# Python i Ja (w sensie Ty)

...

Czyli po co i jak pracować z Pythonem.

# O mnie

- Michał Bultrowicz
- aka. Butla
- programista / (drobny) lead techniczny
- pracuje już tylko na Linuxie
- ulubiony język - PYTHON
- jedno z hobby - PYTHON
- https://github.com/butla

# Tematy na dziś

- Dlaczego Python?
- Jak się wkręcić?
- Jak pracować z Pythonem?
- Wasze pytania

# Filozofia

- Przyjemny język
- Zen of Python (`import this`)
  - “Beautiful is better than ugly”
  - “Explicit is better than implicit.”
  - One way to do it, etc.
- DRY, KISS
- TDD / BDD
- “Language for consenting adults”

# Zalety

- Ekspresywność
- Czytelność
- Szybkość pracy (i zmian)
- Bogaty zasób bibliotek (uniwersalność)
- Wszędobylskość
- Rozszerzenia w C
- Wolność


# Wady

- Wolny w “Number crunching”
  - Tu pomagają rozszerzenia w C (Numpy, Numba, Nuitka)
- Brak równoległości wątków
- Dostarczanie aplikacji / Zarządzanie zależnościami
- Brak statycznego typowania
  - Są typehinty
  - I tak trzeba napisać testy
- Za duża różnorodność

# Interpreter

- Python to język i specyfikacja
- Istnieje kilka implementacji maszyn wirtualnych (interpreterów) go wykonujących
- CPython, Jython, IronPython, PyPy, ActivePython, MicroPython
- Interpreter może działać w trybie interaktywnym

# Confusing live demo!

- Trochę Pythona bez zbytnich podstaw
- Wykorzystane narzędzie - Jupyter (dawniej IPython Notebook)
- Jupyter - aplikacja webowa hostująca interpreter i kawałki kodu

# Python 2 vs Python 3

- 3 jest lepiej zaprojektowany (nawet na poziomie C)
- Większość przydatnych bibliotek dostępnych dla 3
- 2 lekko szybszy w wielu benchmarkach, ale nie zawsze
- 3 ma nowe zabawki (async, type hints)
- 2 - End of Life w 2020
- “raise from”, lepszy print() i dużo więcej

# Nauka (początek)

- Python3
- https://docs.python.org/3/tutorial/index.html (o języku)
- http://docs.python-guide.org/en/latest/ (zwięźle, życiowo, nadal rozwijane)
- https://packaging.python.org/en/latest/
- Learning Python, O’Reilly (ponoć spoko)

# Nauka (dalsza)

- Python Cookbook, O’Reilly (jest super)
- http://planetpython.org/ (RSS feed)
- podglądanie projektów na Githubie
- PyConPL / EuroPython / PyCon?

# virtualenv

- Narzędzie do tworzenia odizolowanych środowisk (Python + biblioteki)
- Kluczowy dla programisty Pythona
- Reprodukowalność developmentu
- virtualenvwrapper - ułatwia globalne zarządzanie
- “venv” (pyvenv) - mniej popularna alternatywa wbudowana od Pythona 3.3.
- Nie polecam virtualenv-burrito

# Wincyj Pythonów!!11!1

sudo add-apt-repository ppa:fkrull/deadsnakes

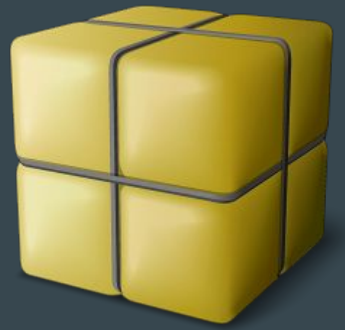
sudo apt-get update

sudo apt-get install -y python2.6 python2.6-dev

Jeden pip i virtualenv dla wszystkich.

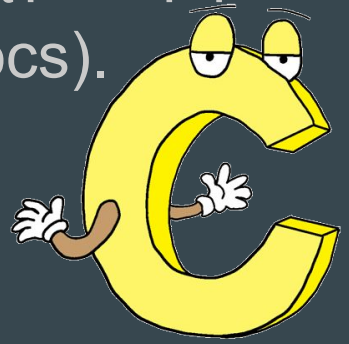
# Package management (Unix)

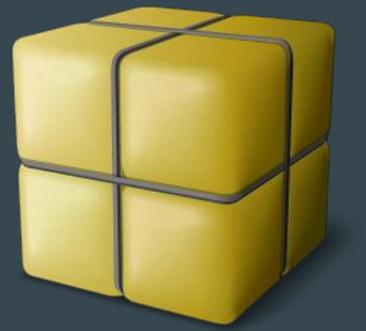
- PyPI/pip - analog Maven, NuGet, npm
- Nie używać easy_install
- Unikać instalacji bibliotek przez apt-get
- `sudo apt-get install -y python-pip3 python3.4-dev`
- `sudo pip3 install --upgrade pip`
- pip może być w Pythonie od 2.7.9 i 3.4



# Package management - zależności natywne

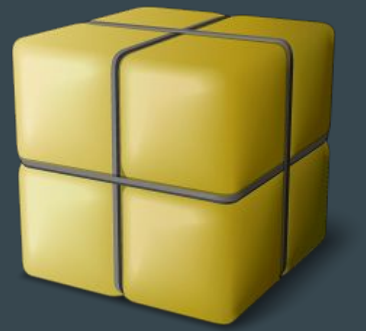
- Czasem wymagane .so nie przychodzą z biblioteką.
- Są paczki pobierane apt-getem, ale mieszają, jeśli używa się globalnie pipa.
- Można zobaczyć zależności paczki systemowej i dociągnąć tylko natywne, a następnie pipem Pythona.
- Patrz w dokumentację (readthedocs).





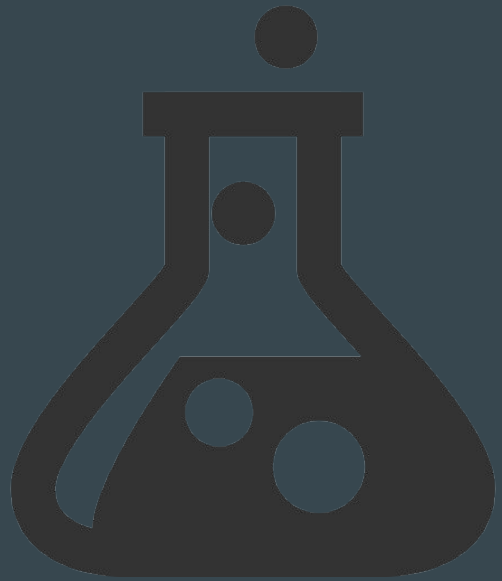
# Package management (Windows)

- Też pip, ale trzeba sobie ustawić kompilator
- http://blog.ionelmc.ro/2014/12/21/compiling-python-extensions-on-windows/
- Instalatory bibliotek - te same problemy co apt-get



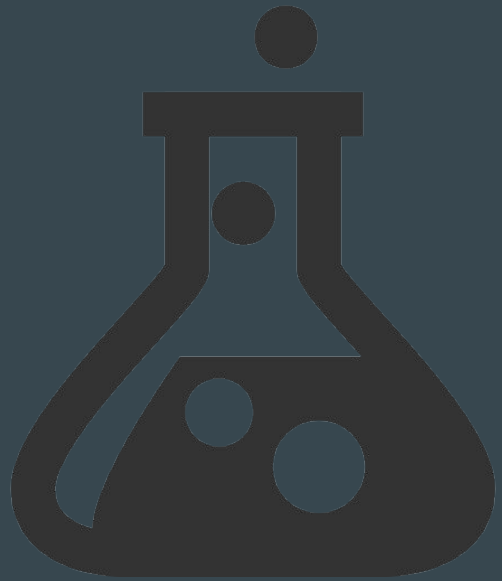
# Eksperymentowanie

- Rozwija i utwierdza wiedzę książkową
- “Odpal i popatrz”
- Interaktywny shell (i “półautomaty”)
- bpython (<3) / IPython / ptpython
- Małe skrypty
- Virtualenv(y)



# Eksperymentowanie

Kolejne “live demo”!



# GIL - Global Interpreter Lock

- Jeden mutex na cały proces Pythona
- Tylko jeden wątek na raz coś robi
- Zapobiega błędom przy liczeniu referencji
- Wszystko thread-safe :)
- Deterministyczne zwalnianie obiektów bez wzajemnych zależności

# Omijanie GILa

- Zbędne dla programów “IO-bound”
- Forkowanie procesu (ProcessPoolExecutor)
- Rozszerzenia w C
- PyPy

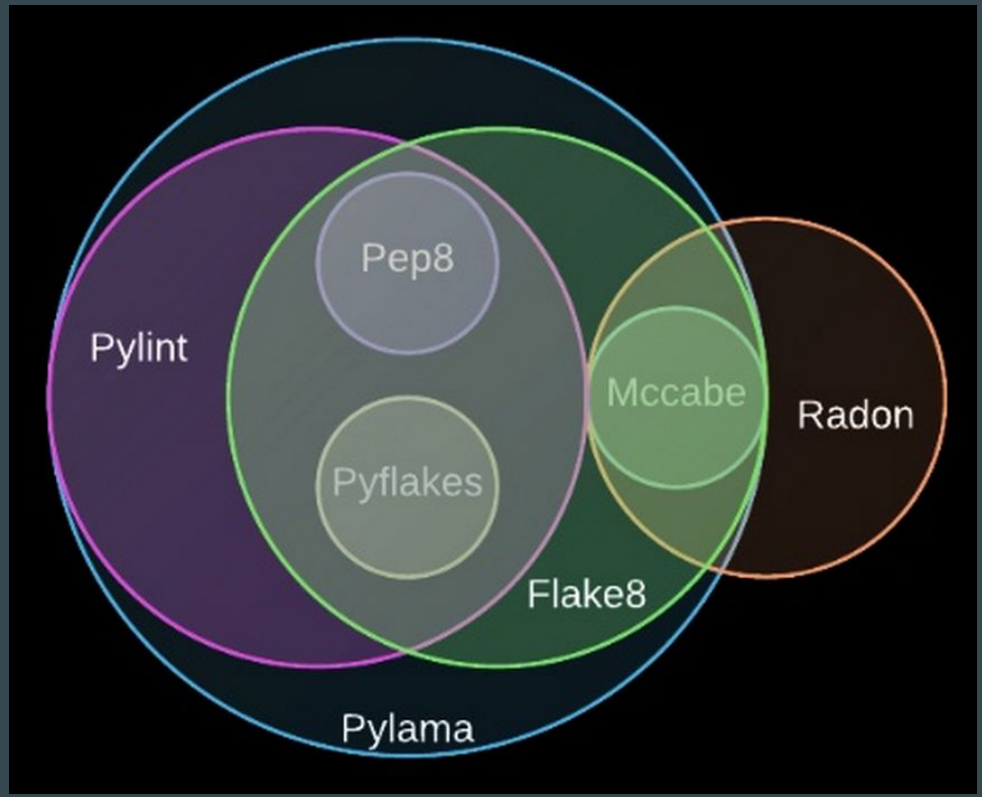
# Frameworki testowe

- unittest - wbudowany, “non-pythonic”
- nose - ?
- pytest - “pythonic”, zwięzły, magiczny (+/-)

```python
import pytest

def foo():
    return 5

def test_foo():
    assert foo() == 5

@pytest.fixture(scope='function')
def db_client():
    return DbClient()

def test_db(db_client):
    assert db_client.get('foo') == 'bar'
```

# Analiza statyczna

- pylint za bardzo się czepia, ale często jest coś na rzeczy
- Radosław Ganczarek: Code quality in Python - tools and reasons



# Docstringi

- Docstring - dokumentacja klasy, metody, wartości...
- Dwa główne formaty - rST, Google style.
- Pycharm generuje podpowiedzi o typach z obu.
- rST (reStructuredText) to format używany przez Sphinxa (narzędzie do generacji stron z dokumentacją).
- Google style chce być mniej zatłoczony wizualnie.
- Napoleon - wtyczka do Sphinxa dla Google style.

# Docstringi - rST

```
def bla(n):
    """
    Does the bla.
    :param int n: some number
    :returns: True if n is even, false otherwise.
    :rtype: bool
    :raises ValueError: When n > 10.
    """
    if n > 10:
        raise ValueError('n can't be > 10')
    return n % 2 == 0
```

# Docstringi - Google style

```
def bla(n):
    """Does the bla.

    Args:
        n (int): some number

    Returns:
        bool: True if n is even, false otherwise.

    Raises:
        ValueError: When n > 10.
    """
    if n > 10:
        raise ValueError('n can't be > 10')
    return n % 2 == 0
```

# Tox

- Puszcza testy na wielu wersjach/konfiguracjach Pythona.
- Można zintegrować z testami pokrycia i analizą statyczną.
- Tworzy virtualenva (można go podpiąć pod IDE)
- Używany trochę jako narzędzie buildowe
- Może wywoływać generację dokumentacji…
- ...i wszystko inne

# Wersjonowanie

- Bumpversion
  - proste podbijanie numerów w plikach
  - commity
  - tagi
- pbr
  - semantyczne wersjonowanie na bazie VCS
  - Bardziej deklaratywny setup.py

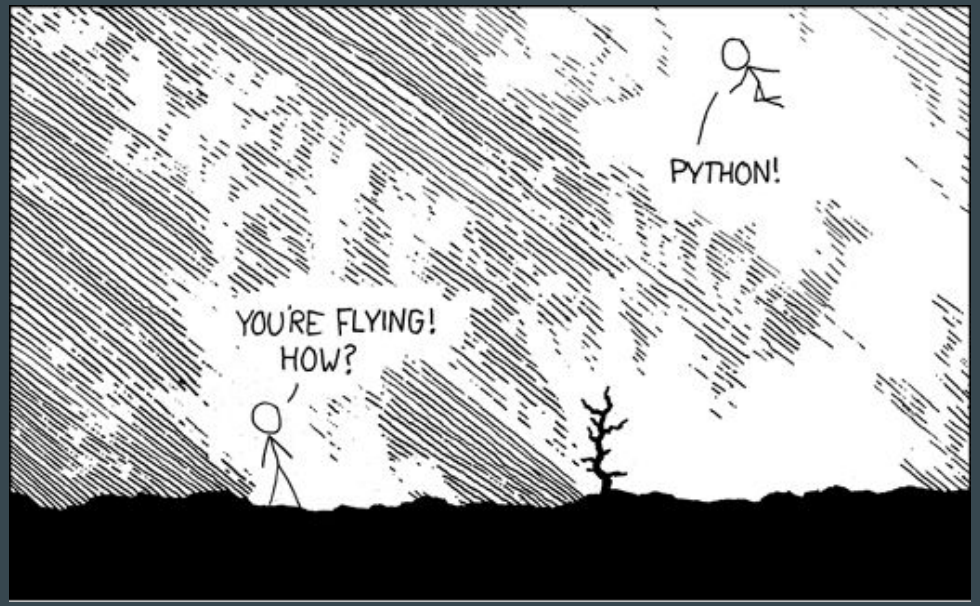
# Szablony projektów

- Pokazują strukturę folderów, konfigurację Travisa, konfigurację projektu (setup.py, requirements.txt), itd.
- cookiecutter - multum szablonów dla konkretnych zastosowań
- pyscaffold - jeden potężny konfigurowalny szablon
- Zawsze można zmodyfikować.

# Backend development - kilka bibliotek

- Falcon - szybki, czysty microframework REST/WSGI
- aiohttp - REST na asynchronicznym IO
- Gunicorn - dobry serwer WSGI
- API-hour - Gunicorn z szybkimi asychnonicznymi runnerami
- ...https://github.com/butla/mountepy?

# Fin?




https://xkcd.com/353/