

# Analiza i bazy danych

## Laboratorium 14: Test Driven Development

Artur Mzyk, 400658, gr. 2b

### 1. Wstęp

Test Driven Development to podejście do tworzenia oprogramowania, którego myślą przewodnią jest to, że przed napisaniem właściwej funkcjonalności tworzy się test, który ma sprawdzać jej działanie.

Cały proces pisania kodu składa się z cykli, które powtarza się jeden po drugim.

Każdy cykl składa się z 3 faz:

- Faza Red:

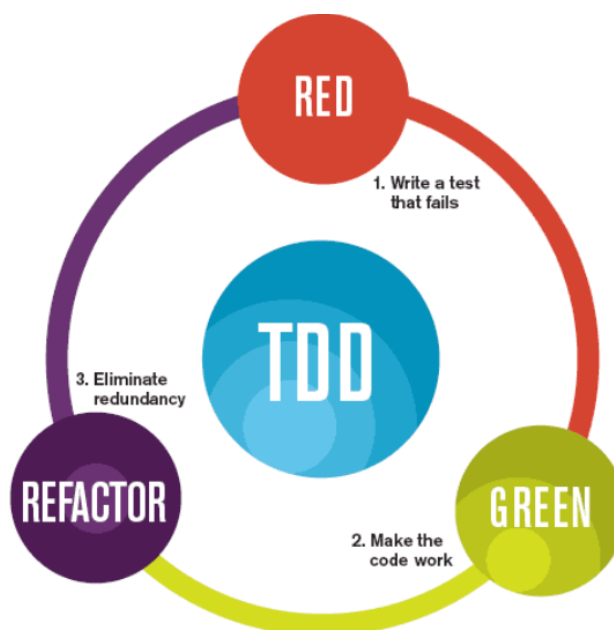
Na samym początku należy napisać test. Nie jest on w stanie wykonać się z sukcesem, gdyż testowana funkcjonalność nie została jeszcze zaimplementowana.

- Faza Green:

Następnym krokiem jest implementacja właściwej funkcjonalności. Wystarczy tylko tyle, aby test przeszedł.

- Faza Refactor:

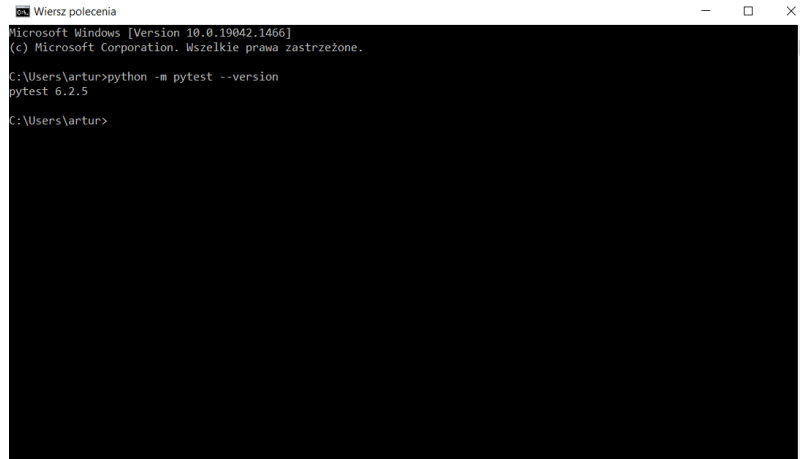
Ostatnia faza cyklu to refaktoryzacja, czyli ulepszanie istniejącego kodu, aby był bardziej czytelny i mniej złożony.



Rysunek 1. Cykl TDD

## 2. Framework

Najpopularniejszym frameworkiem do tworzenia testów w Pythonie jest Pytest. Umożliwia korzystanie z wbudowanych funkcji do porównywania czy korzystanie z pluginów.



```
Wiersz polecenia
Microsoft Windows [Version 10.0.19042.1466]
(c) Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\artur>python -m pytest --version
pytest 6.2.5

C:\Users\artur>
```

Rysunek 2. Potwierdzenie instalacji Pytest

## 3. Struktura katalogów

Dobłą praktyką jest zdefiniowanie struktury katalogów już na samym początku.

```
myapp/
  app.py
  test/
    __init__.py
    test_app.py
    ...
```

Rysunek 3. Zastosowana struktura katalogów

Kod tworzony będzie w katalogu *myapp* w module *app.py*. Testy będą tworzone w katalogu *test* w module *test\_app.py*. Nazewnictwo w przypadku testów jest ważne.

## 4. Pierwsza funkcja

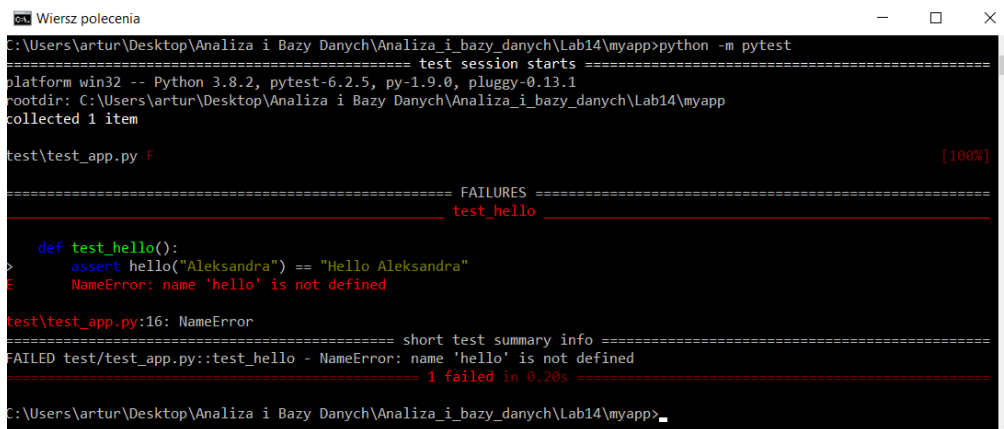
Celem jest napisanie funkcji, która jako argument przyjmuje imię i zwraca przywitanie składające się ze słowa „Hello” i tegoż imienia.

### 4.1. Faza Red

```
def test_hello():  
    assert hello("Aleksandra") == "Hello Aleksandra"
```

Rysunek 4. Pierwszy test

Stworzono i wykonano test. Naturalnie pojawił się komunikat o błędzie, gdyż funkcja nie została jeszcze zaimplementowana.



```
Wiersz polecenia  
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>python -m pytest  
===== test session starts =====  
platform win32 -- Python 3.8.2, pytest-6.2.5, py-1.9.0, pluggy-0.13.1  
rootdir: C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp  
collected 1 item  
  
test\test_app.py F [100%]  
  
===== FAILURES =====  
test_hello  
  
def test_hello():  
> assert hello("Aleksandra") == "Hello Aleksandra"  
E       NameError: name 'hello' is not defined  
  
test\test_app.py:16: NameError  
===== short test summary info =====  
FAILED test\test_app.py::test_hello - NameError: name 'hello' is not defined  
===== 1 failed in 0.20s =====  
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>
```

Rysunek 5. Niepomyślne wykonanie pierwszego testu

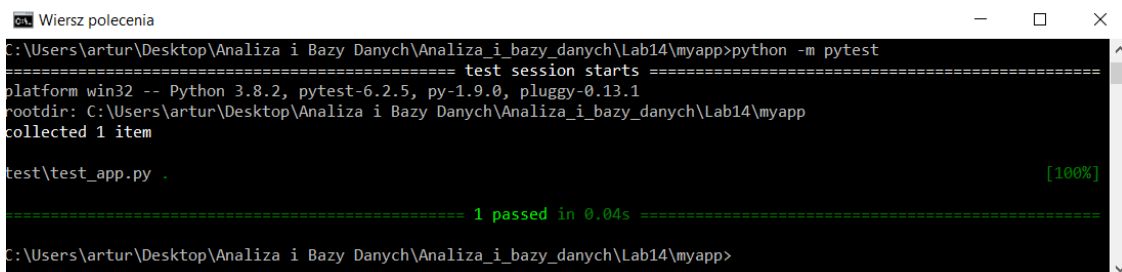
### 4.2. Faza Green

Zaimplementowano funkcję `hello()`.

```
def hello(name: str):  
    return f"Hello {name}"
```

Rysunek 6. Implementacja pierwszej funkcji

Funkcja pomyślnie przeszła test.



```
Wiersz polecenia  
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>python -m pytest  
===== test session starts =====  
platform win32 -- Python 3.8.2, pytest-6.2.5, py-1.9.0, pluggy-0.13.1  
rootdir: C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp  
collected 1 item  
  
test\test_app.py . [100%]  
  
===== 1 passed in 0.04s =====  
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>
```

Rysunek 7. Pomyślne wykonanie pierwszego testu

### 4.3. Faza Refactor

Funkcja jest na tyle prosta, że nie ma potrzeby ulepszania.

## 5. Druga funkcja

Sentyment to wydźwięk emocjonalny wypowiedzi. Analiza sentymentu to mechanizm analizy tekstów online, gdzie oceniany jest ton emocjonalny.

W tym celu można wykorzystać bibliotekę TextBlob, która służy do przetwarzania danych tekstowych.

Celem jest napisanie funkcji, która zwróci wartość większą od 0, jeżeli zdanie ma wydźwięk pozytywny, i mniejszą od 0, jeżeli zdanie ma wydźwięk negatywny.

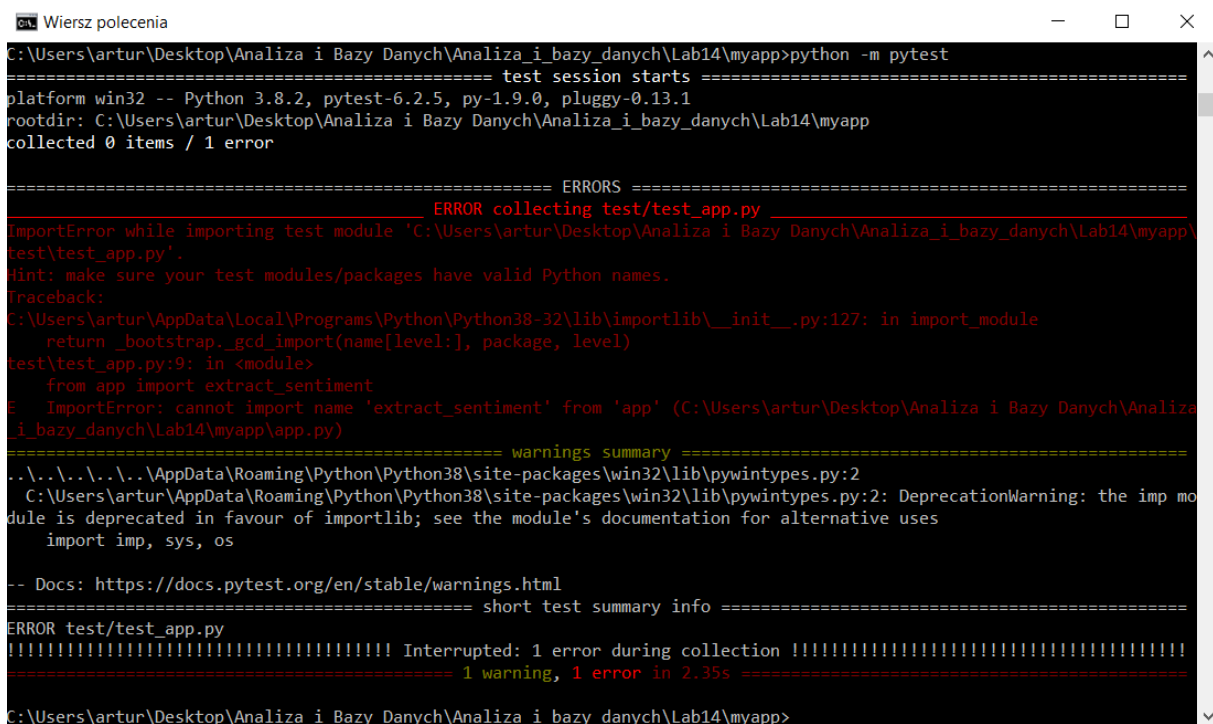
### 5.1. Faza Red

Stworzono i wykonano test. Naturalnie pojawia się komunikat o błędzie, gdyż funkcja nie została jeszcze zaimplementowana. Wykorzystano dekorator `pytest.mark.parametrize()`, który pozwala na przetestowanie wielu danych wejściowych.

```
test_data_1 = ["I think today will be a great day", "I do not think this will turn out well"]

@pytest.mark.parametrize("sample", test_data_1)
def test_extract_sentiment(sample):
    assert extract_sentiment(sample) > 0
```

Rysunek 8. Drugi test



```
Wiersz polecenia
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>python -m pytest
===== test session starts =====
platform win32 -- Python 3.8.2, pytest-6.2.5, py-1.9.0, pluggy-0.13.1
rootdir: C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp
collected 0 items / 1 error

===== ERRORS =====
ERROR collecting test/test_app.py
ImportError while importing test module 'C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp\test\test_app.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
C:\Users\artur\AppData\Local\Programs\Python\Python38-32\lib\importlib\__init__.py:127: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
test\test_app.py:9: in <module>
    from app import extract_sentiment
E ImportError: cannot import name 'extract_sentiment' from 'app' (C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp\app.py)

===== warnings summary =====
..\..\..\AppData\Roaming\Python\Python38\site-packages\win32\lib\pywintypes.py:2
C:\Users\artur\AppData\Roaming\Python\Python38\site-packages\win32\lib\pywintypes.py:2: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp, sys, os

-- Docs: https://docs.pytest.org/en/stable/warnings.html
===== short test summary info =====
ERROR test/test_app.py
!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!!
===== 1 warning, 1 error in 2.35s =====
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>
```

Rysunek 9. Niepomyślne wykonanie drugiego testu

Ostrzeżenie pojawia się ze względu na bibliotekę TextBlob.

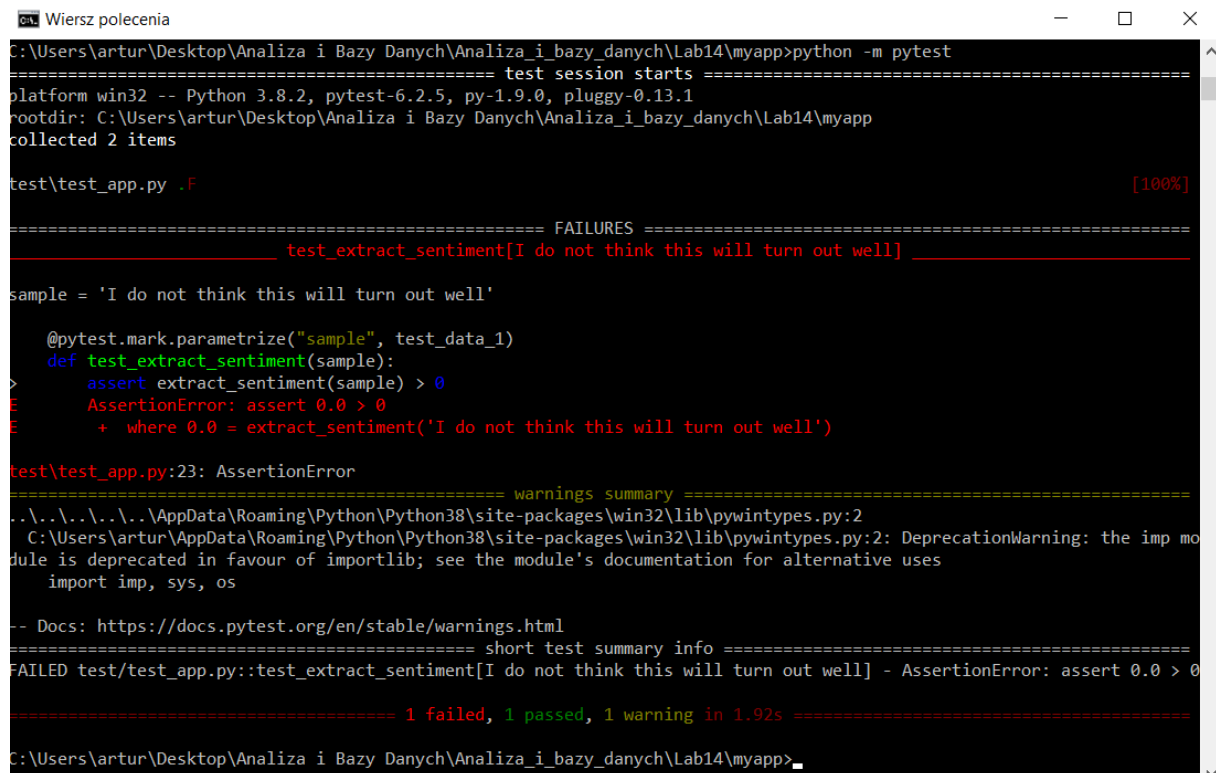
## 5.2. Faza Green

Zaimplementowano funkcję `extract_sentiment()`.

```
def extract_sentiment(text: str):  
    return TextBlob(text).sentiment.polarity
```

Rysunek 10. Implementacja drugiej funkcji

Funkcja dla pierwszych danych wejściowych przeszła test pomyślnie, dla drugich już nie – zgodnie z założeniem.



```
Wiersz polecenia  
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>python -m pytest  
===== test session starts =====  
platform win32 -- Python 3.8.2, pytest-6.2.5, py-1.9.0, pluggy-0.13.1  
rootdir: C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp  
collected 2 items  
  
test\test_app.py .F [100%]  
  
===== FAILURES =====  
_____ test_extract_sentiment[I do not think this will turn out well] _____  
  
sample = 'I do not think this will turn out well'  
  
  @pytest.mark.parametrize("sample", test_data_1)  
  def test_extract_sentiment(sample):  
>     assert extract_sentiment(sample) > 0  
E       AssertionError: assert 0.0 > 0  
E       + where 0.0 = extract_sentiment('I do not think this will turn out well')  
  
test\test_app.py:23: AssertionError  
  
===== warnings summary =====  
..\..\..\AppData\Roaming\Python\Python38\site-packages\win32\lib\pywintypes.py:2  
C:\Users\artur\AppData\Roaming\Python\Python38\site-packages\win32\lib\pywintypes.py:2: DeprecationWarning: the imp mo  
dule is deprecated in favour of importlib; see the module's documentation for alternative uses  
import imp, sys, os  
  
-- Docs: https://docs.pytest.org/en/stable/warnings.html  
===== short test summary info =====  
FAILED test\test_app.py::test_extract_sentiment[I do not think this will turn out well] - AssertionError: assert 0.0 > 0  
  
===== 1 failed, 1 passed, 1 warning in 1.92s =====  
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>
```

Rysunek 11. Pomyślne i niepomyślne wykonanie drugiego testu

## 5.3. Faza Refactor

Funkcja jest na tyle prosta, że nie ma potrzeby ulepszania.

## 6. Trzecia funkcja

Celem jest napisanie funkcji, która zwraca informację, czy zdanie zawiera wybrany wyraz.

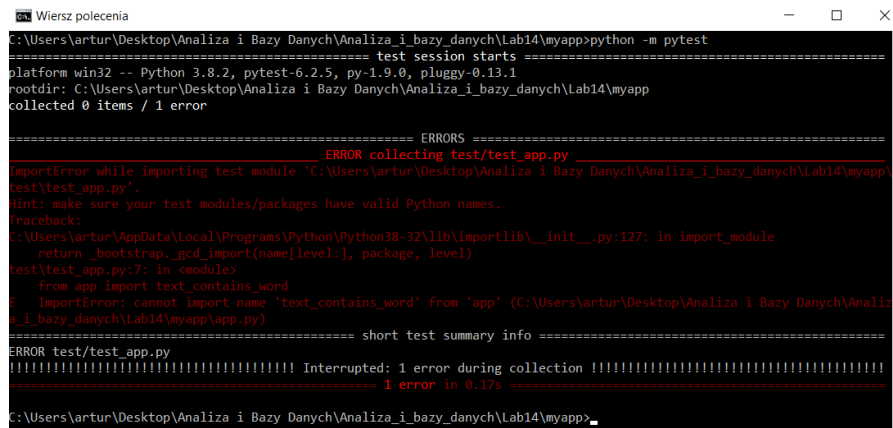
### 6.1. Faza Red

```
test_data_2 = [("There is a duck in this text", "duck", True), ("There is nothing here", "duck", False)]

@pytest.mark.parametrize("sample, word, expected_output", test_data_2)
def test_text_contains_word(sample, word, expected_output):
    assert text_contains_word(word, sample) == expected_output
```

Rysunek 12. Trzeci test

Stworzono i wykonano test. Naturalnie pojawia się komunikat o błędzie, gdyż funkcja nie została jeszcze zaimplementowana. Wykorzystano dekorator `pytest.mark.parametrize()`, który pozwala na przetestowanie wielu danych wejściowych.



```
Wiersz polecenia
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>python -m pytest
===== test session starts =====
platform win32 -- Python 3.8.2, pytest-6.2.5, py-1.9.0, pluggy-0.13.1
rootdir: C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp
collected 0 items / 1 error

===== ERRORS =====
ERROR collecting test/test_app.py
ImportError while importing test module 'C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp\test\test_app.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
C:\Users\artur\AppData\Local\Programs\Python\Python38-32\lib\importlib\_init_.py:127: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
test\test_app.py:7: in <module>
    from app import text_contains_word
E ImportError: cannot import name 'text_contains_word' from 'app' (C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp\app.py)
===== short test summary info =====
ERROR test/test_app.py
!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!!
===== 1 error in 0.17s =====
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>
```

Rysunek 13. Niepomyślne wykonanie trzeciego testu

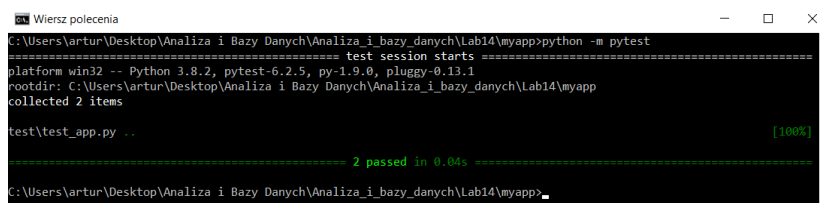
### 6.2. Faza Green

Zaimplementowano funkcję `text_contains_word()`.

```
def text_contains_word(word: str, text: str):
    return word in text
```

Rysunek 14. Implementacja trzeciej funkcji

Funkcja pomyślnie przeszła test.



```
Wiersz polecenia
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>python -m pytest
===== test session starts =====
platform win32 -- Python 3.8.2, pytest-6.2.5, py-1.9.0, pluggy-0.13.1
rootdir: C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp
collected 2 items

test\test_app.py .. [100%]

===== 2 passed in 0.04s =====
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp>
```

Rysunek 15. Pomyślne wykonania trzeciego testu

### 6.3. Faza Refactor

Funkcja jest na tyle prosta, że nie ma potrzeby ulepszania.

## 7. Czwarta funkcja

Celem jest napisanie funkcji realizującej sortowanie bąbelkowe.

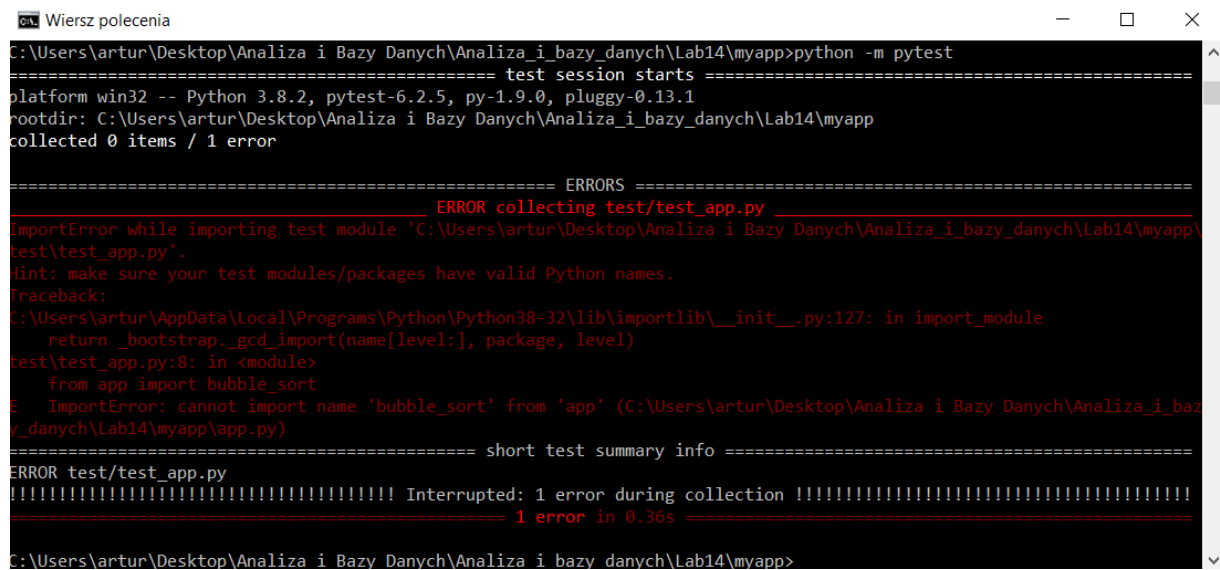
### 7.1. Faza Red

```
float_array = list(np.random.uniform(size=(1, 20)))
sort_arrays = [([3, 0, 6, 9, 8, 7, 4], [0, 3, 4, 6, 7, 8, 9]), ([5, 4, 3, 2, 1], [1, 2, 3, 4, 5]),
               ([i for i in range(20)], [i for i in range(20)]),
               ([i for i in range(30 - 1, -1, -1)], [i for i in range(30)]),
               (float_array, sorted(float_array))]

@pytest.mark.parametrize("input_array, output_array", sort_arrays)
def test_bubble_sort(input_array, output_array):
    assert bubble_sort(input_array) == output_array
```

Rysunek 16. Czwarty test

Stworzono i wykonano test. Naturalnie pojawia się komunikat o błędzie, gdyż funkcja nie została jeszcze zaimplementowana. Wykorzystano dekorator `pytest.mark.parametrize()`, który pozwala na przetestowanie wielu danych wejściowych.



```
Wiersz polecenia
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza i_bazy_danych\Lab14\myapp>python -m pytest
===== test session starts =====
platform win32 -- Python 3.8.2, pytest-6.2.5, py-1.9.0, pluggy-0.13.1
rootdir: C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp
collected 0 items / 1 error

===== ERRORS =====
ERROR collecting test/test_app.py
ImportError while importing test module 'C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_bazy_danych\Lab14\myapp\
test\test_app.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
C:\Users\artur\AppData\Local\Programs\Python\Python38-32\lib\importlib\__init__.py:127: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
test\test_app.py:8: in <module>
    from app import bubble_sort
E   ImportError: cannot import name 'bubble_sort' from 'app' (C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza_i_baz
y_danych\Lab14\myapp\app.py)
===== short test summary info =====
ERROR test/test_app.py
!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!!
===== 1 error in 0.36s =====
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza i_bazy_danych\Lab14\myapp>
```

Rysunek 17. Niepomyślne wykonanie czwartego testu

## 7.2. Faza Green

Zaimplementowano funkcję `bubble_sort()`.

```
def bubble_sort(array: List[int]) -> List[int]:
    lst = array[:]
    n = len(lst) - 1

    if_changed = True

    while n > 0 and if_changed:
        if_changed = False

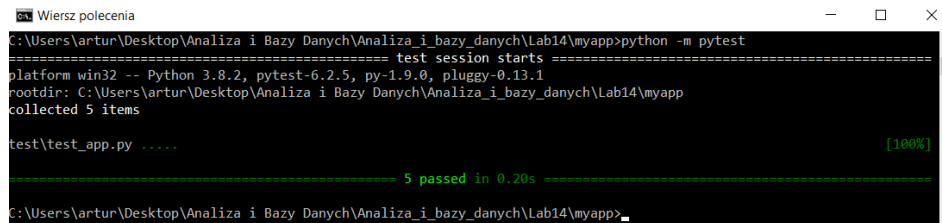
        for i in range(n):
            if lst[i] > lst[i + 1]:
                lst[i], lst[i + 1] = lst[i + 1], lst[i]
                if_changed = True

        n -= 1

    return lst
```

Rysunek 18. Implementacja trzeciej funkcji

Funkcja pomyślnie przeszła test.



```
Wiersz polecenia
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza i bazy danych\Lab14\myapp>python -m pytest
===== test session starts =====
platform win32 -- Python 3.8.2, pytest-6.2.5, py-1.9.0, pluggy-0.13.1
rootdir: C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza i bazy danych\Lab14\myapp
collected 5 items

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

===== 5 passed in 0.20s =====
C:\Users\artur\Desktop\Analiza i Bazy Danych\Analiza i bazy danych\Lab14\myapp>
```

Rysunek 19. Pomyślne wykonania trzeciego testu

## 7.3. Faza Refactor

Funkcja jest na tyle prosta, że nie ma potrzeby ulepszania.

## 8. Podsumowanie

Implementowane funkcje były na tyle proste, że wystarczający był 1 cykl. W ogólnym przypadku może ich być znacznie więcej.

Pytest jest bardzo przydatnym frameworkiem, która pozwala na dogłębne i przejrzyste przetestowanie implementowanych funkcjonalności.

Test Driven Development to na pierwszy rzut oka nietypowe podejście, gdyż zaczyna się od napisania testu, a dopiero potem implementuje się właściwą funkcjonalność. Jednak taka zasada pozwala na uporządkowanie programu i zapewnienie, że na każdym etapie działa on poprawnie.