Assignimant-8 (mon 16-02-26)

Hallticket no:2303A510A6

Batch no: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

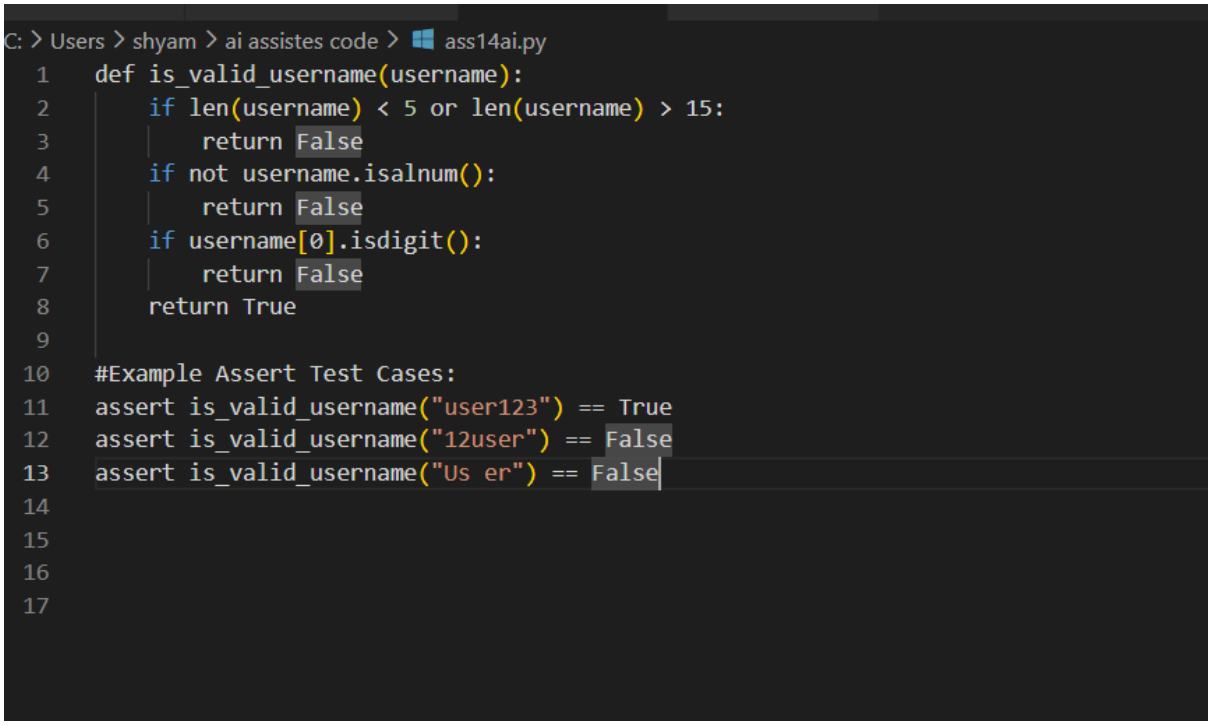
assert is_valid_username("12User") == False

assert is_valid_username("Us er") == False

Expected Output #1:

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

Code:



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

Code:

C: > Users > shyam > ai assistes code > ass14ai.py

```
def classify_value(x):
    if not isinstance(x, int):
        return "invaild"

    if x == 0:
        return "0"
    elif x % 2 == 0:
        return "Even"
    else :
        return "Odd"



# assert test cases
        assert classify_value(8) == "Even"
        assert classify_value(7) == "Odd"
        assert classify_value(0) == "0"
        assert classify_value("abc") == "invaild"
```

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

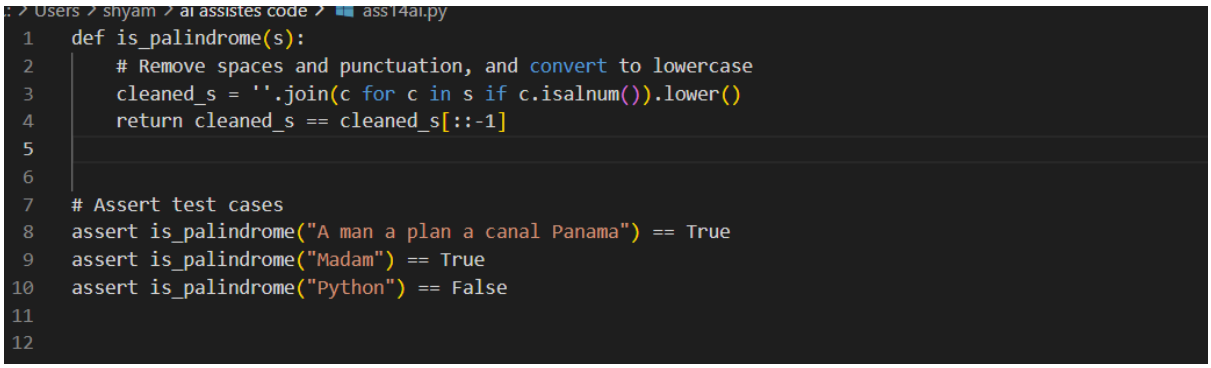
assert is_palindrome("A man a plan a canal Panama") == True

assert is_palindrome("Python") == False

Expected Output #3:

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

Code:



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 funct ion.

• 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

Code:

```python
def validate_email(email) :
    if email.count('@') != 1:
        return False

    local_part, domain_part = email.split('@')

    if not local_part or not domain_part:
        return False

    if '.' not in domain_part:
        return False

    return True

#assert test cases
assert validate_email("user@example.com") == True
assert validate_email("userexample.com") == False
assert validate_email("@gmail.com") == False
```

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.

Code:

```
def is_perfect_number(num):
    if num <= 1:
        return False

    divisors_sum = 0
    for i in range(1, num):
        if num % i == 0:
            divisors_sum += i

    return divisors_sum == num

#assert test cases
assert is_perfect_number(6) == True
assert is_perfect_number(10) == False
assert is_perfect_number(1) == False
assert is_perfect_number(-5) == False
assert is_perfect_number(28) == True
```

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.

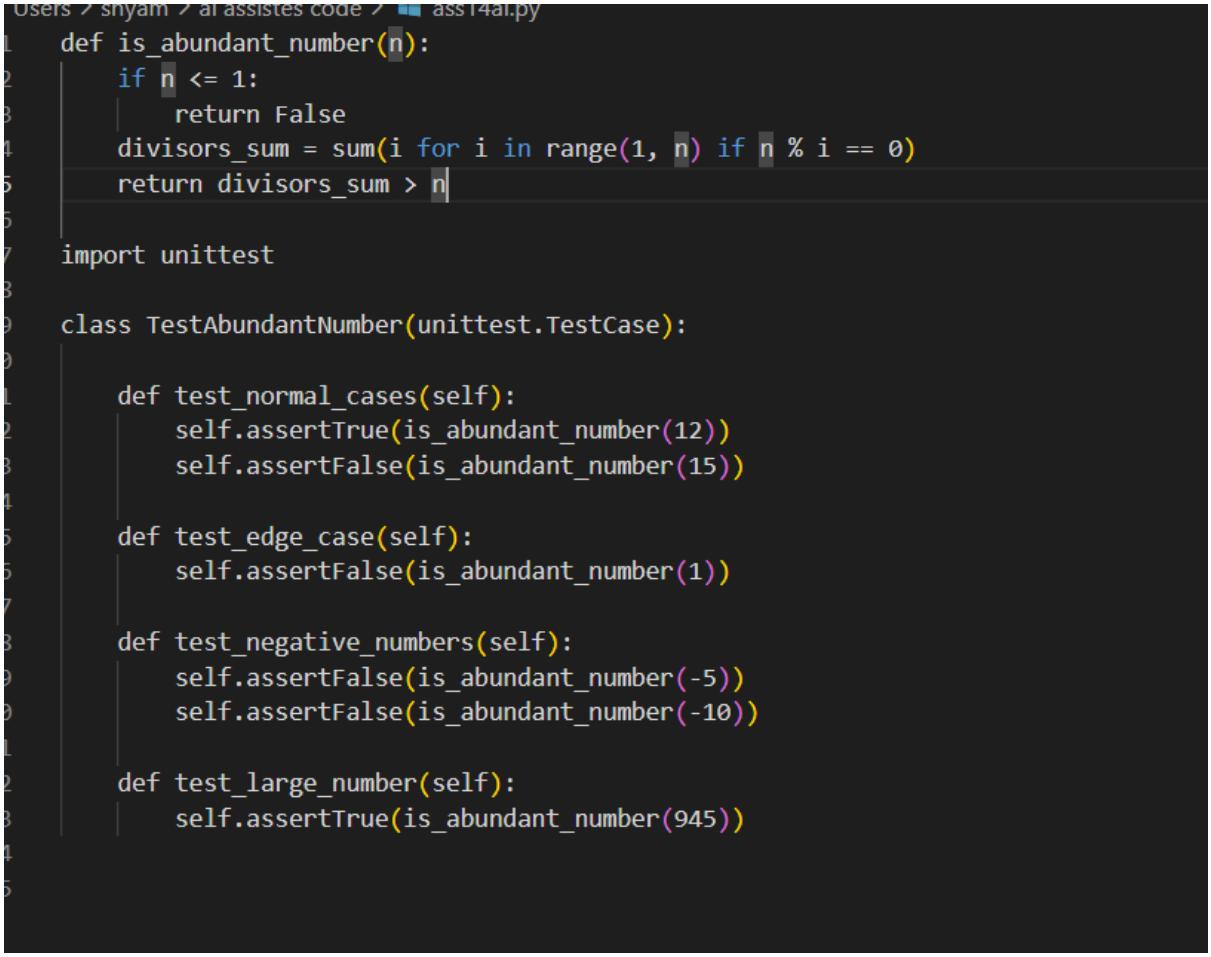
o Edge case: 1.

o Negative number case.

o Large case: 945.

Requirement: Validate correctness with unittest

Code:



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.

Code:

```python
def is_deficient_number(number):
    if number <= 1:
        return False
    divisors_sum = sum(i for i in range(1, number) if number % i == 0)
    return divisors_sum < number


import pytest

# normal cases
def test_deficient_numbers():
    assert is_deficient_number(8) == True
    assert is_deficient_number(12) == False

# edge case
def test_edge_case():
    assert is_deficient_number(1) == False

# negative number case
def test_negative_number():
    assert is_deficient_number(-5) == False

# large number case
def test_large_number():
    assert is_deficient_number(546) == True
```

Task 8 :

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

Code:

```python

def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False
import pytest
def test_leap_year():
    assert is_leap_year(2020) == True
    assert is_leap_year(2021) == False
    assert is_leap_year(1900) == False
    assert is_leap_year(2000) == True
    assert is_leap_year(2024) == True
    assert is_leap_year(2100) == False
    assert is_leap_year(2400) == True
    assert is_leap_year(1996) == True
    assert is_leap_year(1997) == False
    assert is_leap_year(2004) == True


```

Task 9 :

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


```python

def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    else:
        return False
import pytest
def test_leap_year():
    assert is_leap_year(2020) == True
    assert is_leap_year(2021) == False
    assert is_leap_year(1900) == False
    assert is_leap_year(2000) == True
    assert is_leap_year(2024) == True
    assert is_leap_year(2100) == False
    assert is_leap_year(2400) == True
    assert is_leap_year(1996) == True
    assert is_leap_year(1997) == False
    assert is_leap_year(2004) == True


```

Task 10 :

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

Code:

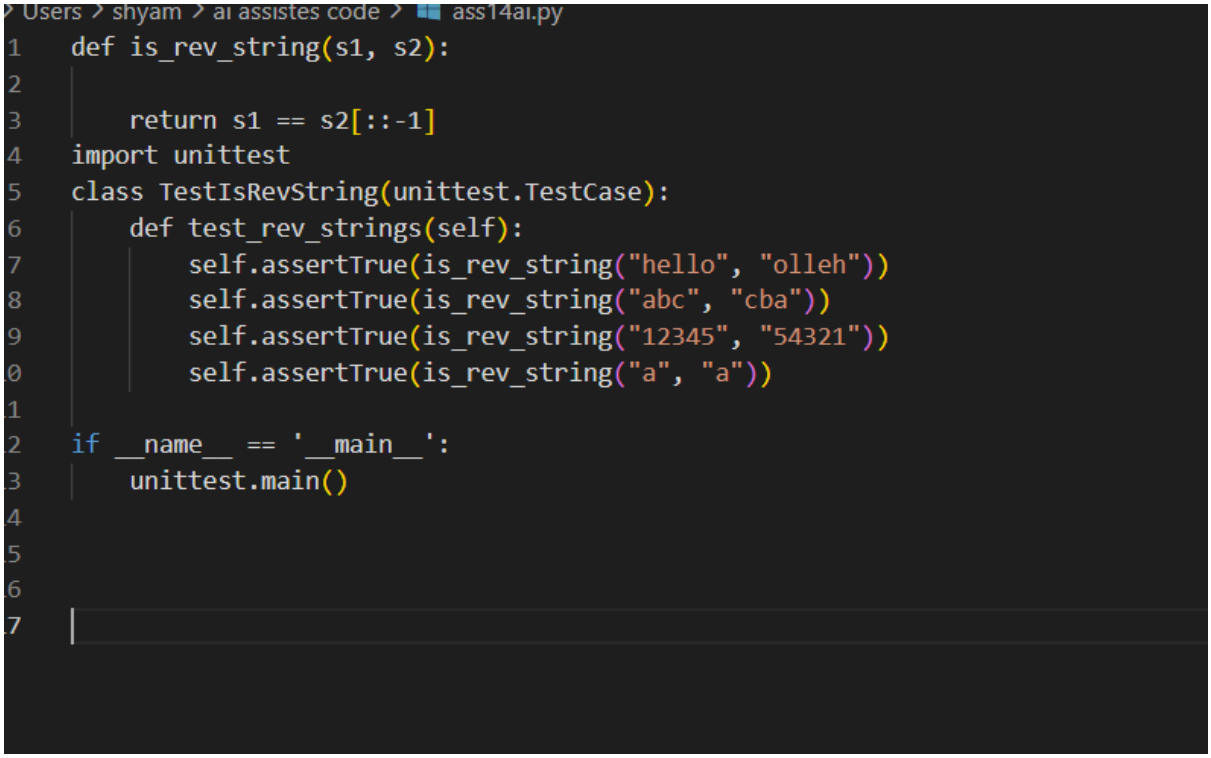
```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# 25 pytest cases
import pytest
def test_bubble_sort():
assert bubble_sort([64, 34, 25, 12, 22, 11, 90]) == [11, 12, 22, 25, 34, 64, 90]
    assert bubble_sort([5, 1, 4, 2, 8]) == [1, 2, 4, 5, 8]
    assert bubble_sort([3, 0, 2, 5, -1, 4, 1]) == [-1, 0, 1, 2, 3, 4, 5]
    assert bubble_sort([]) == []
    assert bubble_sort([1]) == [1]
    assert bubble_sort([2, 1]) == [1, 2]
    assert bubble_sort([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
    assert bubble_sort([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5]
    assert bubble_sort([1, 1, 1, 1]) == [1, 1, 1, 1]
    assert bubble_sort([3, 3, 2, 1]) == [1, 2, 3, 3]
    assert bubble_sort([0, 0, 0, 0]) == [0, 0, 0, 0]
    assert bubble_sort([-1, -3, -2, -5, -4]) == [-5, -4, -3, -2, -1]
    assert bubble_sort([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    assert bubble_sort([10, 9, 8, 7, 6, 5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    assert bubble_sort([1, 3, 5, 7, 9]) == [1, 3, 5, 7, 9]
    assert bubble_sort([9, 7, 5, 3, 1]) == [1, 3, 5, 7, 9]
    assert bubble_sort([1, 2, 2, 3, 4]) == [1, 2, 2, 3, 4]
    assert bubble_sort([4, 3, 2, 1, 1]) == [1, 1, 2, 3, 4]
    assert bubble_sort([1, 1, 1, 2, 2]) == [1, 1, 1, 2, 2]
    assert bubble_sort([2, 2, 1, 1, 1]) == [1, 1, 1, 2, 2]
    assert bubble_sort([1, 2, 3, 4, 5, 5, 4, 3, 2, 1]) == [1, 1, 2, 2, 3, 3, 4, 4, 5, 5]
    assert bubble_sort([5, 4, 3, 2, 1, 1, 2, 3, 4, 5]) == [1, 1, 2, 2, 3, 3, 4, 4, 5, 5]
    assert bubble_sort([100, 100, 100]) == [100, 100, 100]
    assert bubble_sort([3.2, 1.5, 2.8]) == [1.5, 2.8, 3.2]
```

Task 11 :

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

Code:


Users > shyam > ai assistes code > ass14ai.py
def is_rev_string(s1, s2):
return s1 == s2[::-1]
import unittest
class TestIsRevString(unittest.TestCase):
def test_rev_strings(self):
self.assertTrue(is_rev_string("hello", "olleh"))
self.assertTrue(is_rev_string("abc", "cba"))
self.assertTrue(is_rev_string("12345", "54321"))
self.assertTrue(is_rev_string("a", "a"))
if __name__ == '__main__':
unittest.main()

Task 12 :

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

Code:

```python
def is_anagram_checker(str1, str2):
    str1 = str1.replace(" ", "").lower()
    str2 = str2.replace(" ", "").lower()

    return sorted(str1) == sorted(str2)


import unittest

class TestAnagramChecker(unittest.TestCase):

    def test_anagrams(self):
        self.assertTrue(is_anagram_checker("listen", "silent"))
        self.assertTrue(is_anagram_checker("triangle", "integral"))
        self.assertTrue(is_anagram_checker("evil", "vile"))
        self.assertTrue(is_anagram_checker("Dormitory", "Dirty Room"))
        self.assertTrue(is_anagram_checker("The eyes", "They see"))

    def test_non_anagrams(self):
        self.assertFalse(is_anagram_checker("hello", "world"))
        self.assertFalse(is_anagram_checker("python", "java"))
        self.assertTrue(is_anagram_checker("anagram", "nagaram"))  # Fixed
        self.assertFalse(is_anagram_checker("test", "tseta"))
        self.assertFalse(is_anagram_checker("example", "samples"))

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

Task13 :Write a function ArmstrongChecker and validate its implementation

using 8 unittest test cases.

Code:

```python
def is_armstrong_number(num):
    sum = 0
    temp = num
    while temp > 0:
        digit = temp % 10
        sum += digit ** 3
        temp //= 10
    return num == sum
import unittest
class TestArmstrongNumber(unittest.TestCase):
    def test_armstrong_numbers(self):
        self.assertTrue(is_armstrong_number(153))
        self.assertTrue(is_armstrong_number(370))
        self.assertTrue(is_armstrong_number(371))
        self.assertTrue(is_armstrong_number(407))

    def test_non_armstrong_numbers(self):
        self.assertFalse(is_armstrong_number(100))
        self.assertFalse(is_armstrong_number(200))
        self.assertFalse(is_armstrong_number(300))
        self.assertFalse(is_armstrong_number(400))
if __name__ == '__main__':
    unittest.main()
```