

Kurs rozszerzony języka Python

Wykład 12.

Marcin Młotkowski

7 stycznia 2025

Plan wykładu

- 1 Kontrola poprawności podczas biegu programu
- 2 Testowanie oprogramowania
 - PyUnit
 - Alternatywa: pydoc
 - Inne środowiska testowe
- 3 Statyczna analiza kodu
- 4 Debugowanie
- 5 Pomiar wydajności aplikacji

Plan wykładu

- 1 Kontrola poprawności podczas biegu programu
- 2 Testowanie oprogramowania
 - PyUnit
 - Alternatywa: pydoc
 - Inne środowiska testowe
- 3 Statyczna analiza kodu
- 4 Debugowanie
- 5 Pomiar wydajności aplikacji

Asercje

- Asercja to formuła logiczna;
- Asercji używa się do kontrolowania czy np. wartość zmiennej ma odpowiedni typ lub mieści się w pożądanym zakresie;
- Do kontroli używa się instrukcji
`assert wyrażenie`
- W przypadku niespełnienia wyrażenia zgłaszany jest wyjątek `AssertionError`

Przykład użycia asercji

```
def dodaj(x, y):  
    assert type(x) in [int, float]  
    assert type(y) == str, f'y jest typu {type(y)}!'  
    return x + int(y)
```

Przykład użycia asercji

```
def dodaj(x, y):  
    assert type(x) in [int, float]  
    assert type(y) == str, f'y jest typu {type(y)}!'  
    return x + int(y)
```

```
>>> dodaj(2.0, 2.0)
```

```
Traceback (most recent call last):
```

```
File "asercje.py", line 12, in <module>
```

```
    dodaj(2.0, 3.0)
```

```
File "asercje.py", line 7, in dodaj
```

```
    assert type(y) is str, f'Parametr y jest typu {type(y)}'
```

```
AssertionError: Parametr y jest typu <class 'float'> zamiast
```

Uwagi

Asercje spowalniają działanie programu.

Wyłączanie asercji

- Asercje są sprawdzane w zależności od zmiennej logicznej `__debug__` ;
- domyślna wartość `__debug__` : **True**;
- zmiennej `__debug__` nie można modyfikować w czasie wykonywania programu
- W przypadku uruchomienia programu z opcją `'-O'` (optymalizacja) wartością `__debug__` jest **False**.

Inne wykorzystanie `--debug--`

```
import logging
logging.basicConfig(format='%(asctime)s %(message)s',
                    filename='myapp.log', level=logging.INFO)

if __debug__:
    zmienne = locals().copy()
    for v in zmienne:
        logging.info("{}: {}".format(v, zmienne[v]))
```

Plan wykładu

- 1 Kontrola poprawności podczas biegu programu
- 2 **Testowanie oprogramowania**
 - PyUnit
 - Alternatywa: pydoc
 - Inne środowiska testowe
- 3 Statyczna analiza kodu
- 4 Debuggowanie
- 5 Pomiar wydajności aplikacji

Szybkie testy (trace debugging)

```
def fib(n):  
    if n < 2:  
        return 1  
    return fib(n-1) + fib(n-2)  
  
print(fib(2))  
print(fib(3))  
print(fib(4))
```

Niedogodności takiego testowania

```
import fib
```

Jak szybko poprawić

```
def fib(n):  
    if n < 2:  
        return 1  
    return fib(n - 1) + fib(n - 2)  
  
if __name__ == "__main__":  
    print(fib(2))  
    print(fib(3))  
    print(fib(4))
```

Wprowadzenie do testowania automatycznego

Test jednostkowy(ang. unit test)

Test sprawdzający poprawność pojedynczego elementu oprogramowania: metody, klasy czy procedury.

Wprowadzenie do testowania automatycznego

Test jednostkowy(ang. unit test)

Test sprawdzający poprawność pojedynczego elementu oprogramowania: metody, klasy czy procedury.

Zestaw testów (ang. test suite)

Implementuje się zestawy testów, które można uruchomić automatycznie.

Testy jednostkowe

Testy jednostkowe można traktować jako specyfikację klasy bądź modułu.

Testowanie

Co jest testowane:

- czy poprawne dane dają poprawny wynik;
- czy niepoprawne dane dają oczekiwany (np. niepoprawny wynik) lub wyjątek.

Narzędzia do testowania w Pythonie

- PyUnit
- PyDoc

Zadanie

Napisać funkcję `deCapitalize` z argumentem typu `string` i zwracającą `string`

- Unifikacja imienia i nazwiska do postaci 'Imie Nazwisko', np.
`deCapitalize('JAN KOWALSKI'):` `'Jan Kowalski'`
- Kontrola typu, gdy argument nie jest typu `string`, zgłaszany jest wyjątek `ArgumentNotStringError`

PyUnit

- Testy są zebrane w odrębnym pliku (plikach)
- Można wskazywać, jakie testy mają być wykonane

Implementacja funkcji

```
class ArgumentNotStringError(Exception): pass

def deCapitalize(nazwisko):
    """Zamiana napisów (imion i nazwisk) pisanych
    wielkimi literami
    """
    return ""
```

Implementacja testów: unittest

```
import unittest
import types

class TestDeCapitalize(unittest.TestCase):
```

Zamiana na poprawną postać

```
znaneWartosci = [  
    ("jaN KoWaLski", "Jan Kowalski"),  
    ("cLaude leVi-StrAuSs", "Claude Levi-Strauss"),  
    ("JeRzy auGust MniSzEch", "Jerzy August Mniszech")  
]  
  
def testProsty(self):  
    """Proste sprawdzenia"""  
    for zly, dobry in self.znaneWartosci:  
        res = deCapitalize(zly)  
        self.assertEqual(dobry, res)
```

Test na identyczność

```
listaNazwisk = ["Benedykt Polak",  
                "Fryderyk Joliot-Curie"]  
  
def testIdent(self):  
    """Nie zamieni poprawnych nazwisk"""  
    for nazw in self.listaNazwisk:  
        self.assertEqual(nazw, deCapitalize(nazw),  
                          "Zmienił poprawne nazwiska")
```


Niepoprawne wyniki

```
psuj = [  
    ("Jean de Rond d'Alembert", "Jean de Rond d'Alembert"),  
    ("Ludwig van Beethoven", "Ludwig van Beethoven"),  
]  
  
def testZly(self):  
    """Nie radzi sobie"""  
    for zly, dobry in self.psuj:  
        res = deCapitalize(zly)  
        self.assertNotEqual(dobry, res)
```

Przypomnienie

```
class ArgumentNotStringError(Exception): pass

def deCapitalize(nazwisko):
    """
    Zamiana napisów (imion i nazwisk) pisanych
    wielkimi literami"""

    if type(nazwisko) != str:
        raise ArgumentNotStringError
```

Metoda testująca

```
def testDziedzina(self):
    self.assertRaises(ArgumentNotStringError,
                      deCapitalize, 10)
```

Podsumowanie

```
import unittest, types
import testowany_modul

class TestdeCapitalize(unittest.TestCase):
    def testProsty(self): ...
    def testIdent(self): ...
    def testDziedzina(self): ...

if __name__ == "__main__":
    unittest.main()
```

Uzupełnienie

Metoda `TestDeCapitalize.setUp(self)`

Inicjowanie wstępne wykonywane przed każdym testem (zakładanie baz danych i tabel, tworzenie plików/tabel z przykładowymi danymi).

Uzupełnienie

Metoda `TestDeCapitalize.setUp(self)`

Inicjowanie wstępne wykonywane przed każdym testem (zakładanie baz danych i tabel, tworzenie plików/tabel z przykładowymi danymi).

Metoda `TestDeCapitalize.tearDown(self)`

sprzątanie wykonywane po każdym teście (usuwanie tymczasowych plików etc).

Uruchomienie

```
python testy.py -v
```

```
testDziedzina (__main__.testy) ... ok
```

```
Nie zamieni poprawnych nazwisk ... ok
```

```
Proste sprawdzenia ... ok
```

```
Nie radzi sobie ... ok
```

```
-----
```

```
Ran 5 tests in 0.001s
```

```
OK
```

Zarządzanie zestawami testów

```
s1 = TestyDeCapitalize()  
s2 = modul.InneTesty()  
alltests = unittest.TestSuite([s1, s2])  
unittest.TextTestRunner(verbosity=3).run(alltests)
```

Ile testów?

Wskaźnik pokrycia kodu: jaka część funkcji, instrukcji, gałęzi etc. jest przechodzonych podczas testów.

Ile testów?

Wskaźnik pokrycia kodu: jaka część funkcji, instrukcji, gałęzi etc.
jest przechodzonych podczas testów.

Wyliczanie pokrycia w pythonie: coverage

```
$ coverage run -m unittest decapital_unittest.py  
$ coverage report  
$ coverage html
```

Organizacja testów

```
mojprojekt/  
    aplikacja/  
    testy/
```

Skąd testy mają wiedzieć, gdzie są moduły do testowania?

w katalogu `mojprojekt/testy/context.py`

```
import os  
import sys  
sys.path.insert(0, os.path.abspath("../aplikacja/"))
```

w każdym pliku `Test*.py`

```
import context
```

Testy regresyjne

Testowanie przetestowanych modułów z powodu:

- modyfikacja w innej części kodu;
- zmiany środowiska, np. system operacyjny czy środowisko programowania

Testy regresyjne

Testowanie przetestowanych modułów z powodu:

- modyfikacja w innej części kodu;
- zmiany środowiska, np. system operacyjny czy środowisko programowania

Python3.8 vs Python3.9:

What's New In Python 3.9, Other Language Changes:

`"".replace("", s, n)` now returns `s` instead of an empty string for all non-zero `n`. It is now consistent with `"".replace("", s)`.

Testowanie za pomocą pakietu doctest

Przypomnienie

```
print (modul.__doc__)  
help(modul)
```

Testy w komentarzach

```
def deCapitalize(nazwisko):  
    """  
    Zamiana napisów (nazwisk) pisanych wielkimi literami.  
    Przykłady:  
    >>> [deCapitalize(n) for n in ['KazImieRz WieLki', 'Stefan Bato  
    ['Kazimierz Wielki', 'Stefan Batory']  
  
    >>> deCapitalize('Henryk Walezy')  
    'Henryk Walezy'  
  
    >>> deCapitalize(2)  
    Traceback (most recent call last):  
    ...  
    ArgumentNotStringError  
    """  
    ...
```

Uruchomienie testów

```
if __name__ == "__main__":  
    import doctest  
    doctest.testmod()
```

doctest — wynik

Trying:

```
[deCapitalize(n) for n in ['Kaz  
Wielki','Stefan Batory']]
```

Expecting:

```
['Kaz Wielki', 'Stefan Batory']
```

ok

Trying:

```
deCapitalize('Henryk Walezy')
```

Expecting:

```
'Henryk Walezy'
```

ok

Trying:

```
deCapitalize(2)
```

Expecting:

```
Traceback (most recent call last):
```

```
...
```

```
ArgumentNotStringError
```

ok

```
2 items had no tests:
  __main__
  __main__.ArgumentNotStringError
1 items passed all tests:
  3 tests in __main__.deCapitalize
3 tests in 3 items.
3 passed and 0 failed.
Test passed.
```

nose

Rozszerzenie środowiska pyunit, m.in.:

- analiza wyjścia tekstowego (stdout);
- wybrane stesty (np. pominięcie długotrwałych testów);
- zbadanie pokrycia testami.

Selenium

Selenium

Środowisko do testowania aplikacji webowych poprzez symulację działań użytkownika za pomocą przeglądarki.

- nagrywanie scenariuszy za pomocą wtyczki w firefoxie;
- programowanie scenariuszy wraz z asercjami.

Krótki przykład

```
import unittest

from selenium import webdriver
from selenium.webdriver.common.keys import Keys

class IISearch(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
    def tearDown(self):
        self.driver.close()

    def test_prosty(self):
        self.driver.get("http://www.ii.uni.wroc.pl")
        self.assertIn("Instytut Informatyki", self.driver.title)

unittest.main()
```

Plan wykładu

- 1 Kontrola poprawności podczas biegu programu
- 2 Testowanie oprogramowania
 - PyUnit
 - Alternatywa: pydoc
 - Inne środowiska testowe
- 3 Statyczna analiza kodu
- 4 Debugowanie
- 5 Pomiar wydajności aplikacji

Dekorowanie typami

```
def greeting(name: str) -> str:  
    return "Hello" + name
```

Typy generyczne i aliasy

```
Vector = list[float]
```

```
def dodaj_wektory(v1: Vector, v2: Vector) -> Vector:  
    return
```

```
def tosam(value: Any) -> Any:  
    return value
```



```
def greeting(name: str) -> str:  
    return "Hello" + name
```

Co można z tym dalej zrobić

- MyPy
- Pytype (Google)
- Pylance (Microsoft)
- Pyre (Facebook)

Plan wykładu

- 1 Kontrola poprawności podczas biegu programu
- 2 Testowanie oprogramowania
 - PyUnit
 - Alternatywa: pydoc
 - Inne środowiska testowe
- 3 Statyczna analiza kodu
- 4 Debugowanie
- 5 Pomiar wydajności aplikacji

pdb

Wywołanie

```
$ python -m pdb mymodule.py  
(pdb) help  
(pdb)
```

Plan wykładu

- 1 Kontrola poprawności podczas biegu programu
- 2 Testowanie oprogramowania
 - PyUnit
 - Alternatywa: pydoc
 - Inne środowiska testowe
- 3 Statyczna analiza kodu
- 4 Debugowanie
- 5 Pomiar wydajności aplikacji

Pomiar wydajności fragmentu kodu

Klasa `timeit.Timer`

```
import timeit  
t = timeit.Timer(stmt='[6,5,4,3,2,1].sort()')  
print ('czas %.2f sec' % t.timeit())
```

Pomiar wydajności całego programu

Z linii poleceń

```
$ python3 -m timeit '[3,2,1].sort()'
```

Wynik

1000000 loops, best of 3: 0.483 usec per loop

Profilowanie

Profilowanie dostarcza informacji o czasie wykonywania poszczególnych funkcji, liczbie wywołań etc.

Przykład profilowania

Wywołanie

```
$ python3 -m cProfile my_doctest.py
```

Wynik

19287 function calls (19035 primitive calls) in 0.350 CPU seconds

Ordered by: standard name

	ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
	1	0.000	0.000	0.010	0.010	:0(__import__)
	3	0.000	0.000	0.000	0.000	:0(getframe)
	1	0.000	0.000	0.000	0.000	:0(allocate_lock)
	1436	0.000	0.000	0.000	0.000	:0(append)
	4	0.000	0.000	0.000	0.000	:0(callable)

profile vs cProfile

profile

Napisany w Pythonie, działa we wszystkich implementacjach Pythona. Duży narzut czasowy.

cPython

Działa tylko z cPythonem, ale przy małym narzucie czasowym.