Michał Motyl, 401943

TDD methodology

Funkcje do zrealizowania wybrałem jako jedno z wyzwań na stronie hackerrank: https://www.hackerrank.com/challenges/magic-square-forming/problem?isFullScreen=true

Funkcja ma dla macierzy 3x3 w której znajdują się liczby od 1 do 9 (każda tylko i wyłącznie raz) znaleźć minimalny 'koszt' zamienienia danej wprowadzonej macierzy do kwadratu magicznego.

RED

Zgodnie z metodologią zacząłem od sporządzenia odpowiedniego testu:

In [ ]:
```
# Zawartość pliku test_app.py

#####################################################
from app import magic_square
import pytest

# test data, matrixes with correct minimal costs
testdata = [
    ([[5, 3, 4], [1, 5, 8], [6, 4, 2]], 7),
    ([[4, 9, 2], [3, 5, 7], [8, 1, 5]], 1),
    ([[4, 9, 7], [3, 5, 2], [8, 1, 6]], 10)
]

@pytest.mark.parametrize('sample, output', testdata)
def test_magic_square(sample, output):

    got = magic_square(sample)
    want = output

    assert got == want
#####################################################
```

GREEN

Następnie przystąpiłem do pisania funkcji. Poniżej efekt mojej pracy:

In [ ]:
```
# Zawartość pliku app.py przed REFACTOR

#####################################################
import numpy as np
from typing import List

# function should change any 3x3 square into a magic square, calculate the MINIMUM cost of said change
# cost of one change: abs(a - b) where a is the previous number and b is the new number
# sum of all costs is the total cost

# idea from: https://www.hackerrank.com/challenges/magic-square-forming/problem?isFullScreen=true

def magic_square(square: List[List]) -> int:
    np_square = np.array(square)
    magic_constant = 15

    np_sum_axis0 = np.sum(np_square, axis=0)
    axis0_magic = np.all(np_sum_axis0 == magic_constant)

    np_sum_axis1 = np.sum(np_square, axis=1)
    axis1_magic = np.all(np_sum_axis1 == magic_constant)

    cost = 0
    if not axis1_magic:
        sum_axis = np.sum(np_sum_axis1)
        diff = np.abs(sum_axis - 3*magic_constant)
        if diff == 0:
            max_val = np.max(np_sum_axis1)
            cost = (max_val - magic_constant) * 2
        else:
            cost = diff

    if not axis0_magic:
        sum_axis = np.sum(np_sum_axis0)
        diff = np.abs(sum_axis - 3*magic_constant)
        if diff == 0:
            max_val = np.max(np_sum_axis0)
            cost = (max_val - magic_constant) * 2
        else:
            cost = diff

    return cost
#####################################################
```

Wyniki testów przed poprawkami:

```
==================================== FAILURES ====================================
_____________________ test_magic_square[sample2-10] _____________________

sample = [[4, 9, 7], [3, 5, 2], [8, 1, 6]], output = 10

    @pytest.mark.parametrize('sample, output', testdata)
    def test_magic_square(sample, output):

        got = magic_square(sample)
        want = output

>       assert got == want
E       assert 15 == 10

test/test_app.py:17: AssertionError
=========================== short test summary info ============================
FAILED test/test_app.py::test_magic_square[sample2-10] - assert 15 == 10
========================== 1 failed, 2 passed in 0.30s ===========================
(michalmotyl) michal@MacBook-Air-3 TDD %
```

Wyniki testów po poprawkach:

```
============================= test session starts ==============================
platform darwin -- Python 3.8.8, pytest-6.2.5, py-1.10.0, pluggy-0.13.1
rootdir: /Users/michal/Desktop/AiBD_all/AiBD/TDD
plugins: anyio-2.2.0
collected 3 items

test/test_app.py ...                                                     [100%]

============================== 3 passed in 0.19s ===============================
(michalmotyl) michal@macbook-air-3 TDD %
```

REFACTOR

Testy przechodzą, dlatego też przystępuje do poprawek w kodzie. Po za kosmetycznym dodaniem *return* w dwóch miejscach dodałem tylko i wyłącznie komentarze:

In [ ]:
```
# Zawartość pliku app.py po REFACTOR

#####################################################
import numpy as np
from typing import List

# function should change any 3x3 square into a magic square, calculate the MINIMUM cost of said change
# cost of one change: abs(a - b) where a is the previous number and b is the new number
# sum of all costs is the total cost

# idea from: https://www.hackerrank.com/challenges/magic-square-forming/problem?isFullScreen=true

def magic_square(square: List[List]) -> int:

    # transforming list into ndarray, for ease of operations
    np_square = np.array(square)

    # for a 3x3 magic square this is the value that all the columns/rows have to sum up to
    magic_constant = 15

    # checking sum in rows
    np_sum_axis0 = np.sum(np_square, axis=0)
    axis0_magic = np.all(np_sum_axis0 == magic_constant)

    # checking sum in columns
    np_sum_axis1 = np.sum(np_square, axis=1)
    axis1_magic = np.all(np_sum_axis1 == magic_constant)

    # below the minimal cost is calculated the same way for both columns/rows
    cost = 0
    if not axis1_magic:
        # we check the sum of all columns
        sum_axis = np.sum(np_sum_axis1)
        diff = np.abs(sum_axis - 3*magic_constant)

        # if its 45 it means that there are correct numbers in the array, but in wrong order.
        if diff == 0:
            # calculation of cost
            max_val = np.max(np_sum_axis1)
            cost = (max_val - magic_constant) * 2
            return cost

        # else it means that the numbers are wrong - it does not matter what those numbers are,
        # the cost will always be the same
        else:
            cost = diff

    # same idea with second 'sum'
    if not axis0_magic:
        sum_axis = np.sum(np_sum_axis0)
        diff = np.abs(sum_axis - 3*magic_constant)
        if diff == 0:
            max_val = np.max(np_sum_axis0)
            cost = (max_val - magic_constant) * 2
            return cost

        else:
            cost = diff

    # if not, it is indeed a magic square -> cost is equal to 0
    return cost
#####################################################
```

Ostatnim etapem byłoby sprawdzenie czy moje rozwiązanie jest zgodnie z wynikami na hackerrank, niestety już po napisaniu funkcji zorientowałem się że w zaprezentowanym przez nich wyzwaniu biblioteka numpy nie była dozwolona. Niemniej jednak jestem przekonany że funkcja działa poprawnie - a co najważniejsze, podczas jej pisania trzymałem się metodyki TDD.

In [ ]:
```

```