

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.

Prompt:

```
""" E:\AI_ASSISTED_CODING
write a function to validate a username based on the following criteria:
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
"""
```

Code and output:

```
def is_valid_username(username):
    if len(username) < 5 or len(username) > 15:
        return False
    if not username.isalnum():
        return False
    if username[0].isdigit():
        return False
    if ' ' in username:
        return False
    return True
# Test cases
```

```
19     assert is_valid_username("user123") == True
20     assert is_valid_username("123user") == False
21     assert is_valid_username("user name") == False
22     assert is_valid_username("user") == False
23
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\AI_ASSISTED_CODING> python demo12.py
PS E:\AI ASSISTED CODING>
```

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".

Prompt:

```
2 """
3     write a python to classify a given input as "Even", "Odd", "Zero", or "Invalid Input"
4     based on the following criteria:
5     o If input is an integer, classify as "Even" or "Odd".
6     o If input is 0, return "Zero".
7     o If input is non-numeric, return "Invalid Input"
8 """
```

Code and ouput:

```

9  def classify_value(x):
10     if isinstance(x, int):
11         if x == 0:
12             return "Zero"
13         elif x % 2 == 0:
14             return "Even"
15         else:
16             return "Odd"
17     else:
18         return "Invalid Input"
19 # Test cases
20 assert classify_value(4) == "Even"
21 assert classify_value(7) == "Odd"
22 assert classify_value(0) == "Zero"
23 assert classify_value("abc") == "Invalid Input"

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- PS E:\AI_ASSISTED_CODING> **python demo12.py**
- ❖ PS E:\AI_ASSISTED_CODING>

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.

Prompt:

```

"""
write a function to check if a given string is a palindrome or not based on the following
criteria:
o Ignore case, spaces, and punctuation.
o Handle edge cases such as empty strings and single characters.
"""

```

Code and ouput:

```
8  def is_palindrome(text):
9      cleaned_text = ''.join(char.lower() for char in text if char.isalnum())
10     return cleaned_text == cleaned_text[::-1]
11 # Test cases
12 assert is_palindrome("Madam") == True
13 assert is_palindrome("A man a plan a canal Panama") == True
14 assert is_palindrome("") == True
15 assert is_palindrome("x") == True
16 assert is_palindrome("Hello") == False
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS E:\AI_ASSISTED_CODING> python demo12.py

PS E:\AI_ASSISTED_CODING>

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.

Prompt:

```
"""
write a function to validate an email address based on the following criteria:
o Must contain @ and .
o Must not start or end with special characters.
o Should handle invalid formats gracefully.
"""
```

Code and ouput:

```

8     def validate_email(email):
9         if '@' not in email or '.' not in email:
10            return False
11        if email[0] in '@.' or email[-1] in '@.':
12            return False
13        return True
14 # Test cases
15 assert validate_email("user@example.com") == True
16 assert validate_email("userexample.com") == False
17 assert validate_email("@gmail.com") == False
18 assert validate_email("user@.com") == False
19 assert validate_email("user@examplecom") == False
20

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

File "E:\AI_ASSISTED_CODING\demo12.py", line 18, in <module>
    assert validate_email("user@.com") == False
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.

Prompt:

```

"""
write a function to check if a given number is a perfect number or not based on the following
| criteria:
• Test Cases to Design:
o Normal case: 6 → True, 10 → False.
o Edge case: 1.
o Negative number case.
o Larger case: 28.
"""

```

Code and output:

```
11  def is_perfect_number(n):
12      if n < 1:
13          return False
14      divisors_sum = sum(i for i in range(1, n) if n % i == 0)
15      return divisors_sum == n
16
17  # Test cases
18  assert is_perfect_number(6) == True
19  assert is_perfect_number(10) == False
20  assert is_perfect_number(1) == False
21  assert is_perfect_number(-5) == False
22  assert is_perfect_number(28) == True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\AI_ASSISTED_CODING> python demo12.py
● PS E:\AI_ASSISTED_CODING> python demo12.py
❖ PS E:\AI_ASSISTED_CODING> []
```

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.
 - Large case: 945.

Requirement: Validate correctness with unittest

Prompt:

```
"""
write a function to check if a given number is abundant or not based on the following criteria
using unittest for testing:
• Test Cases to Design:
◦ Normal case: 12 → True, 15 → False.
◦ Edge case: 1.
◦ Negative number case.
◦ Large case: 945
"""
```

Code and output:

```

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 TestIsAbundantNumber(unittest.TestCase):
    def test_abundant_numbers(self):
        self.assertTrue(is_abundant_number(12))
        self.assertFalse(is_abundant_number(15))
    def test_edge_case(self):
        self.assertFalse(is_abundant_number(1))
    def test_negative_number(self):
        self.assertFalse(is_abundant_number(-5))
    def test_large_case(self):
        self.assertTrue(is_abundant_number(945))
if __name__ == '__main__':
    unittest.main()

```

● PS E:\AI_ASSISTED_CODING> python demo12.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:
 - Normal case: 8 → True, 12 → False.
 - Edge case: 1.
 - Negative number case.
 - Large case: 546.

Requirement: Validate correctness with pytest.

Prompt:

```

"""
write a function to check if a number is deficient (sum of divisors < number)
| using pytest for testing:
• Test Cases to Design:
o Normal case: 8 → True, 12 → False.
o Edge case: 1.
o Negative number case.
o Large case: 546.
"""

```

Code and output:

```

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
def test_is_deficient_number():
    assert is_deficient_number(8) == True
    assert is_deficient_number(12) == False
    assert is_deficient_number(1) == False
    assert is_deficient_number(-5) == False
    assert is_deficient_number(546) == True

```

```

=====
 FAILURES =====
----- test_is_deficient_number -----
def test_is_deficient_number():
    assert is_deficient_number(8) == True
    assert is_deficient_number(12) == False
>     assert is_deficient_number(1) == False
E     assert True == False
E     +  where True = is_deficient_number(1)

demo12.py:19: AssertionError
=====
 short test summary info =====
FAILED demo12.py::test_is_deficient_number - assert True == False
=====
 1 failed in 0.18s =====

```

Task 8 :

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

Prompt:

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

Code and output:

```
def is_leap_year(year):
    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False

def test_is_leap_year():
    assert is_leap_year(2020) == True
    assert is_leap_year(1900) == False
    assert is_leap_year(2000) == True
    assert is_leap_year(2021) == False
    assert is_leap_year(2400) == True
    assert is_leap_year(1800) == False
    assert is_leap_year(1996) == True
    assert is_leap_year(2100) == False
```

```
22     assert is_leap_year(2100) == False
23     assert is_leap_year(1600) == True
24     assert is_leap_year(2024) == True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

collected 1 item

demo12.py .

[100%]

Task 9 :

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

Prompt:

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

Code and output:

```

3 def sum_of_digits(n):
4     return sum(int(digit) for digit in str(abs(n)) if digit.isdigit())
5 def test_sum_of_digits():
6     assert sum_of_digits(123) == 6
7     assert sum_of_digits(-456) == 15
8     assert sum_of_digits(0) == 0
9     assert sum_of_digits(789) == 24
10    assert sum_of_digits(1001) == 2
11    assert sum_of_digits(-999) == 27
12    assert sum_of_digits(12345) == 15

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS pwsh - AI_ASSISTED_CODING + ↻ ⌂ ⌂ ⌂ ⌂ ⌂

```

rootdir: E:\AI_ASSISTED_CODING
collected 1 item

demo12.py .

=====
===== 1 passed in 0.05s =====

```

Task 10 :

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

Prompt:

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

Code and ouput:

```

def sort_numbers(numbers):
    n = len(numbers)
    for i in range(n):
        for j in range(0, n-i-1):
            if numbers[j] > numbers[j+1]:
                numbers[j], numbers[j+1] = numbers[j+1], numbers[j]
    return numbers
def test_sort_numbers():
    assert sort_numbers([5, 2, 9, 1, 5, 6]) == [1, 2, 5, 5, 6, 9]
    assert sort_numbers([]) == []
    assert sort_numbers([1]) == [1]
    assert sort_numbers([3, 2]) == [2, 3]
    assert sort_numbers([-10, -1, 0]) == [-1, 0, 10]
    assert sort_numbers([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5]
    assert sort_numbers([1.5, 0.5, -0.5]) == [-0.5, 0.5, 1.5]
    assert sort_numbers([100, 50, 25]) == [25, 50, 100]
    assert sort_numbers([-3, -1, -2]) == [-3, -2, -1]
    assert sort_numbers([0]) == [0]
    assert sort_numbers([7, 7, 7]) == [7, 7, 7]
    assert sort_numbers([9, -9]) == [-9, 9]

```

```
23     assert sort_numbers([2.2, 3.3]) == [2.2, 3.3]
24     assert sort_numbers([-1.1, -2.2]) == [-2.2, -1.1]
25     assert sort_numbers([0.0]) == [0.0]
26     assert sort_numbers([5, 3, 8, 1]) == [1, 3, 5, 8]
27     assert sort_numbers([10, 9, 8, 7]) == [7, 8, 9, 10]
28     assert sort_numbers([1, 2, 3, 4, 5]) == [1, 2, 3, 4, 5]
29     assert sort_numbers([5, 5, 5, 5]) == [5, 5, 5, 5]
30     assert sort_numbers([-5, -4, -3, -2, -1]) == [-5, -4, -3, -2, -1]
31     assert sort_numbers([0, -1, 1]) == [-1, 0, 1]
32     assert sort_numbers([3, 1, 2, 3]) == [1, 2, 3, 3]
33     assert sort_numbers([1000, 500, 100]) == [100, 500, 1000]
34     assert sort_numbers([-10, 0, 10]) == [-10, 0, 10]
```

Task 11 :

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

Prompt:

```
import unittest  
  
#Write a function ReverseString and validate its implementation using 5 unittest test cases  
def ReverseString(s):
```

Code and output:

```
2 import unittest
3 def reverse_string(s):
4     return s[::-1]
5 class TestReverseString(unittest.TestCase):
6     def test_reverse_string(self):
7         self.assertEqual(reverse_string("hello"), "olleh")
8         self.assertEqual(reverse_string("Python"), "nohtyP")
9         self.assertEqual(reverse_string(""), "")
10        self.assertEqual(reverse_string("a"), "a")
11        self.assertEqual(reverse_string("12345"), "54321")
12 if __name__ == '__main__':
13     unittest.main()
```

.

Ran 1 test in 0.000s

OK

Task 12 :

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

Prompt:

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

Code and ouput:

```
4  def are_anagrams(str1, str2):
5      return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
6  class TestAnagramChecker(unittest.TestCase):
7      def test_anagrams(self):
8          self.assertTrue(are_anagrams("listen", "silent"))
9          self.assertTrue(are_anagrams("Triangle", "Integral"))
10         self.assertFalse(are_anagrams("hello", "world"))
11         self.assertTrue(are_anagrams("Dormitory", "Dirty Room"))
12         self.assertFalse(are_anagrams("Python", "Java"))
13         self.assertTrue(are_anagrams("Astronomer", "Moon starer"))
14         self.assertFalse(are_anagrams("Test", "Best"))
15         self.assertTrue(are_anagrams("The eyes", "They see"))
16         self.assertFalse(are_anagrams("Anagram", "Nag a ram!"))
17         self.assertTrue(are_anagrams("Conversation", "Voices rant on"))
18     if __name__ == '__main__':
19         unittest.main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS pwsh - AI_ASSISTED_CODING + ▾

```
Ran 1 test in 0.000s
Ran 1 test in 0.000s
```

Task 13 :

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

Prompt:

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

Code and ouput:

```
import unittest
def is_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 TestArmstrongChecker(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(9474))
    def test_non_armstrong_numbers(self):
        self.assertFalse(is_armstrong_number(123))
        self.assertFalse(is_armstrong_number(10))
        self.assertFalse(is_armstrong_number(0))
        self.assertFalse(is_armstrong_number(-153))
if __name__ == '__main__':
    unittest.main()
```

```
=====
FAIL: test_non_armstrong_numbers (__main__.TestArmstrongChecker.test_non_armstrong_numbers)
-----
Traceback (most recent call last):
  File "e:\AI_ASSISTED_CODING\demo12.py", line 18, in test_non_armstrong_numbers
    self.assertFalse(is_armstrong_number(0))
AssertionError: True is not false
-----
Ran 2 tests in 0.002s
FAILED (failures=1)
```