Simeon Patton

April 29, 2021

CS362 Oregon State Spring 2021

In class Activity 3

1. a) write a simple calc app

   b) your app should take two numbers as inputs (a,b)

   c) should have the following methods:

   addition, subtraction, multiplication, division

   d) Create a test module that has your test cases

This is completed with two separate files;

calc.py and test_calc.py

and can both be found at

https://github.com/pattons-OSU/CS362_InClassActivities/tree/master/Week_5

2. Search for another Python Testing Framework. What did you find? Would you be able to write tests for your calculator using the new testing framework that you discovered?

   Through a google search I was able to find a page that had eight different types of testing frameworks that are currently in use in the python environment. Out of the eight, I chose to focus on PyTest and it is a testing framework that I have heard of in the past (although have not had the opportunity to use it yet. With PyTest, it seems as if the user is getting the largest possible community to help out with it, largely because it is opensource and one of the most used testing suits that can be found in the python environment.

   I am sure that the tests can be re-written in PyTest and put to use in the test program. The existing tests, however, could not migrate and begin working within the PyTest framework. The commands are different and the calls would not transfer over. Personally, I would not be able to right off of the bat to write tests in the PyTest framework, but I would be able to give it a try after taking a look at a lot more of the documentation within the framework.

An example of a PyTest test script can be seen below and can also be found at:

https://gist.github.com/AndyLPK247/650b137dbd42c802d1a995a48d4830eb

```
import pytest
from com.automationpanda.example.calc_class import Calculator

# "Constants"

NUMBER_1 = 3.0
NUMBER_2 = 2.0


# Fixtures

@pytest.fixture
def calculator():
    return Calculator()


# Helpers

def verify_answer(expected, answer, last_answer):
    assert expected == answer
    assert expected == last_answer


# Test Cases

def test_last_answer_init(calculator):
    assert calculator.last_answer == 0.0


def test_add(calculator):
    answer = calculator.add(NUMBER_1, NUMBER_2)
    verify_answer(5.0, answer, calculator.last_answer)


def test_subtract(calculator):
    answer = calculator.subtract(NUMBER_1, NUMBER_2)
    verify_answer(1.0, answer, calculator.last_answer)
```

```
def test_subtract_negative(calculator):
    answer = calculator.subtract(NUMBER_2, NUMBER_1)
    verify_answer(-1.0, answer, calculator.last_answer)


def test_multiply(calculator):
    answer = calculator.multiply(NUMBER_1, NUMBER_2)
    verify_answer(6.0, answer, calculator.last_answer)


def test_divide(calculator):
    answer = calculator.divide(NUMBER_1, NUMBER_2)
    verify_answer(1.5, answer, calculator.last_answer)


def test_divide_by_zero(calculator):
    with pytest.raises(ZeroDivisionError) as e:
        calculator.divide(NUMBER_1, 0)
    assert "division by zero" in str(e.value)


@pytest.mark.parametrize("a,b,expected", [
    (NUMBER_1, NUMBER_2, NUMBER_1),
    (NUMBER_2, NUMBER_1, NUMBER_1),
    (NUMBER_1, NUMBER_1, NUMBER_1),
])
def test_maximum(calculator, a, b, expected):
    answer = calculator.maximum(a, b)
    verify_answer(expected, answer, calculator.last_answer)


@pytest.mark.parametrize("a,b,expected", [
    (NUMBER_1, NUMBER_2, NUMBER_2),
    (NUMBER_2, NUMBER_1, NUMBER_2),
    (NUMBER_2, NUMBER_2, NUMBER_2),
])
def test_minimum(calculator, a, b, expected):
    answer = calculator.minimum(a, b)
    verify_answer(expected, answer, calculator.last_answer)
```