

# Testowanie aplikacji

Poziomy testowania, pytest

# Poziomy testowania

**Poziom testów** czyli grupa czynności testowych, które są razem zorganizowane.

*Istotą rozróżniania poziomów testów jest to, że każdy poziom ma inne cele testowania, ma zwykle inną podstawę testów, a także inny obiekt testowania.*

*Typowe poziomy testowania:*

- **jednostkowe** - *testowanie pojedynczych modułów,*
- **integracyjne** - *testowanie wykonywane w celu wykrycia defektów podczas interakcji między komponentami lub systemami,*
- **systemowe** - *testowanie zintegrowanego systemu w celu sprawdzenia jego zgodności z wyspecyfikowanymi wymaganiami,*
- **akceptacyjne** - *testowanie formalnie przeprowadzane w celu umożliwienia użytkownikowi, klientowi lub innemu ustalonemu podmiotowi ustalenia, czy zaakceptować system lub moduł.*



# pytest

*The pytest framework makes it easy to write small tests.*

(źródło: <https://docs.pytest.org/en/latest/>)

Pytest pozwala napisać proste testy z użyciem instrukcji `assert`, dodatkowo potrafi uruchomić testy napisane z użyciem modułu `unittest`.

Do napisania testu z użyciem `pytest` wymagana jest instalacja pakietu `pytest`.

## Kluczowe słownictwo:

- *test fixture* - określa akcje potrzebne do przygotowania testu i jego zakończenia.
- *test case* - przypadek testowy, test.
- *test suite* - kolekcja test case i test suite, służy do grupowania testów, które należy wykonać razem.



Testowanie aplikacji

# pytest - Struktura projektu

Przykładowa struktura plików projektu.

Pliki z testami umieszczamy w katalogu z implementacją lub w osobnym katalogu o nazwie tests.

**Ważne jest, by nazwa pliku z testami zaczynała się od frazy `test_`, inaczej testy się nie uruchomią!**



Testowanie aplikacji

**struktura plików:  
testy w katalogu  
z implementacją**

```
.
|-- kalkulator.py
`-- utils
    |-- strings.py
    |-- test_strings.py
    |-- user.py
    `-- test_user.py
```

**struktura plików:  
testy w osobnym  
katalogu**

```
.
|-- kalkulator.py
|-- utils
|   |-- strings.py
|   `-- user.py
`-- tests
    |-- __init__.py
    |-- utils
    |-- __init__.py
    |-- test_strings.py
    `-- test_user.py
```

**wymagany plik**

**nazwa pliku zaczyna się od `test_`**

# pytest - Tworzenie testów

## Tworzenie testów

rozpoczynamy od utworzenia funkcji. Jedna funkcja odpowiada jednemu przypadkowi testowemu.

**Nazwy poszczególnych testów powinny zaczynać się od frazy `test_`, inaczej testy się nie uruchomią!**

**`tests/utils/test_strings.py:`**

```
from utils.strings import czy_liczba
```

```
def test_poprawna_liczba_dodatnia():  
    result = czy_liczba("5")  
    assert result is True
```



**nazwa zaczyna się od `test_`**



# pytest - Warunki testowe

## Warunki testowe

sprawdzone są za pomocą instrukcji `assert`.

Jeśli warunek instrukcji `assert` nie zostanie spełniony wykonywanie testu zostanie przerwane i pojawi się komunikat o błędzie.

**tests/utils/test\_strings.py:**

```
from utils.strings import czy_liczba
```

```
def test_poprawna_liczba_dodatnia():  
    result = czy_liczba("5")  
    assert result is True
```



# pytest - Warunki testowe

Wybrane warunki w instrukcji `assert`:

- `a == b`
- `a != b`
- `x is True`
- `x is False`
- `x is None`
- `x is not None`
- `a in b`
- `a not in b`

W przypadku, gdy warunek instrukcji `assert` nie zostanie spełniony test zostaje przerwany i oznaczony jako FAIL.



# pytest - Warunki testowe

Funkcja **pytest.raises()**, sprawdza czy instrukcje umieszczone w specjalnym bloku **with** rzuca wyjątek, jako parametr przyjmuje klasę oczekiwanego wyjątku.

```
import pytest

def test_liczba_z_litera():
    with pytest.raises(NotDigit):
        check_digit("3a")
```

<https://docs.pytest.org/en/latest/assert.html>





# pytest - Klasy z testami

Testy mogą zostać  
pogrupowane za pomocą  
klasy.

Nazwa klasy musi zaczynać  
się od słowa Test.

tests/utils/test\_strings.py:

```
from utils.strings import czy_liczba
```

```
class TestCzyLiczba:
```

```
    def test_poprawna_liczba_dodatnia(self):  
        result = czy_liczba("5")  
        assert result is True
```


```
    def test_poprawna_liczba_ujemna(self):  
        result = czy_liczba("-1")  
        assert result is True
```



# pytest - fixture

Testy mogą również wykorzystywać specjalne **metody uruchamiane przed rozpoczęciem testu**, który tego wymaga.

Takie metody oznaczane są za pomocą `@pytest.fixture`.



```
import pytest

@pytest.fixture
def sut():
    return Kalkulator()

def test_dodawanie(sut):
    assert sut.dodaj(2, 3) == 5

def test_true():
    assert True is True
```



# pytest - fixture

Nazwa oznaczonej funkcji musi zgadzać się z nazwą parametru przyjmowanego przez funkcję testową, by ta została uruchomiona dla danego testu.


Funkcja `sut()` zostanie uruchomiona dla `test_dodawanie()`, nie zostanie uruchomiona dla `test_true()`.

```
import pytest

@pytest.fixture
def sut():
    return Kalkulator()

def test_dodawanie(sut):
    assert sut.dodaj(2, 3) == 5

def test_true():
    assert True is True
```



# pytest - Uruchamianie testów

Testy uruchamiamy za pomocą polecenia **pytest** wykonywanego z poziomu katalogu głównego projektu.

Opcjonalnym argumentem do polecenia jest ścieżka do pliku z testem.



Testowanie aplikacji

# pytest - Uruchamianie testów

Przykład uruchomienia programu pytest bez argumentów.

```
$ pytest
===== test session starts =====
platform linux -- Python 3.7.0, pytest-4.3.0,
py-1.8.0, pluggy-0.9.0
rootdir: /home/user/, inifile:
collected 1 item

tests/utils/test_strings.py .                [100%]

===== 1 passed in 0.01 seconds =====
```



Testowanie aplikacji

# pytest - Uruchamianie testów

Przykład uruchomienia programu pytest z podaniem ścieżki do pliku z testami.

```
$ pytest tests/utils/test_strings.py
===== test session starts =====
platform linux -- Python 3.7.0, pytest-4.3.0,
py-1.8.0, pluggy-0.9.0
rootdir: /home/user/, inifile:
collected 1 item

tests/utils/test_strings.py .                [100%]

===== 1 passed in 0.01 seconds =====
```



Testowanie aplikacji

# pytest - Uruchamianie testów

Przykładowy wydruk  
z wykonania testu,  
który się nie powiódł.

```
===== test session starts =====
[...]
tests/utils/test_strings_pytest.py F [100%]

===== FAILURES =====
_____ test_poprawna_liczba_dodatnia _____

    def test_poprawna_liczba_dodatnia():
        result = czy_liczba("5")
>         assert result is True
E         assert False is True

tests/utils/test_strings_pytest.py:8:
AssertionError
===== 1 failed in 0.02 seconds =====
```



Testowanie aplikacji

# Pytania

1. Jaki przedrostek powinny zawierać nazwy plików z testami i nazwy testów?
2. Jakiej instrukcji używa pytest do sprawdzania warunków testowych?
3. Co się stanie jeśli warunek instrukcji `assert` nie zostanie spełniony?





# Literatura

1. Full pytest documentation,  
<https://docs.pytest.org/en/latest/contents.html>
2. Adam Roman, Testowanie i jakość oprogramowania. Metody, narzędzia, techniki, 2015,  
<https://ksiegarnia.pwn.pl/Testowanie-i-jakosc-oprogramowania.-Modele-techniki-narzedzia.,732463348,p.html>



Testowanie aplikacji

