



Politechnika
Wrocławska

Metody Systemowe i Decyzyjne L

Python - konfiguracja i podstawowe narzędzia

Piotr Kawa

W4N, K46

sem. letni 2023/24



Agenda

- 1 Python - podstawowe informacje
- 2 Python - konfiguracja lokalna
 - Edytor kodu (IDE)
 - Manager wirtualnych środowisk
 - Manager bibliotek
 - Jupyter Notebook
- 3 Google Colab
- 4 Podsumowanie

Python - podstawowe informacje

Python to język:

- Interpretowany,
- Wspiera wiele paradygmatów programowania,
- Nastawiony na zwięzłość i czytelność,
- Dynamicznie typowany.

Python - podstawowe informacje

Istnieje szereg implementacji Pythona. Oto najpopularniejsze z nich:

Table: Implementacje języka Python

Nazwa	Implementacja
CPython	C
IronPython	C#
Jython	Java
PyPy	RPython

Python - podstawowe informacje

Preferowany zestaw narzędzi potrzebnych do uczestnictwa w zajęciach. Dla trybu stacjonarnego:

- wybrane IDE,
- wybrany manager wirtualnych środowisk,
- wybrany manager pakietów,
- IPython + Jupyter.

oraz dla trybu online:

- Google Colab

Agenda

- 1 Python - podstawowe informacje
- 2 Python - konfiguracja lokalna
 - Edytor kodu (IDE)
 - Manager wirtualnych środowisk
 - Manager bibliotek
 - Jupyter Notebook
- 3 Google Colab
- 4 Podsumowanie

Python - konfiguracja lokalna : Edytor kodu (IDE)

Najpopularniejsze narzędzia do developmmentu w Pythonie:

- Pycharm,
- Visual Studio Code,
- Vim.

Python - konfiguracja lokalna : Manager wirtualnych środowisk

Język Python dość szybko odpowiedział na potrzebę enkapsulacji środowisk deweloperskich w celu zachowania ich reprodukowalności i oddzielenia konkretnych konfiguracji od instalacji systemowej.

Innymi słowy dobrą praktyką korzystania z języka Python jest tworzenie dedykowanych "instalacji" dla każdego z projektów. Pozwala to (1) zachować higienę w naszym systemie operacyjnym, (2) usprawnić przenoszalność kodu.

Python - konfiguracja lokalna : Manager wirtualnych środowisk

Najpopularniejszymi narzędziami są:

- **Anaconda**
- Pipenv
- Virtualenv
- Poetry

Python - konfiguracja lokalna : Manager wirtualnych środowisk

Ja polecam instalację Condy, a dokładniej Minicondę. Tutaj możecie znaleźć przydatne linki:

- [instalator](#),
- [tutorial dla Linuxów](#),
- [najczęściej zadawane pytania](#).

Python - konfiguracja lokalna : Manager wirtualnych środowisk

Utworzenie nowego środowiska - `conda create`, np.
utworzenie środowiska o nazwie 'msid' z Pythonem 3.9:

```
conda create --name msid python=3.9
```

Python - konfiguracja lokalna : Manager wirtualnych środowisk

Aktywacja środowiska - `conda activate`, np. aktywacja środowiska o nazwie 'msid':

```
conda activate msid
```

Deaktywacja środowiska (tj. powrót do 'base'):

```
conda deactivate
```

Python - konfiguracja lokalna : Manager wirtualnych środowisk

To, jakie środowisko mamy uruchomione możemy sprawdzić na kilka sposobów. Najprostszym z nich jest sprawdzenie co się wyświetla w terminalu. Np. taki komunikat oznacza, że aktywne jest środowisko "base", tj. domyślne:

```
(base) piotrkawa@Piotr's PC ~
```

Możemy też sprawdzić ścieżkę do pliku wykonywalnego interpretera Pythona, tj. dla Unixów komenda:

```
which python
```

Wywołanie jej powinno zwrócić ścieżkę podobną do:
"/usr/local/anaconda3/bin/python"

Python - konfiguracja lokalna : Manager bibliotek

Siłą Pythona są zewnętrzne pakiety (mimo rozbudowanej biblioteki standardowej). Znowu, jak w przypadku wirtualnych środowisk, mamy kilka sposobów na instalowanie paczek. Ja polecam najpopularniejszy z nich - `pip` (tj. Python Package Index).

Najważniejszą zasadą (której niekiedy nie da się ściśle przestrzegać) jest trzymanie się jednego managera, tj. jeśli w danym środowisku instalujemy paczki przez `conda` (tak, to też manager paczek), to nie używamy `pipa`.

Python - konfiguracja lokalna : Manager bibliotek

Instalacja paczki - `pip install`, np. dla pakietu "numpy":

```
pip install numpy
```

Python - konfiguracja lokalna : Manager bibliotek

Odinstalowywanie paczki - `pip uninstall`, np. dla pakietu "pandas":

```
pip uninstall pandas
```


Python - konfiguracja lokalna : Manager bibliotek

Wylistowanie wszystkich zainstalowanych paczek w środowisku - `pip freeze`. Możemy przekazać je od razu do pliku konfiguracyjnego naszego projektu. Np. poniższa komenda:

```
pip freeze > requirements.txt
```

spowoduje utworzenie pliku "requirements.txt" i wypełnienie go listingiem zainstalowanych bibliotek.

Python - konfiguracja lokalna : Manager bibliotek

Możemy także instalować paczki zbiorczo z podanego listingu.
Poniższa komenda

```
pip install -r requirements.txt
```

spowoduje instalacje paczek wypisanych w pliku
"requirements.txt".

Python - konfiguracja lokalna : Jupyter Notebook

Jupyter Notebook jest webowym interaktywnym środowiskiem programistycznym przeznaczonym do obliczeń z kategorii data-science (czyli naszej). Każda sesja wystawia port, pod którym możemy z niego korzystać. Najpopularniejszą metodą jest wpisanie adresu do przeglądarki, ale można też go używać przez IDE.

Plikami, w których pracujemy, są te z rozszerzeniem ".ipynb" (jak IPython Notebook). I są to de facto zwykłe dokumenty "html". Składają się one z komórek, które mogą zawierać w sobie kod lub też dokumentację. Komórki możemy uruchamiać wielokrotnie i w dowolnej kolejności.

W tej materii ważnym pojęciem jest tzw. "kernel", który wskazuje Jupyterowi jak ma wykonać dany kod. Kernele budujemy z wirtualnych środowisk (snippet na następnych slajdach).

Python - konfiguracja lokalna : Jupyter Notebook

Aby zainstalować Jupyter Notebook wystarczy wpisać następującą komendę w terminalu mając aktywowane odpowiednie wirtualne środowisko:

```
pip install jupyter
```

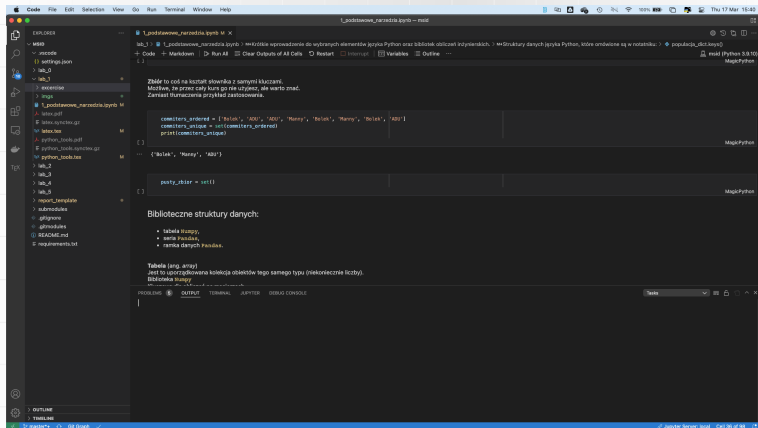
Python - konfiguracja lokalna : Jupyter Notebook

Aby uruchomić Jupytera wchodzimy w terminal i wpisujemy:

```
jupyter notebook
```

Python - konfiguracja lokalna : Jupyter Notebook

Innym, według mnie lepszym sposobem obsługi Notebooka jest obsługa go przez VS Code. Możemy bez problemu odpalać tam pliki .ipynb.



```

L_podstawowe_struktury.ipynb -- main
+ Code + Markdown + Run All + Clear Outputs of All Cells + Restart + Interrupt + Variables + Outline
Złóż to coś na kształt słownika z danymi kluczami.
Możliwe, że przez cały kurs go nie użyjemy, ale warto znać.
Zanimś tłumaczenia przykład zastosowania.

comitters_ordered = ['Belok', 'ADD', 'ADD', 'Manny', 'Belok', 'Manny', 'Belok', 'ADD']
comitters_unique = set(comitters_ordered)
print(comitters_unique)

MaggiePython

{'Belok', 'Manny', 'ADD'}

MaggiePython

posty_tzbar = set()

MaggiePython

Biblioteczne struktury danych:

• tabele NumPy,
• słowniki Pandas,
• tabele danych Pandas.

Tabela (arg: array)
Jest to uporządkowana kolekcja obiektów tego samego typu (niekoniecznie liczby).
Biblioteczne struktury

PROBLEMS OUTPUT TERMINAL JUPYTER DEBUG CONSOLE

```

Agenda

- 1 Python - podstawowe informacje
- 2 Python - konfiguracja lokalna
 - Edytor kodu (IDE)
 - Manager wirtualnych środowisk
 - Manager bibliotek
 - Jupyter Notebook
- 3 Google Colab
- 4 Podsumowanie

Google Colab : Jupyter Notebook

Dobłą alternatywą dla konfiguracji lokalnej jest skorzystanie z webowego narzędzia - Colab. Jest to platforma dedykowana zagadnieniom z data-science, utrzymywana przez Google. Jest także oparta o Jupytera, z tym, że obliczenia są wykonywane na zewnętrznych serwerach¹.

Platforma ta szczególnie przydaje się gdy nasz sprzęt nie posiada wystarczających zasobów, np. GPU. Dodatkowo jest silnie zintegrowana z całym ekosystemem Google, co niesie za sobą szereg zalet :)

¹udostępniana moc obliczeniowa jest tu podstawą modelu biznesowego, niemniej na nasze potrzeby powinna wystarczyć wersja darmowa

Google Colab : Jupyter Notebook

Platforma dostępna jest pod adresem:

<https://colab.research.google.com>

Agenda

- 1 Python - podstawowe informacje
- 2 Python - konfiguracja lokalna
 - Edytor kodu (IDE)
 - Manager wirtualnych środowisk
 - Manager bibliotek
 - Jupyter Notebook
- 3 Google Colab
- 4 Podsumowanie

Podsumowanie :

To, jakie narzędzie wybierze Państwo zależy od Was.

Dziękuję za uwagę!