# Testowanie aplikacji

Poziomy testowania, unittest

# 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.



#### Unittest

The unittest unit testing framework was originally inspired by **JUnit** and has a similar flavor as major unit testing frameworks in other languages. (źródło: https://docs.python.org/3/library/unittest.html)

unittest jest biblioteką dostarczaną razem z Pythonem, nie wymaga instalacji.

#### Kluczowe słownictwo:

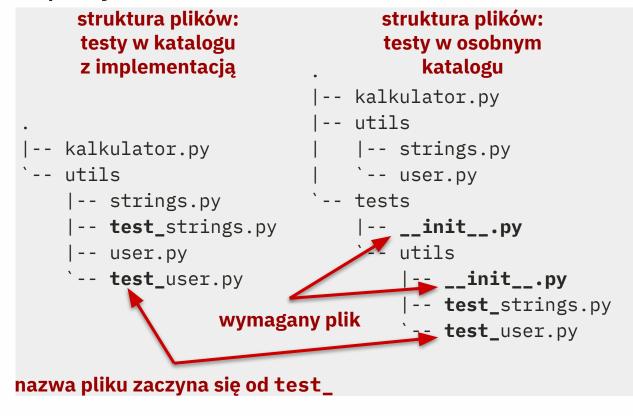
- *test fixture* określa akcje potrzebne do przygotowania testu i jego zakończenia.
- test case przypadek testowy, test.
- test suite kolekcja test case i test suite, służy do grupowania testów, które należy wykonać razem.

## Unittest - 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!





#### Unittest - Tworzenie testów

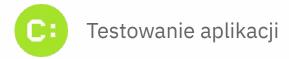
#### Tworzenie testów

rozpoczynamy od utworzenia klasy, w której umieszczone będą poszczególne przypadki testowe.

Klasa ta musi dziedziczyć po unittest. TestCase.

Nazwy poszczególnych testów powinny zaczynać się od frazy test\_, inaczej testy się nie uruchomią!

```
pamietamy o zaimportowaniu
                                      biblioteki
    tests/utils/test strings.py:
    import unittest *
    from utils.strings import czy_liczba
                                   konieczne dziedziczenie
    class CzyLiczba(unittest.TestCase):
        def test_poprawna_liczba_dodatnia(self):
            result = czy_liczba("5")
            self.assertTrue(result)
nazwa zaczyna się od test
```



#### Warunki testowe

sprawdzane są za pomocą metod typu assert\*.

W przykładzie została użyta metoda **assertTrue()**, sprawdza ona czy podany argument ma wartość True, jeśli nie, wykonywany test zostanie przerwany i pojawi się komunikat o błędzie.

```
tests/utils/test strings.py:
import unittest
from utils.strings import czy_liczba
class CzyLiczba(unittest.TestCase):
    def test_poprawna_liczba_dodatnia(self):
        result = czy liczba("5")
        self.assertTrue(result)
```

#### Wybrane metody typu assert\*():

- assertEqual(a, b)
- assertNotEqual(a, b)
- assertTrue(x)
- assertFalse(x)
- assertIs(a, b)
- assertIsNot(a, b)
- assertIsNone(x)
- assertIsNotNone(x)
- assertIn(a, b)
- assertNotIn(a, b)

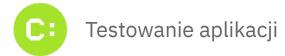
$$a == b$$

$$a != b$$

$$bool(x)$$
 is True

$$bool(x)$$
 is False

$$x$$
 is not None



Wybrane metody typu assert\*():

```
assertAlmostEqual(a, b) round(a-b, 7) == 0
assertNotAlmostEqual(a, b) round(a-b, 7) != 0
assertGreater(a, b) a > b
assertGreaterEqual(a, b) a >= b
assertLess(a, b) a < b</li>
assertLessEqual(a, b) a <= b</li>
```

W metodach typu **assert\*()** za parametr **a** podajemy wartość uzyskaną w wyniku działania testu, parametr **b** powinien przyjmować wartość oczekiwaną. Ma to znaczenie podczas wyświetlania komunikatu o błędzie.

W przypadku, gdy warunek metody typu assert\*() nie zostanie spełniony test zostaje przerwany i oznaczony jako FAIL.

Metoda assertRaises(), sprawdza czy instrukcje umieszczone w specjalnym bloku with rzucą wyjątek, jako parametr przyjmuje klasę oczekiwanego wyjątku.

```
import unittest

class CheckDigit(unittest.TestCase):

    def test_liczba_z_litera(self):
        with self.assertRaises(NotDigit):
        check_digit("3a")
```

## Unittest - setUp, tearDown

Testy mogą również wykorzystywać specjalne metody setUp() oraz tearDown() do przygotowania i zakończenia testu.

Metody te zostaną wywołane przed rozpoczęciem i po zakończeniu każdego z testów.

```
import unittest
class TestKalkulator(unittest.TestCase):
    def setUp(self):
        self.kalkulator = Kalkulator()
    def test_dodaj(self):
        result = self.kalkulator.dodaj(2, 3)
        self.assertEqual(result, 5)
    def tearDown(self):
        self.kalkulator = None
```

Testy uruchamiamy za pomocą polecenia **python -m unittest** wykonywanego z poziomu katalogu głównego projektu.

#### Argumentem do polecenia jest:

- ścieżka w formie importu,
- ścieżka do pliku z testem,
- polecenie discover automatycznie znajduje testy.

Przykład uruchomienia z argumentem:

- ścieżki w formie importu,
- ścieżki do pliku z testem,
- poleceniem discover.

# Przykład uruchomienia z argumentem:

- ścieżki w formie importu,
- ścieżki do pliku z testem,
- poleceniem discover.

# Przykład uruchomienia z argumentem:

- ścieżki w formie importu,
- ścieżki do pliku z testem,
- poleceniem discover.

Warto wiedzieć, że polecenia te przyjmują dodatkowy argument - v, który powoduje wyświetlanie większej ilości informacji.

```
$ python -m unittest discover -v
test_poprawna_liczba_dodatnia
      (tests.utils.test_strings.CzyLiczba) ... ok
Ran 1 test in 0.000s
0K
$ python -m unittest -v tests/utils/test_strings.py
test_poprawna_liczba_dodatnia
      (tests.utils.test_strings.CzyLiczba) ... ok
Ran 1 test in 0.000s
0K
```

Przykładowy wydruk z wykonania testu, który się nie powiódł.

```
$ python -m unittest discover
FAIL: test poprawna liczba dodatnia
           (tests.utils.test_strings.CzyLiczba)
Traceback (most recent call last):
  File "tests/utils/test strings.py", line 10, \
              in test poprawna liczba dodatnia
    self.assertTrue(result)
AssertionError: False is not true
Ran 1 test in 0.000s
FAILED (failures=1)
```

## Pytania

- 1. Jaki przedrostek powinny zawierać nazwy plików z testami i nazwy testów?
- 2. Jakich funkcji używa unittest do sprawdzenia warunków testowych?
- 3. Co się stanie jeśli warunek funkcji typu assert\*() nie zostanie spełniony?

#### Literatura

- unittest Unit testing framework, https://docs.python.org/3/library/unittest.html
- 2. Adam Roman, Testowanie i jakość oprogramowania. Metody, narzędzia, techniki, 2015, <a href="https://ksiegarnia.pwn.pl/Testowanie-i-jakosc-oprogramowania.-Modele-techniki-narzedzia..732463348.p.html">https://ksiegarnia.pwn.pl/Testowanie-i-jakosc-oprogramowania.-Modele-techniki-narzedzia..732463348.p.html</a>



