

Sprawozdanie Analiza i Bazy Danych

Paweł Szwarnowski, laboratorium nr 13 – TDD

1. Faza Red

W tej fazie zostały zaimplementowane jedynie testy, sprawdzające czy lista zwrócona przez wywołanie funkcji bubblesort jest posortowana. Ich wynik jest negatywny, ponieważ testowana funkcja jeszcze nie istnieje.

```
3 test_data = [[0,1,2,3,4], [9,8,7,6,5,4,3,2,1],[100,1,200,2,30,4,], [1,1,2,1,1],[2,1,10,5,3,8,7,11,0]]
4
5 @pytest.mark.parametrize('list_to_sort', test_data)
6 def test_bubblesort(list_to_sort):
7     sorted_list = bubblesort(list_to_sort)
8     expected_output = list_to_sort.sort()
9     assert sorted_list == expected_output

test_bubblesort()

Terminal: Local x + v
FAILED test/test_app.py::test_bubblesort[list_to_sort1] - NameError: name 'bubblesort' is not defined
FAILED test/test_app.py::test_bubblesort[list_to_sort2] - NameError: name 'bubblesort' is not defined
FAILED test/test_app.py::test_bubblesort[list_to_sort3] - NameError: name 'bubblesort' is not defined
FAILED test/test_app.py::test_bubblesort[list_to_sort4] - NameError: name 'bubblesort' is not defined
===== 5 failed in 0.14s =====
(myapp) PS D:\uczelnia\semestr_5\AiBD\cw\myapp> $
```

2. Faza Green

W następnym kroku została wykonana początkowa implementacja funkcji bubblesort w taki sposób, aby testy dały wynik pozytywny.

```
1 def bubblesort(list_to_sort):
2     for i in range(len(list_to_sort)):
3         for j in range(len(list_to_sort)-1):
4             if list_to_sort[j] > list_to_sort[j+1]:
5                 temp = list_to_sort[j+1]
6                 list_to_sort[j+1] = list_to_sort[j]
7                 list_to_sort[j] = temp
8     return list_to_sort

bubblesort() > for i in range(len(list_to_sort)): > for j in range(len(list_to_sort)-1): > if list_to_sort[j] > list_to_sort[j+1]:

Terminal: Local x + v
(myapp) PS D:\uczelnia\semestr_5\AiBD\cw\myapp> py -m pytest
===== test session starts =====
platform win32 -- Python 3.9.2, pytest-7.2.0, pluggy-1.0.0
rootdir: D:\uczelnia\semestr_5\AiBD\cw\myapp
collected 6 items

test\test_app.py .....

===== 6 passed in 1.99s =====
```

3. Faza Refactor

Jak można łatwo zauważyć, stworzona w poprzednim punkcie nie jest optymalna oraz brakuje jej opisu. Dlatego w ostatniej fazie dokonano refaktoryzacji kodu – zwiększamy jego czytelność oraz zmniejszamy złożoność obliczeniową.

```
1 def bubblesort(list_to_sort):
2     for i in range(len(list_to_sort)):
3         # n_swap parameter tells us, how many swaps were made in one iteration
4         n_swap = 0
5
6         for j in range(len(list_to_sort)-1):
7             # checking, if the very next number is greater than current one
8             if list_to_sort[j] > list_to_sort[j+1]:
9                 # if condition is true, then we are swapping those numbers
10                list_to_sort[j], list_to_sort[j+1] = list_to_sort[j+1], list_to_sort[j]
11                n_swap += 1
12
13            # if there were no swaps, that means that our list is sorted and we can return it
14            if n_swap == 0:
15                return list_to_sort
16        return list_to_sort
```

bubblesort() > for i in range(len(list_to_sort...

Terminal: Local × + ▾

rootdir: D:\uczelnia\semestr_5\AiBD\cw\myapp
collected 6 items

test\test_app.py

===== 6 passed in 0.03s =====

Ponieważ jeden z testów sprawdzał sortowanie posortowanej listy 5000-elementowej, to zrefaktoryzowana funkcja poradziła sobie z tym znacznie lepiej, gdyż była do tego przystosowana.