AI ASSISTED CODING

ASSIGNMENT-8

Name: Harshitha Guda

H.T.No:2303A51102

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.

```
79  def is_valid_username(username):
80      if len(username) < 5 or len(username) > 15:
81          return False
82      if not username.isalnum():
83          return False
84      if username[0].isdigit():
85          return False
86      if " " in username:
87          return False
88      return True
89  assert is_valid_username("User123") == True
90  assert is_valid_username("12User") == False
91  assert is_valid_username("Us er") == False
92  assert is_valid_username("User") == False
```

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

```
95  def classify_value(x):
96      if x==0:
97          return "Zero"
98      if isinstance(x, (int, float)):
99          if x%2 == 0:
100             return "Even"
101         else:
102             return "Odd"
103     else:
104         return "Invalid Input"
105 assert classify_value(8) == "Even"
106 assert classify_value(7) == "Odd"
107 assert classify_value("abc") == "Invalid Input"
```

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(text):
    cleaned_text = ''.join(text.split()).lower()  # Remove spaces and convert to lowercase
    return cleaned_text == cleaned_text[::-1]  # Check if the cleaned text is the same as its reverse
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.

```python
def validate_email(email):
    if "@" not in email or "." not in email:
        return False
    if email.count("@") != 1:
        return False
    if email.startswith("@") or email.endswith("@"):
        return False
    if email.startswith(".") or email.endswith("."):
        return False
    return True

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.

```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(1) == False
assert perfect_number(0) == False

```

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

```python
#generate a python function abundant_number that checks if a number is abundant and validate the function using unit tests
def 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
# Unit tests for the abundant_number function
import unittest
class TestAbundantNumber(unittest.TestCase):
    def test_abundant_number(self):
        self.assertTrue(abundant_number(12))  # 1 + 2 + 3 + 4 + 6 = 16 > 12
    def test_non_abundant_number(self):
        self.assertFalse(abundant_number(28))  # 1 + 2 + 4 + 7 + 14 = 28 == 28
    def test_edge_cases(self):
        self.assertFalse(abundant_number(1))   # No proper divisors
        self.assertFalse(abundant_number(0))   # Not a valid input
    def test_negative_number(self):
        self.assertFalse(abundant_number(-5))  # Not a valid input
if __name__ == "__main__":
    unittest.main()  # Run the unit tests

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\gudah> & C:/Python314/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_8.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

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

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.

```python
#generate a python function number is deficient and validate the function using pytest
def 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
# Unit tests for the deficient_number function using pytest
def test_deficient_number():
    assert deficient_number(8) == True  # 1 + 2 + 4 = 7 < 8
def test_non_deficient_number():
    assert deficient_number(12) == False # 1 + 2 + 3 + 4 + 6 = 16 > 12
def test_edge_cases():
    assert deficient_number(1) == True
    assert deficient_number(0) == False   # Not a valid input
def test_negative_number():
    assert deficient_number(-5) == False  # Not a valid input

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\gudah\OneDrive\Documents\AIAC> python -m pytest Lab_assignment_8.py
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:/Python314/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_8.py
PS C:\Users\gudah\OneDrive\Documents\AIAC> python -m pytest Lab_assignment_8.py
============================= test session starts =============================
platform win32 -- Python 3.14.2, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\gudah\OneDrive\Documents\AIAC
plugins: anyio-4.12.1
collected 4 items

Lab_assignment_8.py ....

============================== 4 passed in 0.09s ==============================
```

Task 8 : Write a function LeapYearChecker and validate its implementation using 10 pytest test cases

```python
#Write a function LeapYearChecker and validate its implementation using 10 pytest test cases
def is_leap_year(year):
    if year % 4 == 0:
        if year % 100 == 0:
            return year % 400 == 0
        return True
    return False

def test_leap_year():
    assert is_leap_year(2024) == True   # Divisible by 4, not by 100
    assert is_leap_year(2023) == False   # Not divisible by 4
    assert is_leap_year(1900) == False   # Divisible by 100 but not by 400
    assert is_leap_year(2000) == True   # Divisible by 400
    assert is_leap_year(2100) == False   # Divisible by 100 but not by 400
    assert is_leap_year(2400) == True   # Divisible by 4, and also divisible by 400
    assert is_leap_year(1656) == True   # Divisible by 4, not by 100
    assert is_leap_year(1776) == True   # Divisible by 4, not by 100
    assert is_leap_year(1888) == True   # Divisible by 4, not by 100
    assert is_leap_year(1999) == False   # Not divisible by 4
```

Task 9 :Write a function SumOfDigits and validate its implementation using 7 pytest test cases.

```python
#Write a function SumOfDigits and validate its implementation using 7 pytest test cases.
def sum_of_digits(n):
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    return sum(int(digit) for digit in str(n))
def test_sum_of_digits():
    assert sum_of_digits(123) == 6   # 1 + 2 + 3 = 6
    assert sum_of_digits(0) == 0     # Sum of digits in 0 is 0
    assert sum_of_digits(999) == 27   # 9 + 9 + 9 = 27
    assert sum_of_digits(4567) == 22   # 4 + 5 + 6 + 7 = 22
    assert sum_of_digits(1001) == 2   # 1 + 0 + 0 + 1 = 2
    assert sum_of_digits(12345) == 15   # 1 + 2 + 3 + 4 + 5 = 15
    try:
        sum_of_digits(-123)   # Should raise ValueError for negative input
    except ValueError:
        pass

```

Task 10 : Write a function SortNumbers (implement bubble sort) and validate its implementation using 25 pytest test cases.

```python
#Write a function SortNumbers (implement bubble sort) and validate its implementation using 25 pytest test cases.
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
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([]) == []
    assert bubble_sort([1]) == [1]
    assert bubble_sort([2, 1]) == [1, 2]
    assert bubble_sort([3, 3, 3]) == [3, 3, 3]
    assert bubble_sort([-1, -5, -3]) == [-5, -3, -1]
    assert bubble_sort([0]) == [0]
    assert bubble_sort([10, -10]) == [-10, 10]
    assert bubble_sort([1.5, 0.5, -0.5]) == [-0.5, 0.5, 1.5]
    assert bubble_sort([1000000, -1000000]) == [-1000000, 1000000]
    assert bubble_sort([9, -9]) == [-9, 9]
    assert bubble_sort([2.71828]) == [2.71828]
    assert bubble_sort([-3.14]) == [-3.14]
    assert bubble_sort([1e-10]) == [1e-10]
    assert bubble_sort([-1e-10]) == [-1e-10]
    assert bubble_sort([123456789]) == [123456789]
```

```
    assert bubble_sort([1e-10]) == [1e-10]
    assert bubble_sort([-1e-10]) == [-1e-10]
    assert bubble_sort([123456789]) == [123456789]
    assert bubble_sort([-123456789]) == [-123456789]
    assert bubble_sort([0.00001]) == [0.00001]
    assert bubble_sort([-0.00001]) == [-0.00001]

EMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
\Users\gudah\OneDrive\Documents\AIAC> python -m pytest Lab_assignment_8.py
============================= test session starts ==============================
orm win32 -- Python 3.14.2, pytest-9.0.2, pluggy-1.6.0
orm win32 -- Python 3.14.2, pytest-9.0.2, pluggy-1.6.0
ir: C:\Users\gudah\OneDrive\Documents\AIAC
ns: anyio-4.12.1
cted 1 item

ssignment_8.py .

============================== 1 passed in 0.11s ===============================
\Users\gudah\OneDrive\Documents\AIAC>
```

Task 11 :Write a function ReverseString and validate its implementation using 5 unittest test cases

```python
#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")
        self.assertEqual(reverse_string("12345"), "54321")
if __name__ == "__main__":
    unittest.main()  # Run the unit tests

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\gudah> & C:/Python314/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_8.py
PS C:\Users\gudah> & C:/Python314/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_8.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

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

Task 12 : Write a function AnagramChecker and validate its implementation using 10 unittest test cases.

```python
#Write a function AnagramChecker and validate its implementation using 10 unittest test cases.
def are_anagrams(str1, str2):
    return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
import unittest
class TestAnagramChecker(unittest.TestCase):
    def test_anagrams(self):
        self.assertTrue(are_anagrams("listen", "silent"))
        self.assertTrue(are_anagrams("Triangle", "Integral"))
        self.assertTrue(are_anagrams("Dormitory", "Dirty Room"))
        self.assertFalse(are_anagrams("Hello", "World"))
        self.assertFalse(are_anagrams("Python", "Java"))
        self.assertTrue(are_anagrams("A gentleman", "Elegant man"))
        self.assertFalse(are_anagrams("Test", "Best"))
        self.assertTrue(are_anagrams("Clint Eastwood", "Old West Action"))
        self.assertTrue(are_anagrams("School master", "The classroom"))
if __name__ == "__main__":
    unittest.main()  # Run the unit tests

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\gudah> & C:/Python314/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_8.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

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

Task 13 : Write a function ArmstrongChecker and validate its implementation using 8 unittest test cases.

```python
def is_armstrong_number(n):
    if n < 0:
        return False
    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 TestArmstrongChecker(unittest.TestCase):
    def test_armstrong_numbers(self):
        self.assertTrue(is_armstrong_number(153))  # 1^3 + 5^3 + 3^3 = 153
        self.assertTrue(is_armstrong_number(370))  # 3^3 + 7^3 + 0^3 = 370
        self.assertTrue(is_armstrong_number(9474)) # 9^4 + 4^4 + 7^4 + 4^4 = 9474
        self.assertFalse(is_armstrong_number(123)) # Not an Armstrong number
        self.assertFalse(is_armstrong_number(-153)) #negative numbers cannot be Armstrong numbers
        self.assertTrue(is_armstrong_number(1))   # 1^1 = 1
        self.assertFalse(is_armstrong_number(10))  # Not an Armstrong number

if __name__ == "__main__":
    unittest.main()  # Run the unit tests
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\gudah> & C:/Python314/python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_8.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

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