PODSTAWY PROGRAMOWANIA W PYTHON

Dzień 12

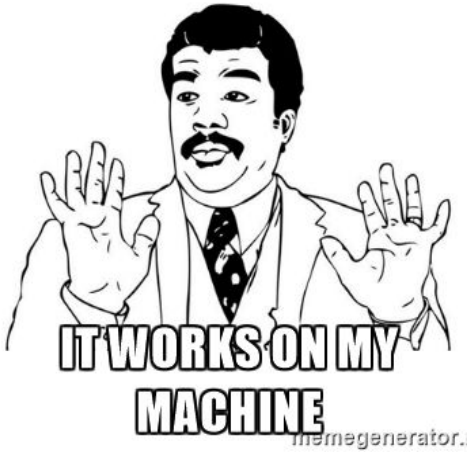
AGENDA
DAY 12

- testowanie kodu:
  - rodzaje testów
  - moduł unit test
  - moduł pytest

testowanie kodu


I SEE YOU TEST YOUR CODE IN PRODUCTION
I TOO LIKE TO LIVE DANGEROUSLY
quickmeme.com

TESTY



Po co testować kod?

- sprawdzanie poprawności implementacji
- dbanie o stabilność i ciągłość (consistency) kodu przy wprowadzaniu zmian, rozszerzaniu, refaktorowaniu
- odnajdywanie przypadków krańcowych
- zapewnienie niezależności od maszyny / środowiska developerskiego

TESTY

- **jednostkowe** – testujemy jednostkę logiczną naszego kodu
- integracyjne – sprawdzamy jak komponenty ze sobą współpracują
- systemowe – jak integracyjne, ale testujemy na poziomie systemów, usług zewnętrznych
- UI – testy interfejsu użytkownika

- manualne
- automatyczne

- black box
- white box

PYTHON
moduł unittest

```
from unittest import TestCase

class TestClass(TestCase):

    def test_test_a(self):
        …
        …
        self.assertEqual(a, b)
```

https://docs.python.org/3/library/unittest.html https://youtu.be/6tNS--WetLI

TESTY JEDNOSTKOWE

- testujemy poszczególne klasy, metody, properties

- muszą być niezależne od siebie
- muszą dawać ten sam rezultat niezależnie od kolejności wykonania

- możemy wspomóc się **code coverage** (pokrycie kodu testami) aby zobaczyć które części nie są przetestowane

- jeśli mamy sporo przypadków testowych dla jednego testu warto posłużyć się metodologią **DDT – Data Driven Tests** (testy sterowane danymi), najczęściej jako dekoratory metody testowej, lub zewnętrzny plik z przypadkami testowymi

PYTHON
moduł unittest

```
assertEqual
assertAlmostEqual
assertNotEqual
assertIsInstance
assertTrue
assertFalse
assertIsNone
assertIn
```

PYTHON
Data Driven Tests

instalujemy i importujemy moduł **ddt**

```
@ddt
class FooTestCase(unittest.TestCase):
    def test_undecorated(self):
        self.assertTrue(larger_than_two(24))

    @data(3, 4, 12, 23)
    def test_larger_than_two(self, value):
        self.assertTrue(larger_than_two(value))
```

dokumentacja: http://ddt.readthedocs.io/en/latest/

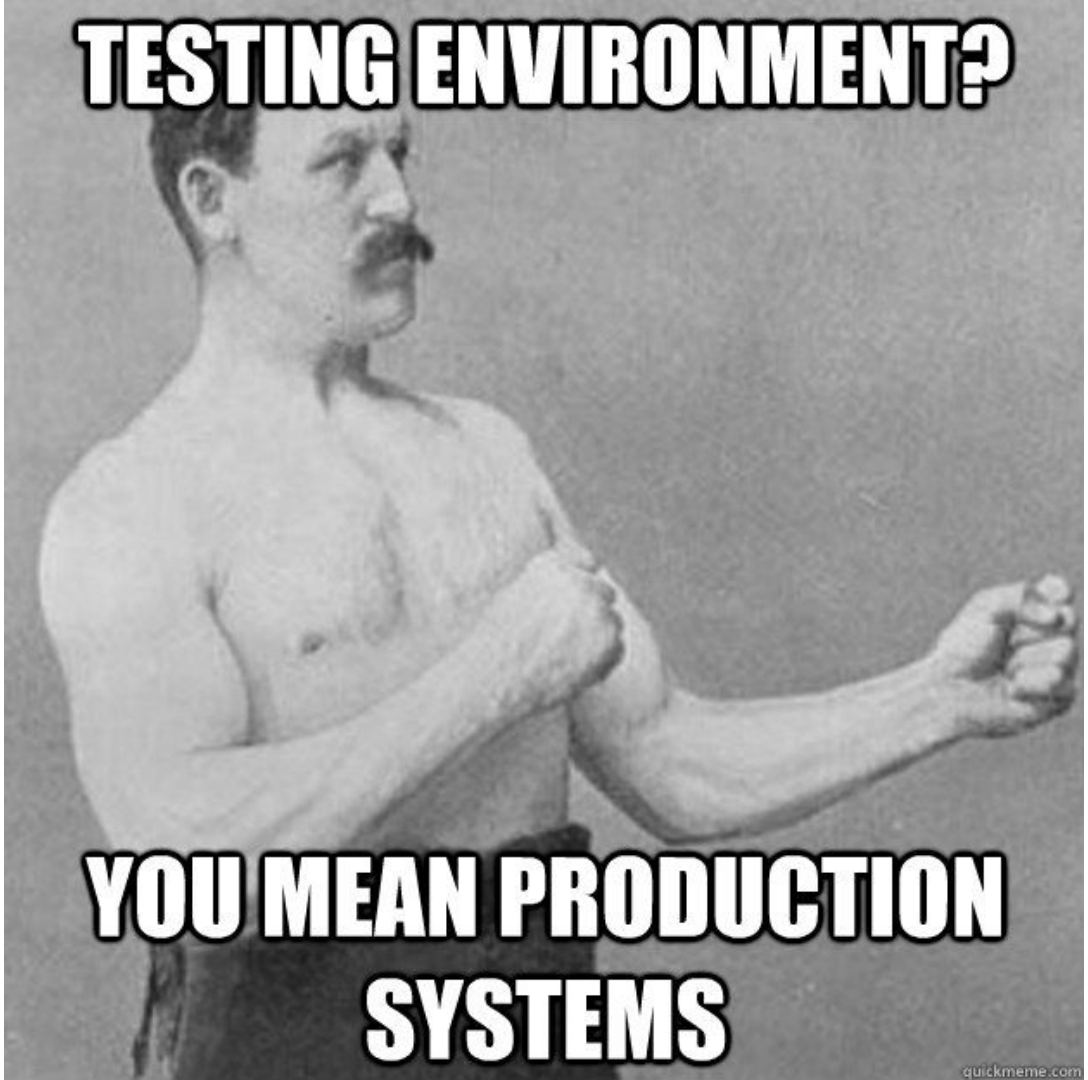
TDD

Test Driven Development - tworzenie kodu sterowane testami

Pisanie kodu składa się z trzech stanów:

- doprowadzenie do sytuacji, w której kod nie spełnia testów (test fails, red-phase)
- doprowadzenie do sytuacji, w której kod spełnia testy (test pass, green phase) - oczywiście poprawiając kod, nie zmieniając testy ;)
- refaktoryzacja kodu (refactor, blue phase) - optymalizowanie rozwiązania, bez zmiany funkcjonalności

Proces powtarzać w pętli, do momentu zaimplementowania docelowej funkcjonalności.


TESTING ENVIRONMENT?
YOU MEAN PRODUCTION SYSTEMS
quickmeme.com

Thanks!!