AI Assisted Coding

Assignment 8

Name : Maripelli Chodhitha

HT No : 2303A52039

Batch : 31

Question 1: Username Validator

The function `is_valid_username()` checks whether a given username meets a set of rules. It returns `True` only when all conditions are satisfied, and `False` the moment any rule is broken.

It checks:

- Length is between 5 and 15 characters.
- First character is a letter.
- No spaces.
- All characters are only letters or digits.

Code:

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

```
assert is_valid_username('user1') == True
assert is_valid_username('1user') == False
assert is_valid_username('user_name') == False

print('All test cases passed!')
```

Output:

```
All test cases passed!
```

Question 2: Even–Odd & Type Classification

The function `classify_value()` determines what category a given input belongs to.

It checks:

- Input must be an `int` or `float` and not a `bool`.
- If the value is `0`, return `'Zero'`.
- Otherwise check parity using modulo.

Code:

```
def classify_value(x):
    if not isinstance(x, (int, float)) or isinstance(x, boo
l):
        return 'Invalid Input'
    if x == 0:
        return 'Zero'
    elif int(x) % 2 == 0:
        return 'Even'
    else:
        return 'Odd'

assert classify_value(0) == 'Zero'
assert classify_value(2) == 'Even'
assert classify_value(3) == 'Odd'
assert classify_value('i') == 'Invalid Input'

print('All test cases passed!')
```

Output:

```
All test cases passed!
```

Question 3: Palindrome Checker

The function `is_palindrome()` checks whether a piece of text reads the same forwards and backwards, ignoring case, spaces, and punctuation.

Code:

```
import string

def is_palindrome(text):
    if not text:
        return True
    cleaned = ''.join(char.lower() for char in text if char.i
salnum())
    if len(cleaned) <= 1:
        return True
    return cleaned == cleaned[::-1]

assert is_palindrome('') == True
assert is_palindrome('A man a plan a canal Panama') == True
assert is_palindrome("No 'x' in Nixon") == True

print('All test cases passed!')
```

Output:

```
All test cases passed!
```

Question 4: Email ID Validation

The function `validate_email()` checks if an email address follows the correct format.

Rules enforced:

- Must be a non-empty string.
- Must not contain spaces.
- Must contain exactly one `@`.
- Local part and domain must be non-empty.
- Local part and domain cannot start or end with `.`.
- Domain must contain at least one `.`.
- TLD must be at least 2 characters.

Code:

```
def validate_email(email):
    if not isinstance(email, str) or not email:
        return False
    if ' ' in email:
        return False
    if email.count('@') != 1:
        return False

    local, domain = email.split('@')
    if not local or not domain:
        return False

    if local.startswith('.') or local.endswith('.'):
        return False
    if domain.startswith('.') or domain.endswith('.'):
        return False
    if '.' not in domain:
        return False
    if len(domain.split('.')[-1]) < 2:
        return False

    return True

assert validate_email('user@example.com') == True
assert validate_email('user@.example.com') == False
assert validate_email('user@example.') == False
assert validate_email('user@exam ple.com') == False
assert validate_email('user@example.c') == False

print('All test cases passed!')
```

Output:

```
All test cases passed!
```

Question 5: Perfect Number Checker

A perfect number is one whose proper divisors (all divisors excluding the number itself) add up to exactly the number.

Code:

```
def is_perfectnumber(n):
    if n <= 1:
        return False

    sum_of_divisors = 1
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            sum_of_divisors += i
            if i != n // i:
                sum_of_divisors += n // i

    return sum_of_divisors == n
```

```
assert is_perfectnumber(6) == True
assert is_perfectnumber(28) == True
assert is_perfectnumber(12) == False
assert is_perfectnumber(-1) == False

print('All test cases passed!')
```

Output:

```
All test cases passed!
```

Question 6: Abundant Number Checker

An abundant number is one where the sum of its proper divisors is greater than the number itself.

Code:

```
def abundant(n):
    if n <= 1:
        return False

    sum_of_divisors = 0
    for i in range(1, n):
        if n % i == 0:
            sum_of_divisors += i

    return sum_of_divisors > n



import unittest

class TestAbundant(unittest.TestCase):
    def test_abundant_normal_true(self):
        self.assertTrue(abundant(12))
    def test_abundant_normal_false(self):
        self.assertFalse(abundant(15))

    def test_abundant_edge_one(self):
        self.assertFalse(abundant(1))

    def test_abundant_negative(self):
        self.assertFalse(abundant(-12))

    def test_abundant_large(self):
        self.assertTrue(abundant(945))


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

Output:

```
.....
---------------------------------------------------------------
---------
Ran 5 tests in 0.000s

OK
```

Question 7: Deficient Number Checker

A deficient number is one where the sum of its proper divisors is less than the number.

Code:

```
def Deficient(n):
    if n <= 1:
        return n == 1
    sum_of_divisors = 1
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            sum_of_divisors += i
            if i != n // i:
                sum_of_divisors += n // i

    return sum_of_divisors < n



def test_deficient():
    assert Deficient(8) == True
    assert Deficient(15) == True



def test_edge_cases():
    assert Deficient(1) == True



def test_negative():
    assert Deficient(-5) == False



def test_large():
    assert Deficient(28) == False
```

Output:

```
python -m pytest 7.py
collected 4 items

7.py ....                 [100%]

4 passed in 0.02s
```

Question 8: Leap Year Checker

A leap year follows two rules:

- Divisible by 4 but not by 100, **or**
- Divisible by 400.

Code:

```
def LeapYearChecker(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 ==
0):
        return True
    return False



def test_leap_year_checker():
    assert LeapYearChecker(2020) == True
    assert LeapYearChecker(1900) == False
    assert LeapYearChecker(2000) == True
    assert LeapYearChecker(0) == True
    assert LeapYearChecker(1) == False
    assert LeapYearChecker(4) == True
    assert LeapYearChecker(100) == False
    assert LeapYearChecker(-4) == True
    assert LeapYearChecker(-100) == False
    assert LeapYearChecker(-400) == True
```

Output:

```
python -m pytest 8.py
collected 1 item

8.py .                    [100%]

1 passed in 0.02s
```

Question 9: Sum of Digits

The function `SumOfDigits()` extracts and adds each individual digit of a number.

Code:

```
def SumOfDigits(n):
    n = abs(n)
    total = 0
    while n > 0:
        digit = n % 10
        total += digit
        n //= 10
    return total


def test_sum_of_digits():
    assert SumOfDigits(123) == 6
    assert SumOfDigits(456) == 15
    assert SumOfDigits(0) == 0
    assert SumOfDigits(9) == 9
    assert SumOfDigits(123456789) == 45
    assert SumOfDigits(-123) == 6
    assert SumOfDigits(-987) == 24
```

Output:

```
python -m pytest 9.py
collected 1 item

9.py .                       [100%]

1 passed in 0.02s
```

Question 10: Sort Numbers (Bubble Sort)

Bubble Sort is a comparison-based sorting algorithm. In each pass, adjacent elements are compared and swapped if they are in the wrong order.

Code:

```
def SortNumbers_bubble(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_normal_unsorted_array():
    assert SortNumbers_bubble([5, 2, 9, 1, 5, 6]) == [1, 2,
5, 5, 6, 9]

# ... (25 total test cases)
```

Output:

```
python -m pytest 10.py
collected 25 items

10.py .........................     [100%]

25 passed in 0.10s
```

Question 11: Reverse String

The function `ReverseString()` uses Python slicing `[::-1]` to reverse a string.

Code:

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

```
import unittest

class TestReverseString(unittest.TestCase):
    def test_simple_cases(self):
        self.assertEqual(ReverseString('hello'), 'olleh')

    def test_empty_string(self):
        self.assertEqual(ReverseString(''), '')

    def test_single_character(self):
        self.assertEqual(ReverseString('a'), 'a')

    def test_palindrome(self):
        self.assertEqual(ReverseString('madam'), 'madam')

    def test_with_spaces(self):
        self.assertEqual(ReverseString('hello world'), 'dlrow
olleh')


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

Output:

```
.....
-------------------------------------------------------------
---------
Ran 5 tests in 0.000s

OK
```

Question 12: Anagram Checker

Two words are anagrams if they contain the same letters arranged differently.

Code:

```
import unittest

def AnagramChecker(word1, word2):
    return sorted(word1.lower().replace(' ', '')) == sorted(w
ord2.lower().replace(' ', ''))


class TestAnagramChecker(unittest.TestCase):
    def test_simple_anagrams(self):
        self.assertTrue(AnagramChecker('listen', 'silent'))

    def test_non_anagrams(self):
        self.assertFalse(AnagramChecker('hello', 'world'))

    # ... (10 total test cases)


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

Output:

```
..........
-------------------------------------------------------------
---------
Ran 10 tests in 0.000s

OK
```

Question 13: Armstrong Number Checker

An Armstrong number is one where the sum of each digit raised to the power of the total number of digits equals the number itself.

Code:

```
def ArmstrongChecker(num):
    if num < 0:
        return False

    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 and num != 0


import unittest

class TestArmstrongChecker(unittest.TestCase):
    def test_armstrong_numbers(self):
        self.assertTrue(ArmstrongChecker(153))
        self.assertTrue(ArmstrongChecker(370))
        self.assertTrue(ArmstrongChecker(371))
        self.assertTrue(ArmstrongChecker(407))

    def test_non_armstrong_numbers(self):
        self.assertFalse(ArmstrongChecker(123))
        self.assertFalse(ArmstrongChecker(0))
        self.assertFalse(ArmstrongChecker(-153))

    def test_single_digit_armstrong_numbers(self):
        self.assertTrue(ArmstrongChecker(1))


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

Output:

```
...
-------------------------------------------------------------
---------
Ran 3 tests in 0.000s

OK
```