ROLL NO:2303A510H4

BATCH:23

Lab Assignment 8

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:

```
def is_valid_username(username):     if len(username) < 5 or len(username) > 15:
return False     if not username[0].isalpha():
        return False     if not all(char.isalnum() or char == '_' for char in username):         return False     return True
assert is_valid_username("User123")==True assert is_valid_username("123User")==False assert is_valid_username("User 123")==False print(“Username
```

validation logic successfully passing all AI- generated test

cases.")

Output: 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.

Code:

```
def classify_value(x):
if isinstance(x, int):
if x == 0:
```

```
        return "Zero"
elif x % 2 == 0:
return "Even"       else:
        return "Odd"
else:
      return "Invalid Input" assert
classify_value(8) == "Even" assert
classify_value(7) == "Odd" assert
classify_value("abc") == "Invalid Input"
print("All test cases passed!") output : All test
cases passed.
```

 # 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.
```

```
import re def
is_palindrome(text):
   # Remove non-alphanumeric characters and convert to lowercase     cleaned_text =
re.sub(r'[^A-Za-z0-9]', '', text).lower()

   # Check if the cleaned text is equal to its reverse     return
cleaned_text == cleaned_text[::-1]
# Assert Test Cases assert is_palindrome("Madam") == True assert
is_palindrome("A man a plan a canal Panama") == True assert
is_palindrome("Python") == True assert is_palindrome("") == True
# Edge case: empty string assert is_palindrome("A") == True  #
Edge case: single character print("All test cases passed!") Output :
All test cases 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.
```

```
import re def
validate_email(email):
   # Regular expression for validating an Email     regex = r'^[a-
zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'


   # Check if the email matches the regex pattern     return
re.match(regex, email) is not None
# Assert Test Cases assert validate_email("user@example.com") == True assert
validate_email("userexample.com") == False assert validate_email("@gmail.com") ==
False assert validate_email("user@.com") == False  # Edge case: missing domain name
assert validate_email("user@com") == False  # Edge case: missing top-level domain
assert validate_email("user@domain.c") == False  # Edge case: top-level domain too
short assert validate_email("user@domain..com") == False  # Edge case: double dots in
domain name
print("All test cases passed!")  Output :

"C:/Program Files/Python312/python.exe"
"c:/Users/Ganne/OneDrive/Desktop/Ai_Assisted_ Coding/Wed.py/Assignment-8.py"
Traceback (most recent call last):
  File "c:\Users\Ganne\OneDrive\Desktop\Ai_Assisted_ Coding\Wed.py\Assignment-
8.py", line 64, in <module>     assert validate_email("user@domain..com") == False  #
Edge case: double dots in domain name
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError
```

 # 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.

def is_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 Test Cases assert

is_perfect_number(6) == True # Normal case assert

is_perfect_number(10) == False # Normal case assert

is_perfect_number(1) == False # Edge case assert

is_perfect_number(-5) == False # Negative number case assert

is_perfect_number(28) == True # Larger case

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

All test cases 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

```
import unittest  def
is_abundant_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 class
TestAbundantNumber(unittest.TestCase):     def
test_normal_cases(self):
       self.assertTrue(is_abundant_number(12))  # Normal case
self.assertFalse(is_abundant_number(15))  # Normal case


   def test_edge_case(self):
       self.assertFalse(is_abundant_number(1))  # Edge case


   def test_negative_case(self):        self.assertFalse(is_abundant_number(-5))  # Negative
number case


   def test_large_case(self):
       self.assertTrue(is_abundant_number(945))  # Large case if
__name__ == '__main__':
   unittest.main()  Output
:
 "C:/Program Files/Python312/python.exe"
"c:/Users/Ganne/OneDrive/Desktop/Ai_Assisted_ Coding/Wed.py/Assignment-8.py"
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s


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.

```
import pytest def
is_deficient_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 # Test Cases def
test_normal_cases():
   assert is_deficient_number(8) == True  # Normal case     assert
is_deficient_number(12) == False  # Normal case     assert
is_deficient_number(1) == False  # Edge case     assert
is_deficient_number(-5) == False  # Negative number case     assert
is_deficient_number(546) == True  # Large case if __name__ ==
'__main__':
   pytest.main() Output
:
```

All test cases passed!

 # Task 8 :

Write a function LeapYearChecker and validate its implementation

```
# using 10 pytest test cases


import re import pytest def LeapYearChecker(year):     if (year
% 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
return False # Test
Cases def
test_leap_years():
    assert LeapYearChecker(2020) == True  # Leap year     assert
LeapYearChecker(1900) == False  # Not a leap year     assert
LeapYearChecker(2000) == True  # Leap year

    assert LeapYearChecker(2021) == False  # Not a leap year
assert LeapYearChecker(2400) == True  # Leap year     assert
LeapYearChecker(1800) == False  # Not a leap year     assert
LeapYearChecker(1996) == True  # Leap year     assert
LeapYearChecker(2100) == False  # Not a leap year     assert
LeapYearChecker(1600) == True  # Leap year     assert
LeapYearChecker(2024) == True  # Future leap year print("All
test cases passed!") Output :
All test cases passed!
All test cases passed!
```

 # Task 9 :

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


import re import pytest def
SumOfDigits(number):
```

```
    return sum(int(digit) for digit in str(abs(number)) if digit.isdigit())
# Test Cases def
test_sum_of_digits():
    assert SumOfDigits(123) == 6  # Normal case     assert
SumOfDigits(-456) == 15  # Negative number case     assert
SumOfDigits(0) == 0  # Edge case: zero     assert
SumOfDigits(9999) == 36  # Large number case     assert
SumOfDigits(1001) == 2  # Case with zeros     assert
SumOfDigits(-789) == 24  # Negative number case     assert
SumOfDigits(12345) == 15  # Normal case print("All test cases
passed!") Output :
All test cased are passed!
```

 # Task 10 :

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

import re import
pytest def
SortNumbers(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 # Test Cases def
test_sort_numbers():
    assert SortNumbers([5, 2, 9, 1, 5, 6]) == [1, 2, 5, 5, 6, 9]  # Normal case     assert
SortNumbers([]) == []  # Edge case: empty list     assert SortNumbers([1]) == [1]  # Edge
```

case: single element assert SortNumbers([3, 3, 3]) == [3, 3, 3] # Case with duplicates

assert SortNumbers([-1, -5, -3]) == [-5, -3, -1] # Case with negative numbers assert

SortNumbers([0, -1, 1]) == [-1, 0, 1] # Case with zero and negative numbers assert

SortNumbers([10, 9, 8, 7]) == [7, 8, 9, 10] # Reverse sorted case assert

SortNumbers([2.5, 1.2, -0.5]) == [-0.5, 1.2, 2.5] # Case with floats assert

SortNumbers([1000000, -1000000]) == [-1000000, 1000000] # Case with large numbers

assert SortNumbers([5]*10) == [5]*10 # Case with all elements the same

assert SortNumbers([3.14]) == [3.14] # Edge case: single float element assert

SortNumbers([-2.5, -1.2]) == [-2.5, -1.2] # Case with negative floats assert

SortNumbers([0.0]) == [0.0] # Edge case: single zero float element assert

SortNumbers([1e-10, -1e-10]) == [-1e-10, 1e-10] # Case with very small numbers

assert SortNumbers([float('inf'), float('-inf')]) == [float('-inf'), float('inf')] # Case with infinity

assert SortNumbers([float('nan'), float('nan')]) == [float('nan'), float('nan')] # Case with NaN values

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

All test cases passed!

 # Task 11 :

Write a function ReverseString and validate its implementation

using 5 unittest test cases import unittest

def ReverseString(s): return s[::-1] class

TestReverseString(unittest.TestCase): def

test_reverse_string(self):

self.assertEqual(ReverseString("hello"), "olleh") # Normal case

self.assertEqual(ReverseString(""), "") # Edge case: empty string

self.assertEqual(ReverseString("a"), "a") # Edge case: single character

self.assertEqual(ReverseString("12345"), "54321") # Case with numbers

self.assertEqual(ReverseString("!@#$%"), "%$#@!") # Case with special characters if

__name__ == '__main__':

  unittest.main() Output

: "C:/Program

Files/Python312/python.

exe"

"c:/Users/Ganne/OneDrive/Desktop/Ai_Assisted_ Coding/Wed.py/Assignment-8.py"

.

Ran 1 test in 0.000s

OK

 # Task 12 :

Write a function AnagramChecker and validate its implementation

using 10 unittest test cases.

import unittest def

AnagramChecker(str1, str2):

  return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())

class TestAnagramChecker(unittest.TestCase): def test_anagram_checker(self):

    self.assertTrue(AnagramChecker("listen", "silent")) # Normal case

self.assertTrue(AnagramChecker("Triangle", "Integral")) # Case with different cases

self.assertFalse(AnagramChecker("hello", "world")) # Not anagrams

self.assertTrue(AnagramChecker("Dormitory", "Dirty Room")) # Case with spaces

self.assertFalse(AnagramChecker("abc", "def")) # Not anagrams

self.assertTrue(AnagramChecker("A gentleman", "Elegant man")) # Case with spaces and

different cases self.assertFalse(AnagramChecker("Clint Eastwood", "Old West Action"))

Not anagrams self.assertTrue(AnagramChecker("School master", "The classroom"))

Case with spaces and different cases print("All test cases passed!") Output :

All test cases passed!

 # Task 13 :

Write a function ArmstrongChecker and validate its implementation

using 8 unittest test cases.

import unittest 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 class

TestArmstrongChecker(unittest.TestCase): def

test_armstrong_checker(self):

    self.assertTrue(ArmstrongChecker(153)) # Normal case

self.assertTrue(ArmstrongChecker(370)) # Normal case

self.assertTrue(ArmstrongChecker(371)) # Normal case

self.assertFalse(ArmstrongChecker(123)) # Not an Armstrong number

self.assertTrue(ArmstrongChecker(0)) # Edge case: zero

self.assertTrue(ArmstrongChecker(1)) # Edge case: single digit

self.assertFalse(ArmstrongChecker(-153)) # Negative number case print("All

test cases passed!") Output :

All test cases passed!