

Assignment-8

2303A51272

A.Srivani

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.

The screenshot shows a code editor interface with the following details:

- Code Editor Area:** Displays the file `xyz.py` containing Python code for a `is_valid_username` function and three test cases using `assert` statements.
- Terminal Tab:** Shows the command line output of running the code. It includes two terminal sessions (PS) showing the execution of `xyz.py` and the resulting output: "All test cases for is_valid_username passed!"
- Status Bar:** Shows the current file is `xyz.py`, the line count is 20, column 1, and the character count is 4. It also indicates the file is saved in UTF-8 format with CRLF line endings, and the Python version is 3.14.0. A "Go Live" button is also present.

```
def is_valid_username(username):
    if len(username) < 5 or len(username) > 15:
        return False
    if not username[0].isalpha():
        return False
    for char in username:
        if not (char.isalnum() or char == '_'):
            return False
    return True

#test cases for the is_valid_username function
assert is_valid_username("User123") == True
assert is_valid_username("12User") == False
assert is_valid_username("Us er") == False
print("All test cases for is_valid_username passed!")
```

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:
 - If input is an integer, classify as "Even" or "Odd".
 - If input is 0, return "Zero".
 - 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.

```
1  def classify_value(x):  
2      if x < 0:  
3          return "Negative"  
4      elif x == 0:  
5          return "Zero"  
6      elif x%2 == 0:  
7          return "Even"  
8      else:  
9          return "Odd"  
10  
11  # Test cases for the classify_value function  
12  assert classify_value(8) == "Even"  
13  assert classify_value(7) == "Odd"  
14  assert classify_value("abc") == "Invalid Input"  
15  
16
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> & C:/Users/sriva/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/sriva/OneDrive/Documents/AI Assisted Code/xyz.py"  
Traceback (most recent call last):  
  File "c:/Users/sriva/OneDrive/Documents/AI Assisted Code/xyz.py", line 14, in <module>  
    assert classify_value("abc") == "Invalid Input"  
           ~~~~~~  
  File "c:/Users/sriva/OneDrive/Documents/AI Assisted Code/xyz.py", line 2, in classify_value  
    if x < 0:  
        ^^^^  
TypeError: '<' not supported between instances of 'str' and 'int'  
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code>
```

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:
 - Ignore case, spaces, and punctuation.
 - 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
```

The screenshot shows a terminal window with the following content:

```
1 def is_palindrome(text):  
2     cleaned_text = ''.join(char.lower() for char in text if char.isalnum())  
3     return cleaned_text == cleaned_text[::-1]  
4  
5 # Test cases for the is_palindrome function  
6 assert is_palindrome("Madam") == True  
7 assert is_palindrome("A man a plan a canal Panama") == True  
8 assert is_palindrome("Python") == False  
9 print("All test cases for is_palindrome passed!")  
10  
11  
12
```

Below the code, there are several terminal tabs:

- PROBLEMS
- OUTPUT
- DEBUG CONSOLE
- TERMINAL**
- PORTS

The TERMINAL tab contains the following output:

- PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code & C:/Users/sriva/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/sriva/xyz.py"
- All test cases for is_palindrome passed!
- PS C:\Users\sriva\OneDrive\Documents\AI Assisted 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 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
```

Email validation function passing all AI-generated test cases and handling edge cases correctly.

```

1 def validate_email(email):
2     if '@' not in email or '.' not in email:
3         return False
4     at_index = email.index('@')
5     dot_index = email.rindex('.')
6     if at_index < 1 or dot_index < at_index + 2 or dot_index >= len(email) - 1:
7         return False
8     return True
9
10 # Test cases for the validate_email function
11 assert validate_email("user@example.com") == True
12 assert validate_email("userexample.com") == False
13 assert validate_email("@gmail.com") == False
14 print("All test cases for validate_email passed!")
15

```

TERMINAL

```

PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> & C:/Users/sriva/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/sriva/OneDrive/Documents/AI Assisted xyz.py"
All test cases for validate_email passed!
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code>

```

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:
 - Normal case: 6 → True,
 - 10 → False.
 - Edge case: 1.
 - Negative number case.
 - Larger case: 28.
- Requirement: Validate correctness with assertions.

```

1 #generate a python code to display whether the given number is perfect number or not
2 def is_perfect_number(num):
3     if num < 1:
4         return False
5     divisors_sum = sum(i for i in range(1, num) if num % i == 0)
6     return divisors_sum == num
7
8 # Test cases for the is_perfect_number function
9 assert is_perfect_number(6) == True
10 assert is_perfect_number(10) == False
11 assert is_perfect_number(28) == True
12 assert is_perfect_number(1) == False
13 print("All test cases for is_perfect_number passed!")
14
15

```

TERMINAL

```

PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> & C:/Users/sriva/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/sriva/OneDrive/Documents/AI Assisted xyz.py"
All test cases for is_perfect_number passed!
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code>

```

Task 6 (Abundant Number Checker – Test Case Design)

- Function: Check if a number is abundant (sum of divisors > number).
- Test Cases to Design:
 - Normal case: 12 → True, 15 → False.
 - Edge case: 1.
 - Negative number case.

- o Large case: 945.

Requirement: Validate correctness with unit test

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

Task 9 :

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

```

xyz.py > test_sum_of_digits
1 def SumOfDigits(num):
2     return sum(int(digit) for digit in str(abs(num)))
3
4 def test_sum_of_digits():
5     assert SumOfDigits(123) == 6
6     assert SumOfDigits(-456) == 15
7     assert SumOfDigits(0) == 0
8     assert SumOfDigits(9999) == 36
9     assert SumOfDigits(-1001) == 2
10    print("All test cases for SumOfDigits passed!")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + × ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

```

PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> python -m pytest xyz.com
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> python -m pytest xyz.py
===== test session starts =====
platform win32 -- Python 3.14.0, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\sriva\OneDrive\Documents\AI Assisted Code
collected 1 item

xyz.py .

===== 1 passed in 0.02s =====
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> & C:/Users/sriva/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/sriva/OneDrive/Documents/AI Assisted Code/"

```

Task 10 :

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

```

1 def SortNumbers(numbers):
2     n = len(numbers)
3     for i in range(n):
4         for j in range(0, n-i-1):
5             if numbers[j] > numbers[j+1]:
6                 numbers[j], numbers[j+1] = numbers[j+1], numbers[j]
7     return numbers
8
9 def test_sort_numbers():
10    assert SortNumbers([5, 2, 9, 1, 5, 6]) == [1, 2, 5, 5, 6, 9]
11    assert SortNumbers([1]) == []
12    assert SortNumbers([3]) == [3]
13    assert SortNumbers([3, 2]) == [2, 3]
14    assert SortNumbers([1, 2, 3]) == [1, 2, 3]
15    assert SortNumbers([3, 2, 1]) == [1, 2, 3]
16    assert SortNumbers([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5]
17    assert SortNumbers([1, 1, 1, 1]) == [1, 1, 1, 1]
18    assert SortNumbers([9, 8, 7, 6, 5]) == [5, 6, 7, 8, 9]
19    assert SortNumbers([10, 9, 8, 7, 6]) == [6, 7, 8, 9, 10]
20    assert SortNumbers([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
21    assert SortNumbers([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5]
22    assert SortNumbers([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
23    assert SortNumbers([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5]
24    assert SortNumbers([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
25    print("All test cases for SortNumbers passed!")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + × ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

```

PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code & C:/Users/sriva/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/sriva/OneDrive/Documents/AI Assisted Code/xyz.py"
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> python -m pytest xyz.py
===== test session starts =====
platform win32 -- Python 3.14.0, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\sriva\OneDrive\Documents\AI Assisted Code
collected 1 item

xyz.py .

===== 1 passed in 0.03s =====
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code>

```

Task 11 :

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



```
xyz.py > ...
1 def ReverseString(s):
2     return s[::-1]
3
4 import unittest
5
6 class TestReverseString(unittest.TestCase):
7     def test_reverse_string(self):
8         self.assertEqual(ReverseString("Hello"), "olleH")
9         self.assertEqual(ReverseString("Python"), "nohtyP")
10        self.assertEqual(ReverseString(""), "")
11        self.assertEqual(ReverseString("A"), "A")
12        self.assertEqual(ReverseString("12345"), "54321")
13
14 if __name__ == '__main__':
15     unittest.main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

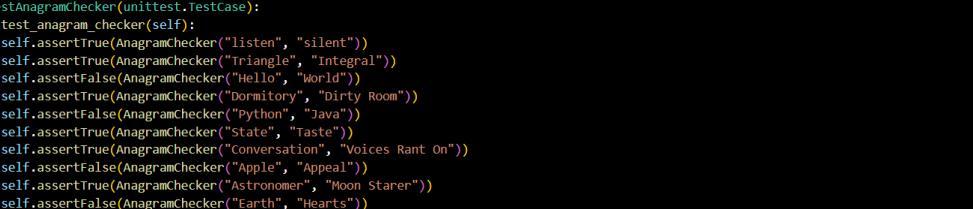
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> & C:/Users/sriva/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/sriva/OneDrive/Documents/AI Assisted Code/xyz.py"
.

Ran 1 test in 0.002s

OK
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code>

Task 12 :

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



```
xyz.py > ...
1 def AnagramChecker(str1, str2):
2     |     return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
3 import unittest
4 class TestAnagramChecker(unittest.TestCase):
5     def test_anagram_checker(self):
6         self.assertTrue(AnagramChecker("listen", "silent"))
7         self.assertTrue(AnagramChecker("triangle", "integral"))
8         self.assertFalse(AnagramChecker("Hello", "World"))
9         self.assertTrue(AnagramChecker("dormitory", "dirty Room"))
10        self.assertFalse(AnagramChecker("Python", "Java"))
11        self.assertTrue(AnagramChecker("State", "Taste"))
12        self.assertTrue(AnagramChecker("Conversation", "Voices Rant On"))
13        self.assertFalse(AnagramChecker("Apple", "Appeal"))
14        self.assertTrue(AnagramChecker("Astronomer", "Moon Starer"))
15        self.assertFalse(AnagramChecker("Earth", "Hearts"))
16 if __name__ == '__main__':
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - ×
● PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> & C:/Users/sriva/AppData/Local/Programs/Python/Python314/python.exe "c:/users/sriva/OneDrive/Documents/AI Assisted Code/xyz.py"
.
-----
Ran 1 test in 0.000s
OK
○ PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code>
```

Task 13 :

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

