Assignment 8.4

2403A52089

Batch 04

- **Email Validator**:

  Task: to create a email validator.

  Prompt: Create test cases to validate an email address (e.g., contains @ and .com); Use AI assistance to implement the validate_email() function.

  Code:

```python
import re

def validate_email(email):
    """
    Validate an email address format.

    This function checks if an email address has a valid format by ensuring:
    - Contains exactly one @ symbol
    - Has a valid local part (before @)
    - Has a valid domain part (after @)
    - Domain has a valid extension (like .com, .org, etc.)

    Args:
        email (str): The email address to validate

    Returns:
        bool: True if the email format is valid, False otherwise

    Examples:
        >>> validate_email("user@example.com")
        True
        >>> validate_email("invalid-email")
        False
        >>> validate_email("user@domain")
        False
    """
    # Check if email is empty or None
    if not email or not isinstance(email, str):
        return False

    # Strip whitespace
    email = email.strip()

    # Check if email is empty after stripping
    if not email:
        return False

    # Check for exactly one @ symbol
    if email.count('@') != 1:
        return False

    # Split email into local and domain parts
    local_part, domain_part = email.split('@')

    # Check if local part is empty
    if not local_part:
        return False

    # Check if domain part is empty
    if not domain_part:
        return False

    # Check for invalid characters in local part
    # Local part should not start or end with a dot
    if local_part.startswith('.') or local_part.endswith('.'):
        return False

    # Check for consecutive dots in local part
    if '..' in local_part:
        return False

    # Check for spaces in local part
    if ' ' in local_part:
        return False

    # Check for spaces in domain part
    if ' ' in domain_part:
        return False
```

Name: B.Nishant
Enrol : 2403A52089
Batch : 04

```
    # Check if domain has at least one dot (for extension)
    if '.' not in domain_part:
        return False

    # Check if domain doesn't start or end with a dot
    if domain_part.startswith('.') or domain_part.endswith('.'):
        return False

    # Check for consecutive dots in domain
    if '..' in domain_part:
        return False

    # Check if extension is at least 2 characters long
    extension = domain_part.split('.')[-1]
    if len(extension) < 2:
        return False

    # Check if extension contains only letters
    if not extension.isalpha():
        return False

    # Additional regex check for more comprehensive validation
    # This pattern ensures the email follows a more strict format
    email_pattern = r'^[a-zA-Z0-9._+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

    if not re.match(email_pattern, email):
        return False

    return True

# Alternative implementation using only regex (more concise but less readable)
def validate_email_regex_only(email):
    """
    Alternative email validation using only regex.
    This is more concise but less readable and debuggable.
    """
    if not email or not isinstance(email, str):
        return False

    email = email.strip()
    if not email:
        return False

    # Comprehensive regex pattern for email validation
    pattern = r'^[a-zA-Z0-9._+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return bool(re.match(pattern, email))
```

Test:

```
import unittest

from email_validator import validate_email

class TestEmailValidation(unittest.TestCase):
    """Test cases for the email validation function."""

    def test_valid_emails(self):
        """Test valid email addresses."""
        # Standard valid emails
        self.assertTrue(validate_email("user@example.com"))
        self.assertTrue(validate_email("test@domain.org"))
        self.assertTrue(validate_email("admin@company.net"))
        self.assertTrue(validate_email("info@website.co.uk"))

        # Emails with numbers and special characters
        self.assertTrue(validate_email("user123@example.com"))
```

Name: B.Nishant
Enrol : 2403A52089
Batch : 04

```
        self.assertTrue(validate_email("test.email@domain.com"))
        self.assertTrue(validate_email("user+tag@example.com"))
        self.assertTrue(validate_email("user_name@example.com"))

        # Short domain names
        self.assertTrue(validate_email("a@b.co"))
        self.assertTrue(validate_email("test@example.co"))

        # Long but valid emails
        self.assertTrue(validate_email("very.long.email.address@very.long.domain.name.com"))

    def test_invalid_emails_missing_at(self):
        """Test emails missing @ symbol."""
        self.assertFalse(validate_email("userexample.com"))
        self.assertFalse(validate_email("test.domain.com"))
        self.assertFalse(validate_email("invalid-email"))
        self.assertFalse(validate_email("justtext"))

    def test_invalid_emails_missing_domain(self):
        """Test emails missing domain part."""
        self.assertFalse(validate_email("user@"))
        self.assertFalse(validate_email("test@"))
        self.assertFalse(validate_email("@domain.com"))
        self.assertFalse(validate_email("@"))

    def test_invalid_emails_missing_extension(self):
        """Test emails missing .com or similar extension."""
        self.assertFalse(validate_email("user@domain"))
        self.assertFalse(validate_email("test@example"))
        self.assertFalse(validate_email("admin@company"))

    def test_invalid_emails_multiple_at(self):
        """Test emails with multiple @ symbols."""
        self.assertFalse(validate_email("user@@example.com"))
        self.assertFalse(validate_email("test@domain@com"))
        self.assertFalse(validate_email("user@example@domain.com"))

    def test_invalid_emails_invalid_characters(self):
        """Test emails with invalid characters."""
        self.assertFalse(validate_email("user name@example.com"))
        self.assertFalse(validate_email("user@exam ple.com"))
        self.assertFalse(validate_email("user@example.c om"))
        self.assertFalse(validate_email("user@example..com"))
        self.assertFalse(validate_email("user@@example.com"))

    def test_invalid_emails_empty_or_whitespace(self):
        """Test empty or whitespace-only emails."""
        self.assertFalse(validate_email(""))
        self.assertFalse(validate_email("   "))
        self.assertFalse(validate_email(" "))
        self.assertFalse(validate_email("\t"))
        self.assertFalse(validate_email("\n"))

    def test_invalid_emails_edge_cases(self):
```

Name: B.Nishant
Enrol : 2403A52089
Batch : 04

```
        """Test edge cases and malformed emails."""
        self.assertFalse(validate_email("@.com"))
        self.assertFalse(validate_email("user@.com"))
        self.assertFalse(validate_email("@example.com"))
        self.assertFalse(validate_email("user@example."))
        self.assertFalse(validate_email(".user@example.com"))
        self.assertFalse(validate_email("user.@example.com"))
        self.assertFalse(validate_email("user@example.com."))

    def test_invalid_emails_wrong_extension(self):
        """Test emails with invalid or missing extensions."""
        self.assertFalse(validate_email("user@example"))
        self.assertFalse(validate_email("user@example."))
        self.assertFalse(validate_email("user@example.c"))
        self.assertFalse(validate_email("user@example.123"))

    def test_valid_emails_various_extensions(self):
        """Test valid emails with different extensions."""
        self.assertTrue(validate_email("user@example.com"))
        self.assertTrue(validate_email("user@example.org"))
        self.assertTrue(validate_email("user@example.net"))
        self.assertTrue(validate_email("user@example.edu"))
        self.assertTrue(validate_email("user@example.gov"))
        self.assertTrue(validate_email("user@example.co.uk"))
        self.assertTrue(validate_email("user@example.info"))
        self.assertTrue(validate_email("user@example.biz"))

if __name__ == '__main__':
    # Run the tests
    unittest.main(verbosity=2)
```

OP:

```
PS D:\Nishant\AI_Assisted_Coding\lab8.4> & C:/Python313/python.exe d:/Nishant/AI_Assisted_Coding/lab8.4/test_email_validation.py
test_invalid_emails_edge_cases (__main__.TestEmailValidation.test_invalid_emails_edge_cases)
Test edge cases and malformed emails. ... ok
test_invalid_emails_empty_or_whitespace (__main__.TestEmailValidation.test_invalid_emails_empty_or_whitespace)
Test empty or whitespace-only emails. ... ok
test_invalid_emails_invalid_characters (__main__.TestEmailValidation.test_invalid_emails_invalid_characters)
Test emails with invalid characters. ... ok
test_invalid_emails_missing_at (__main__.TestEmailValidation.test_invalid_emails_missing_at)
Test emails missing @ symbol. ... ok
test_invalid_emails_missing_domain (__main__.TestEmailValidation.test_invalid_emails_missing_domain)
Test emails missing domain part. ... ok
test_invalid_emails_missing_extension (__main__.TestEmailValidation.test_invalid_emails_missing_extension)
Test emails missing .com or similar extension. ... ok
test_invalid_emails_multiple_at (__main__.TestEmailValidation.test_invalid_emails_multiple_at)
Test emails with multiple @ symbols. ... ok
test_invalid_emails_wrong_extension (__main__.TestEmailValidation.test_invalid_emails_wrong_extension)
Test emails with invalid or missing extensions. ... ok
test_valid_emails (__main__.TestEmailValidation.test_valid_emails)
Test valid email addresses. ... ok
test_valid_emails_various_extensions (__main__.TestEmailValidation.test_valid_emails_various_extensions)
Test valid emails with different extensions. ... ok

----------------------------------------------------------------------
Ran 10 tests in 0.007s

OK
PS D:\Nishant\AI_Assisted_Coding\lab8.4>
```

Name: B.Nishant
Enrol : 2403A52089
Batch : 04

➢ Max of three:

Task: To find the max of three:

Prompt: Write test cases for a function that returns the maximum of three numbers. Prompt Copilot/Cursor to write the logic based on tests

```
"""
Implement max_of_three(a, b, c) to return the maximum of three numbers.
Follow the tests in test_max_of_three.py.
Constraints:
- Inputs can be int or float, including negatives.
- Handle duplicates (e.g., (5,5,3) -> 5, (4,4,4) -> 4).
- Must work for special floats like inf and -inf.
"""


def max_of_three(a, b, c):
    """Return the maximum among a, b, and c.

    Works with integers and floats, including special float values like
    positive and negative infinity. Duplicate values are handled naturally.
    """
    return max(a, b, c)
```

Test:

```
import unittest
from max_of_three import max_of_three


class TestMaxOfThree(unittest.TestCase):
    def test_all_positive_distinct(self):
        self.assertEqual(max_of_three(1, 2, 3), 3)
        self.assertEqual(max_of_three(10, 5, 7), 10)
        self.assertEqual(max_of_three(3, 9, 6), 9)

    def test_with_negatives(self):
        self.assertEqual(max_of_three(-1, -2, -3), -1)
        self.assertEqual(max_of_three(-10, 0, -5), 0)
        self.assertEqual(max_of_three(-7, -1, -4), -1)

    def test_duplicates(self):
        self.assertEqual(max_of_three(5, 5, 3), 5)
        self.assertEqual(max_of_three(2, 7, 7), 7)
        self.assertEqual(max_of_three(4, 4, 4), 4)

    def test_mixed_int_float(self):
        self.assertEqual(max_of_three(1.5, 2, 1.9), 2)
        self.assertEqual(max_of_three(2.2, 2.21, 2.19), 2.21)
        self.assertEqual(max_of_three(-1.1, -1.2, -1.05), -1.05)

    def test_edge_values(self):
        self.assertEqual(max_of_three(float('inf'), 1, 2), float('inf'))
        self.assertEqual(max_of_three(float('-inf'), -1, -2), -1)


if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Name: B.Nishant
Enrol : 2403A52089
Batch : 04

Op:

```
PS D:\Nishant\AI_Assisted_Coding\lab8.4> & C:/Python313/python.exe d:/Nishant/AI_Assisted_Coding/lab8.4/test_max_of_three.py
test_all_positive_distinct (__main__.TestMaxOfThree.test_all_positive_distinct) ... ok
test_duplicates (__main__.TestMaxOfThree.test_duplicates) ... ok
test_edge_values (__main__.TestMaxOfThree.test_edge_values) ... ok
test_mixed_int_float (__main__.TestMaxOfThree.test_mixed_int_float) ... ok
test_with_negatives (__main__.TestMaxOfThree.test_with_negatives) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.003s

OK
PS D:\Nishant\AI_Assisted_Coding\lab8.4>
```

➢ Shopping cart:

Task: to create a code for shopping cart.

Prompt: Use TDD to write a shopping cart class with methods to add, remove, and get total price.First write tests for each method, then generate code using AI.

CODE:

```python
class ShoppingCart:
    """Simple shopping cart to add, remove items and compute total price."""

    def __init__(self):
        # items: dict[name] = {"price": float, "quantity": int}
        self._items = {}

    def add_item(self, name, price, quantity=1):
        if price <= 0:
            raise ValueError("price must be positive")
        if quantity <= 0:
            raise ValueError("quantity must be positive")

        if name in self._items:
            # Accumulate quantity, keep latest price for simplicity
            self._items[name]["quantity"] += quantity
            self._items[name]["price"] = price
        else:
            self._items[name] = {"price": float(price), "quantity": int(quantity)}

    def remove_item(self, name, quantity=1):
        if quantity <= 0:
            raise ValueError("quantity must be positive")

        if name not in self._items:
            return  # noop

        current_qty = self._items[name]["quantity"]
        if quantity >= current_qty:
            del self._items[name]
        else:
            self._items[name]["quantity"] = current_qty - quantity
```

Name: B.Nishant
Enrol : 2403A52089
Batch : 04

```
    def get_total_price(self):
        total = 0.0
        for item in self._items.values():
            total += item["price"] * item["quantity"]
        return round(total, 2)
```

Test:

```
import unittest

from shopping_cart import ShoppingCart


class TestShoppingCart(unittest.TestCase):
    def setUp(self):
        self.cart = ShoppingCart()

    def test_initial_total_is_zero(self):
        self.assertEqual(self.cart.get_total_price(), 0.0)

    def test_add_single_item(self):
        self.cart.add_item("apple", price=1.20, quantity=1)
        self.assertEqual(self.cart.get_total_price(), 1.20)

    def test_add_multiple_quantities(self):
        self.cart.add_item("banana", price=0.50, quantity=4)
        self.assertEqual(self.cart.get_total_price(), 2.00)

    def test_add_multiple_different_items(self):
        self.cart.add_item("milk", price=2.50, quantity=1)
        self.cart.add_item("bread", price=1.75, quantity=2)
        self.assertEqual(self.cart.get_total_price(), 2.50 + 1.75 * 2)

    def test_add_same_item_accumulates_quantity(self):
        self.cart.add_item("eggs", price=0.25, quantity=6)
        self.cart.add_item("eggs", price=0.25, quantity=6)
        self.assertEqual(self.cart.get_total_price(), 0.25 * 12)

    def test_remove_item_reduces_total(self):
        self.cart.add_item("cheese", price=3.00, quantity=1)
        self.cart.add_item("ham", price=2.00, quantity=2)
        self.cart.remove_item("ham", quantity=1)
        self.assertEqual(self.cart.get_total_price(), 3.00 + 2.00)

    def test_remove_item_entirely_when_quantity_exceeds(self):
        self.cart.add_item("juice", price=1.50, quantity=2)
        self.cart.remove_item("juice", quantity=5)
        self.assertEqual(self.cart.get_total_price(), 0.0)

    def test_remove_nonexistent_item_is_noop(self):
        self.cart.add_item("yogurt", price=1.00, quantity=3)
```

Name: B.Nishant
Enrol : 2403A52089
Batch : 04

```
        self.cart.remove_item("not-in-cart", quantity=1)
        self.assertEqual(self.cart.get_total_price(), 3.00)

    def test_invalid_add_raises(self):
        with self.assertRaises(ValueError):
            self.cart.add_item("apple", price=-1.0, quantity=1)
        with self.assertRaises(ValueError):
            self.cart.add_item("apple", price=1.0, quantity=0)

    def test_invalid_remove_raises(self):
        with self.assertRaises(ValueError):
            self.cart.remove_item("apple", quantity=0)


if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Op:

```
test_add_multiple_different_items (__main__.TestShoppingCart.test_add_multiple_different_items) ... ok
test_add_multiple_quantities (__main__.TestShoppingCart.test_add_multiple_quantities) ... ok
test_add_same_item_accumulates_quantity (__main__.TestShoppingCart.test_add_same_item_accumulates_quantity) ... ok
test_add_single_item (__main__.TestShoppingCart.test_add_single_item) ... ok
test_initial_total_is_zero (__main__.TestShoppingCart.test_initial_total_is_zero) ... ok
test_invalid_add_raises (__main__.TestShoppingCart.test_invalid_add_raises) ... ok
test_invalid_remove_raises (__main__.TestShoppingCart.test_invalid_remove_raises) ... ok
test_remove_item_entirely_when_quantity_exceeds (__main__.TestShoppingCart.test_remove_item_entirely_when_quantity_exceeds) .
.. ok
test_remove_item_reduces_total (__main__.TestShoppingCart.test_remove_item_reduces_total) ... ok
test_remove_nonexistent_item_is_noop (__main__.TestShoppingCart.test_remove_nonexistent_item_is_noop) ... ok

----------------------------------------------------------------------
Ran 10 tests in 0.003s
```

Name: B.Nishant
Enrol : 2403A52089
Batch : 04

➢ Square:

Task: To find the square of the numbers.

Prompt: Write a test case to check if a function returns the square of a number. Then write the function

CODE:

```
def square(number):
    """
    Calculate the square of a number.

    Args:
        number (int or float): The number to be squared

    Returns:
        int or float: The square of the input number

    Examples:
        >>> square(5)
        25
        >>> square(-3)
        9
        >>> square(2.5)
        6.25
    """
    return number ** 2
```

Test:

```
import unittest

from square_function import square

class TestSquareFunction(unittest.TestCase):
    """Test cases for the square function."""

    def test_positive_integers(self):
        """Test square function with positive integers."""
        self.assertEqual(square(1), 1)
        self.assertEqual(square(2), 4)
        self.assertEqual(square(5), 25)
        self.assertEqual(square(10), 100)

    def test_negative_integers(self):
        """Test square function with negative integers."""
        self.assertEqual(square(-1), 1)
        self.assertEqual(square(-2), 4)
        self.assertEqual(square(-5), 25)
        self.assertEqual(square(-10), 100)
```

Name: B.Nishant
Enrol : 2403A52089
Batch : 04

```python
    def test_zero(self):
        """Test square function with zero."""
        self.assertEqual(square(0), 0)

    def test_positive_floats(self):
        """Test square function with positive floating point numbers."""
        self.assertAlmostEqual(square(1.5), 2.25, places=5)
        self.assertAlmostEqual(square(2.5), 6.25, places=5)
        self.assertAlmostEqual(square(0.5), 0.25, places=5)

    def test_negative_floats(self):
        """Test square function with negative floating point numbers."""
        self.assertAlmostEqual(square(-1.5), 2.25, places=5)
        self.assertAlmostEqual(square(-2.5), 6.25, places=5)
        self.assertAlmostEqual(square(-0.5), 0.25, places=5)

    def test_large_numbers(self):
        """Test square function with large numbers."""
        self.assertEqual(square(1000), 1000000)
        self.assertEqual(square(-1000), 1000000)

    def test_small_numbers(self):
        """Test square function with very small numbers."""
        self.assertAlmostEqual(square(0.001), 0.000001, places=8)
        self.assertAlmostEqual(square(-0.001), 0.000001, places=8)

if __name__ == '__main__':
    # Run the tests
    unittest.main(verbosity=2)
```

OP:

```
test_large_numbers (__main__.TestSquareFunction.test_large_numbers)
Test square function with large numbers. ... ok
test_negative_floats (__main__.TestSquareFunction.test_negative_floats)
Test square function with negative floating point numbers. ... ok
test_negative_integers (__main__.TestSquareFunction.test_negative_integers)
Test square function with negative integers. ... ok
test_positive_floats (__main__.TestSquareFunction.test_positive_floats)
Test square function with positive floating point numbers. ... ok
test_positive_integers (__main__.TestSquareFunction.test_positive_integers)
Test square function with positive integers. ... ok
test_small_numbers (__main__.TestSquareFunction.test_small_numbers)
Test square function with very small numbers. ... ok
test_zero (__main__.TestSquareFunction.test_zero)
Test square function with zero. ... ok

----------------------------------------------------------------------
Ran 7 tests in 0.005s
```

Name: B.Nishant
Enrol : 2403A52089
Batch : 04