

Wizualizacja danych - wykład 1

dr Piotr Jastrzębski

Sprawy organizacyjne

Sprawy organizacyjne

- Sylabus jest dostępny w systemie USOS.
- Regulamin zajęć dostępny jest na stronie prowadzącego zajęcia <http://wmii.uwm.edu.pl/~piojas/>.
- Forma zaliczenia: egzamin.
- Wykład - 15 godzin, zajęcia w ustalonych terminach.
- Terminarz, prezentacje i inne materiały związane z wykładem będą udostępniane w repozytorium na Githubie <https://github.com/pjastr/WizualizacjaDanychNStac> .

Ostatnia aktualizacja pliku: 2019-02-22 18:26:16.

Wymagania wstępne

- Znajomość podstawowych konstrukcji programistycznych (ze wstępu do programowania).
- Matematyka z zakresu szkoły średniej/z przedmiotu repozytorium matematyki elementarnej.

Ewentualne braki należy opanować w samodzielnym zakresie.

W razie problemów zapraszam na konsultacje.

Wstęp to języka Python

Język Python

- Poprawna wymowa: pajton.
- Język Python stworzył we wczesnych latach 90. Guido van Rossum – jako następcę języka ABC.
- Nazwa języka pochodzi od serialu komediowego emitowanego w latach siedemdziesiątych przez BBC – „Monty Python’s Flying Circus” (Latający cyrk Monty Pythona). Projektant, będąc fanem serialu i poszukując nazwy krótkiej, unikalnej i nieco tajemniczej, uznał tę za świetną.

Przełomowy rok - 2008

- Utworzenie drugiej gałęzi rozwoju 3.x. Początkowe obie gałęzie były rozwijane niezależnie, lecz na dziś zostało ogłoszone zakończenia wsparcia Pythona 2.x na rok 2020.
- Wykład będzie oparty o wersję 3.7.2 32-bitową (choć bardzo rzadko będzie korzystać z ostatnich nowości).

Podstawowe różnice między 2.x a 3.x

- funkcja print

Python 2:

```
print 'Hello, World!'
print('Hello, World!')
print "text", ; print 'print more text on the same line'
```

Python 3

```
print('Hello, World!')
print("some text,", end='')
print(' print more text on the same line')
```


Dzielenie zmiennych typu int

Python 2:

```
print '3 / 2 =', 3 / 2
print '3 // 2 =', 3 // 2
print '3 / 2.0 =', 3 / 2.0
print '3 // 2.0 =', 3 // 2.0
```

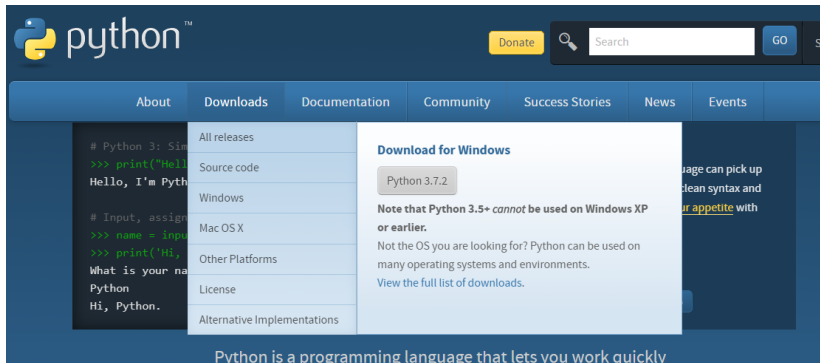
Python 3

```
print('3 / 2 =', 3 / 2)
print('3 // 2 =', 3 // 2)
print('3 / 2.0 =', 3 / 2.0)
print('3 // 2.0 =', 3 // 2.0)
```

Warto doczytać np. tutaj.

Instalacja - Windows

- <https://python.org/>



Rysunek 1: Strona www

Linux

Sprawdzenie wersji na Ubuntu 18.04:

```
piotrekwd@piotrekwd-VirtualBox:~$ python3
Python 3.6.5 (default, Apr 1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

Ręczna instalacja:

```
sudo apt install python3
```

Wybór IDE do Pythona

- IDLE (domyślny)
- PyCharm <https://www.jetbrains.com/pycharm/> (na ćw. i wykład)
- Spyder IDE <https://www.spyder-ide.org/>
- Visual Studio
<https://visualstudio.microsoft.com/pl/vs/features/python/>
- Visual Studio Code + odpowiednie rozszerzenia
<https://code.visualstudio.com/>
- Atom + ide-python <https://atom.io/packages/ide-python>
- i wiele innych...

PyCharm

- IDE używany na wykładzie i sugerowany do użycia w trakcie ćwiczeń, na egzaminie można użyć dowolny IDE zainstalowany w pracowni (choć zalecane jest użycie PyCharm),
- edytor z tzw. “inteligentnymi podpowiedziami”
- graficzny debugger
- inspekcja kodu, refaktoryzacja, wsparcie dla systemów kontroli wersji

Dla m.in. studentów dostępna jest za darmo wersja Professional.

Informacje o niej dostępne są tutaj:

<https://www.jetbrains.com/student/>.

Sugerowane jest użycie uczelnianego maila w domenie

@student.uwm.edu.pl (więcej informacji o niej jest tutaj:

<http://www.uwm.edu.pl/studenci/uslugi-informatyczne>) lub karty ISIC.

Styl PEP8

- wymowa: pi-i-pi-ejt
- standaryzacja kodu używana m.in. przy rozwijaniu nowych funkcjonalności
- używanie daje lepszą organizację i czytelność kod
- pełna wersja <https://www.python.org/dev/peps/pep-0008/>

Znaki odstępu:

- we wcięciach stosujemy spacje (a nie tabulatory)
- każdy poziom wcięcia powinien składać się z 4 spacji
- wiersz powinien składać się z maksymalnie 79 znaków

Dobrze:

```
foo = long_function_name(var_one, var_two,  
                          var_three, var_four)
```

```
def long_function_name(  
    var_one, var_two, var_three,  
    var_four):  
    print(var_one)
```

```
foo = long_function_name(  
    var_one, var_two,  
    var_three, var_four)
```


Źle:

```
foo = long_function_name(var_one, var_two,  
                          var_three, var_four)
```

```
def long_function_name(  
    var_one, var_two, var_three,  
    var_four):  
    print(var_one)
```

Instrukcje warunkowe:

```
if (this_is_one_thing and
    that_is_another_thing):
    do_something()
```

```
if (this_is_one_thing and
    that_is_another_thing):
    # dodatkowy komentarz
    do_something()
```

```
if (this_is_one_thing
    and that_is_another_thing):
    do_something()
```

Listy:

```
my_list = [  
    1, 2, 3,  
    4, 5, 6,  
]  
  
result = some_function_that_takes_arguments(  
    'a', 'b', 'c',  
    'd', 'e', 'f',  
)
```

Listy - druga wersja:

```
my_list = [  
    1, 2, 3,  
    4, 5, 6,  
]  
result = some_function_that_takes_arguments(  
    'a', 'b', 'c',  
    'd', 'e', 'f',  
)
```

Operatory arytmetyczne a przenoszenie:

Źle:

```
income = (gross_wages +  
          taxable_interest +  
          (dividends - qualified_dividends) -  
          ira_deduction -  
          student_loan_interest)
```

Dobrze:

```
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

Puste linie:

- dwie linie między funkcjami najwyższego poziomu i między klasami.
- pojedyncza linia między funkcjami w klasie

Kodowanie:

- dla Pythona 3 sugerowane i domyślne to UTF-8.

Importowanie bibliotek

Dobrze:

```
import os  
import sys
```

Źle:

```
import sys, os
```

Ale dobrze też:

```
from subprocess import Popen, PIPE
```

Kolejność:

- 1 Biblioteki systemowe.
- 2 Biblioteki zewnętrzne tzw. third-party imports.
- 3 Biblioteki lokalne.

Stringi:

- można używać pojedynczych apostrofów jak i podwójnych cudzysłówów
- ważne, aby stosować wybraną notację konsekwentnie
- jedyny wyjątek to gdy wewnątrz stringu chcemy użyć cudzysłów np.

```
print('Oglądam film "Player One"')
```

Spacje w wyrażeniach:

- należy unikać ich nadużywania

Dobrze:

```
spam(ham[1], {eggs: 2})
```

Źle:

```
spam( ham[ 1 ], { eggs: 2 } )
```

Dobrze:

```
foo = (0,)
```

Źle:

```
bar = (0, )
```

Dobrze:

```
spam(1)
```

Źle:

```
spam (1)
```

Dobrze:

```
dct['key'] = lst[index]
```

Źle:

```
dct ['key'] = lst [index]
```

```
x = 1  
y = 2  
long_variable = 3
```

Źle:

```
x           = 1  
y           = 2  
long_variable = 3
```

Cechy języka Python

- Python wspiera różne paradygmaty programowania: obiektowy, imperatywny oraz funkcyjny.
- Posiada w pełni dynamiczny system typów i automatyczne zarządzanie pamięcią (garbage collector).
- Często używany jako język skryptowy. Interpretery Pythona są dostępne na wiele systemów operacyjnych. Różne implementacje Pythona: CPython (język C), IronPython (platforma .NET), Jython (Java), PyPy (Python).
- Prosta i czytelna składnia ułatwiająca utrzymywanie, używanie i rozumienie kodu.

Zen

```
import this
```

The Zen of Python, by Tim Peters

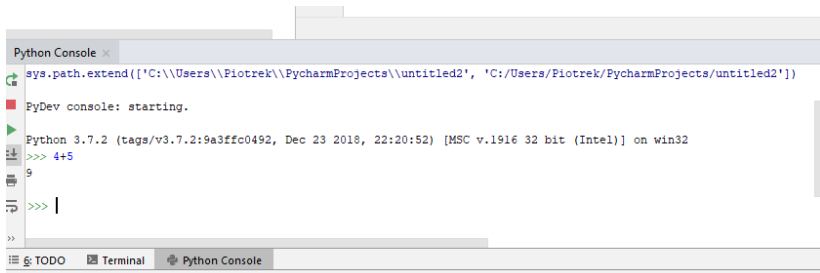
```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
...
```

PL - [https:](https://pl.python.org/forum/index.php?topic=392.msg1844#msg1844)

[//pl.python.org/forum/index.php?topic=392.msg1844#msg1844](https://pl.python.org/forum/index.php?topic=392.msg1844#msg1844)

Dwa tryby pracy

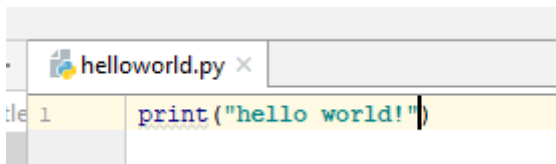
Python umożliwia dwa tryby pracy. Pierwszym z nich jest tzw. programowanie w trybie interaktywnym za pomocą konsoli.



```
Python Console x
sys.path.extend(['C:\\Users\\Piotrek\\PycharmProjects\\untitled2', 'C:/Users/Piotrek/PycharmProjects/untitled2'])
PyDev console: starting.
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit (Intel)] on win32
>>> 4+5
9
>>> |
```

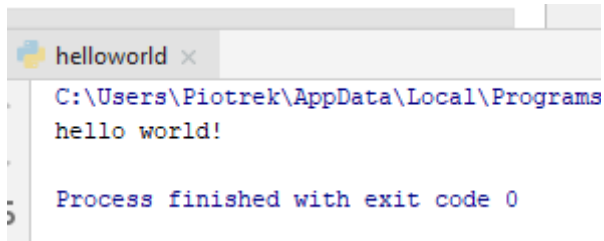
Rysunek 2: Konsola Pythona w PyCharm.

Drugim sposobem pracy jest tzw. tryb skryptowy.



```
helloworld.py x  
1 print("hello world!")
```

Rysunek 3: Skrypt.



```
helloworld x  
C:\Users\Piotrek\AppData\Local\Programs  
hello world!  
  
Process finished with exit code 0
```

Rysunek 4: Wyjście po wykonaniu skryptu.

Powstawowe typy danych w Pythonie

- liczby (int, float, complex): 22, 5.2, 2j + 9
- łańcuchy znaków (str): 'tekst1', "tekst2"
- lista (list): [3, 22, 'tekst', False]
- krotka (tuple): (6, 17, 'tekst', False)
- słownik (dict): {'klucz': 'wartość', 23: 33, 'status': False}
- typ logiczny (bool): True, False

Liczby

```
print(type(5))
```

```
## <class 'int'>
```

```
print(type(4.5))
```

```
## <class 'float'>
```

```
print(type(55+3j))
```

```
## <class 'complex'>
```

```
print(type(4e+4))
```

```
## <class 'float'>
```

```
print(type(40000))
```

```
## <class 'int'>
```

Łańcuchy znaków - stringi

```
str = 'Hello World!'
print(str)
```

```
## Hello World!
```

```
print(str[0])
```

```
## H
```

```
print(str[2:5])
```

```
## llo
```

```
print(str[2:])
```

```
## llo World!
```

```
print(str * 2)
```

```
## Hello World!Hello World!
```

```
print(str + 'WMII')
```

```
## Hello World!WMII
```

Podstawowa instrukcja wyjścia - funkcja print

Składnia wg dokumentacji:

```
print(*objects, sep=' ', end='\n', file=sys.stdout,  
      flush=False)
```

- `objects` - to co ma być wyświetlone
- `sep` - separator, domyślnie znak spacji
- `end` - co co ma być wyświetlone na końcu, domyślnie znak końca linii
- `file` - określa gdzie mają być `objects` wyświetlone, domyślnie `sys.stdout` (domyślny ekran)
- `flush` - określa czy "wyjście" ma być buforowane przed przekazaniem do `file`, domyślne `False`

```
print(1, 2, 3, 4)
```

```
## 1 2 3 4
```

```
print(1, 2, 3, 4, sep='*')
```

```
## 1*2*3*4
```

```
print(1, 2, 3, 4 ,sep='#', end='&')
```

```
## 1#2#3#4&
```

```
print('x', 'y', 'z', sep='', end='')  
print('a', 'b', 'c', sep='', end='')
```

```
## xyzabc
```

```
print('a', 'b', '\n', 'c')
```

```
## a b
```

```
## c
```


\t - przesunięcie do następnego "tab"=8 spacji

```
print('sdf', 3456, -2, sep='\t')
```

```
## sdf 3456 -2
```

\r - przesunięcie do lewej strony po każdym wyświetleniu

```
print(345, 'y', 'abc', sep='\r')
```

```
## abc
```

```
a = 16  
b = 2.25  
z = 45  
print('{:5d} {:6.3f} {:10d}'.format(a, b, z))
```

```
##      16  2.250           45
```

```
print('{2:5d} {1:6.3f} {0:10d}'.format(a, b, z))
```

```
##      45  2.250           16
```

Input - podstawowe wejście

```
name= input('Podaj imię \n')  
print('typ:', type(name))
```

Podaj imię

Jan

typ: <class 'str'>

```
number= int(input('Podaj liczbę \n'))  
print('typ:', type(number))
```

Podaj liczbę

32

typ: <class 'int'>

Operacje arytmetyczne

```
print(5+3)
```

```
## 8
```

```
print(4*5.2)
```

```
## 20.8
```

```
print(9-7)
```

```
## 2
```

```
print(25%7)
```

```
## 4
```

```
print(4/5)
```

```
## 0.8
```

```
print(4//5)
```

```
## 0
```

```
print(4/5.0)
```

```
## 0.8
```

```
print(4//5.0)
```

```
## 0.0
```

```
print(3**0)
```

```
## 1
```

```
print(0**0)
```

```
## 1
```

```
print(4/0)
```

```
ZeroDivisionError: division by zero
```

Operacje na stringach

```
print('raz'+ ' '+'dwa')
```

```
## raz dwa
```

```
print('tekst'*3)
```

```
## tekstteksttekst
```


Operatory przypisania

- = standardowy
- +=, -=, *=, /=, %=, **=, //=

```
a = 5  
a += 1  
print(a)
```

6

```
a **= 2  
print(a)
```

36

Operatory porównania

Znak	Znaczenie	Przykład
>	większe niż	$x > y$
<	mniejsze niż	$x < y$
==	równa się	$x == y$
!=	nie równa się	$x != y$
>=	większe lub równe	$x >= y$
<=	mniejsze lub równe	$x <= y$

Operatory logiczne i typ logiczny

Operator	Znaczenie	Przykład
and	i	x and y
or	lub	x or y
not	negacja	not x

```
a = True  
b = False  
print(a or b)
```

```
## True
```

```
print(type(a))
```

```
## <class 'bool'>
```

Kolejność operatorów

Od ostatniego:

- lambda
- if - else
- or
- and
- not x
- in, not in, is, is not, <, <=, >, >=, !=, ==
- |
- ^
- &
- <<, >>
- +, -
- *, @, /, //, %
- +x, -x, ~x

- `**`
- `await x`
- `x[index]`, `x[index:index]`, `x(arguments...)`,
`x.attribute`
- `(expressions...)`, `[expressions...]`, `{key:
value...}`, `{expressions...}`

Źródło: <https://docs.python.org/3/reference/expressions.html#operator-precedence>.

Listy

Listy w Pythonie mogą przechowywać elementy różnych typów.

```
list1 = ['raz', 'dwa', 5, 5];  
list2 = [1, 2, 3, 4, 5 ];  
list3 = ["a", "b", "c", "d"];  
print(list3)
```

```
## ['a', 'b', 'c', 'd']
```

```
list4 = ['s', 'ww', True, 5]  
print(list4[3])
```

```
## 5
```

```
list4[1] = True  
print(list4[1])
```

```
## True
```

```
print(list4[-1])
```

```
## 5
```

```
print(list4[2:])
```

```
## [True, 5]
```

```
print(len([2, 3, 4]))
```

```
## 3
```

```
print([1, 2, 3] + [4, 5, 6])
```

```
## [1, 2, 3, 4, 5, 6]
```

```
print(['Hi!'] * 4)
```

```
## ['Hi!', 'Hi!', 'Hi!', 'Hi!']
```

```
print(3 in [1, 2, 3])
```

```
## True
```


Pytanie do przemyślenia na kolejny wykład

Co oznacza w Pythonie, że wartości przekazywane są przez referencję?

```
a = 5  
b = a  
b += 2  
print(a)
```

```
## 5
```

```
print(b)
```

```
## 7
```

```
list1 = [1, 2, 3, 4]
list2 = list1
list1[2] = 'a'
print(list1)
```

```
## [1, 2, 'a', 4]
```

```
print(list2)
```

```
## [1, 2, 'a', 4]
```

Instrukcje warunkowe

```
a = 5
if a > 0:
    print('liczba dodatnia')
elif a == 0:
    print('zero')
else:
    print('liczba ujemna')
```

liczba dodatnia

Pętle

```
words = ['kot', 'pies', 'chomik']  
for w in words:  
    print(w, len(w))
```

```
## kot 3  
## pies 4  
## chomik 6
```

```
i = None  
for i in range(2):  
    print(i)
```

```
## 0  
## 1
```

```
i = 1
j = 1
while i < 4:
    j = 1
    while j < 4:
        print(i, '*', j, '=', i * j)
        j += 1
    i += 1
```

Bibliografia

- <https://pl.wikipedia.org/wiki/Python>, dostęp online 12.02.2019.
- <https://bulldogjob.pl/news/264-java-php-ruby-jak-wlasciwie-wymawiac-nazwy-technologiei>.
dostęp online 12.02.2019.
- https://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html, dostęp online 14.02.2019.
- K. Ropiak, Wprowadzenie do języka Python,
<http://wmii.uwm.edu.pl/~kropiak/wd/Wprowadzenie%20do%20j%C4%99zyka%20Python.pdf>, dostęp online 14.02.2019.
- B. Slatkin, Efektywny Python. 59 sposobów na lepszy kod,
Helion 2015.
- <https://www.python.org/dev/peps/pep-0008/>, dostęp online 14.02.2019.

Bibliografia - cd2

- <https://www.flynerd.pl/2017/05/python-4-typy-i-zmienne.html>, dostęp online 14.02.2019.
- <http://pytolearn.csd.auth.gr/p0-py/01/print.html>, dostęp online 15.02.2019.
- https://www.tutorialspoint.com/python3/python_lists.htm, dostęp online 17.02.2019.