Assignment-8.4

Task1: Developing a Utility Function Using TDD

Scenario: You are working on a small utility library for a larger software system. One of the required functions should calculate the square of a given number, and correctness is critical because other modules depend on it.

Task Description

Following the Test Driven Development (TDD) approach:

1. First, write unit test cases to verify that a function correctly returns the square of a number for multiple inputs.

2. After defining the test cases, use GitHub Copilot or Cursor AI to generate the function implementation so that all tests pass.

Ensure that the function is written only after the tests are created.

Expected Outcome

• A separate test file and implementation file

• Clearly written test cases executed before implementation

• AI-assisted function implementation that passes all tests

• Demonstration of the TDD cycle: test → fail → implement → pass

square.py > square

```python
def square(x):
    """Return the square of a number.

    Accepts ints and floats. Raises TypeError for non-numeric input.
    """
    if not isinstance(x, (int, float)):
        raise TypeError("square() requires a numeric argument")
    return x * x

```

```python
import unittest

from square import square


class TestSquare(unittest.TestCase):
    def test_square_positive_int(self):
        self.assertEqual(square(3), 9)

    def test_square_zero(self):
        self.assertEqual(square(0), 0)

    def test_square_negative_int(self):
        self.assertEqual(square(-4), 16)

    def test_square_positive_float(self):
        self.assertAlmostEqual(square(2.5), 6.25)

    def test_square_large_int(self):
        self.assertEqual(square(10_000), 100_000_000)


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

OUTPUT:

```
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/test_square.
py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/square.py
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/test_square.
py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s
```

Task2: Email Validation for a User Registration System

Scenario: You are developing the backend of a user registration system. One requirement is to validate user email addresses before storing them in the database.

Task Description

Apply Test Driven Development by:

1. Writing unit test cases that define valid and invalid email formats (e.g., missing @, missing domain, incorrect structure).

2. Using AI assistance to implement the validate_email() function based

strictly on the behavior described by the test cases.

The implementation should be driven entirely by the test expectations.

Expected Outcome

- Well-defined unit tests using unittest or pytest
- An AI-generated email validation function
- All test cases passing successfully
- Clear alignment between test cases and function behavior

email_validator.py > ...

```python
import re

# Basic email validation regex matching the test expectations:
# - local part: letters, digits, . _ % + - (no leading/trailing dot)
# - single @ separator
# - domain: labels separated by dots, labels start/end with alnum, may contain hyphens
# - no surrounding spaces
_EMAIL_RE = re.compile(
    r"^[A-Za-z0-9]([A-Za-z0-9._%+-]{0,63}[A-Za-z0-9])?@"
    r"(?:[A-Za-z0-9](?:[A-Za-z0-9-]{0,61}[A-Za-z0-9])?\.)+"
    r"[A-Za-z]{2,}$"
)


def validate_email(email):
    """Return True if `email` is a valid email address per test rules.

    The function expects a string and enforces a conservative pattern:
    - no surrounding whitespace
    - exactly one '@'
    - reasonable local and domain label rules
    """
    if not isinstance(email, str):
        return False
    # Reject surrounding whitespace explicitly
    if email.strip() != email:
        return False
    return bool(_EMAIL_RE.fullmatch(email))

```

test_email_validator.py

test_email_validator.py > ...

```python
import unittest

from email_validator import validate_email


class TestValidateEmail(unittest.TestCase):
    # Valid email formats
    def test_valid_simple(self):
        self.assertTrue(validate_email('user@example.com'))

    def test_valid_subdomain(self):
        self.assertTrue(validate_email('user@mail.example.co.uk'))

    def test_valid_plus_addressing(self):
        self.assertTrue(validate_email('user+tag@example.com'))

    # Invalid formats
    def test_invalid_missing_at(self):
        self.assertFalse(validate_email('userexample.com'))

    def test_invalid_missing_domain(self):
        self.assertFalse(validate_email('user@'))

    def test_invalid_missing_local(self):
        self.assertFalse(validate_email('@example.com'))

    def test_invalid_multiple_at(self):
        self.assertFalse(validate_email('user@@example.com'))

    def test_invalid_spaces(self):
        self.assertFalse(validate_email(' user@example.com '))

    def test_invalid_domain_label_starts_with_dash(self):
        self.assertFalse(validate_email('user@-example.com'))


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

```

OUTPUT:

```
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/test_email_v
alidator.py
.........
----------------------------------------------------------------------
Ran 9 tests in 0.001s

OK
PS C:\Users\tloke\OneDrive\AI>
```

Task 3: Decision Logic Development Using TDD

Scenario: In a grading or evaluation module, a function is required to determine the maximum value among three inputs. Accuracy is essential, as incorrect results could affect downstream decision logic.

Task Description

Using the TDD methodology:

1. Write test cases that describe the expected output for different combinations of three numbers.

2. Prompt GitHub Copilot or Cursor AI to implement the function logic

based on the written tests. Avoid writing any logic before test cases are completed.

Expected Outcome

• Comprehensive test cases covering normal and edge cases

• AI-generated function implementation

• Passing test results demonstrating correctness

• Evidence that logic was derived from tests, not assumptions

max_of_three.py > max_of_three

```python
def max_of_three(a, b, c):
    """Return the maximum of three numeric values.

    Accepts ints and floats and returns the largest value. Uses Python's
    built-in comparison semantics so mixed numeric types are handled.
    """
    return a if (a >= b and a >= c) else (b if b >= c else c)

```

test_max_of_three.py > ...

```python
import unittest

from max_of_three import max_of_three


class TestMaxOfThree(unittest.TestCase):
    def test_increasing_order(self):
        self.assertEqual(max_of_three(1, 2, 3), 3)

    def test_decreasing_order(self):
        self.assertEqual(max_of_three(3, 2, 1), 3)

    def test_mixed_order(self):
        self.assertEqual(max_of_three(2, 3, 1), 3)

    def test_negative_numbers(self):
        self.assertEqual(max_of_three(-10, -5, -7), -5)

    def test_with_zero_and_positive(self):
        self.assertEqual(max_of_three(-1, 0, 1), 1)

    def test_floats_and_ints(self):
        self.assertEqual(max_of_three(2.5, 2, 2.7), 2.7)

    def test_all_equal(self):
        self.assertEqual(max_of_three(5, 5, 5), 5)

    def test_two_equal_max(self):
        self.assertEqual(max_of_three(7, 7, 3), 7)

    def test_large_numbers(self):
        self.assertEqual(max_of_three(10_000_000, 9_999_999, 10_000_000), 10_000_000)


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

```

OUTPUT:

```
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/test_max_of_
three.py
.........
----------------------------------------------------------------------
Ran 9 tests in 0.002s

OK
```

Task 4: Shopping Cart Development with AI-Assisted TDD

Scenario: You are building a simple shopping cart module for an e-commerce application. The cart must support adding items, removing items, and calculating the total price accurately.

Task Description

Follow a test-driven approach:

1. Write unit tests for each required behavior:

o Adding an item

o Removing an item

o Calculating the total price

2. After defining all tests, use AI tools to generate the Shopping Cart class and its methods so that the tests pass. Focus on behavior-driven testing rather than implementation details.

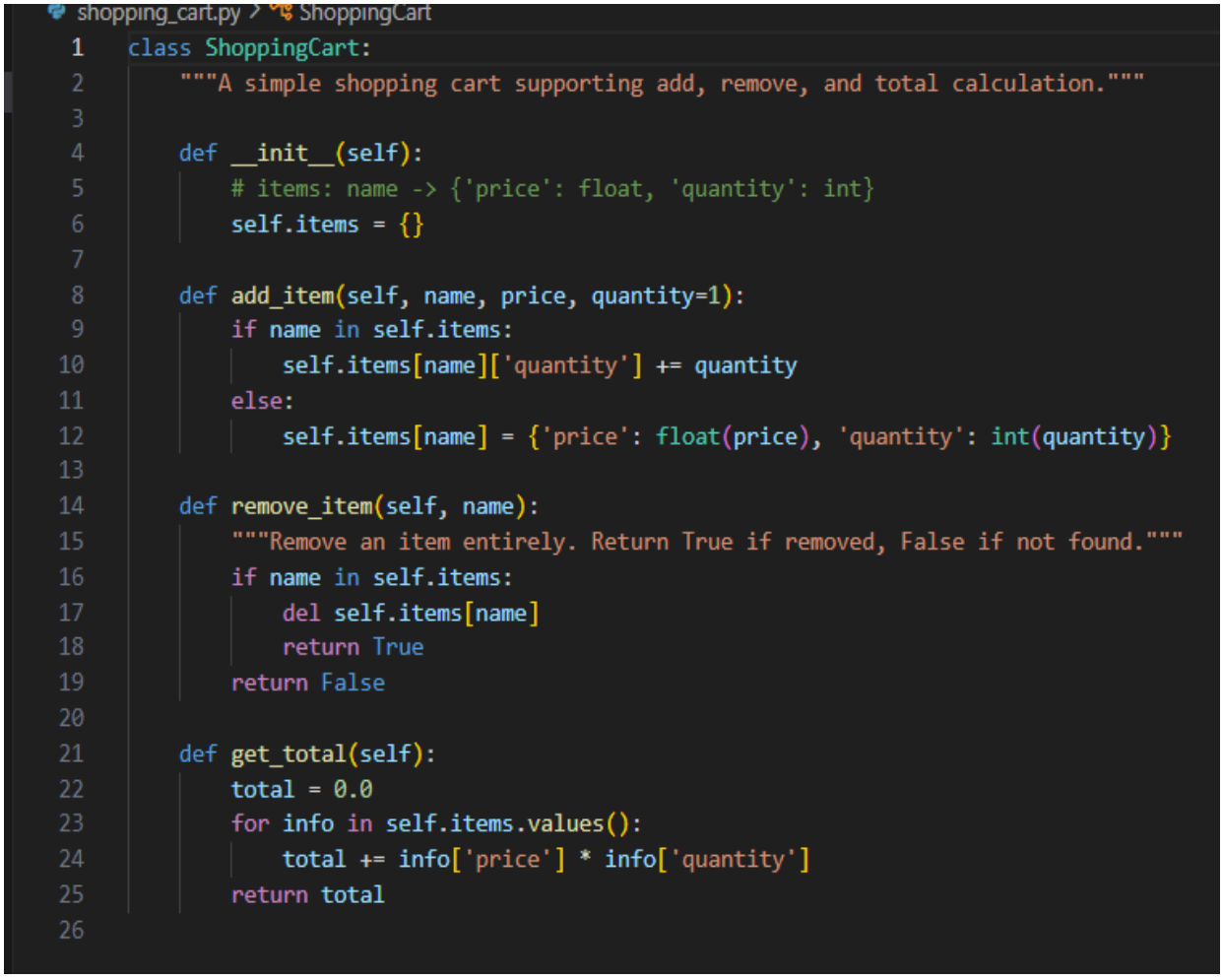
Expected Outcome

• Unit tests defining expected shopping cart behavior

• AI-generated class implementation

• All tests passing successfully

• Clear demonstration of TDD applied to a class-based design



test_shopping_cart.py > ...

```python
import unittest

from shopping_cart import ShoppingCart


class TestShoppingCart(unittest.TestCase):
    def test_add_item_increases_total(self):
        cart = ShoppingCart()
        cart.add_item('apple', 1.0)
        self.assertEqual(cart.get_total(), 1.0)
        cart.add_item('apple', 1.0, quantity=2)
        self.assertEqual(cart.get_total(), 3.0)

    def test_remove_item_updates_total(self):
        cart = ShoppingCart()
        cart.add_item('apple', 1.0, quantity=2)
        cart.add_item('banana', 0.5)
        self.assertEqual(cart.get_total(), 2.5)
        removed = cart.remove_item('apple')
        self.assertTrue(removed)
        self.assertEqual(cart.get_total(), 0.5)

    def test_remove_nonexistent_returns_false(self):
        cart = ShoppingCart()
        cart.add_item('banana', 0.5)
        self.assertFalse(cart.remove_item('apple'))

    def test_calculate_total_multiple_items(self):
        cart = ShoppingCart()
        cart.add_item('apple', 1.25, quantity=3)
        cart.add_item('orange', 0.75, quantity=2)
        expected = 1.25 * 3 + 0.75 * 2
        self.assertAlmostEqual(cart.get_total(), expected)


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

```

OUTPUT:

```
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/test_shoppin
g_cart.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
```

Task 5: String Validation Module Using TDD

Scenario: You are working on a text-processing module where a function is required to identify whether a given string is a palindrome. The function must handle different cases and inputs reliably.

Task Description

Using Test Driven Development:

1. Write test cases for a palindrome checker covering:

o Simple palindromes

o Non-palindromes

o Case variations

2. Use GitHub Copilot or Cursor AI to generate the is palindrome() function based on the test case expectations. The function should be implemented only after tests are written.

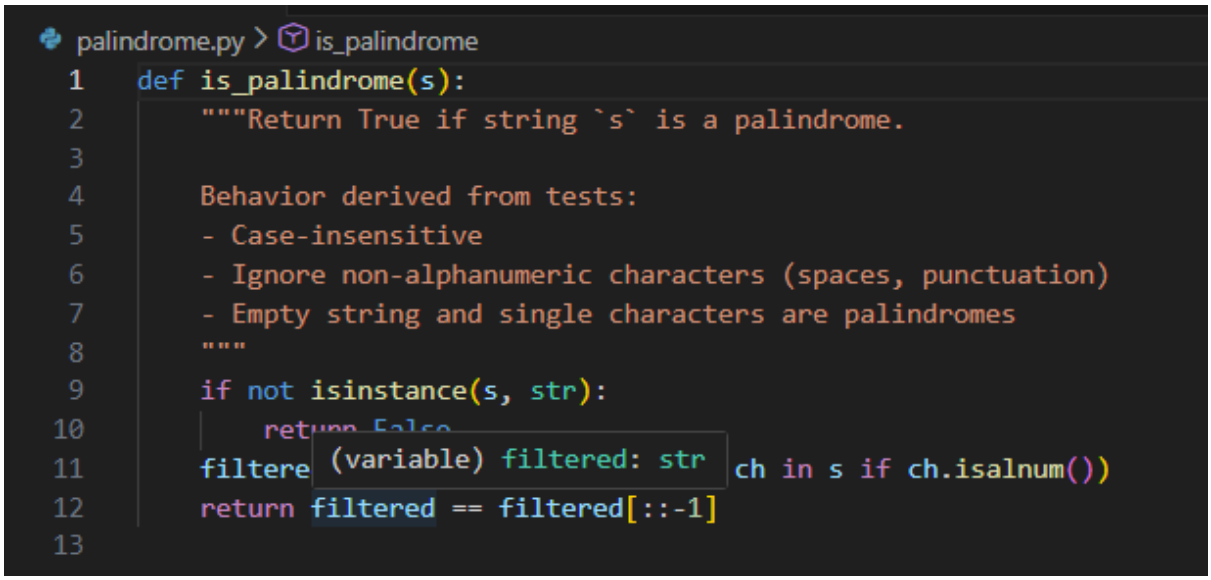
Expected Outcome

• Clearly written test cases defining expected behavior

• AI-assisted implementation of the palindrome checker

• All test cases passing successfully

• Evidence of TDD methodology applied correctly



test_is_palindrome.py > ...

```python
import unittest

from palindrome import is_palindrome


class TestIsPalindrome(unittest.TestCase):
    def test_simple_palindrome(self):
        self.assertTrue(is_palindrome('radar'))

    def test_non_palindrome(self):
        self.assertFalse(is_palindrome('hello'))

    def test_case_insensitive(self):
        self.assertTrue(is_palindrome('RaceCar'))

    def test_ignore_spaces_punctuation(self):
        self.assertTrue(is_palindrome('A man, a plan, a canal: Panama'))

    def test_empty_string(self):
        self.assertTrue(is_palindrome(''))

    def test_single_char(self):
        self.assertTrue(is_palindrome('x'))

    def test_numeric_palindrome(self):
        self.assertTrue(is_palindrome('12321'))


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

OUTPUT:

```
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/test_is_palindrome.py
.......
----------------------------------------------------------------------
Ran 7 tests in 0.001s

OK
```