

PODSTAWY PROGRAMOWANIA W PYTHON

Dzień 13



AGENDA

DAY 13

- virtual env
- pip
- testowanie kodu
- praca z plikami
- modyfikowanie plików (zdjęcia)

| virtualenv

virtualenv

Wirtualne środowisko Python – kopia Pythona, ze specyficznymi ustawieniami, zainstalowanymi modułami itp.

Dzięki virtualenv możemy mieć środowiska z różnymi wersjami tych samych modułów.

pip install virtualenv

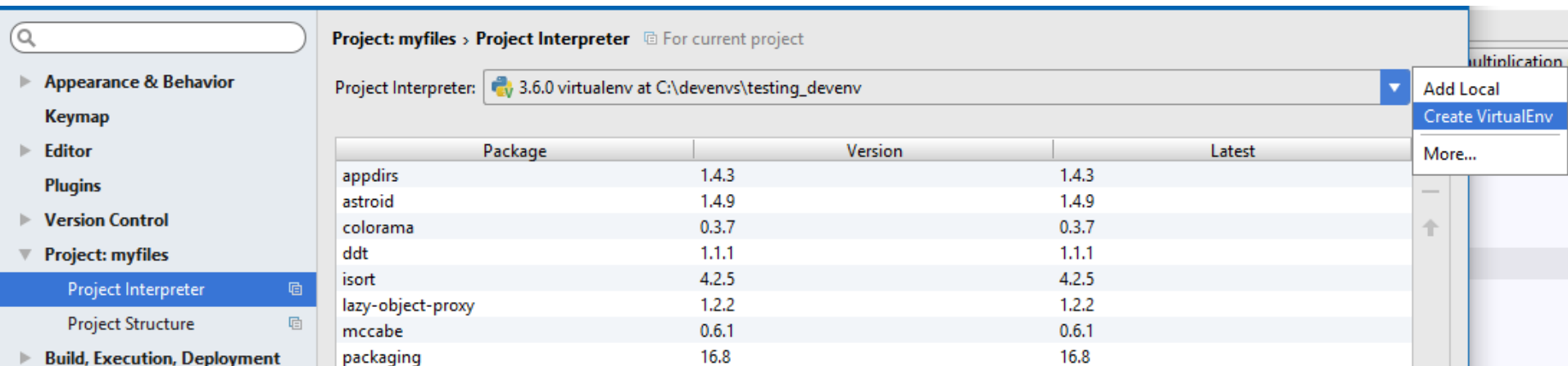
virtualenv mytestingenv – tworzymy virtualenv

mytestingenv/Scripts/activate – aktywujemy

(mytestingenv/**bin**/activate - Mac i Linux)

virtualenv w PyCharm

file -> settings -> project settings -> project interpreter



The screenshot shows the 'Project Interpreter' settings window in PyCharm. The left sidebar contains a search bar and a list of settings categories: Appearance & Behavior, Keymap, Editor, Plugins, Version Control, and Project: myfiles. Under 'Project: myfiles', 'Project Interpreter' is selected. The main panel shows the 'Project Interpreter' for 'myfiles' set to '3.6.0 virtualenv at C:\devenvs\testing_devenv'. Below this is a table of installed packages with columns for Package, Version, and Latest. A context menu is open on the right, showing options: Add Local, Create VirtualEnv, and More...

Project: myfiles > Project Interpreter For current project

Project Interpreter: 3.6.0 virtualenv at C:\devenvs\testing_devenv

Package	Version	Latest
appdirs	1.4.3	1.4.3
astroid	1.4.9	1.4.9
colorama	0.3.7	0.3.7
ddt	1.1.1	1.1.1
isort	4.2.5	4.2.5
lazy-object-proxy	1.2.2	1.2.2
mccabe	0.6.1	0.6.1
packaging	16.8	16.8

Context Menu:

- Add Local
- Create VirtualEnv
- More...

PyPI & pip

Menadżer pakietów Python

PyPI Python Package Index

lista dostępnych pakietów

pypi.python.org/pypi

pip

Menadżer pakietów instalowany razem z Python.
Komendy w wierszu poleceń:

pip help – ogólna pomoc

pip help install – pomoc dot. polecenia

pip list – lista zainstalowanych pakietów

pip search – szuka pakietów w repozytorium online

pip install *pakiet* – instalowanie modułu

pip uninstall *pakiet* - odinstalowanie

pip list -o -sprawdzenie nieaktualnych pakietów

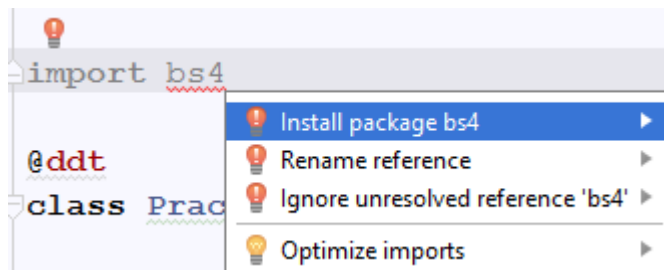
pip install -U *pakiet* - update pakietu

pip freeze > plik.txt – zapisanie informacji do pliku o pakietach

pip install -r plik.txt – zainstaluje wszystkie wymagane pakiety

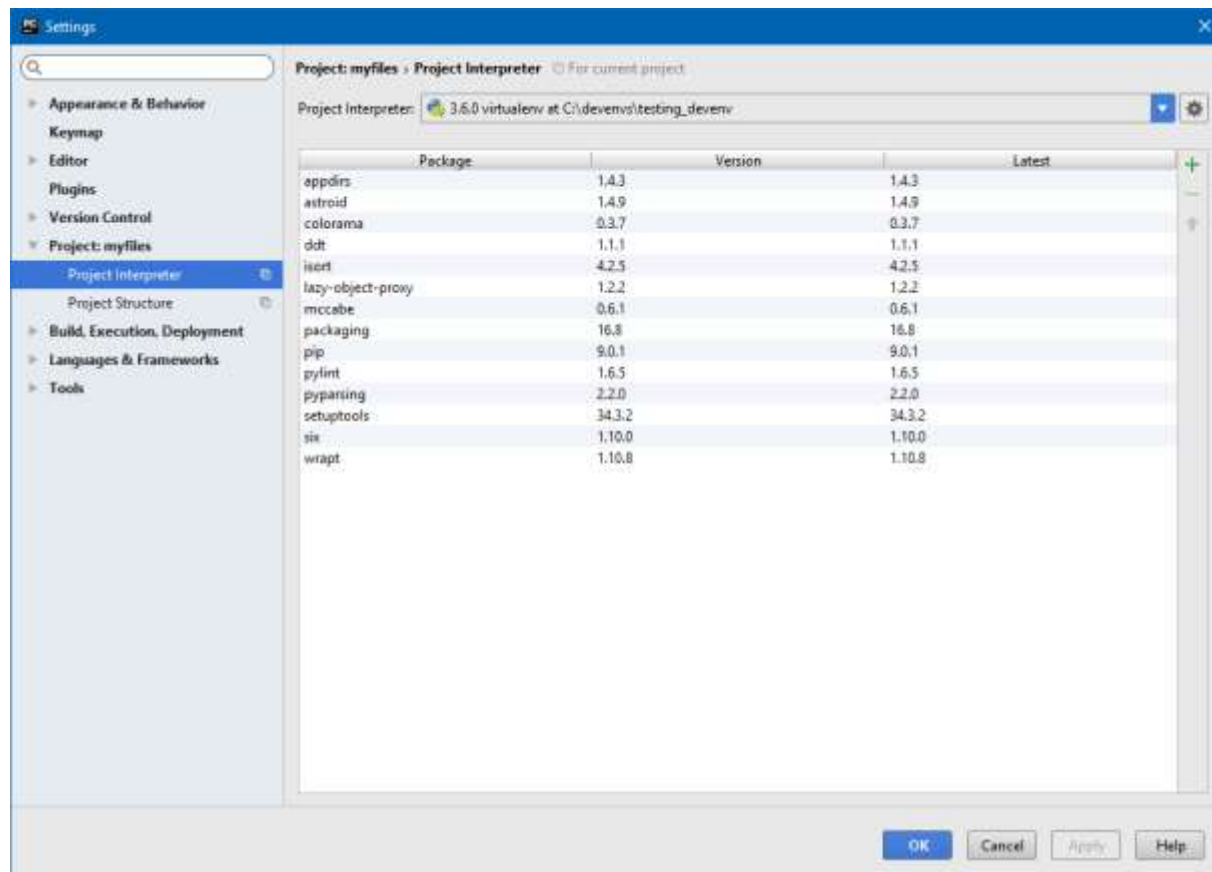
INSTALOWANIE PAKIETÓW W PYCHARM

- podpowiedzi przy pisaniu kodu – alt + enter



INSTALOWANIE PAKIETÓW W PYCHARM

file ->
settings ->
project ->
project interpreter



| testowanie kodu

**I SEE YOU TEST YOUR CODE IN
PRODUCTION**

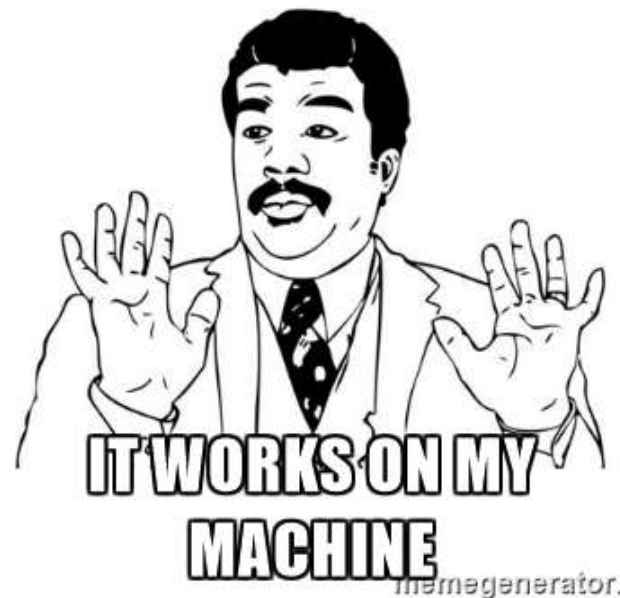
I TOO LIKE TO LIVE DANGEROUSLY

quickmeme.com

TESTY

Po co testować kod?

- sprawdzanie poprawności implementacji
- dbanie o stabilność i konsystentność kodu przy wprowadzaniu zmian, rozszerzaniu, refaktorowaniu
- odnajdywanie przypadków krańcowych
- niezależność 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

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

TDD

Test Driven Development

Tworzenie kodu sterowane testami

- piszemy test
 - uruchamiamy test
 - test fail
 - dopisujemy kawałek kodu
 - uruchamiamy test...
-
- ... test przechodzi, gdy cała funkcjonalność jest zaimplementowana

PYTHON

moduł unittest

```
from unittest import TestCase

class TestClass(TestCase):

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

PYTHON

modul 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/>

TESTING ENVIRONMENT?

**YOU MEAN PRODUCTION
SYSTEMS**

| praca z plikami

OS

- moduł os służy do pracy z plikami, ścieżkami, zmiennymi systemowymi

mkdir, chdir, getcwd, unlink, rmdir, listdir, walk

- os.path – działania na ścieżkach
split, join, abspath



shutil

wrapper dla poleceń systemowych, w pewnych sytuacjach ułatwia wykonanie poleceń

copytree, move, rmtree,

send2trash

przenosi do kosza

```
send2trash.send2trash(plik)
```


PIL – pillow działania na zdjęciach

```
pip install Pillow
```

```
from PIL import Image
```

```
foto = Image.open(plik)
```



Thanks!!