Adrian Poniatowski

Test Driven Development(TDD)

Sprawozdanie

Wykonanie tej serii laboratorium rozpocząłem od zapoznania się z instrukcją do ćwiczenia oraz teorii przedstawionej w konspekcie. Muszę tutaj zaznaczyć, że wszystko zostało czytelnie opisane i nie stanowiło żadnych trudności. Po uruchomieniu przygotowanych plików zawierających logikę oraz jej testowanie otrzymałem rezultat mówiący o tym, że wszystkie testy zostały zaliczone. Zgodnie z instrukcją postanowiłem przeprowadzić kilka eksperymentów z tym związanych, jednak poniżej postanowiłem zamieścić tylko zrzut ekranu pokazujący prawidłowe działanie przykładów.

```
============================= test session starts =============================
platform win32 -- Python 3.6.5, pytest-6.2.5, py-1.11.0, pluggy-1.0.0 -- C:\Users\adria\AppData\Local\Programs\Python\Python36\python.exe
cachedir: .pytest_cache
rootdir: E:\Adrian-studia\Semestr_5\Analiza_i_Bazy_Danych\Lab_14\test
collecting ... collected 4 items

test_app.py::test_hello PASSED                                           [ 25%]
test_app.py::test_extract_sentiment[I think today will be a great day] PASSED [ 50%]
test_app.py::test_text_contain_word[There is a duck in this text-duck-True] PASSED [ 75%]
test_app.py::test_text_contain_word[There is nothing here-duck-False] PASSED [100%]

============================== 4 passed in 2.38s ==============================

Process finished with exit code 0
```

Następnie przystąpiłem do samodzielnego wykonania przydzielonego zadania. Miałem napisać funkcję, która będzie implementowała algorytm sortowania bąbelkowego.

Zgodnie z podejściem TDD, najpierw napisałem funkcje testującą, która przedstawia się następująco:

```
34
35      testdata3 = [
36          [[1, 5, 7, 9, 2, 4, 6, 8, 3], [1, 2, 3, 4, 5, 6, 7, 8, 9]],
37          [[9, 8, 7, 6, 5, 4, 3, 2, 1], [1, 2, 3, 4, 5, 6, 7, 8, 9]],
38          [[7, 8, 9, 4, 5, 6, 1, 2, 3], [1, 2, 3, 4, 5, 6, 7, 8, 9]],
39          [[1, 9, 2, 8, 3, 7, 4, 6, 5], [1, 2, 3, 4, 5, 6, 7, 8, 9]],
40          [[3, 5, 7, 1, 9, 2, 6, 4, 8], [1, 2, 3, 4, 5, 6, 7, 8, 9]]
41      ]
42
43
44      @pytest.mark.parametrize('sample, expected_output', testdata3)
45      def test_bubble_sort(sample, expected_output):
46
47          assert bubble_sort(sample) == expected_output
```

Jak podpowiada sam program, występuje tutaj błąd, ponieważ nie istnieje jeszcze funkcja sortująca. Zgodnie z fazą RED, niemożliwe więc będzie, aby program przeszedł testy. Sprawdźmy to.

```
================================== FAILURES ===================================
_________________ test_bubble_sort[sample0-expected_output0] _________________

sample = [1, 5, 7, 9, 2, 4, ...], expected_output = [1, 2, 3, 4, 5, 6, ...]

    @pytest.mark.parametrize('sample, expected_output', testdata3)
    def test_bubble_sort(sample, expected_output):

>       assert bubble_sort(sample) == expected_output
E       NameError: name 'bubble_sort' is not defined

test_app.py:43: NameError
=========================== short test summary info ===========================
FAILED test_app.py::test_bubble_sort[sample0-expected_output0] - NameError: n...
========================= 1 failed, 4 passed in 2.49s =========================

Process finished with exit code 1

Assertion failed

Assertion failed
```

Jak widać, test zwrócił odpowiedni błąd, więc wszystko jest w porządku. Należało więc napisać odpowiednią funkcję, która prezentuje się następująco:

```
19
20  def bubble_sort(arr: list):
21
22      for i in range(len(arr)):
23
24          for j in range(0, len(arr)-i-1):
25              if arr[j] > arr[j+1]:
26                  arr[j], arr[j+1] = arr[j+1], arr[j]
27
28      return arr
29
```

Teraz już, jeśli tylko algorytm został dobrze zaimplementowany, testy powinny przejść. Aby się o tym przekonać, jeszcze raz uruchomiłem pytest. Wyniki testowania są następujące:

```
============================= test session starts =============================
platform win32 -- Python 3.6.5, pytest-6.2.5, py-1.11.0, pluggy-1.0.0 -- C:\Users\adria\AppData\Local\Programs\Python\Python36\python.exe
cachedir: .pytest_cache
rootdir: E:\Adrian-studia\Semestr_5\Analiza_i_Bazy_Danych\Lab_14\test
collecting ... collected 9 items

test_app.py::test_hello PASSED                                           [ 11%]
test_app.py::test_extract_sentiment[I think today will be a great day]
test_app.py::test_text_contain_word[There is a duck in this text-duck-True] PASSED [ 22%]
test_app.py::test_text_contain_word[There is nothing here-duck-False]
test_app.py::test_bubble_sort[sample0-expected_output0] PASSED [ 33%]PASSED [ 44%]
test_app.py::test_bubble_sort[sample1-expected_output1]
test_app.py::test_bubble_sort[sample2-expected_output2]
test_app.py::test_bubble_sort[sample3-expected_output3]
test_app.py::test_bubble_sort[sample4-expected_output4]

============================== 9 passed in 2.32s ==============================

Process finished with exit code 0
PASSED          [ 55%]PASSED          [ 66%]PASSED          [ 77%]PASSED          [ 88%]PASSED          [100%]
```

Wszystko zostało zaliczone pomyślnie, więc zadanie można uznać za zrobione i ukończone. W tym przypadku nie podejmowałem się trzeciej fazy REFACTOR, ponieważ nie było takiej potrzeby. Funkcja sortująca była prosta i przejrzysta, więc nie należało jej poprawiać.