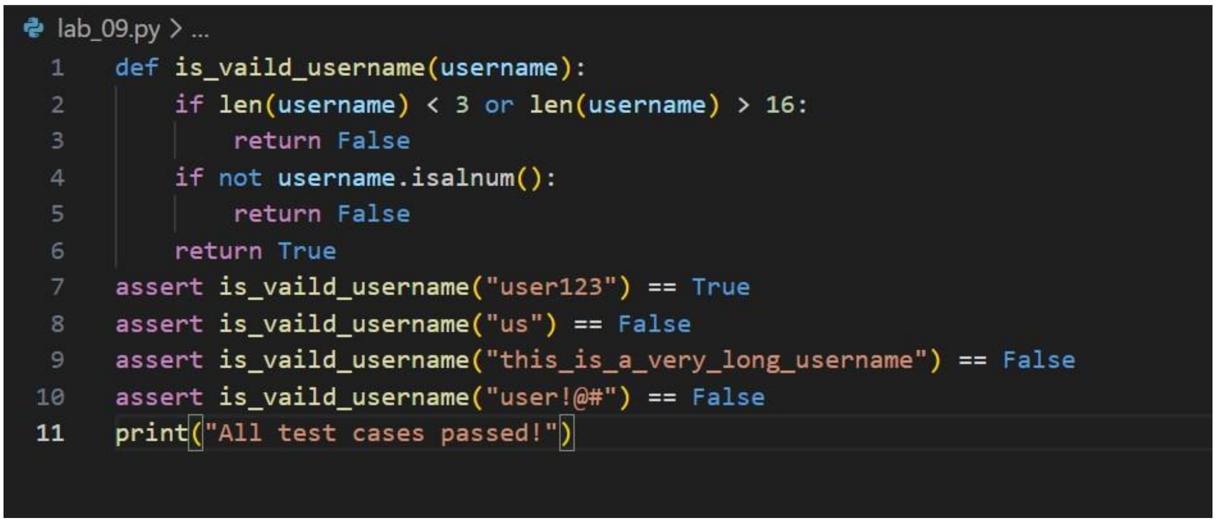
```
assert is_valid_username("User123") == True
assert is_valid_username("12User") == False
assert is_valid_username("Us er") == False
```

Expected Output #1:

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


lab_09.py > is_vaild_username
< 1/2 > Accept Tab Accept Word Ctrl

```python
def is_vaild_username(username):
    if len(username) < 3 or len(username) > 16:
        return False
    if not username.isalnum():
        return False
    return True
```

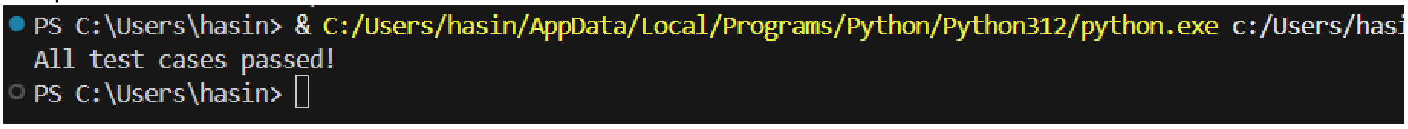


```
lab_09.py > ...
def is_vaild_username(username):
    if len(username) < 3 or len(username) > 16:
        return False
    if not username.isalnum():
        return False
    return True
assert is_vaild_username("user123") == True
assert is_vaild_username("us") == False
assert is_vaild_username("this_is_a_very_long_username") == False
assert is_vaild_username("user!@#") == False
print("All test cases passed!")
```

Code:

```python
def is_vaild_username(username):
    if len(username) < 3 or len(username) > 16:
        return False
    if not username.isalnum():
        return False
    return True
assert is_vaild_username("user123") == True
assert is_vaild_username("us") == False
assert is_vaild_username("this_is_a_very_long_username") == False
assert is_vaild_username("user!@#") == False
print("All test cases passed!")
```

Output:



```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hasi
All test cases passed!
PS C:\Users\hasin>
```

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"

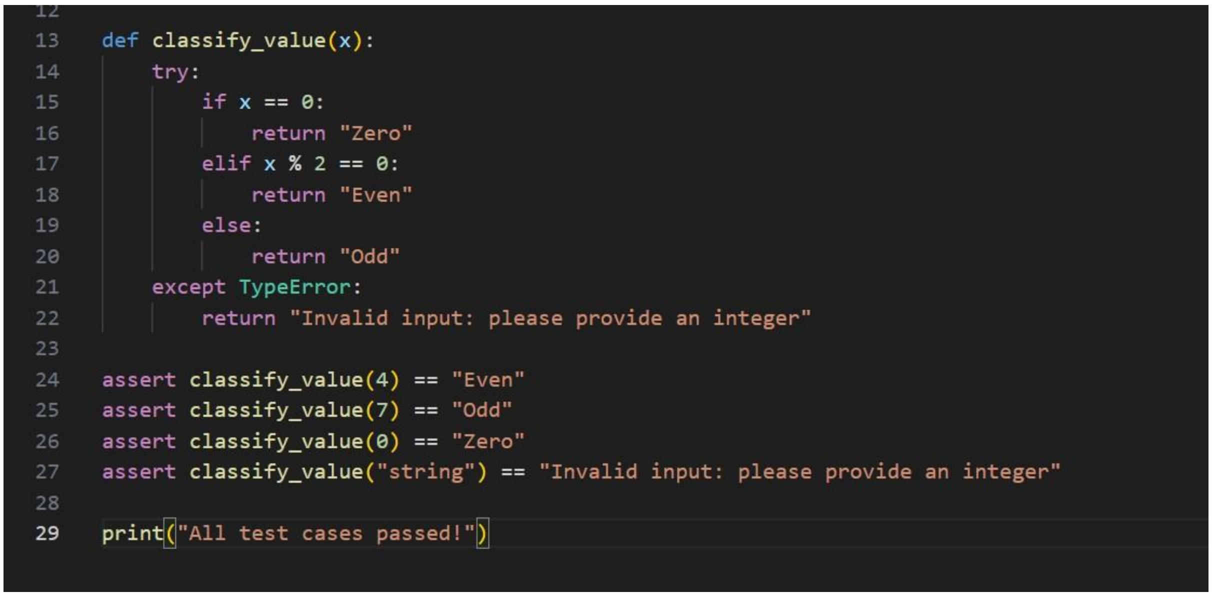
assert classify_value(7) == "Odd"

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

Expected Output #2:

• Function correctly classifying values and passing all test

cases.

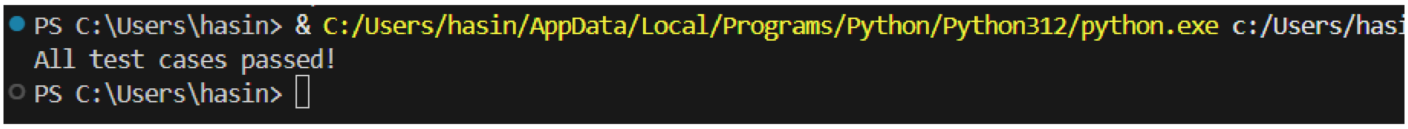


```python
def classify_value(x):
    try:
        if x == 0:
            return "Zero"
        elif x % 2 == 0:
            return "Even"
        else:
            return "Odd"
    except TypeError:
        return "Invalid input: please provide an integer"

assert classify_value(4) == "Even"
assert classify_value(7) == "Odd"
assert classify_value(0) == "Zero"
assert classify_value("string") == "Invalid input: please provide an integer"

print("All test cases passed!")
```

Code:

```python
def classify_value(x):
    try:
        if x == 0:
            return "Zero"
        elif x % 2 == 0:
            return "Even"
        else:
            return "Odd"
    except TypeError:
        return "Invalid input: please provide an integer"

assert classify_value(4) == "Even"
assert classify_value(7) == "Odd"
assert classify_value(0) == "Zero"
assert classify_value("string") == "Invalid input: please provide an integer"

print("All test cases passed!")
```

output:



```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hasi
All test cases passed!
PS C:\Users\hasin>
```

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:

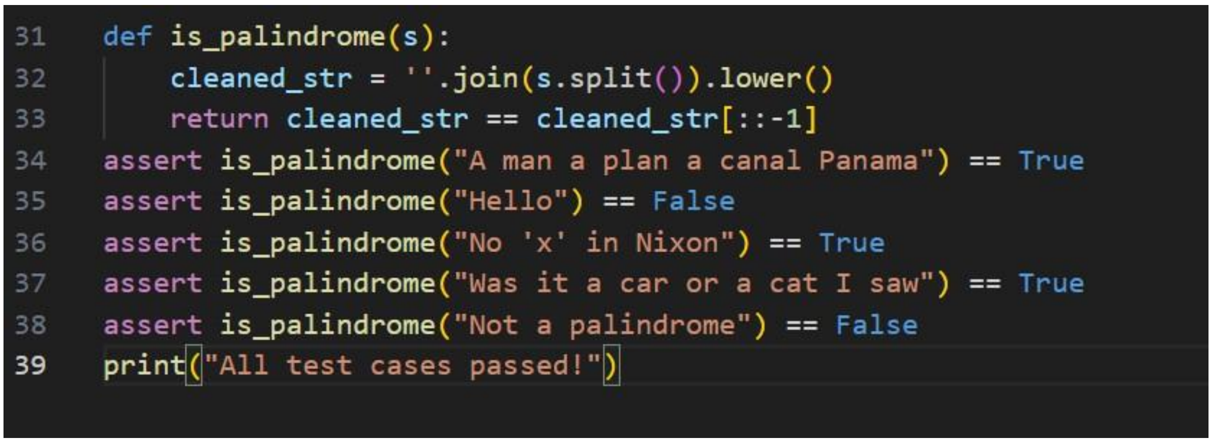
assert is_palindrome("Madam") == True

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

assert is_palindrome("Python") == False

Expected Output #3:

- Function correctly identifying palindromes and passing all AI-generated tests.

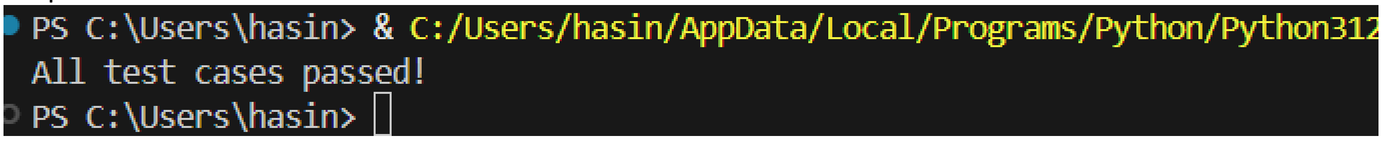


```python
def is_palindrome(s):
    cleaned_str = ''.join(s.split()).lower()
    return cleaned_str == cleaned_str[::-1]
assert is_palindrome("A man a plan a canal Panama") == True
assert is_palindrome("Hello") == False
assert is_palindrome("No 'x' in Nixon") == True
assert is_palindrome("Was it a car or a cat I saw") == True
assert is_palindrome("Not a palindrome") == False
print("All test cases passed!")
```

Code:

```python
def is_palindrome(s):
    cleaned_str = ''.join(s.split()).lower()
    return cleaned_str == cleaned_str[::-1]
assert is_palindrome("A man a plan a canal Panama") == True
assert is_palindrome("Hello") == False
assert is_palindrome("No 'x' in Nixon") == True
assert is_palindrome("Was it a car or a cat I saw") == True
assert is_palindrome("Not a palindrome") == False
print("All test cases passed!")
```

output:



```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312
All test cases passed!
PS C:\Users\hasin>
```

<u>Task Description #4 (Email ID Validation – Apply AI for Data</u>

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

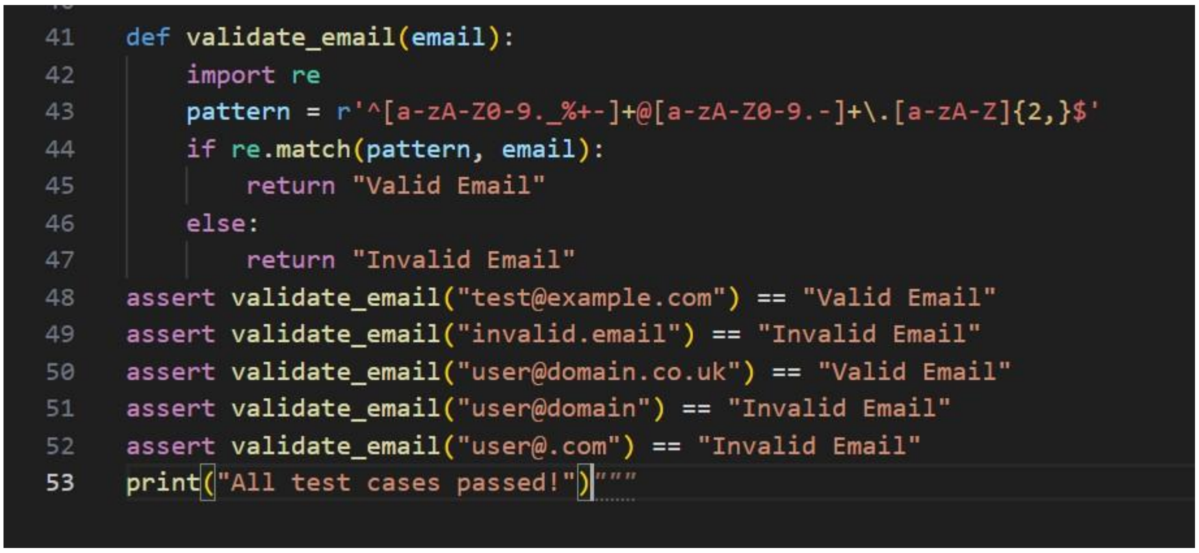
assert validate_email("userexample.com") == False

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

Expected Output #5:

- Email validation function passing all AI-generated test cases

  and handling edge cases correctly.

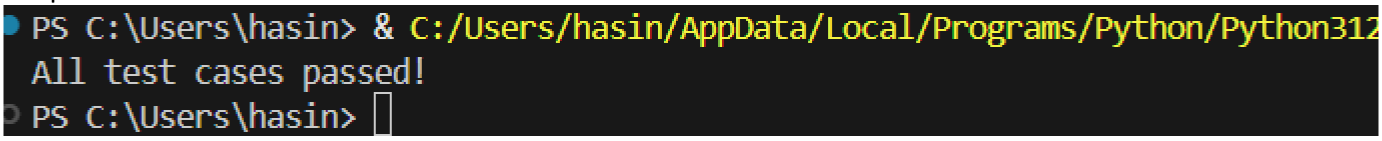


```
def validate_email(email):
    import re
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if re.match(pattern, email):
        return "Valid Email"
    else:
        return "Invalid Email"
assert validate_email("test@example.com") == "Valid Email"
assert validate_email("invalid.email") == "Invalid Email"
assert validate_email("user@domain.co.uk") == "Valid Email"
assert validate_email("user@domain") == "Invalid Email"
assert validate_email("user@.com") == "Invalid Email"
print("All test cases passed!")"""
```

Code:

```python
def validate_email(email):
    import re
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if re.match(pattern, email):
        return "Valid Email"
    else:
        return "Invalid Email"
assert validate_email("test@example.com") == "Valid Email"
assert validate_email("invalid.email") == "Invalid Email"
assert validate_email("user@domain.co.uk") == "Valid Email"
assert validate_email("user@domain") == "Invalid Email"
assert validate_email("user@.com") == "Invalid Email"
print("All test cases passed!")
```

output:



```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312
All test cases passed!
PS C:\Users\hasin>
```

<u>Task 5 (Perfect Number Checker – Test Case Design)</u>

• Function: Check if a number is a perfect number (sum of

divisors = number).

• Test Cases to Design:

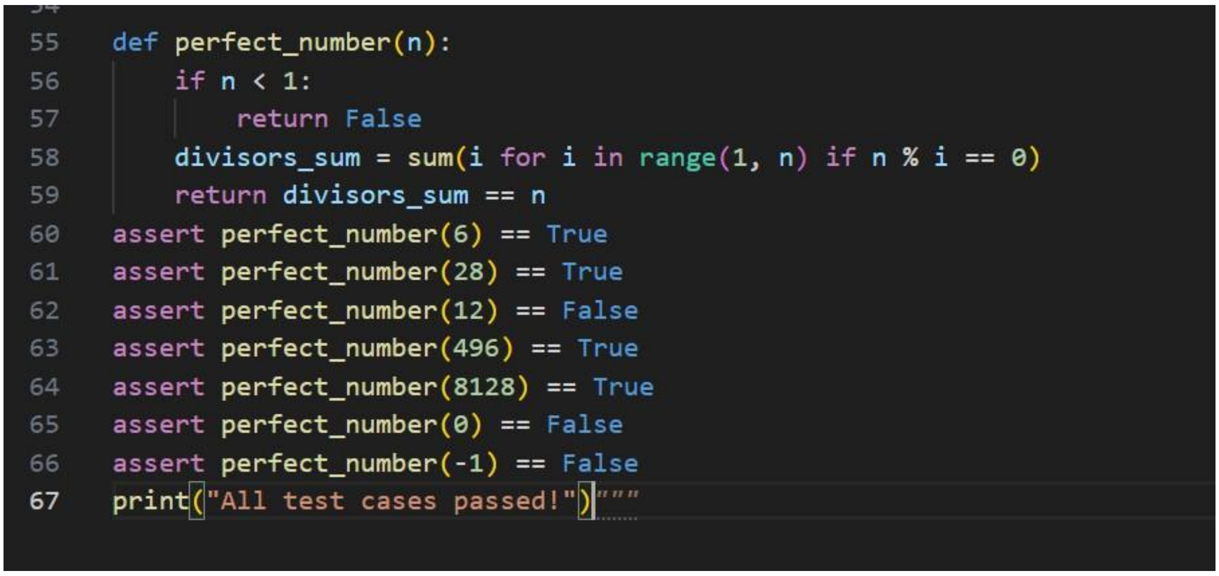
o Normal case: 6 → True, 10 → False.

o Edge case: 1.

o Negative number case.

o Larger case: 28.

• Requirement: Validate correctness with assertions.

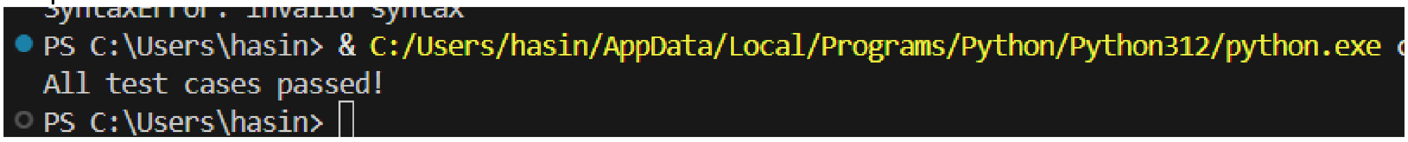


```python
def perfect_number(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum == n
assert perfect_number(6) == True
assert perfect_number(28) == True
assert perfect_number(12) == False
assert perfect_number(496) == True
assert perfect_number(8128) == True
assert perfect_number(0) == False
assert perfect_number(-1) == False
print("All test cases passed!")"""
```

Code:

```python
def perfect_number(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum == n
assert perfect_number(6) == True
assert perfect_number(28) == True
assert perfect_number(12) == False
assert perfect_number(496) == True
assert perfect_number(8128) == True
assert perfect_number(0) == False
assert perfect_number(-1) == False
print("All test cases passed!")
```

output:



```
SyntaxError: invalid syntax
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c
All test cases passed!
PS C:\Users\hasin>
```

<u>Task 6 (Abundant Number Checker – Test Case Design)</u>

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

number).

• Test Cases to Design:

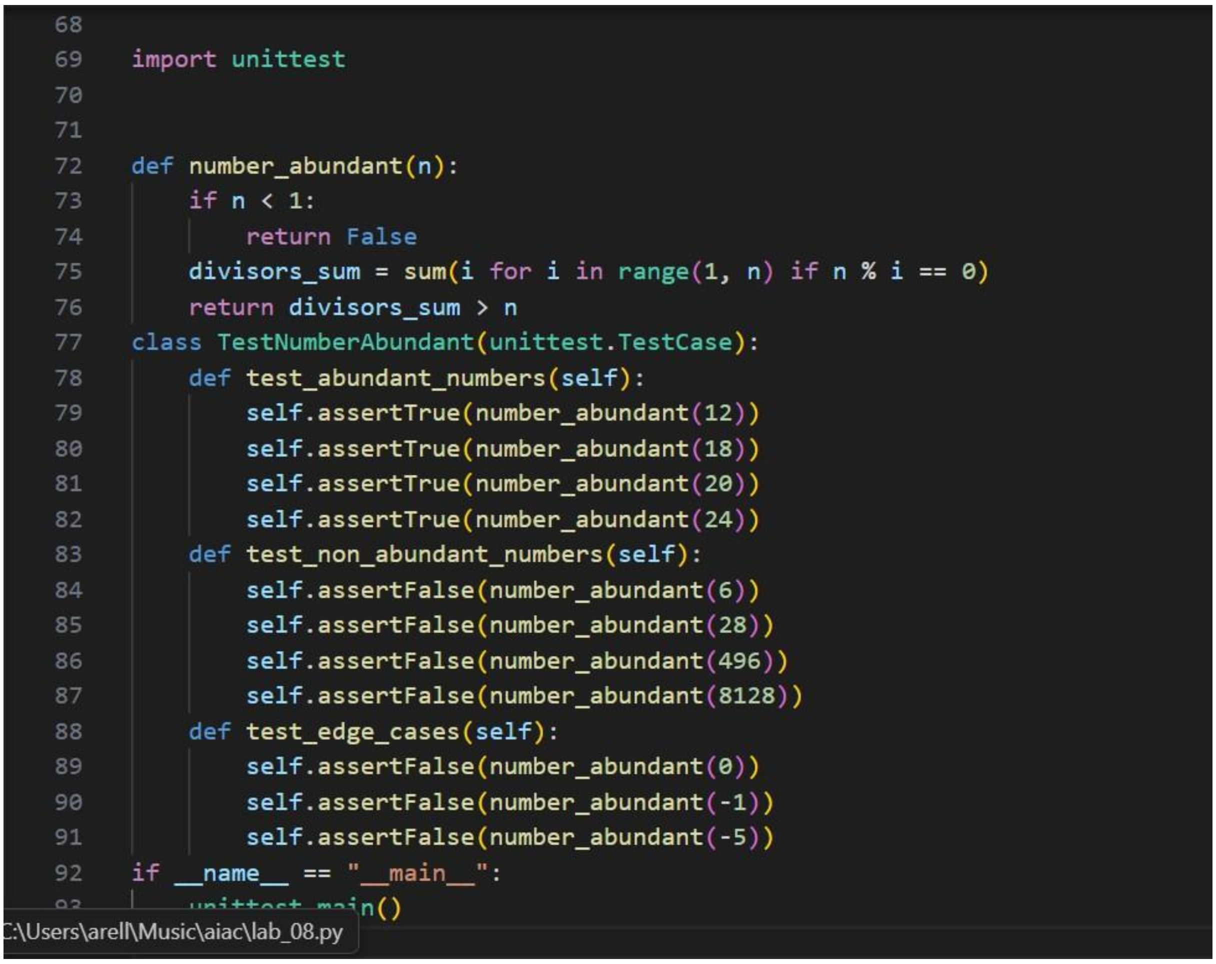
o Normal case: 12 → True, 15 → False.

o Edge case: 1.

o Negative number case.

o Large case: 945.

Requirement: Validate correctness with unittest



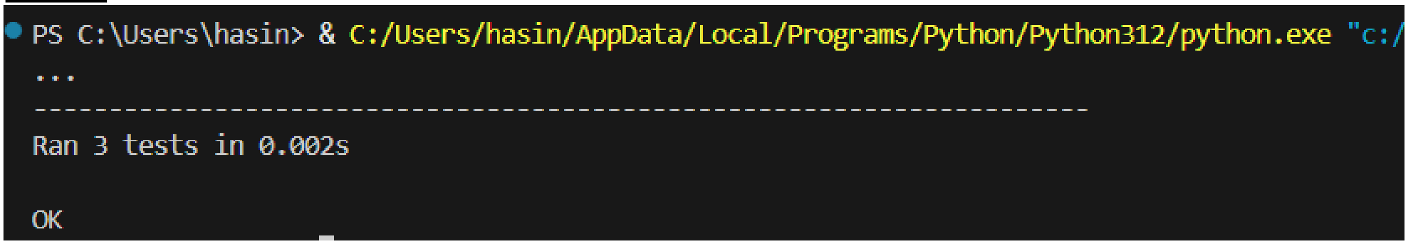
```python

import unittest


def number_abundant(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum > n
class TestNumberAbundant(unittest.TestCase):
    def test_abundant_numbers(self):
        self.assertTrue(number_abundant(12))
        self.assertTrue(number_abundant(18))
        self.assertTrue(number_abundant(20))
        self.assertTrue(number_abundant(24))
    def test_non_abundant_numbers(self):
        self.assertFalse(number_abundant(6))
        self.assertFalse(number_abundant(28))
        self.assertFalse(number_abundant(496))
        self.assertFalse(number_abundant(8128))
    def test_edge_cases(self):
        self.assertFalse(number_abundant(0))
        self.assertFalse(number_abundant(-1))
        self.assertFalse(number_abundant(-5))
if __name__ == "__main__":
    unittest.main()
```

:\Users\arell\Music\aiac\lab_08.py

<u>Code:</u>

```python
import unittest

def number_abundant(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum > n
class TestNumberAbundant(unittest.TestCase):
    def test_abundant_numbers(self):
        self.assertTrue(number_abundant(12))
        self.assertTrue(number_abundant(18))
        self.assertTrue(number_abundant(20))
        self.assertTrue(number_abundant(24))
    def test_non_abundant_numbers(self):
        self.assertFalse(number_abundant(6))
        self.assertFalse(number_abundant(28))
        self.assertFalse(number_abundant(496))
        self.assertFalse(number_abundant(8128))
    def test_edge_cases(self):
        self.assertFalse(number_abundant(0))
        self.assertFalse(number_abundant(-1))
        self.assertFalse(number_abundant(-5))
if __name__ == "__main__":
    unittest.main()
```

Output:

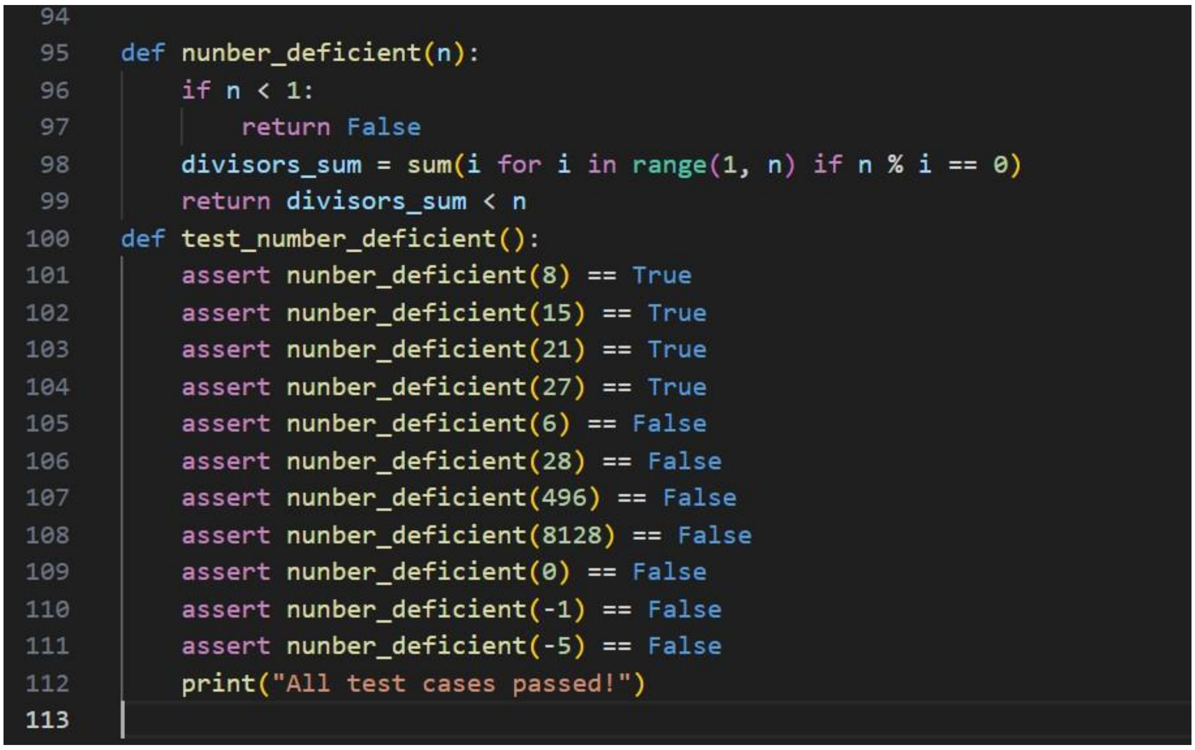


```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe "c:/
...
----------------------------------------------------------------------
Ran 3 tests in 0.002s

OK
```

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.

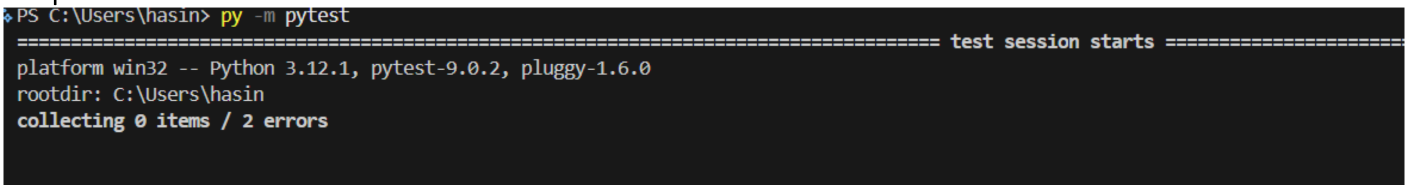


```

def nunber_deficient(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum < n
def test_number_deficient():
    assert nunber_deficient(8) == True
    assert nunber_deficient(15) == True
    assert nunber_deficient(21) == True
    assert nunber_deficient(27) == True
    assert nunber_deficient(6) == False
    assert nunber_deficient(28) == False
    assert nunber_deficient(496) == False
    assert nunber_deficient(8128) == False
    assert nunber_deficient(0) == False
    assert nunber_deficient(-1) == False
    assert nunber_deficient(-5) == False
    print("All test cases passed!")

```

Code:

```python
def nunber_deficient(n):
    if n < 1:
        return False
    divisors_sum = sum(i for i in range(1, n) if n % i == 0)
    return divisors_sum < n
def test_number_deficient():
    assert nunber_deficient(8) == True
    assert nunber_deficient(15) == True
    assert nunber_deficient(21) == True
    assert nunber_deficient(27) == True
    assert nunber_deficient(6) == False
    assert nunber_deficient(28) == False
    assert nunber_deficient(496) == False
    assert nunber_deficient(8128) == False
    assert nunber_deficient(0) == False
    assert nunber_deficient(-1) == False
    assert nunber_deficient(-5) == False
    print("All test cases passed!")
```

Output:

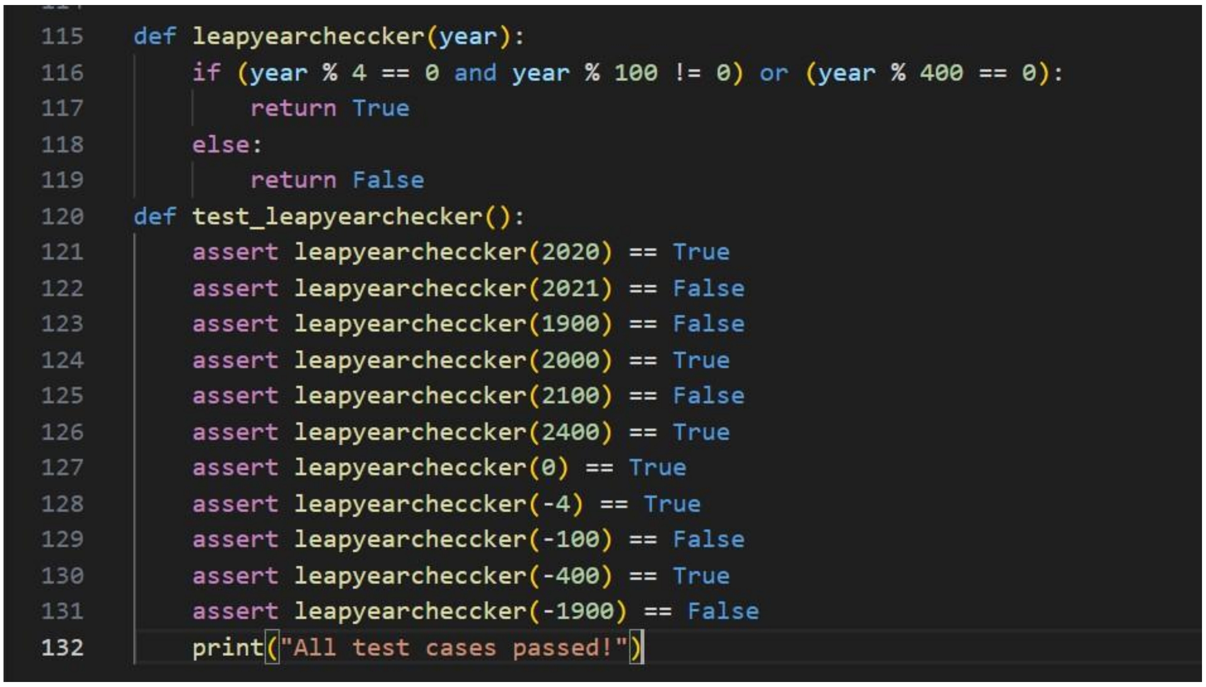


```
PS C:\Users\hasin> py -m pytest
================================================================================ test session starts ======================
platform win32 -- Python 3.12.1, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\hasin
collecting 0 items / 2 errors
```

Task 8 :

Write a function LeapYearChecker and validate its implementation

using 10 pytest test cases

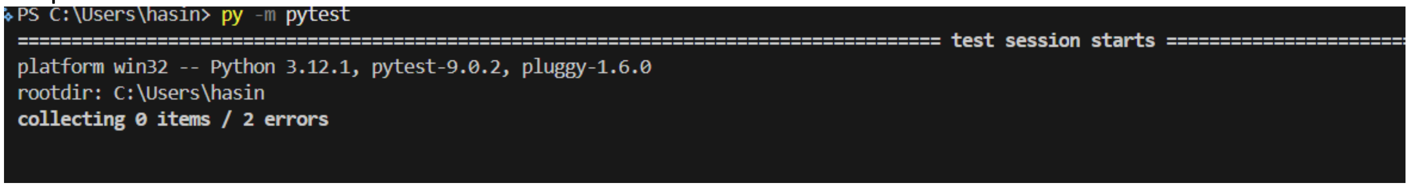


```python
def leapyearcheccker(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False
def test_leapyearchecker():
    assert leapyearcheccker(2020) == True
    assert leapyearcheccker(2021) == False
    assert leapyearcheccker(1900) == False
    assert leapyearcheccker(2000) == True
    assert leapyearcheccker(2100) == False
    assert leapyearcheccker(2400) == True
    assert leapyearcheccker(0) == True
    assert leapyearcheccker(-4) == True
    assert leapyearcheccker(-100) == False
    assert leapyearcheccker(-400) == True
    assert leapyearcheccker(-1900) == False
    print("All test cases passed!")
```

Code:

```python
def leapyearcheccker(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False
def test_leapyearchecker():
    assert leapyearcheccker(2020) == True
    assert leapyearcheccker(2021) == False
    assert leapyearcheccker(1900) == False
    assert leapyearcheccker(2000) == True
    assert leapyearcheccker(2100) == False
    assert leapyearcheccker(2400) == True
    assert leapyearcheccker(0) == True
    assert leapyearcheccker(-4) == True
    assert leapyearcheccker(-100) == False
    assert leapyearcheccker(-400) == True
    assert leapyearcheccker(-1900) == False
    print("All test cases passed!")
```

output:

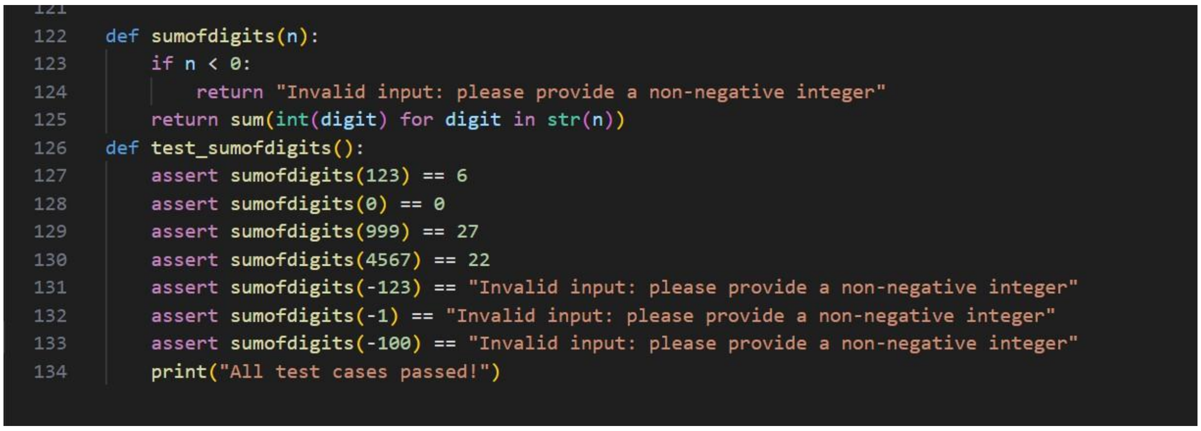


```
PS C:\Users\hasin> py -m pytest
=================================================================================== test session starts ======================
platform win32 -- Python 3.12.1, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\hasin
collecting 0 items / 2 errors
```

Task 9 :

Write a function SumOfDigits and validate its implementation

using 7 pytest test cases.

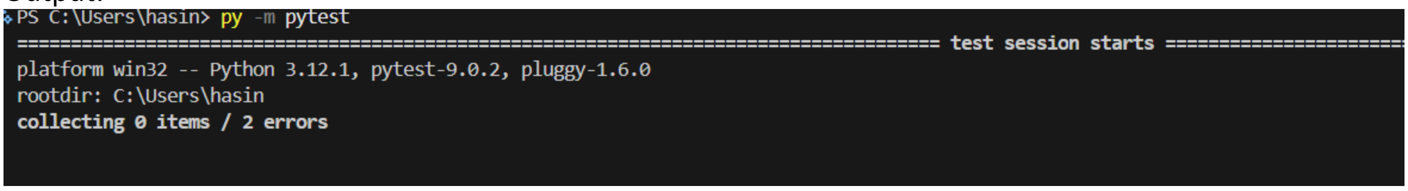


```
def sumofdigits(n):
    if n < 0:
        return "Invalid input: please provide a non-negative integer"
    return sum(int(digit) for digit in str(n))
def test_sumofdigits():
    assert sumofdigits(123) == 6
    assert sumofdigits(0) == 0
    assert sumofdigits(999) == 27
    assert sumofdigits(4567) == 22
    assert sumofdigits(-123) == "Invalid input: please provide a non-negative integer"
    assert sumofdigits(-1) == "Invalid input: please provide a non-negative integer"
    assert sumofdigits(-100) == "Invalid input: please provide a non-negative integer"
    print("All test cases passed!")
```

Code:

```
def sumofdigits(n):
    if n < 0:
        return "Invalid input: please provide a non-negative integer"
    return sum(int(digit) for digit in str(n))
def test_sumofdigits():
    assert sumofdigits(123) == 6
    assert sumofdigits(0) == 0
    assert sumofdigits(999) == 27
    assert sumofdigits(4567) == 22
    assert sumofdigits(-123) == "Invalid input: please provide a non-negative
integer"
    assert sumofdigits(-1) == "Invalid input: please provide a non-negative
integer"
    assert sumofdigits(-100) == "Invalid input: please provide a non-negative
integer"
    print("All test cases passed!")
```

Output:

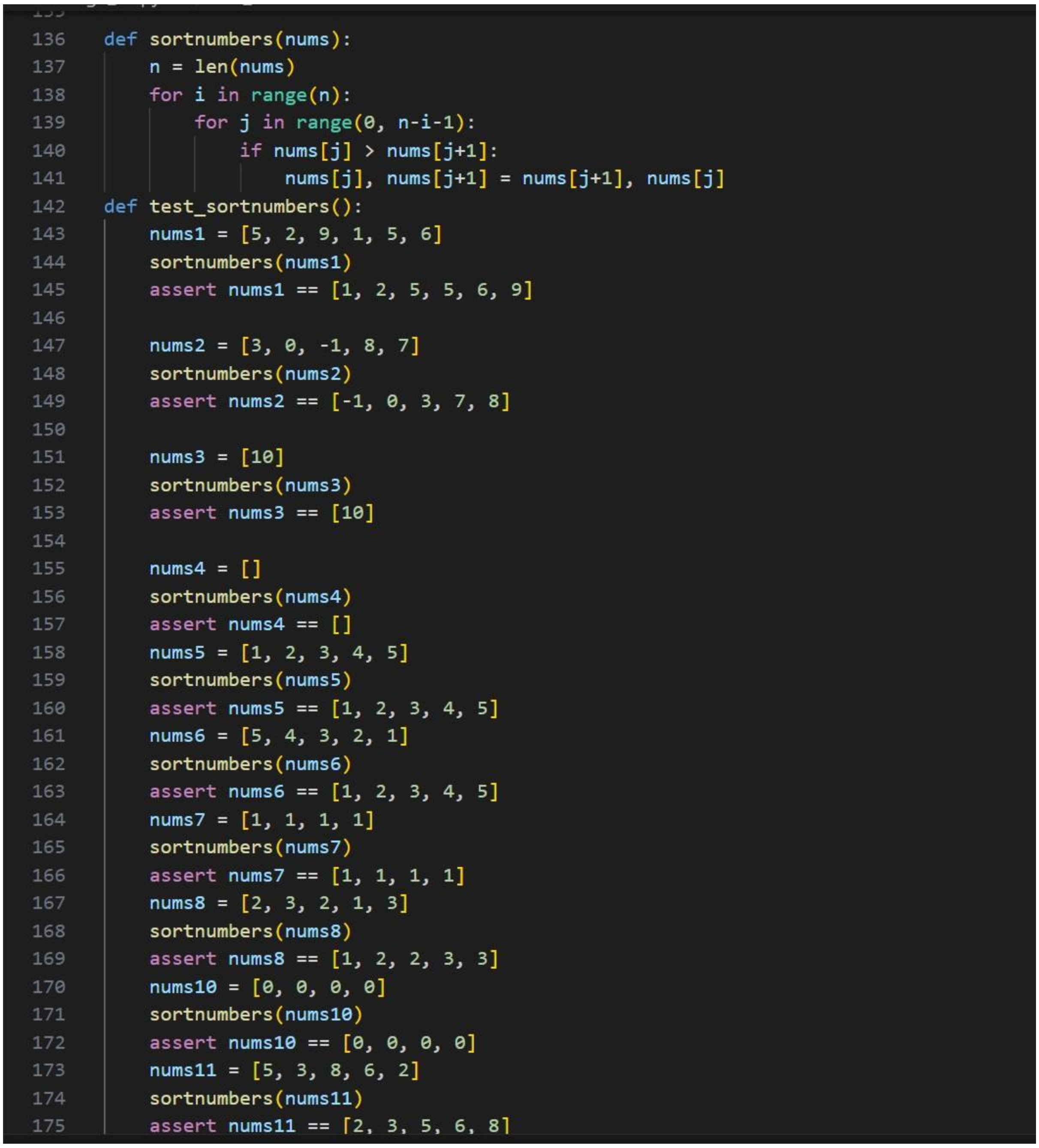


```
PS C:\Users\hasin> py -m pytest
=========================================================================== test session starts ======================
platform win32 -- Python 3.12.1, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\hasin
collecting 0 items / 2 errors
```

Task 10 :

Write a function SortNumbers (implement bubble sort) and validate

its implementation using 25 pytest test cases.



```
def sortnumbers(nums):
    n = len(nums)
    for i in range(n):
        for j in range(0, n-i-1):
            if nums[j] > nums[j+1]:
                nums[j], nums[j+1] = nums[j+1], nums[j]
def test_sortnumbers():
    nums1 = [5, 2, 9, 1, 5, 6]
    sortnumbers(nums1)
    assert nums1 == [1, 2, 5, 5, 6, 9]

    nums2 = [3, 0, -1, 8, 7]
    sortnumbers(nums2)
    assert nums2 == [-1, 0, 3, 7, 8]

    nums3 = [10]
    sortnumbers(nums3)
    assert nums3 == [10]

    nums4 = []
    sortnumbers(nums4)
    assert nums4 == []
    nums5 = [1, 2, 3, 4, 5]
    sortnumbers(nums5)
    assert nums5 == [1, 2, 3, 4, 5]
    nums6 = [5, 4, 3, 2, 1]
    sortnumbers(nums6)
    assert nums6 == [1, 2, 3, 4, 5]
    nums7 = [1, 1, 1, 1]
    sortnumbers(nums7)
    assert nums7 == [1, 1, 1, 1]
    nums8 = [2, 3, 2, 1, 3]
    sortnumbers(nums8)
    assert nums8 == [1, 2, 2, 3, 3]
    nums10 = [0, 0, 0, 0]
    sortnumbers(nums10)
    assert nums10 == [0, 0, 0, 0]
    nums11 = [5, 3, 8, 6, 2]
    sortnumbers(nums11)
    assert nums11 == [2, 3, 5, 6, 8]
```

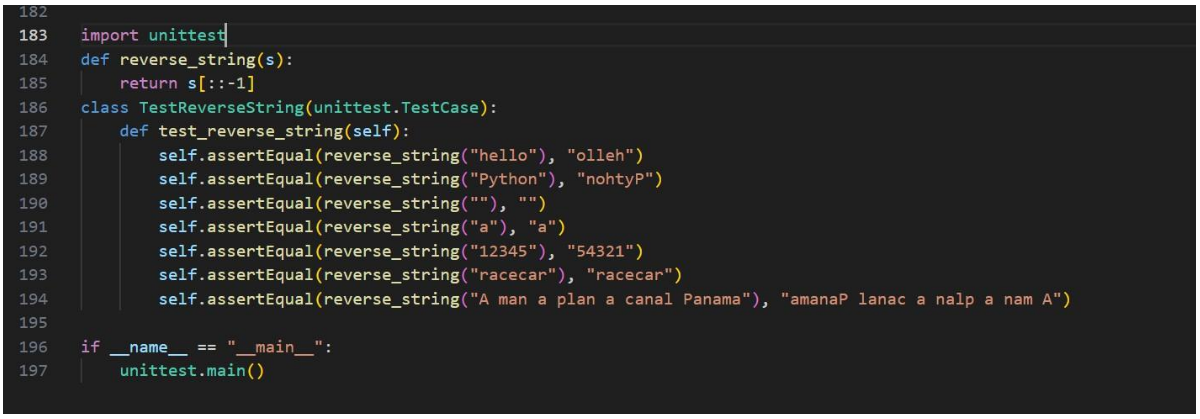
<u>Code:</u>

```
def sortnumbers(nums):
    n = len(nums)
    for i in range(n):
        for j in range(0, n-i-1):
            if nums[j] > nums[j+1]:
                nums[j], nums[j+1] = nums[j+1], nums[j]
def test_sortnumbers():
    nums1 = [5, 2, 9, 1, 5, 6]
    sortnumbers(nums1)
    assert nums1 == [1, 2, 5, 5, 6, 9]

    nums2 = [3, 0, -1, 8, 7]
    sortnumbers(nums2)
    assert nums2 == [-1, 0, 3, 7, 8]

    nums3 = [10]
    sortnumbers(nums3)
    assert nums3 == [10]

    nums4 = []
    sortnumbers(nums4)
    assert nums4 == []
    nums5 = [1, 2, 3, 4, 5]
    sortnumbers(nums5)
    assert nums5 == [1, 2, 3, 4, 5]
    nums6 = [5, 4, 3, 2, 1]
    sortnumbers(nums6)
    assert nums6 == [1, 2, 3, 4, 5]
    nums7 = [1, 1, 1, 1]
    sortnumbers(nums7)
    assert nums7 == [1, 1, 1, 1]
    nums8 = [2, 3, 2, 1, 3]
    sortnumbers(nums8)
    assert nums8 == [1, 2, 2, 3, 3]
    nums10 = [0, 0, 0, 0]
    sortnumbers(nums10)
    assert nums10 == [0, 0, 0, 0]
    nums11 = [5, 3, 8, 6, 2]
    sortnumbers(nums11)
    assert nums11 == [2, 3, 5, 6, 8]
    nums12 = [9, 7, 5, 3, 1]
    sortnumbers(nums12)
    assert nums12 == [1, 3, 5, 7, 9]

    print("All test cases passed!")
```
Output:

<u>Task 11 :</u>

Write a function ReverseString and validate its implementation

using 5 unittest test cases

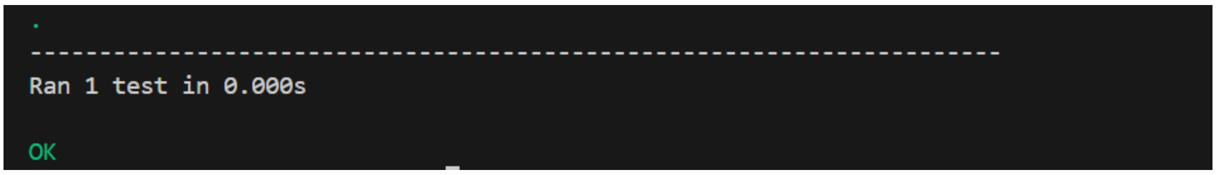


```
import unittest
def reverse_string(s):
    return s[::-1]
class TestReverseString(unittest.TestCase):
    def test_reverse_string(self):
        self.assertEqual(reverse_string("hello"), "olleh")
        self.assertEqual(reverse_string("Python"), "nohtyP")
        self.assertEqual(reverse_string(""), "")
        self.assertEqual(reverse_string("a"), "a")
        self.assertEqual(reverse_string("12345"), "54321")
        self.assertEqual(reverse_string("racecar"), "racecar")
        self.assertEqual(reverse_string("A man a plan a canal Panama"), "amanaP lanac a nalp a nam A")

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

code:

```
import unittest

def reverse_string(s):
    return s[::-1]
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")
        self.assertEqual(reverse_string("12345"), "54321")
        self.assertEqual(reverse_string("racecar"), "racecar")
        self.assertEqual(reverse_string("A man a plan a canal Panama"),
"amanaP lanac a nalp a nam A")

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

output:

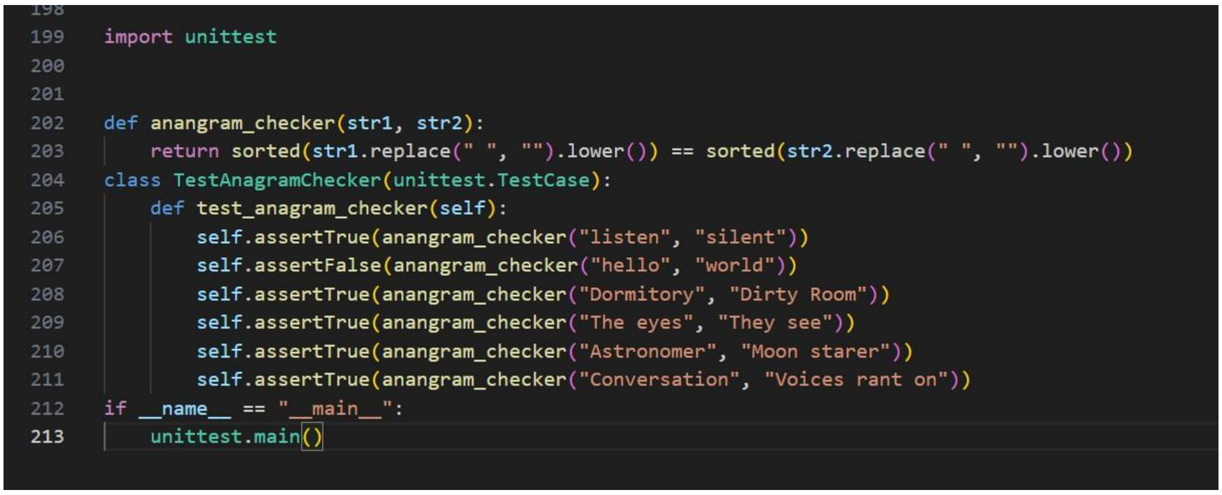


```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Task 12 :

Write a function AnagramChecker and validate its implementation

using 10 unittest test cases.

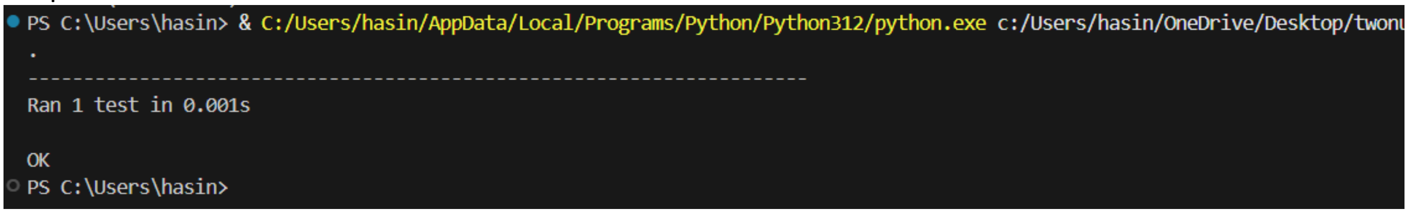


```python
import unittest


def anangram_checker(str1, str2):
    return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
class TestAnagramChecker(unittest.TestCase):
    def test_anagram_checker(self):
        self.assertTrue(anangram_checker("listen", "silent"))
        self.assertFalse(anangram_checker("hello", "world"))
        self.assertTrue(anangram_checker("Dormitory", "Dirty Room"))
        self.assertTrue(anangram_checker("The eyes", "They see"))
        self.assertTrue(anangram_checker("Astronomer", "Moon starer"))
        self.assertTrue(anangram_checker("Conversation", "Voices rant on"))
if __name__ == "__main__":
    unittest.main()
```

Code:

```python
import unittest

def anangram_checker(str1, str2):
    return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ",
"").lower())
class TestAnagramChecker(unittest.TestCase):
    def test_anagram_checker(self):
        self.assertTrue(anangram_checker("listen", "silent"))
        self.assertFalse(anangram_checker("hello", "world"))
        self.assertTrue(anangram_checker("Dormitory", "Dirty Room"))
        self.assertTrue(anangram_checker("The eyes", "They see"))
        self.assertTrue(anangram_checker("Astronomer", "Moon starer"))
        self.assertTrue(anangram_checker("Conversation", "Voices rant on"))
if __name___ == "__main__":
    unittest.main()
```

output:

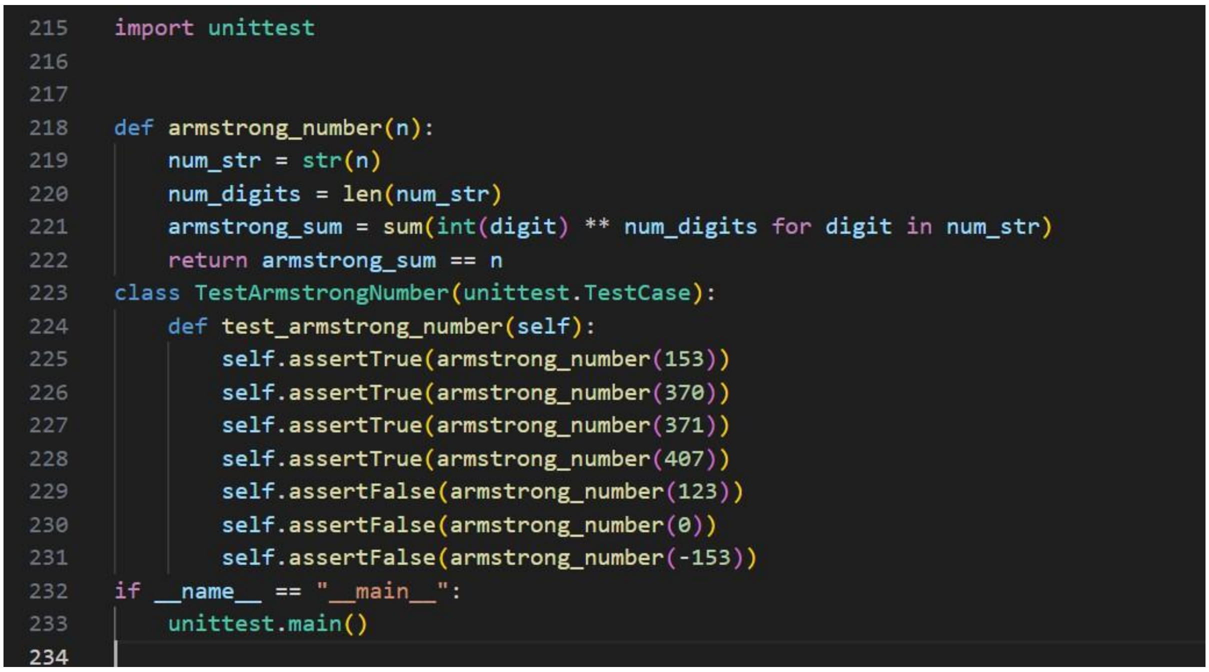


```
PS C:\Users\hasin> & C:/Users/hasin/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hasin/OneDrive/Desktop/twonu
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
PS C:\Users\hasin>
```

Task 13 :

Write a function ArmstrongChecker and validate its implementation

using 8 unittest test cases.

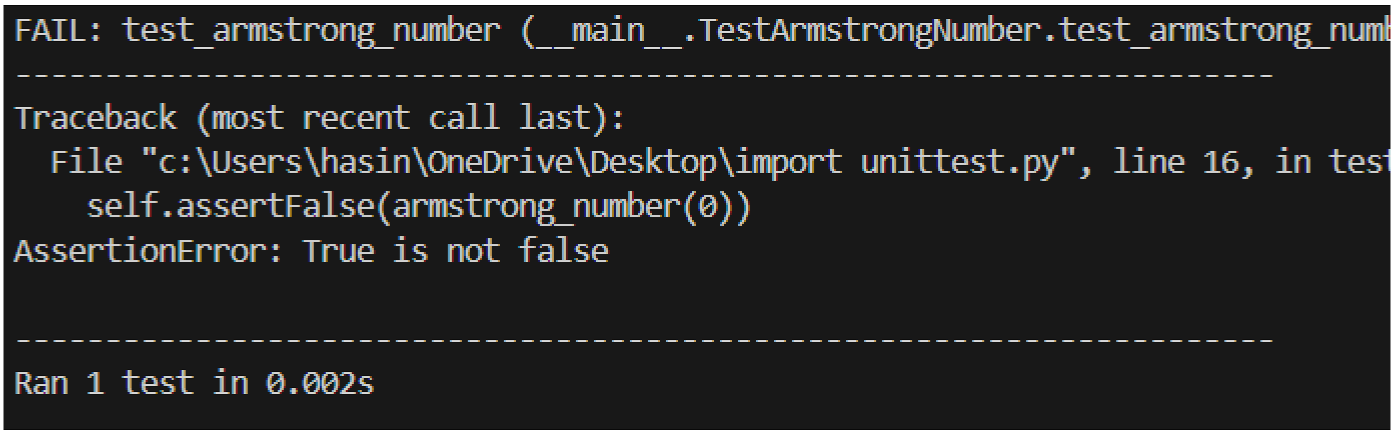


```python
import unittest


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
class TestArmstrongNumber(unittest.TestCase):
    def test_armstrong_number(self):
        self.assertTrue(armstrong_number(153))
        self.assertTrue(armstrong_number(370))
        self.assertTrue(armstrong_number(371))
        self.assertTrue(armstrong_number(407))
        self.assertFalse(armstrong_number(123))
        self.assertFalse(armstrong_number(0))
        self.assertFalse(armstrong_number(-153))
if __name__ == "__main__":
    unittest.main()

```

<u>Code:</u>

```python
import unittest

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
class TestArmstrongNumber(unittest.TestCase):
    def test_armstrong_number(self):
        self.assertTrue(armstrong_number(153))
        self.assertTrue(armstrong_number(370))
        self.assertTrue(armstrong_number(371))
        self.assertTrue(armstrong_number(407))
        self.assertFalse(armstrong_number(123))
        self.assertFalse(armstrong_number(0))
        self.assertFalse(armstrong_number(-153))
if __name__ == "__main__":
    unittest.main()
```

Output:



```
FAIL: test_armstrong_number (__main__.TestArmstrongNumber.test_armstrong_numb
----------------------------------------------------------------------
Traceback (most recent call last):
  File "c:\Users\hasin\OneDrive\Desktop\import unittest.py", line 16, in test
    self.assertFalse(armstrong_number(0))
AssertionError: True is not false

----------------------------------------------------------------------
Ran 1 test in 0.002s
```