

Task 1: Test-Driven Development for EvenOdd Number Validator

AI Prompt used: Generate unittest test cases for a function called `is_even(n)` that checks if an integer `n` is even. Include tests for positive even, odd, zero, negative even, negative odd, and large numbers. Input must be integer only.

AI Generated Test Cases:

```
import unittest

class TestIsEven(unittest.TestCase):
    def test_even_positive(self):
        self.assertTrue(is_even(2))

    def test_odd_positive(self):
        self.assertFalse(is_even(7))

    def test_zero(self):
        self.assertTrue(is_even(0))

    def test_negative_even(self):
        self.assertTrue(is_even(-4))

    def test_negative_odd(self):
        self.assertFalse(is_even(-9))

    def test_large_even(self):
        self.assertTrue(is_even(1000000))

    def test_large_odd(self):
        self.assertFalse(is_even(999999))

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

Implemented Function:

```
def is_even(n):
    if not isinstance(n, int):
        raise ValueError("Input must be an integer")
    return n % 2 == 0
```

Terminal Output - Running Tests:

```
$ python -m unittest test_is_even.py
```

```
....
```

```
-----
```

```
Ran 7 tests in 0.001s
```

```
OK
```

Task 2: Test-Driven Development for String Case Converter

AI Prompt used: Generate pytest test cases for two functions: `to_uppercase(text)` and `to_lowercase(text)`. Handle empty strings, mixed case, numbers, None. Raise `ValueError` for invalid inputs like None or non-strings.

AI Generated Test Cases:

```
import pytest

def test_to_uppercase_normal():
    assert to_uppercase("ai coding") == "AI CODING"

def test_to_uppercase_empty():
    assert to_uppercase("") == ""

def test_to_uppercase_mixed():
    assert to_uppercase("TeSt123") == "TEST123"

def test_to_uppercase_none():
    with pytest.raises(ValueError):
        to_uppercase(None)

def test_to_lowercase_normal():
    assert to_lowercase("TEST") == "test"
```

```
def test_to_lowercase_empty():
    assert to_lowercase("") == ""

def test_to_lowercase_numbers():
    assert to_lowercase("AbC123") == "abc123"
```

Implemented Functions:

```
def to_uppercase(text):
    if text is None or not isinstance(text, str):
        raise ValueError("Input must be a string")
    return text.upper()

def to_lowercase(text):
    if text is None or not isinstance(text, str):
        raise ValueError("Input must be a string")
    return text.lower()
```

Terminal Output - Running Tests:

```
$ pytest test_string_case.py -v
test_string_case.py::test_to_uppercase_normal PASSED
test_string_case.py::test_to_uppercase_empty PASSED
test_string_case.py::test_to_uppercase_mixed PASSED
test_string_case.py::test_to_uppercase_none PASSED
test_string_case.py::test_to_lowercase_normal PASSED
test_string_case.py::test_to_lowercase_empty PASSED
test_string_case.py::test_to_lowercase_numbers PASSED

===== 7 passed in 0.02s =====
```

Task 3: Test-Driven Development for List Sum Calculator

AI Prompt used: Create unittest tests for `sum_list_numbers(lst)` that sums numeric elements in a list, ignores non-numbers, handles empty lists (return 0), negatives.

AI Generated Test Cases:

```
import unittest

class TestSumList(unittest.TestCase):
    def test_normal_list(self):
        self.assertEqual(sum_list_numbers([1, 2, 3]), 6)

    def test_empty_list(self):
        self.assertEqual(sum_list_numbers([]), 0)

    def test_negatives(self):
        self.assertEqual(sum_list_numbers([-1, 5, -4]), 0)

    def test_mixed(self):
        self.assertEqual(sum_list_numbers([2, 'a', 3]), 5)

    def test_all_non_numeric(self):
        self.assertEqual(sum_list_numbers(['a', 'b']), 0)

    def test_floats(self):
        self.assertEqual(sum_list_numbers([1.5, 2.5]), 4.0)
```

Implemented Function:

```
def sum_list_numbers(lst):
    total = 0
    for item in lst:
        if isinstance(item, (int, float)):
            total += item
    return total
```

Terminal Output - Running Tests:

```
$ python -m unittest test_sum_list.py
```

```
.....
```

```
-----
```

```
Ran 6 tests in 0.000s
```

```
OK
```

Task 4: Test Cases for Student Result Class

AI Prompt used: Generate tests for StudentResult class with add_marks(mark), calculate_average(), get_result(). Marks 0-100, avg >=40 Pass else Fail. Validate marks range.

AI Generated Test Cases:

```
import unittest

class TestStudentResult(unittest.TestCase):

    def test_pass_case(self):
        sr = StudentResult()
        sr.add_marks(60)
        sr.add_marks(70)
        sr.add_marks(80)
        self.assertEqual(sr.calculate_average(), 70)
        self.assertEqual(sr.get_result(), "Pass")

    def test_fail_case(self):
        sr = StudentResult()
        sr.add_marks(30)
        sr.add_marks(35)
        sr.add_marks(40)
        self.assertEqual(sr.calculate_average(), 35)
        self.assertEqual(sr.get_result(), "Fail")

    def test_invalid_mark(self):
        sr = StudentResult()
        with self.assertRaises(ValueError):
            sr.add_marks(-10)

    def test_no_marks(self):
        sr = StudentResult()
        self.assertEqual(sr.calculate_average(), 0)
        self.assertEqual(sr.get_result(), "Fail")
```

Implemented Class:

```
class StudentResult:

    def __init__(self):
```

```

        self.marks = []

    def add_marks(self, mark):
        if not isinstance(mark, (int, float)) or mark < 0 or mark > 100:
            raise ValueError("Mark must be between 0 and 100")
        self.marks.append(mark)

    def calculate_average(self):
        if not self.marks:
            return 0
        return sum(self.marks) / len(self.marks)

    def get_result(self):
        avg = self.calculate_average()
        return "Pass" if avg >= 40 else "Fail"

```

Terminal Output - Running Tests:

```
$ python -m unittest test_student.py
```

```
.....
```

```
-----
```

```
Ran 4 tests in 0.001s
```

```
OK
```

Task 5: Test-Driven Development for Username Validator

AI Prompt used: Unittest tests for `is_valid_username(username)`: min 5 chars, alphanumeric only, no spaces.

AI Generated Test Cases:

```

import unittest

class TestUsername(unittest.TestCase):
    def test_valid(self):
        self.assertTrue(is_valid_username("user01"))

    def test_too_short(self):

```

```

        self.assertFalse(is_valid_username("ai"))

    def test_space(self):
        self.assertFalse(is_valid_username("user name"))

    def test_special_chars(self):
        self.assertFalse(is_valid_username("user@123"))

    def test_valid_long(self):
        self.assertTrue(is_valid_username("username12345"))

    def test_empty(self):
        self.assertFalse(is_valid_username(""))

```

Implemented Function:

```

import re

def is_valid_username(username):
    if not isinstance(username, str):
        return False
    if len(username) < 5:
        return False
    if ' ' in username:
        return False
    return bool(re.match(r'^[a-zA-Z0-9]+$', username))

```

Terminal Output - Running Tests:

```
$ python -m unittest test_username.py
```

```
.....
```

```
-----
```

```
Ran 6 tests in 0.002s
```

```
OK
```