

[Dismiss](#)

## Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)Branch: master ▾ [Python](#) / [4\\_Teoria](#) / [python\\_teoria\\_01.md](#)[Find file](#) [Copy path](#) Stach Poprawa struktury katalogow

4493d4e on Feb 15, 2019

0 contributors

933 lines (567 sloc) 18.4 KB

[Raw](#) [Blame](#) [History](#)   

# Python

## Teoria 1

### Program kursu

- wprowadzenie
  - podstawowe typy danych i operatory
  - wyrażenia warunkowe i pętle
  - funkcje i moduły
  - struktury danych
  - klasy i obiekty
  - operacje wejścia/wyjścia
  - obsługa wyjątków
- 
- obliczenia numeryczne
  - debugowanie i profilowanie
  - tworzenie graficznego interfejsu użytkownika
  - dekoratory, funkcje lambda, generatory ...

### Materiały do zajęć

- dokumentacja: <https://www.python.org/>
- 
- A. B. Downey, J. Elkner, C. Meyers, "Think Python. How to Think Like a Computer Scientist":  
<http://www.greenteapress.com/thinkpython/>
  - M. Pilgrim, "Dive into Python": <http://www.diveintopython.net/> -> [Zanurkuj\\_w\\_Pythonie.pdf](#)
  - Swaroop CH "A Byte of Python": <https://python.swaroopch.com/> (tłumaczenie: <http://python.edu.pl>)

/byteofpython/)

## Język kompilowany

---

- przykłady: C / C++, Pascal
- zalety: wydajność, błędy na etapie kompilacji, utrzymanie i rozwijanie kodu
- wady: czas pisania, kompilacja dużych projektów

kod źródłowy -> kompilator -> kod maszynowy -> wynik

## Język interpretowany

---

- przykłady: Python, Perl
- zalety: prosty, szybki i przyjemny
- wady: wydajność, debugowanie (brak kompilacji)

kod źródłowy -> interpreter -> wynik

## Python

---

| Python is powerful... and fast; plays well with others; runs everywhere; is friendly & easy to learn; is Open.

- otwarte oprogramowanie (*open source*)
- przenośny
  - Linux, Mac OS - prawdopodobnie już jest
  - Windows - instalator ze strony <https://www.python.org>
  - Anaconda - <https://www.continuum.io/>
- zwięzły i elegancki (prawie jak pseudokod)
- automatyczne zarządzanie pamięcią (*garbage collection*)
- bogata biblioteka standardowa i niezliczona ilość bibliotek zewnętrznych
- struktury wysokiego poziomu

## Hello World! - w różnych językach

---

```
# Python
print("Hello World!")
```

```
// C++
#include <iostream>

int main()
{
    std::cout << "Hello World!";
}
```

```
// Java
public class Hello {
    public static void main(String []args) {
        System.out.println("Hello World!");
    }
}
```

## Zastosowanie

---

- internet: Yahoo, Google, ...
- gry: Battlefield 2, Civilization 4, ...
- grafika: Walt Disney, Blender 3D, ...
- finanse: Altis Investment Management, Bellco Credit Union, ...
- nauka: NASA, Los Alamos National Laboratory, ...
- interfejs programistyczny aplikacji (*Application Programming Interface = API*): tensorflow (Google's machine learning), Amazon, Facebook, Youtube, ...

## Python 2 vs Python 3

---

- *Short version: Python 2.x is legacy, Python 3.x is the present and future of the language* (<https://wiki.python.org/moin/Python2orPython3>)
- Python 3 jest gotowy, jednak nie wszystkie zewnętrzne biblioteki są już z nim kompatybilne

## Python 2 vs Python 3

---

- Niektóre zmiany są kosmetyczne

```
# Python 2

print "Hello World!"

# Python 3

print("Hello World!")
```

## Python 2 vs Python 3

---

- Niektóre niebezpieczne

```
# Python 2

3 / 2.0 # = 1.5
3 / 2    # = 1

# Python 3

3 / 2.0 # = 1.5
3 / 2    # = 1.5
```

```
3 // 2 # = 1
```

## Python 2 vs Python 3

---

- A niektóre gruntowne: iteratory
- Niektóre elementy Pythona 3 można zainportować w Pythonie 2

```
Python 2.7.12 (default, Jul 1 2016, 15:12:24)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 3 / 2
1
>>> from __future__ import division
>>> 3 / 2
1.5
>>>
```

- Więcej informacji: <https://wiki.python.org/moin/Python2orPython3>
- Na kursie omawiany będzie Python 2

## Tryb interaktywny

---

```
$ python
Python 2.7.15
Type "help", "copyright", "credits" or "license" for more information.
>>> oceny_uwr = {2.0: "niedostateczny", 3.0: "dostateczny",
...             3.5: "dostateczny plus", 4.0: "dobry",
...             4.5: "dobry plus", 5.0: "bardzo dobry"}
>>> for ocena in sorted(oceny_uwr, reverse=True):
...     print(ocena, oceny_uwr[ocena])
...
5.0 bardzo dobry
4.5 dobry plus
4.0 dobry
3.5 dostateczny plus
3.0 dostateczny
2.0 niedostateczny
>>>
```

## Tryb skryptowy - linux

---

```
$ cat skala_ocen.py
#!/usr/bin/env python

oceny_uwr = {2.0: "niedostateczny", 3.0: "dostateczny",
            3.5: "dostateczny plus", 4.0: "dobry",
            4.5: "dobry plus", 5.0: "bardzo dobry"}

for ocena in sorted(oceny_uwr, reverse=True):
    print(ocena, oceny_uwr[ocena])

$ python skala_ocen.py
5.0 bardzo dobry
```

4.5 dobry plus  
4.0 dobry  
3.5 dostateczny plus  
3.0 dostateczny  
2.0 niedostateczny

## Tryb skryptowy - linux

---

```
$ ./skala_ocen.py
bash: ./skala_ocen.py: Permission denied

$ chmod +x skala_ocen.py

$ ./skala_ocen.py
5.0 bardzo dobry
4.5 dobry plus
4.0 dobry
3.5 dostateczny plus
3.0 dostateczny
2.0 niedostateczny
```

## Czytelność kodu

---

- przejrzysty kod
- zrozumiałe nazwy zmiennych
- komentarzy nigdy za wiele
- dzielenie linii nie boli
- konsekwencja w konwencji
- najlepiej trzymać się standardu PEP 8: <https://www.python.org/dev/peps/pep-0008/>
- ale bez przesady - przede wszystkim kod ma być czytelny

## Czytelność kodu - good way

---

```
import random

# skala ocen obowiązująca na Uniwersytecie Wrocławskim
oceny_uwr = {2.0: "niedostateczny", 3.0: "dostateczny",
            3.5: "dostateczny plus", 4.0: "dobry",
            4.5: "dobry plus", 5.0: "bardzo dobry"}

# lista studentów uczęszczających na zajęcia z Pythona
lista_studentow = ["Kasia", "Basia", "Józek", "Marek"]

# przez lenistwo prowadzącego oceny wystawiane losowo
for student in lista_studentow:
    # losowa ocena
    ocena = random.choice(list(oceny_uwr))
    # przypisanie oceny
    print(student, " -> ", oceny_uwr[ocena])
```

```
Kasia -> dobry plus
Basia -> dobry plus
```

Józek -> dostateczny plus  
Marek -> niedostateczny

## Czytelność kodu - wrong way

```
import random
o = {2.0: "niedostateczny", 3.0: "dostateczny", 3.5: "dostateczny plus", 4.0: "dobry", 4.5: "dobry plus", 5.0: "bardzo dobry"}
s = ["Kasia", \
      "Basia", "Józek",
      "Marek"]
for i in range(len(s)): x = random.randint(2, len(o) - 1); print(s[i], " -> ", o[x])
```

Kasia -> dostateczny  
Basia -> niedostateczny  
Józek -> dostateczny  
Marek -> bardzo dobry

## Podstawowe typy danych

- typ całkowity (*int*)
- typ zmiennoprzecinkowy (*float*)
- typ logiczny (*bool*)
- typ tekstowy (*str*)
- zmienne są typowane dynamicznie

```
x = 1    # [zmienna] [operator przypisania] [wartość]
y = 1.0
z = True

# drukuj typy zmiennych x, y i z (oddzielone przecinkiem)
print(type(x), type(y), type(z))

(<type 'int'>, <type 'float'>, <type 'bool'>)
```

## Operacje na liczbach

```
2 + 2 # suma dwóch liczb całkowitych -> liczba całkowita
```

4

```
2 + 2.0 # wynikiem tej sumy jest liczba zmiennoprzecinkowa
```

4.0

```
5 - 2
```

3

```
-5 * 2
```

-10

```
5 / 2 # w Pythonie 2 dzielenie zawsze zwraca liczbę całkowitą
```

2

```
5 // 2 # chyba że użyjemy // -> dla pythona 3 żeby zwrócił liczbę całkowitą, w innym przypadku zwróci zmieni
```

2

## Operacje na liczbach

---

```
5 % 2 # reszta z dzielenia (operator modulo)
```

1

```
int(2.5) # jawne rzutowanie na int
```

2

```
float(1) # jawne rzutowanie na float
```

1.0

## Operacje na liczbach

---

```
abs(-10) # moduł
```

10

```
abs(-2.5)
```

2.5

## Operacje na liczbach

---

```
pow(2, 3) # pow(a, b) = a^b
```

8

```
pow(2, 3.0) # zwróć uwagę na typy
```

8.0

```
2 ** 3 # inna wersja a^b
```

8

## Moduł math

---

```
import math # import [moduł]  
  
dir (math) # lista dostępnych funkcji  
  
['__doc__',  
 '__file__',  
 '__loader__',  
 '__name__',  
 '__package__',  
 '__spec__',  
 'acos',  
 'acosh',  
 'asin',  
 'asinh',  
 'atan',  
 'atan2',  
 'atanh',  
 'ceil',  
 'copysign',  
 'cos',  
 'cosh',  
 'degrees',  
 'e',  
 'erf',  
 'erfc',  
 'exp',  
 'expm1',  
 'fabs',  
 'factorial',  
 'floor',  
 'fmod',  
 'frexp',  
 'fsum',
```

```
'gamma',
'gcd',
'hypot',
'inf',
'isclose',
'isfinite',
'isinf',
'isnan',
'lDEXP',
'lgamma',
'log',
'log10',
'log1p',
'log2',
'modf',
'nan',
'pi',
'pow',
'radians',
'sin',
'sinh',
'sqrt',
'tan',
'tanh',
'trunc']
```

## Moduł math

---

```
help(math.floor) # dokumentacja funkcji floor z math
```

```
Help on built-in function floor in module math:
```

```
floor(...)
    floor(x)
```

```
    Return the floor of x as an Integral.
    This is the largest integer <= x.
```

```
help(math.ceil) # zwróć uwagę na: moduł.funkcja
```

```
Help on built-in function ceil in module math:
```

```
ceil(...)
    ceil(x)
```

```
    Return the ceiling of x as an Integral.
    This is the smallest integer >= x.
```

## Moduł math

---

```
int(2.8) # rzutowanie float na int "ucina wszystko po kropce"
```

```
math.floor(2.8) # tak jak funkcja math.floor
```

2

```
math.ceil(2.8) # funkcja math.ceil zaokragla w góre
```

3

```
math.ceil(2.1) # zawsze w góre (patrz definicja)
```

3

## Typ tekstowy bez *unicode*

```
Python 2.7.12 (default, Jul 1 2016, 15:12:24)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> s = "zażółć gęślą jaźń"
>>> s
'za\xc5\xbc\xc3\xb3\xc5\x82\xc4\x87 g\xc4\x99\xc5\x9b\xc4\x85 ja\xc5\xba\xc5\x84'
>>>
```

## Typ tekstowy

- Literał łańcuchowy można wprowadzić przez:

- pojedynczy cudzysłów

```
s = 'zażółć gęślą jaźń'
```

- podwójny cudzysłów

```
s = "zażółć gęślą jaźń"
```

- potrójny cudzysłów

```
s = '''zażółć gęślą jaźń'''
```

```
s = """zażółć gęślą jaźń"""
```

## Znak ucieczki

- zmienia interpretację znaku
- w Pythonie (i prawie zawsze) jest to ""

```
print("Ala powiedziała: \"Mam kota.\" ...") # cudzysłów kończy literał
```

```
File "<ipython-input-44-2bf66ab2182a>", line 1
    print("Ala powiedziała: "Mam kota." ...") # cudzysłów kończy literał
                                         ^
SyntaxError: invalid syntax

print("Ala powiedziała: \"Mam kota.\" ...") # chyba że "uciekniemy"

Ala powiedziała: "Mam kota." ...

print('Ala powiedziała: "Mam kota." ...') # kombinacja ' i "

Ala powiedziała: "Mam kota." ...

print("Ala powiedziała: 'Mam kota.' ...") # działa w obie strony

Ala powiedziała: 'Mam kota.' ...
```

## Wybrane sekwencje specjalne

---

- \n - nowa linia
- \t - tabulacja

```
print("\n wstawia nową linię.\nNowa linia.") # zwróć uwagę na \\n\n wstawia nową linię.  
Nowa linia.
```

```
print("\tTabulacja też się przydaje.") # brak spacji po \t  
  
Tabulacja też się przydaje.
```

## Długie łańcuchy

---

```
tekst = "Lorem Ipsum jest tekstem stosowanym jako przykładowy wypełniacz w przemyśle poligraficznym. Został
print(tekst)
```

Lorem Ipsum jest tekstem stosowanym jako przykładowy wypełniacz w przemyśle poligraficznym. Został po raz pierwszy użyty w XV w. przez nieznanego drukarza do wypełnienia tekstem próbkowej książki. Pięć wieków później zaczął być używany przemyśle elektronicznym, pozostając praktycznie niezmienionym. Spopularyzował się w latach 60. XX w. wraz z publikacją arkuszy Letrasetu, zawierających fragmenty Lorem Ipsum, a ostatnio z zawierającym różne wersje Lorem Ipsum oprogramowaniem przeznaczonym do realizacji druków na komputerach osobistych, jak Aldus PageMaker

## Długie łańcuchy

---

```
tekst = "Lorem Ipsum jest tekstem stosowanym jako przykładowy \
wypełniacz w przemyśle poligraficznym. Został po raz \
pierwszy użyty w XV w. przez nieznanego drukarza do \
wypełnienia tekstem próbnej książki. Pięć wieków później \
zaczął być używany przemyśle elektronicznym, \
pozostając praktycznie niezmienionym. Spopularyzował się \
w latach 60. XX w. wraz z publikacją arkuszy Letrasetu, \
zawierających fragmenty Lorem Ipsum, a ostatnio z zawierającym \
różne wersje Lorem Ipsum oprogramowaniem przeznaczonym do \
realizacji druków na komputerach osobistych, jak Aldus PageMaker"

print(tekst)
```

  Lorem Ipsum jest tekstem stosowanym jako przykładowy                         wypełniacz w przemyśle poligraficznym.  
  Został po raz                     pierwszy użyty w XV w. przez nieznanego drukarza do                     wypełnienia tekstem  
  próbnej książki. Pięć wieków później                     zaczął być używany przemyśle elektronicznym,  
  pozostając praktycznie niezmienionym. Spopularyzował się                     w latach 60. XX w. wraz z publikacją  
  arkuszy Letrasetu,                     zawierających fragmenty Lorem Ipsum, a ostatnio z zawierającym                     różne  
  wersje Lorem Ipsum oprogramowaniem przeznaczonym do                     realizacji druków na komputerach osobistych,  
  jak Aldus PageMaker

## Długie łańcuchy

---

```
tekst = "Lorem Ipsum jest tekstem stosowanym jako przykładowy " \
"wypełniacz w przemyśle poligraficznym. Został po raz " \
"pierwszy użyty w XV w. przez nieznanego drukarza do " \
"wypełnienia tekstem próbnej książki. Pięć wieków później " \
"zaczął być używany przemyśle elektronicznym, " \
"pozostając praktycznie niezmienionym. Spopularyzował się " \
"w latach 60. XX w. wraz z publikacją arkuszy Letrasetu, " \
"zawierających fragmenty Lorem Ipsum, a ostatnio z zawierającym " \
"różne wersje Lorem Ipsum oprogramowaniem przeznaczonym do " \
"realizacji druków na komputerach osobistych, jak Aldus PageMaker "
```

```
print(tekst) # "słowo" "słowo" = "słowo" + "słowo"
```

  Lorem Ipsum jest tekstem stosowanym jako przykładowy wypełniacz w przemyśle poligraficznym. Został po raz  
  pierwszy użyty w XV w. przez nieznanego drukarza do wypełnienia tekstem próbnej książki. Pięć wieków  
  później zaczął być używany przemyśle elektronicznym, pozostając praktycznie niezmienionym. Spopularyzował  
  się w latach 60. XX w. wraz z publikacją arkuszy Letrasetu, zawierających fragmenty Lorem Ipsum, a  
  ostatnio z zawierającym różne wersje Lorem Ipsum oprogramowaniem przeznaczonym do realizacji druków na  
  komputerach osobistych, jak Aldus PageMaker

## Długie łańcuchy

---

```
tekst = "Lorem Ipsum jest tekstem stosowanym jako przykładowy\n" \
"wypełniacz w przemyśle poligraficznym. Został po raz\n" \
"pierwszy użyty w XV w. przez nieznanego drukarza do\n" \
"wypełnienia tekstem próbnej książki. Pięć wieków później\n" \
"zaczął być używany przemyśle elektronicznym,\n" \
"pozostając praktycznie niezmienionym. Spopularyzował się\n" \
```

```
"w latach 60. XX w. wraz z publikacją arkuszy Letrasetu,\n" \
"zawierających fragmenty Lorem Ipsum, a ostatnio z zawierającym\n" \
"różne wersje Lorem Ipsum oprogramowaniem przeznaczonym do\n" \
"realizacji druków na komputerach osobistych, jak Aldus PageMaker\n"

print(tekst)
```

Lorem Ipsum jest tekstem stosowanym jako przykładowy wypełniacz w przemyśle poligraficznym. Został po raz pierwszy użyty w XV w. przez nieznanego drukarza do wypełnienia tekstem próbnej książki. Pięć wieków później zaczął być używany przemyśle elektronicznym, pozostając praktycznie nieniemionym. Spopularyzował się w latach 60. XX w. wraz z publikacją arkuszy Letrasetu, zawierających fragmenty Lorem Ipsum, a ostatnio z zawierającym różne wersje Lorem Ipsum oprogramowaniem przeznaczonym do realizacji druków na komputerach osobistych, jak Aldus PageMaker

## Długie łańcuchy

---

```
tekst = """Lorem Ipsum jest tekstem stosowanym jako przykładowy
wypełniacz w przemyśle poligraficznym. Został po raz
pierwszy użyty w XV w. przez nieznanego drukarza do
wypełnienia tekstem próbnej książki. Pięć wieków później
zaczął być używany przemyśle elektronicznym,
pozostając praktycznie nieniemionym. Spopularyzował się
w latach 60. XX w. wraz z publikacją arkuszy Letrasetu,
zawierających fragmenty Lorem Ipsum, a ostatnio z zawierającym
różne wersje Lorem Ipsum oprogramowaniem przeznaczonym do
realizacji druków na komputerach osobistych, jak Aldus PageMaker
"""

print(tekst)
```

Lorem Ipsum jest tekstem stosowanym jako przykładowy wypełniacz w przemyśle poligraficznym. Został po raz pierwszy użyty w XV w. przez nieznanego drukarza do wypełnienia tekstem próbnej książki. Pięć wieków później zaczął być używany przemyśle elektronicznym, pozostając praktycznie nieniemionym. Spopularyzował się w latach 60. XX w. wraz z publikacją arkuszy Letrasetu, zawierających fragmenty Lorem Ipsum, a ostatnio z zawierającym różne wersje Lorem Ipsum oprogramowaniem przeznaczonym do realizacji druków na komputerach osobistych, jak Aldus PageMaker

## Przykładowy program

---

```
import turtle # biblioteka do rysowania

pen = turtle.Turtle()

n_steps = 90 # w ilu krokach robimy 360 stopni

for i in range(n_steps):

    pen.forward(100) # rysuj 100 do przodu
    pen.right(45)    # obróć się w prawo o 45 stopni
    pen.forward(50)  # rysuj 50 do przodu
```

```
pen.left(90)      # obróć się w lewo o 90 stopni
pen.forward(100)  # rysuj do przodu o 100
pen.right(45)     # obróź się w prawo o 45 stopni

pen.penup()       # podnieś długopis (nie rysuj przy przemieszczaniu)
pen.setposition(0, 0) # wróć na środek
pen.pendown()     # rysujemy dalej

pen.right(360/n_steps) # obróć się w prawo o n_stepną część 360
```

[Dismiss](#)

## Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)Branch: master ▾ [Python](#) / [4\\_Teoria](#) / [python\\_teoria\\_02.md](#)[Find file](#) [Copy path](#) Stach Poprawa struktury katalogow

4493d4e on Feb 15, 2019

0 contributors

1685 lines (775 sloc) 18.2 KB

[Raw](#) [Blame](#) [History](#)   

# Python

## Teoria 2

- sekwencyjne typy danych
- wyrażenia logiczne

## Łańuch znaków

- zmienna *str* przechowuje tak naprawdę ciąg znaków
- w którym każdy znak ma określoną pozycję

```
s = "Python"  
s[0] # str[i] -> i-ty znak w ciągu
```

'p'

```
s[1] # pierwszy to drugi?
```

'y'

```
s[2]
```

't'

## Indeksowanie

- indeksowanie zaczyna się od 0 a kończy na  $n-1$ , gdzie  $n$  - długość ciągu

```
s = "Python"

n = len(s) # długość łańcucha s

print(n)
```

6

```
s[n-1] # ostatni element
```

```
'n'
```

```
s[n] # poza zakresem
```

```
-----
IndexError                                Traceback (most recent call last)

<ipython-input-7-1dea7ae0782c> in <module>()
----> 1 s[n] # poza zakresem
```

```
IndexError: string index out of range
```

## Wycinki

---

```
s = "Python"

s[2:4] # elementy od 2 do 3 (czyli "od 3 do 4")
```

```
'th'
```

```
s[2:] # od indeksu 2 do końca
```

```
'thon'
```

```
s[:2] # od początku do 1
```

```
'Py'
```

```
s[2:1000] # nie ma błędu o wyjściu poza zakres
```

'thon'

## Wycinki z krokiem

---

```
s = "0123456789"
```

```
s[2:8:2] # sekwencja[początek:koniec:krok]
```

'246'

```
s[2::3] # od 2 do końca co 3
```

'258'

```
s[::-2] # od początku do końca co 2
```

'02468'

## Indeksy ujemne

---

- indeks  $-k$  oznacza  $n-k$ , gdzie  $n$  - długość łańcucha

```
s = "Python"
```

```
s[-1] # 6 - 1 = 5 -> ostatni znak
```

'n'

```
s[-2] # 6 - 2 = 4 -> przedostatni
```

'o'

```
s[-2:] # od 4 do końca -> dwa ostatnie
```

'on'

```
s[:-2] # wszystko oprócz dwóch ostatnich
```

'Pyth'

## Dygresja o klasach

---

- więcej o klasach będzie na dalszych zajęciach
- klasa definiuje obiekt
- przypisując wartość zmiennej tworzymy obiekt danej klasy

```
n = 10 # tworzymy obiekt klasy int
```

## Dygresja o klasach

---

- klasa może posiadać zmienne i metody (czyli funkcje działające na obiekt)

```
x = 1.0 # tworzymy obiekt klasy float
```

```
# metody/zmienne klasy wywołujemy .  
x.is_integer() # [obiekt].metoda
```

True

```
s = "Python" # tworzymy obiekt klasy str  
s.replace('yt', 'yyyyt') # [obiekt].metoda
```

'Pyyyyython'

## Lista (*list*)

---

- sekwencja zmiennych dowolnego typu
- dynamiczny rozmiar
- inicjowana jest:
  - nawiasy kwadratowe
  - jawnie *list([iterable])* (czyli konstruktor klasy *list*)
  - lista składana (*list comprehension*)

## Lista przez [ ]

---

```
lista = [1, 2, 'a', "Python"]
```

```
len(lista) # długość listy
```

```
lista[0] # pierwszy element listy
```

```
1
```

```
lista[-1] # ostatni element listy
```

```
'Python'
```

## Lista przez konstruktor

---

```
lista = list([1,2,3]) # bez sensu, ale można
```

```
print(lista)
```

```
[1, 2, 3]
```

```
lista = list("Python") # można stworzyć listę ze string
```

```
print(lista)
```

```
['P', 'y', 't', 'h', 'o', 'n']
```

```
lista = list(1) # ale już nie z int
```

```
-----  
TypeError Traceback (most recent call last)
```

```
<ipython-input-28-5b92e1b521a9> in <module>()  
----> 1 lista = list(1) # ale już nie z int
```

```
TypeError: 'int' object is not iterable
```

## Lista składana

---

```
cyfry = ["jeden", "dwa", "trzy"] # definiujemy listę
```

```
# [wartość] dla [obiektu] z [listy]  
# pętle zostaną szerzej omówione na kolejnych listach  
dlugosc = [len(cyfra) for cyfra in cyfry]
```

```
print(dlugosc)
```

```
[5, 3, 4]
```

```
cyfry = ["jeden", "dwa", "trzy"] # definiujemy listę  
  
# [wartość] dla [obiektu] z [listy] jeśli [warunek]  
dlugosc = [len(cyfra) for cyfra in cyfry if cyfra != 'dwa']  
  
print(dlugosc)
```

```
[5, 4]
```

## Lista jako rezultat funkcji

---

```
help(str.split)
```

Help on method\_descriptor:

```
split(...)  
S.split(sep=None, maxsplit=-1) -> list of strings  
  
Return a list of the words in S, using sep as the  
delimiter string. If maxsplit is given, at most maxsplit  
splits are done. If sep is not specified or is None, any  
whitespace string is a separator and empty strings are  
removed from the result.
```

```
("Python is awesome, fast, and friendly!").split()
```

```
['Python', 'is', 'awesome,', 'fast,', 'and', 'friendly!']
```

```
("Python is awesome, fast, and friendly!").split(',')
```

```
['Python is awesome', ' fast', ' and friendly!']
```

## Krotka (*tuple*)

---

- sekwencja zmiennych dowolnego typu
- stały rozmiar
- inicjowana jest:
  - nawiasy okrągłe
  - ciąg elementów oddzielonych przecinkiem
  - jawnie *tuple([iterable])*

## Krotka

---

```
krotka = (1, 2, 3, "Python") # jak lista, ale () zamiast []
```

```
print(krotka)

(1, 2, 3, 'Python')

krotka = 1, 2, 3, "Python" # brak nawiasów = tuple

print(krotka)

(1, 2, 3, 'Python')

lista = list("Python") # lista ze stringa

krotka = tuple(lista) # krotka z listy

print(lista)
print(krotka)

['P', 'y', 't', 'h', 'o', 'n']
('P', 'y', 't', 'h', 'o', 'n')
```

## Lista vs Krotka

---

- krotka ma stały rozmiar a lista jest dynamiczna
- krotka jest "niezmienna" (*immutable*) w przeciwieństwie do listy (*mutable*)
- więcej o *mutable* vs *immutable* w dalszej części teorii
- jeśli sekwencja obiektów jest stała w czasie działania programu, lepiej używać krotek
  - szybsze
  - bezpieczniejsze
  - mogą być kluczami w słowniku (*o słownikach na kolejnych zajęciach*)

## Lista vs Krotka

---

```
lista = [1, 2, 3]

print(lista)

lista[0] = 2

print(lista)

[1, 2, 3]
[2, 2, 3]

krotka = (1, 2, 3)

krotka[0] = 2
```

```
-----  
TypeError                                Traceback (most recent call last)  
  
<ipython-input-38-9a057d2d95b0> in <module>()  
      1 krotka = (1, 2, 3)  
      2  
----> 3 krotka[0] = 2  
  
TypeError: 'tuple' object does not support item assignment
```

## Range

- uwaga: w Pythonie 2 *range()* jest funkcją wbudowaną; w Pythonie 3 jest to typ danych (klasa)
- reprezentuje (niezmienniczą) sekwencję liczb
- zajmuje mniej pamięci niż *list* lub *tuple* (przechowuje tylko informację o początku, końca i kroku)

## Range

```
x = range(10) # od 0 do 10  
  
print(x) # w Pythonie 2 zobaczylibyśmy [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

## Range

```
cyfry = range(0, 10)      # range(początek = 0, koniec)  
parzyste = range(2, 10, 2) # range(początek, koniec, krok)  
nieparzyste = range(1, 10, 2)  
  
print(cyfry)  
print(parzyste)  
print(nieparzyste)  
  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
[2, 4, 6, 8]  
[1, 3, 5, 7, 9]
```

## Sekwencyjne typy danych

- *list* - dynamiczny ciąg zmiennych dowolnego typu
- *tuple* - niezmienniczy ciąg zmiennych dowolnego typu
- *range* - niezmienniczy ciąg liczba całkowitych
- *str* - niezmienniczy ciąg znaków
- (dla kompletności) są jeszcze binarne sekwencyjne typy danych: *bytes*, *bytearray*, *memoryview*

## Operacje na sekwencjach

- *in* - prawda jeśli element należy do sekwencji, inaczej fałsz
- *not in* - fałsz jeśli element należy do sekwencji, inaczej prawda

```
s = "Python"
```

```
"y" in s
```

True

```
"p" not in s
```

True

- więcej o prawdzie i fałszu w dalszej części teorii

## Operacje na sekwencjach

- splot (*concatenation*)
- nie dla *range*

```
begin = "Python "
middle = "is "
end = "awesome!"
```

```
begin + middle + end
```

```
'Python is awesome!'
```

```
lista = [1, 2, 3, 4]
dodatek = [5, 6, 7, 8]
```

```
dodatek + lista # kolejność ma znaczenie
```

```
[5, 6, 7, 8, 1, 2, 3, 4]
```

## Operacje na sekwencjach

- mnożenie przez liczbę całkowitą
- nie dla *range*

```
znak = '-'
```

```
znak * 10
```

```
'-----'
```

```
krotka = (1, 2, 3)
```

```
2 * krotka
```

```
(1, 2, 3, 1, 2, 3)
```

## Operacje na sekwencjach

---

- wycinki

```
lista = [1, 2, 3, "Python"]
```

```
lista[:2] # dwa pierwsze
```

```
[1, 2]
```

```
lista[3][-2:] # dwa ostatnie ostatniego
```

```
'on'
```

```
range(3, 15, 3)[:2] # dla range też działa
```

```
range(3, 9, 3)
```

## Funkcje wbudowane dla sekwencji

---

- *len* - długość sekwencji
- *min, max* - najmniejszy, największy element

```
len([1,2,-3]) # liczba elementów
```

```
3
```

```
min([1,2,-3]) # minimum
```

```
-3
```

```
max("Python") # maximum
```

'y'

## Wspólne funkcje klasowe sekwencji

- *index* - indeks pierwszego znalezionego elementu
- *count* - ilość wystąpień elementu

```
x = [1, 2, 3, 4, 5] * 2
```

```
print(x)
```

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

```
x.count(4) # liczba 4 pojawia się 2 razy
```

```
2
```

```
x.index(4) # pierwszy raz dla i=3
```

```
3
```

## Operacje na sekwencjach "zmiennych" (*mutable*)

- na razie znamy: *list*
- ale na kolejnych listach poznamy kolejne
- sekwencje "niezmienne" (*immutable*): *tuple*, *range* i *str*
- ale na kolejnych listach poznamy kolejne
- więcej o *mutable* vs *immutable* w dalszej części teorii

## Operacje na sekwencjach "zmiennych" (*mutable*)

```
lista = [1, 2, 3]
```

```
lista[1] = 4 # zmień wartość drugiego elementu
```

```
print(lista)
```

```
[1, 4, 3]
```

```
lista = [1, 2, 3]
```

```
lista[1:2] = [5, 5] # zmień wycinek
```

```
print(lista)
```

```
[1, 5, 5, 3]
```

## Operacje na sekwencjach "zmiennych" (*mutable*)

---

```
lista = [1, 2, 3]  
  
del lista[1:2] # usuń wycinek  
  
print(lista)
```

```
[1, 3]
```

## Funkcje klasowe sekwencji "zmiennych" (*mutable*)

---

```
lista = [1, 2, 3, 4, 5]  
  
lista.append(6) # append dodaje element na koniec listy  
  
print(lista)  
  
lista.append([7, 8, 9]) # lista dodana jako element  
  
print(lista)
```

```
[1, 2, 3, 4, 5, 6]  
[1, 2, 3, 4, 5, 6, [7, 8, 9]]
```

```
lista = [1, 2, 3, 4, 5]  
  
lista.extend([6, 7, 8, 9]) # extend rozwija listę  
  
print(lista)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

## Funkcje klasowe sekwencji "zmiennych" (*mutable*)

---

```
lista = [1, 2, 3]  
  
lista.clear() # usuwa wszystkie elementy  
  
print(lista)  
  
[]
```

```
lista = [1, 2, 3]

kopia = lista.copy() # kopiuje całą listę

print(kopia)
```

```
[1, 2, 3]
```

## Funkcje klasowe sekwencji "zmiennych" (*mutable*)

---

```
lista = [1, 2, 3]

lista.insert(1, 4) # wstaw 4 pod indeks 2

print(lista)
```

```
[1, 4, 2, 3]
```

```
lista = [1, 2, 3]

lista.pop(1) # zwróć i usuń *i*-ty element
```

```
2
```

```
print(lista)
```

```
[1, 3]
```

## Funkcje klasowe sekwencji "zmiennych" (*mutable*)

---

```
lista = ['a', 'b', 3] * 3

print(lista)

lista.remove('b') # usuń element (pierwsze wystąpienie)

print(lista)
```

```
['a', 'b', 3, 'a', 'b', 3, 'a', 'b', 3]
['a', 3, 'a', 'b', 3, 'a', 'b', 3]
```

```
lista = [1, 2, 3]

lista.reverse() # odwróć kolejność

print(lista)
```

```
[3, 2, 1]
```

## Funkcja `sort` - tylko dla list

```
help(list.sort)
```

```
Help on method_descriptor:
```

```
sort(...)  
L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
```

```
lista = ['e', 'D', 'a', 'b', 'C']
```

```
lista.sort() # sortuj
```

```
print(lista)
```

```
lista.sort(key=str.lower) # sortuj wg funkcji key
```

```
print(lista)
```

```
['C', 'D', 'a', 'b', 'e']  
['a', 'b', 'C', 'D', 'e']
```

## Limit pamięci

- Architektura 32-bitowa dysponuje  $2^{32}$  adresami

```
B = 1          # bajt  
KB = 1024 * B # kilobajt  
MB = 1024 * KB # megabajt
```

```
2**32 / MB
```

```
4096.0
```

- stąd limit pamięci operacyjnej w systemach 32-bitowych to ok. 4GB
- co często daje limit na pamięć RAM rzędu 3GB (bo karta graficzna)

## Int w Pythonie

```
# zwrócić uwagę na sposób importowania:  
# from [moduł] import [funkcja] as [nazwa]  
from sys import getsizeof as rozmiar  
  
help(rozmiar)
```

```
Help on built-in function getsizeof in module sys:
```

```
getsizeof(...)  
    getsizeof(object, default) -> int  
  
    Return the size of object in bytes.
```

```
rozmiar(0)
```

24

```
rozmiar(2**100)
```

40

## Kolory RGB

- w 24-bitowym RGB:
  - czerwony: 0 - 255
  - zielony: 0 - 255
  - niebieski: 0 - 255
- czemu do 255? 1 bajt = 8 bitów

$\$\$11111111_2 = 255\$\$$

- w 32-bitowym dochodzi kanał alpha (przezroczystość)

## Kolory RGB

- często spotka się zapis: #RRGGBB
- gdzie RR - kolor czerwony w zapisie szesnastkowym itd.
- np. #ff96cb
  - czerwony: \$\\text{ff}\\\_{16} = 255\$
  - zielony: \$96\\\_{16} = 150\$
  - niebieski: \$\\text{cb}\\\_{16} = 203\$

## Systemy liczbowe w Pythonie

```
bin(438) # liczba 438 w systemie binarnym
```

```
'0b110110110'
```

```
0b110110110 # 0b sygnalizuje system binarny
```

438

```
0b000000000010110110 # wiodące zera nic nie zmieniają
```

438

```
hex(438) # liczba 438 w systemie heksadecymalnym
```

'0x1b6'

```
0x1b6 # 0x sygnalizuje system szesnastkowy
```

438

## Typ logiczny (boolowski, *boolean*)

- przyjmuje wartości: prawda lub fałsz
- w Pythonie następujące wartości są utożsamiane z fałszem:
  - False
  - None
  - zero dowolnego typu (0, 0.0, 0j)
  - pusta sekwencja "", (), [] lub mapowanie {}
- pozostałe wartości uznawane są za prawdę

## Operacje logiczne

Operacja	Wynik
x or y	jeśli x jest fałszem, to y, inaczej x
x and y	jeśli x jest fałszem, to x, inaczej y
not x	jeśli x jest fałszem, to True, inaczej False

## Operacje logiczne - or

```
print("0 or 0 -> " + str(0 or 0))
print("1 or 0 -> " + str(1 or 0))
print("0 or 1 -> " + str(0 or 1))
print("1 or 1 -> " + str(1 or 1))
```

```
0 or 0 -> 0
1 or 0 -> 1
0 or 1 -> 1
1 or 1 -> 1
```

## Operacje logiczne - and

```
print("0 and 0 -> " + str(0 and 0))
print("1 and 0 -> " + str(1 and 0))
print("0 and 1 -> " + str(0 and 1))
print("1 and 1 -> " + str(1 and 1))
```

## Operacje logiczne - not

```
not True
```

```
False
```

```
not False
```

```
True
```

## Dygresja - słowa kluczowe / zastrzeżone

```
x = 1      # zmiennej x przypisz wartość 1
y = 2      # zmiennej y przypisz wartość 2
False = 3 # zmiennej False przypusz wartość 3
```

```
File "<ipython-input-88-6d6e4a6b0a3f>", line 3
    False = 3 # zmiennej False przypusz wartość 3
               ^
SyntaxError: can't assign to keyword
```

```
import keyword
print(keyword.kwlist) # lista słów kluczowych w Pythonie
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif',
'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

## Dygresja - niedozwolone nazwy zmiennych

- słowa kluczowe
- zaczynające się od cyfry
- zawierające polskie znaki
- zawierające cokolwiek innego niż litery, cyfry i \_

## Porównania

Operacja	Znaczenie
<code>==</code>	równe
<code>!=</code>	różne
<code>&lt;</code>	mniejsze
<code>&gt;</code>	większe
<code>&lt;=</code>	mniejsze lub równe
<code>&gt;=</code>	większe lub równe
<code>is</code>	ten sam identyfikator
<code>is not</code>	inny identyfikator

## Porównania

```
1 > 0 # większy niż
```

```
True
```

```
"Prowadzący" > "Student" # znak po znaku
```

```
False
```

```
"prowadzący" > "Student" # wielkość ma znaczenie
```

```
True
```

```
"prowadzący" > "2 studentów" # litera > cyfra
```

```
True
```

## Identyfikator zmiennej

```
help(id)
```

```
Help on built-in function id in module builtins:
```

```
id(obj, /)
    Return the identity of an object.
```

This is guaranteed to be unique among simultaneously existing objects.  
(CPython uses the object's memory address.)

```
x = 2 # zmiennej x przypisz wartość 2  
id(x) # wyświetl jej id
```

140685772794432

## id dla int

---

```
x = 2 # zmiennej x przypisz wartość 2  
id(x) # wyświetl jej id
```

140685772794432

```
y = 2  
id(y) # to samo id co x
```

140685772794432

```
x is y # zmienne x i y wskazują w to samo miejsce
```

True

## id dla list

---

```
x = [1, 2, 3] # lista  
y = [1, 2, 3] # taka sama lista  
  
x == y # rzeczywiście taka sama
```

True

```
x is y # ale nie ta sama
```

False

```
print(id(x), id(y)) # listy mają różne id
```

140685552423752 140684962057672

```
x[0] is y[0] # ale elementy już nie
```

True

```
x = y # przypisz zmiennej x obiekty y  
x is y # teraz x i y wskazują to samo
```

True

## Mutable vs Immutable

- immutable: *int, float, complex, str, tuple, (frozen set, bytes)*
  - przypisanie zmiennej nowej wartości tworzy nowy obiekt w pamięci
- mutable: *list, (set, dict, byte array)*
  - możliwa modyfikacja obiektu

## Mutable vs Immutable

```
x = 2  
id(x)
```

140685772794432

```
x = 3  
id(x)
```

140685772794464

```
x = [2]  
id(x)
```

140685552461384

```
x.append(3)  
id(x)
```

140685552461384

## Konsekwencje

```
x = 2 # utwórz obiekt typu int
y = x # y wskazuje na to samo co x

y = 5 # zmieniam y - nowe miejsce w pamięci

print(x) # x dalej wskazuje na 2
```

2

```
x = [1, 2, 3] # utwórz obiekt typu list
y = x # y wskazuje na to samo co x

print(id(x), id(y))
```

140684962055816 140684962055816

```
y[0] = 4 # modyfikuję y

print(x) # a zmienia się ...
```

[4, 2, 3]

## Funkcja copy

```
x = [1, 2, 3] # utwórz obiekt typu list
y = x.copy() # stwórz kopię tego obiektu

print(id(x), id(y)) # teraz są to dwa różne obiekty
```

140685552556616 140685552460232

[Dismiss](#)

## Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)Branch: master ▾ [Python](#) / [4\\_Teoria](#) / [python\\_teoria\\_03.md](#)[Find file](#) [Copy path](#) Stach Poprawa struktury katalogow

4493d4e on Feb 15, 2019

0 contributors

1306 lines (787 sloc) 17.8 KB

[Raw](#) [Blame](#) [History](#)   

# Python

## Teoria 3

- pakowanie / odpakowywanie sekwencji
- kontrola przepływu:
  - instrukcje warunkowe *if*
  - pętle *for* i *while*
- formatowanie tekstu

### "Odpakowywanie" (*unzip*)

```
lista = [1, 2]
a, b = lista # a = lista[0], b = lista[1]
print(a, b)
```

1 2

- przykład: zamiana zmiennych

```
a, b = b, a # a, b = tuple(b, a)
print(a, b)
```

2 1

- dla porównania przykład zamiany zmiennych w C++

```
a = a + b - (b = a);
```

## splat, czyli tzw. asterisk

```
x = [1, 2, 3]

# print(arg1, arg2, arg3, ..., +opcje)

print(x) # drukuje listę (1 argument)
print(*x) # drukuje "rozpakowaną" listę (3 argumenty)
print(x[0], x[1], x[2]) # równoważny zapis
```

```
[1, 2, 3]
1 2 3
1 2 3
```

```
x = "Python"

print(x)
print(*x) # zip/unzip działa na wszystkich sekwencjach
```

## unzip i splat

```
lista = [1, 2, 3, 4, 5]

# a, b = lista # ValueError: too many values to unpack (expected 2)

a, *b = lista # a = lista[0], b = reszta

print(a, b)
```

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

```
lista = [1, 2, 3, 4, 5]

a, *b, c = lista # a, b[0], b[1], b[2], c

print(a, b, c)
```

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

## "Pakowanie" (zip)

```
x = [1, 2, 3]
y = ['a', 'b', 'c']
```

```
zipped = zip(x, y) # pary (x[i], y[i])  
  
print(*zipped)  
  
(1, 'a') (2, 'b') (3, 'c')  
  
x = [1, 2, 3]  
y = ['a', 'b', 'c', 'd'] # długość nie ma znaczenia  
  
zipped = zip(x, y) # pary (x[i], y[i])  
  
print(*zipped)  
  
(1, 'a') (2, 'b') (3, 'c')
```

## zip / unzip - test

---

```
x = [1, 2, 3]  
y = [4, 5, 6]  
  
x_copy, y_copy = zip(*zip(x, y))  
  
x == list(x_copy) and y == list(y_copy)
```

True

```
print(list(zip(x,y)))  
print(*zip(x,y))  
print(list(zip(*zip(x, y))))
```

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

## Paradygmaty programowania

---

- w Pythonie możliwe jest:
  - programowanie strukturalne (np. Fortran, C)
  - programowanie obiektowe (np. C++, Java)
  - programowanie funkcyjne (np. Lisp, Haskell)

## Programowanie strukturalne

---

- trzy struktury sterujące:
  - *sekwencja* - wykonanie instrukcji w zadanej kolejności
  - *wybór* - wykonanie instrukcji w zależności od stanu

- iteracja - wykonywanie instrukcji dopóki (nie)spełniony jest warunek

```
ps_dia = """
digraph {
    label="Programowanie strukturalne"

    rankdir=LR;

    start[shape="box", style=rounded];
    end[shape="box", style=rounded];

    if[shape="diamond", style=""];

    i1[shape="box", style=""];
    i2[shape="box", style=""];
    i3[shape="box", style=""];

    start -> i1;
    i1 -> if;
    if -> i2[label="nie"];
    if -> i3[label="tak"];
    i3 -> end;
    i2 -> i3;
}

from graphviz import Source

Source(ps_dia)
```

## Instrukcja blokowa

---

- wydzielona część kodu źródłowego (traktowana jak pojedyncza instrukcja)
- w C++ wykorzystuje się klamry

```
int x = 0;

{
    int y = 10; // zmienna lokalna
    x = y + 2;
}

cout << x; // OK
cout << y; // ERROR: nieznana zmienna
```

## Składnia Pythona

---

- instrukcje blokowe wprowadza się za pomocą wcięć

```
// przykład C++
double bezpieczne_dzielenie (int a, int b)
{
    if (b != 0) return a / b;
    else return 0;
}
```

```
# przykład Python
def bezpieczne_dzielenie (a, b):
    if b != 0:
        return a / b
    else:
        return 0
```

## Wcięcia

---

- spacje (zalecane) lub tabulacje
- w Pythonie 3 nie wolno mieszać (w Pythonie 2 się nie zaleca)
- najczęściej wcięcie = 4 spacje (czytelność)
- wiele edytorów umożliwia zamianę *tab* na spacje

## Instrukcja warunkowa *if*

---

- wykonaj instrukcje, jeśli spełniony jest warunek

```
if warunek: # zwróć uwagę na :
    instrukcja1
    instrukcja2
    ...
```

```
if 2 > 1:
    print("2 jest większe od 1")
```

2 jest większe od 1

## Instrukcja warunkowa *if else*

---

- wykonaj instrukcje, jeśli spełniony jest warunek, lub wykonaj inne instrukcje

```
if warunek:
    instrukcja1
    instrukcja2
    ...
else:
    instrukcja3
    instrukcja4
    ...

if 2 > 3:
    print("2 jest większe od 3")
else:
    print("2 nie jest większe od 3")
```

2 nie jest większe od 3

## Instrukcja warunkowa *if elif else*

- wykonaj jeśli, lub wykonaj jeśli, ..., lub wykonaj

```
if warunek1:  
    instrukcja1  
    instrukcja2  
    ...  
elif warunek2:  
    instrukcja3  
    instrukcja4  
    ...  
. . .  
. . .  
else:  
    instrukcja5  
    instrukcja6
```

## Instrukcja warunkowa *if elif else*

```
if 2 > 3:  
    print("2 jest większe od 3")  
elif 2 == 3:  
    print("2 jest równe 3")  
else:  
    print("2 jest mniejsze od 3")
```

2 jest mniejsze od 3

## *if* - przykład

```
print("Podaj liczbę:", end=' ')  
  
raw_x = input() # pobierz stringa z wejścia standardowego  
x = eval(raw_x) # zinterpretuj jako wyrażenie Pythona  
  
# x % 2 zwraca resztę z dzielenia  
# każda wartość != 0 jest traktowana jako prawda  
if x % 2:  
    print("Podana liczba jest nieparzysta.")  
else:  
    print("Podana liczba jest parzysta.")
```

```
Podaj liczbę: 2  
Podana liczba jest parzysta.
```

## Dygresja: *input* vs *raw\_input*

- w Pythonie 2

- *raw\_input* pobiera "surowego" stringa
- *input* dodatkowo go parsuje
- w Pythonie 3
  - *input* pobiera "surowego" stringa
  - *eval(input)* odtworzy zachowanie *input* z Pythona 2

## Dygresja: *input* i *eval*

---

```
help(input)
```

```
Help on method raw_input in module ipykernel.kernelbase:
```

```
raw_input(prompt='') method of ipykernel.ipkernel.IPythonKernel instance
    Forward raw_input to frontends

    Raises
    -----
    StdinNotImplementedError if active frontend doesn't support stdin.
```

```
type(input()) # pobiera wejście jako string
```

```
2
```

```
str
```

## Dygresja: *input* i *eval*

---

```
help(eval)
```

```
Help on built-in function eval in module builtins:
```

```
eval(source, globals=None, locals=None, /)
    Evaluate the given source in the context of globals and locals.

    The source may be a string representing a Python expression
    or a code object as returned by compile().
    The globals must be a dictionary and locals can be any mapping,
    defaulting to the current globals and locals.
    If only globals is given, locals defaults to it.
```

```
type(eval(input())) # parsuje wejście jako komendę
```

```
2
```

```
int
```

## Iteratory

---

- iterator wskazuje element sekwencji oraz umożliwia dostęp do następnego

```
help(iter)
```

```
Help on built-in function iter in module builtins:
```

```
iter(...)
    iter(iterable) -> iterator
    iter(callable, sentinel) -> iterator

    Get an iterator from an object. In the first form, the argument must
    supply its own iterator, or be a sequence.
    In the second form, the callable is called until it returns the sentinel.
```

### *iter(iterable)*

---

```
lista = ['a', 'b', 'c', 'd'] # zwykła lista

it = iter(lista) # iterator listy (wskazuje na początek)

print(next(it)) # zwraca 1 element i przesuwa "wskaźnik"
print(next(it)) # zwraca 2 element i przesuwa "wskaźnik"
print(next(it)) # zwraca 3 element i przesuwa "wskaźnik"
print(next(it)) # zwraca 4 element i przesuwa "wskaźnik"
```

```
a
b
c
d
```

### *iter(callable, sentinel)*

---

```
i = 0

# funkcje omówimy w przyszłości
def funkcja():
    """Z każdym wywołaniem zwraca kolejną liczbę całkowitą."""
    global i # użyj globalnej zmiennej i
    i += 1 # zwiększ
    return i # i zwróć

# kolejne wartości zwracane przez funkcję
# iterowane aż zwróci 4
it = iter(funkcja, 4)
```

```
print(next(it))
print(next(it))
print(next(it))
# print(next(it)) # StopIteration

1
2
3
```

## Pętla *for*

---

- pętla po sekwencji
- np. w Pascalu pętla po (arytmetycznym) ciągu liczb
- np. w C++ obie możliwości

```
// przykład w C++
for (unsigned int i = 0; i < N; i++)
{
    // wykonaj coś na i-tym elemencie
}
for (auto it = v.begin(); it != v.end(); ++it)
{
    // wykonaj coś na iteratorze it
}
```

## *for* po liście

---

```
lista = ['a', 'b', 'c', 'd']

for element in lista: # pętla po liście, w każdym kroku
    print(element)    # element zmienia swoją wartość

a
b
c
d
```

## *for* "krok po kroku"

---

```
lista = ['a', 'b', 'c', 'd']

it = iter(lista)

element = next(it)
print(element)
element = next(it)
print(element)
element = next(it)
print(element)
element = next(it)
print(element)
```

```
a  
b  
c  
d
```

## for i next

---

```
lista = ['a', 'b', 'c', 'd']

it = iter(lista)

# pętla po iteratorze it
# wewnętrz "ręcznie" wywołujemy next()
for i in it:
    print("i = " + str(i))
    print("next(it) = " + str(next(it)))
```

```
i = a
next(it) = b
i = c
next(it) = d
```

## for po wycinkach

---

```
lista = ['a', 'b', 'c', 'd']

for element in lista[-2:]:
    print(element, end=' ')
```

```
c d
```

```
for i in range(10)[::2]:
    print(i, end=' ')
```

```
0 2 4 6 8
```

## for po krotce

---

```
krotka = ('a', 'b', 'c', 'd')

for element in krotka: # pętla po krotce
    print(element)      # działa jak po liście
```

```
a  
b  
c
```

d

## for po stringu

---

```
slowo = "Python"

for litera in slowo: # pętla po stringu
    print(litera)    # iteruje po literach
```

P  
y  
t  
h  
o  
n

## for po range

---

```
for liczba in range(5): # pętla po range
    print(liczba)        # "odpowiednik" for (i = 0; ...)
```

0  
1  
2  
3  
4

## Kontrola przepływu - przykład 1

---

```
lista = [1, 2.0, 'Python', 4]

for element in lista:
    if type(element) is not str: # nie drukuj stringów
        print(element) # zwróć uwagę na wcięcia

1
2.0
4
1j
```

## Kontrola przepływu - przykład 2

---

```
lista = [1, 2.0, 'Python', 4, 1j]

for element in lista:
    if type(element) is str:    # tylko dla stringów
```

```
for litera in element: # pętla po stringu
    print(litera) # zwróć uwagę na wcięcia
```

P  
y  
t  
h  
o  
n

## Kontrola przepływu - przykład 3

---

```
lista = [1, 2.0, 'Python', 4, 1j]

for i in range(len(lista)): # pętla indeksach od 0 do len(lista)
    print(str(i + 1) + ". " + str(lista[i])) # mało czytelne...
```

1. 1
2. 2.0
3. Python
4. 4
5. 1j

## *unzip i for*

---

```
lista = [[1, 'a'], [2, 'b'], [3, 'c']]

for cyfra, litera in lista: # cyfra, litera = "lista[i]"
    print(str(cyfra) + ". " + litera)
```

1. a
2. b
3. c

```
cyfry = [1, 2, 3]
litery = ['a', 'b', 'c']

for cyfra, litera in zip(cyfry, litery):
    print(str(cyfra) + ". " + litera)
```

1. a
2. b
3. c

## *enumerate*

---

```
lista = ['a', 'b', 'c']
```

```
list(enumerate(lista)) # tworzy pary (indeks, element)

[(0, 'a'), (1, 'b'), (2, 'c')]

for index, element in enumerate(lista):
    print(str(index + 1) + ". " + element)

1. a
2. b
3. c
```

## Pętla *while*

---

- wykonuj blok instrukcji dopóki warunek jest spełniony

```
while warunek:
    instrukcje
```

- przykład:

```
i = 0

while i < 5: # wykonuj dopóki i < 5
    i += 1    # bez tego mamy nieskończoną pętlę
    print(i, end=' ')
```

1 2 3 4 5

## *while* - przykład

---

```
n = input("Podaj liczbę całkowitą: ")

if n.isdigit():
    print("Twoja liczba to:", n)
else:
    print(n, "nie jest liczbą całkowitą...")
```

```
Podaj liczbę całkowitą: 23
Twoja liczba to: 23
```

## *while* - przykład

---

```
n = input("Podaj liczbę całkowitą: ")

while not n.isdigit():
```

```
n = input("Spróbuj jeszcze raz: ")  
print("Twoja liczba to:", n)
```

```
Podaj liczbę całkowitą: a  
Spróbuj jeszcze raz: 2  
Twoja liczba to: 2
```

## while - nieskończona pętla

---

```
i = 1  
  
while i != 10:  
    i += 2
```

## Dodatkowe instrukcje sterujące

---

- *break* - przerwij pętlę
- *continue* - przerwij obecną iterację
- *pass* - nie rób nic
- *else* - wykonuj jeśli pętla zakończyła się inaczej niż *break*

### ***break***

---

```
for i in range(10): # drukuj liczby z range(10)  
    print(i, end=' ')  
    if i > 5:      # przerwij pętlę jeśli i > 5  
        break
```

```
0 1 2 3 4 5 6
```

```
i = 0  
  
while True:    # wykonuj w nieskończoność  
    print(i, end=' ')  
    if i > 5: # przerwij pętlę jeśli i > 5  
        break  
    i += 1     # bez tego byłaby nieskończona pętla zer
```

```
0 1 2 3 4 5 6
```

### ***continue***

---

```
for i in range(10):
```

```
if i % 2:    # jeśli i jest nieparzyste
    continue # pomini
print(i, end=' ')
```

0 2 4 6 8

- znowu uwaga na nieskończoną *while*

```
i = 0

# pętla wydrukuje 0 i utknie na 1
while i < 10:
    if i % 2:
        continue
    print(i, end=' ')
    i += 1
```

## pass

---

```
for i in range(10):
    if i % 2:    # jeśli i jest nieparzyste
        pass    # nie rób nic
    else:        # w innej sytuacji drukuj
        print(i, end=' ')
```

0 2 4 6 8

```
i = 0

# w praktyce napisalibyśmy: if not i % 2: print...
while i < 10:
    if i % 2:
        pass
    else:
        print(i, end=' ')
    i += 1
```

0 2 4 6 8

## else

---

```
for i in range(10):
    if i < 5: # drukuj mniejsze od 5
        print(i, end=' ')
    else:      # dla pozostałych nie rób nic
        pass
else: # wykonaj po zakończeniu pętli
    print("Koniec pętli.")
```

0 1 2 3 4 Koniec pętli.

```
for i in range(10):
    if i < 5: # drukuj mniejsze od 5
        print(i, end=' ')
    else:      # przerwij pętlę
        break
else: # pętla nie doszła do końca przez break
    print("Koniec pętli.")
```

0 1 2 3 4

## Formatowanie tekstu

---

```
help(print)
```

Help on built-in function print in module builtins:

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

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
  file:  a file-like object (stream); defaults to the current sys.stdout.
  sep:   string inserted between values, default a space.
  end:   string appended after the last value, default a newline.
  flush: whether to forcibly flush the stream.
```

## *print*

---

```
a, b, c = 1, 2, 3
```

```
print(a, b, c) # domyślnym separatorem jest spacja
```

1 2 3

```
print(a, b, c, sep='_') # ale można go zmienić
```

1\_2\_3

```
# domyślnie print kończy sekwencję nową linią
# ale można to zmienić
print(a, b, c, sep="...", end=" koniec")
```

1...2...3 koniec

## *format - podstawy*

---

```
x = 2

"x jest równe " + str(x) # skuteczne, ale mało wygodne

'x jest równe 2'

x = 2

# klasa string ma metodę format
"x jest równe {}".format(x)

'x jest równe 2'

# która w miejscu {} wstawia kolejne argumenty
"x jest równe {}, a x**2 = {}".format(x, x**2)

'x jest równe 2, a x**2 = 4'
```

## ***format - kolejność***

---

```
x = 2
y = 2.5
z = "trzy"

# domnie pod {} wstawiane są kolejne argument format
"x, y, z = {}, {}, {}".format(x, y, z)

'x, y, z = 2, 2.5, trzy'

# ale kolejność można zmienić
"x, y, z = {2}, {0}, {1}".format(x, y, z)

'x, y, z = trzy, 2, 2.5'
```

## ***format - deklaracja typu***

---

```
x = 1234567890
y = 1234567890.1234567890
z = "Python"

# z reguły nie ma potrzeby jawnej deklaracji typu
print("x, y, z = {}, {}, {}".format(x, y, z))

x, y, z = 1234567890, 1234567890.1234567, Python
```

```
# ale można to zrobić  
# d - int; f - float; s - str  
print("x, y, z = {:d}, {:f}, {:s}".format(x, y, z))
```

x, y, z = 1234567890, 1234567890.123457, Python

```
# e - wygodny format dla dużych liczb  
# XeY = X * 10^Y  
print("x, y, z = {:f}, {:e}, {}".format(x, y, z))
```

x, y, z = 1234567890.000000, 1.234568e+09, Python

## ***format - dokładność***

---

```
from math import pi  
  
"pi = {}".format(pi)  
  
'pi = 3.141592653589793'  
  
# ograniczamy się do dwóch liczb po przecinku  
"pi = {:.2f}".format(pi)  
  
'pi = 3.14'  
  
# ograniczamy się do 50 liczb po przecinku  
# zwróć uwagę na zera na końcu  
"pi = {:.50f}".format(pi)  
  
'pi = 3.14159265358979311599796346854418516159057617187500'
```

## ***format - keyword arguments***

---

```
# żeby się nie pogubić  
# warto "tagować" kolejne argumenty  
"{student} otrzymał {ocena}".format(student="Jan Nowak", ocena=5)  
  
'Jan Nowak otrzymał 5'  
  
# wtedy kolejność nie ma znaczenia  
"{student} otrzymał {ocena}".format(ocena=5, student="Jan Nowak")
```

'Jan Nowak otrzymał 5'

```
# lista studentów
studenci = ["Kasia", "Basia", "Marek", "Józek"]
# każdy element odpowiada liście ocen danego studenta
dziennik = [[3, 4, 5], [], [5, 3], [3, 2, 2, 2]]

# [znak]^N -> centruj w szerokości N wypełniając [znakiem]
print("{:~-^42}".format("OCENY"), end="\n\n")

# enumerate zwraca dwa obiekty: index i parę: (student, oceny)
for i, (student, oceny) in enumerate(zip(studenci, dziennik)):
    # policz średnią; chyba że brak ocen
    if len(oceny):
        srednia = sum(oceny) / len(oceny)
    else:
        srednia = 0

    # {oceny:<15} działa jak ljust; {oceny:>15} działa jak rjust
    print("{index}. {imie}: {oceny:<15} => srednia = {srednia:.1f}"
          .format(index=i+1,
                  imie=student,
                  oceny=str(oceny),
                  srednia=srednia))

-----OCENY-----
1. Kasia: [3, 4, 5]      => srednia = 4.0
2. Basia: []              => srednia = 0.0
3. Marek: [5, 3]           => srednia = 4.0
4. Józek: [3, 2, 2, 2]   => srednia = 2.2
```

[Dismiss](#)

## Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)Branch: master ▾ [Python](#) / [4\\_Teoria](#) / [python\\_teoria\\_04.md](#)[Find file](#) [Copy path](#)

Stach Poprawa struktury katalogow

4493d4e on Feb 15, 2019

0 contributors

1587 lines (914 sloc) 24 KB

[Raw](#) [Blame](#) [History](#)

# Python

## Teoria 4

- typy mapujące: *dict*
- funkcje
- dokumentacja
- wyrażenie lambda

## Złożone typy danych

- typy sekwencyjne ("mutowalne"): *list*
- typy sekwencyjne ("niemutowalne"): *tuple, range, str*
- typy mapujące ("mutowalne"): *dict*

## Słownik (*dict*)

- mapuje obiekty hashowalne (*hashable*) w dowolne obiekty, czyli: *klucz: wartość*
  - *klucz* - stały hash
  - *wartość* - dowolny obiekt
- np. krotka może być kluczem, ale lista już nie
- słowniki tworzymy umieszczając oddzielone przecinkiem pary *key: value* w nawiasach klamrowych, np.

```
# student: numer indeksu
słownik = {"Kasia": 7236, "Basia": 5286, "Marek": 9807, "Darek": 7738}

print(słownik)
```

```
{'Kasia': 7236, 'Basia': 5286, 'Darek': 7738, 'Marek': 9807}
```

## Słownik przez konstruktor

```
# {key1: value1, key2: value2...}
a = {"jeden": 1, "dwa": 2, "trzy": 3}
# dict(key1=value1, key2=value2...)
b = dict(jeden=1, dwa=2, trzy=3)
# dict([(key1, value1), (key2, value2)...])
c = dict(zip(["jeden", "dwa", "trzy"], [1, 2, 3]))
# dict(słownik)
d = dict(a)

a == b == c == d
```

```
True
```

## Słownik - dostęp do wartości

```
# student: numer indeksu
słownik = {"Kasia": 7236, "Basia": 5286, "Marek": 9807, "Darek": 7738}

słownik["Kasia"] # dostęp -> dict[key]
```

```
7236
```

```
słownik[0] # słownik nie jest uporządkowany
```

```
-----  
KeyError Traceback (most recent call last)
```

```
<ipython-input-4-8f2cb9210d58> in <module>()
----> 1 słownik[0] # słownik nie jest uporządkowany
```

```
KeyError: 0
```

## Słowniki - dostęp do wartości

```
# student: numer indeksu
słownik = {"Kasia": 7236, "Basia": 5286, "Marek": 9807, "Darek": 7738}

print(słownik["Kasia"]) # Kasia jest w słowniku -> OK
print(słownik["Ania"]) # Ani nie ma -> KeyError
```

```
7236
```

```
-----  
KeyError                                 Traceback (most recent call last)  
  
<ipython-input-5-7f55cd27f397> in <module>()  
      3  
      4 print(slownik["Kasia"]) # Kasia jest w słowniku -> OK  
----> 5 print(slownik["Ania"]) # Ani nie ma -> KeyError  
  
KeyError: 'Ania'
```

## Słowniki - dostęp do wartości

```
# student: numer indeksu  
slownik = {"Kasia": 7236, "Basia": 5286, "Marek": 9807, "Darek": 7738}  
  
print(slownik.get("Kasia")) # Kasia jest w słowniku -> OK  
print(slownik.get("Ania")) # Ani nie ma -> default = None  
print(slownik.get("Ania", 1234)) # Ani nie ma -> default = 1234
```

```
7236  
None  
1234
```

```
# czemu domyślnie None?  
if slownik.get("Ania"):  
    print(slownik["Ania"])  
else:  
    print("Brak studenta.")
```

```
Brak studenta.
```

```
# lub prościej  
print(slownik.get("Ania") or "Brak studenta.")
```

```
Brak studenta.
```

## Słownik - klucze i wartości

```
# student: numer indeksu  
slownik = {"Kasia": 7236, "Basia": 5286, "Marek": 9807, "Darek": 7738}  
  
slownik.keys() # lista kluczy  
  
dict_keys(['Kasia', 'Basia', 'Darek', 'Marek'])
```

```
słownik.values() # lista wartości  
  
dict_values([7236, 5286, 7738, 9807])  
  
słownik.items() # lista par  
  
dict_items([('Kasia', 7236), ('Basia', 5286), ('Darek', 7738), ('Marek', 9807)])
```

## Słownik - Python 2 vs 3

---

- w Pythonie 2 `dict.keys()`, `dict.values()` i `dict.items()` zwracają listy (czyli dopuszczalne jest np. `dict.keys()[0]`)
- w Pythonie 3:

```
import collections # for Iterable  
  
# student: numer indeksu  
słownik = {"Kasia": 7236, "Basia": 5286, "Marek": 9807, "Darek": 7738}  
  
# słownik.keys()[0] # TypeError: 'dict_keys' object does not support indexing  
  
isinstance(słownik.keys(), list)  
  
False
```

```
isinstance(słownik, collections.Iterable) # pętla po keys() -> OK
```

```
True
```

## Słownik - modyfikacja kluczy

---

```
# student: numer indeksu  
słownik = {"Kasia": 7236, "Basia": 5286, "Marek": 9807, "Darek": 7738}  
  
słownik["Kasia"] = 1234 # dict[key] = value  
  
print(słownik)
```

```
{'Kasia': 1234, 'Basia': 5286, 'Darek': 7738, 'Marek': 9807}
```

```
słownik["Kasia"] += 1 # analogicznie do listy: list[index]  
  
print(słownik)
```

```
{'Kasia': 1235, 'Basia': 5286, 'Darek': 7738, 'Marek': 9807}
```

## Słownik - usuwanie kluczy

---

```
# student: numer indeksu
słownik = {"Kasia": 7236, "Basia": 5286, "Marek": 9807, "Darek": 7738}

del słownik["Marek"] # del dict[key] -> usuń klucz

print(słownik)

{'Kasia': 7236, 'Basia': 5286, 'Darek': 7738}

słownik.clear() # wyczyść słownik

print(słownik)

{}
```

## Słownik - dodawanie kluczy

---

```
# student: numer indeksu
słownik = {"Kasia": 7236, "Basia": 5286, "Marek": 9807, "Darek": 7738}

słownik["Ania"] = 3384 # jeśli jest to zmień, jeśli nie ma to dodaj

print(słownik)

{'Kasia': 7236, 'Ania': 3384, 'Basia': 5286, 'Darek': 7738, 'Marek': 9807}

nowi_studenci = {"Romek": 3343, "Basia": 8573}

słownik.update(nowi_studenci) # "dodaj/scal" słowniki

print(słownik) # zwrócić uwagę na Basię

{'Basia': 8573, 'Romek': 3343, 'Darek': 7738, 'Kasia': 7236, 'Ania': 3384, 'Marek': 9807}

słownik.update(Józek=2276)

print(słownik)

{'Basia': 8573, 'Romek': 3343, 'Darek': 7738, 'Józek': 2276, 'Kasia': 7236, 'Ania': 3384, 'Marek': 9807}
```

## Pętla po słowniku

---

```
# student: numer indeksu
```

```
słownik = {"Kasia": 7236, "Basia": 5286, "Marek": 9807, "Darek": 7738}
```

```
for student in słownik.keys(): # pętla po kluczach
    print(student, end=' ')
```

Kasia Basia Darek Marek

```
for index in słownik.values(): # pętla po wartościach
    print(index, end=' ')
```

7236 5286 7738 9807

```
for student, index in słownik.items(): # pętla po (klucz, wartość)
    print(student, index)
```

Kasia 7236  
Basia 5286  
Darek 7738  
Marek 9807

## Przykład - tworzenie słownika

---

```
studenci = {} # stwórz pusty słownik

while True:
    # pobierz imię studenta
    student = input("Imię: ")
    # przerwij jeśli puste
    if not student: break # niezalecane, ale możliwe
    # pobierz numer indeksu
    index = input("Nr indeksu: ")
    # aktualizuj słownik
    studenci.update({student: index})

print(studenci)
```

```
Imię: Kasia
Nr indeksu: 1234
Imię: Jasiu
Nr indeksu: 0987
Imię:
{'Kasia': '1234', 'Jasiu': '0987'}
```

## Słownik w słowniku

---

```
Kasia = {"Wiek": 20, "Wzrost": 190, "Waga": 70}
Marek = {"Wiek": 22, "Wzrost": 180, "Waga": 80}
```

```
# klucz musi być hashowalny
# wartością może być dowolny obiekt, np. słownik
```

```
studenci = {"Kasia": "Kasia", "Marek": "Marek"}  
  
# wydrukuj w pętli wszystkich studentów i ich atrybuty  
for imie, wlasnosci in studenci.items():  
    print("{student} ma {wiek} lat, "  
          "{wzrost} cm wzrostu "  
          "i waży {waga} kg.".format(  
            student=imie,  
            wiek=studenci[imie]["Wiek"],  
            wzrost=studenci[imie]["Wzrost"],  
            waga=studenci[imie]["Waga"]))  
    )
```

Kasia ma 20 lat, 190 cm wzrostu i waży 70 kg.  
Marek ma 22 lat, 180 cm wzrostu i waży 80 kg.

## Funkcje

---

- funkcje (w programowaniu) to "wywoływalne bloki instrukcji"
- mogą (ale nie muszą) przyjmować argumenty
- mogą (ale nie muszą) zwracać obiekt
- raz napisana funkcja może być wykorzystywana wiele razy
- zwiększają czytelność kodu
- łatwiejsze debugowanie

## Definiowanie funkcji

---

```
def nazwa_funkcji(argumenty):  
    instrukcje  
    ...  
    return obiekt
```

1. słowo kluczowe *def*
2. nazwa funkcji
3. lista argumentów (opcjonalnie)
4. dwukropki
5. (wcięte) instrukcje
6. słowo kluczowe *return* (opcjonalnie)

## Funkcja "Hello World"

---

```
def hello(): # brak argumentów  
    print("Hello World!")  
    # nic nie zwraca  
  
hello() # wywołanie funkcji
```

Hello World!

## Argumenty funkcji

```
# funkcja dzialanie przyjmuje trzy argumenty a, b i c
# następnie zwraca wyrażenie a + b - c
def dzialanie(a, b, c):
    return a + b - c
```

```
dzialanie(1, 2, 3) # zwróci 1 + 2 - 3
```

```
0
```

```
x = 1
y = 2
```

```
dzialanie(x, y, 2*x + 3*y) # zwróci 1 + 2 - 2*1 - 3*2
```

```
-5
```

## Argumenty funkcji

```
def pobierz(komunikat): # funkcja przyjmuje jeden argument
    user_input = input(komunikat.ljust(20))
    return user_input # zwraca dane użytkownika

imie = pobierz("Jak masz na imię?") # komunikat = "Jak masz na imię?"
wiek = pobierz("Ile masz lat?")      # komunikat = "Ile masz lat?"

print(imie, wiek)
```

```
Jak masz na imię? Józek
Ile masz lat? 20
Józek 20
```

## Argumenty ze słowem kluczowym

```
# funkcja dzialanie przyjmuje trzy argumenty a, b i c
# następnie zwraca wyrażenie a + b - c
def dzialanie(a, b, c):
    return a + b - c
```

```
dzialanie(1, 2, 3) # 3 argumenty pozycyjne
```

```
0
```

```
dzialanie(a=1, b=2, c=3) # 3 argumenty kluczowe
0

dzialanie(b=2, c=3, a=1) # kolejność nieważna dla argumentów kluczowych
0
```

## Mieszanie argumentów

```
# funkcja dzialanie przyjmuje trzy argumenty a, b i c
# następnie zwraca wyrażenie a + b - c
def dzialanie(a, b, c):
    return a + b - c
```

```
dzialanie(1, c = 3, b = 2) # jeden pozycyjny, dwa kluczowe
```

```
0
```

```
dzialanie(a = 1, b = 2, 3) # najpierw pozycyjne, potem kluczowe!
```

```
File "<ipython-input-38-60286ad5b769>", line 1
    dzialanie(a = 1, b = 2, 3) # najpierw pozycyjne, potem kluczowe!
          ^
SyntaxError: positional argument follows keyword argument
```

## Domyślne wartości argumentów

```
def pobierz(komunikat):      # funkcja przyjmuje jeden argument
    user_input = input(komunikat) # pobiera dane od użytkownika
    return user_input           # i je zwraca

x = pobierz() # funkcja oczekuje argumentu "komunikat"
```

```
-----
TypeError                                Traceback (most recent call last)

<ipython-input-40-6184fdb92a47> in <module>()
----> 1 x = pobierz() # funkcja oczekuje argumentu "komunikat"

TypeError: pobierz() missing 1 required positional argument: 'komunikat'
```

## Domyślne wartości argumentów

```
def pobierz(komunikat='> '):      # domyślna wartość = "> "
    user_input = input(komunikat) # pobiera dane od użytkownika
    return user_input            # i je zwraca
```

```
x = pobierz() # brak argumentu -> domyślna wartość
```

```
> 1
```

```
x = pobierz("Napisz coś: ") # nadpisanie wartości domyślnej
```

```
Napisz coś: 1
```

## Domyślne wartości argumentów

```
# dwa ostatnie argumenty mają wartości domyślne
def dzialanie(a, b=2, c=3):
    return a + b - c
```

```
dzialanie(1, 2, 3) == dzialanie(1)
```

```
True
```

```
# argumenty bez wartości domyślnych
# nie mogą występować po tych
# które wartości domyślne posiadają
def dzialanie(a=1, b, c):
    return a + b - c
```

```
File "<ipython-input-46-21b67a48022e>", line 4
  def dzialanie(a=1, b, c):
  ^
SyntaxError: non-default argument follows default argument
```

## Argumenty "niemutowalne"

```
def zwiększ(x):
    print("wewnętrz funkcji (przed) id(x) =", id(x))
    x += 1
    print("wewnętrz funkcji (po) id(x)=", id(x))

x = 0 # zmienna x wskazuje na dany obszar pamięci
```

```
print("Przed zwiększeniem x =", x)
print("id(x) = ", id(x))
zwiększ(x)
print("id(x) = ", id(x))
print("Po zwiększeniu x =", x)

Przed zwiększeniem x = 0
id(x) = 140266666990080
wewnętrz funkcji (przed) id(x) = 140266666990080
wewnętrz funkcji (po) id(x)= 140266666990112
id(x) = 140266666990080
Po zwiększeniu x = 0
```

## Argumenty "mutowalne"

---

```
def zwiększ(x):
    print("wewnętrz funkcji (przed) id(x) =", id(x))
    x[0] += 1
    print("wewnętrz funkcji (po) id(x) () =", id(x))

x = [0] # zmienna x wskazuje na dany obszar pamięci

print("Przed zwiększeniem x =", x)
print("id(x) = ", id(x))
zwiększ(x)
print("id(x) = ", id(x))
print("Po zwiększeniu x =", x)

Przed zwiększeniem x = [0]
id(x) = 140266455152520
wewnętrz funkcji (przed) id(x) = 140266455152520
wewnętrz funkcji (po) id(x) () = 140266455152520
id(x) = 140266455152520
Po zwiększeniu x = [1]
```

## "Mutowalna" wartość domyślna

---

```
# wartość domyślana jest inicjowana tylko raz
def update(N, L = []):
    L.append(N)
    return L

print(update(1)) # L wskazuje na obszar w pamięci
print(update(2)) # i tak już zostaje
print(update(3)) # dla każdego kolejnego wywołania
```

[1]

```
[1, 2]  
[1, 2, 3]
```

## Obejście problemu

```
def update(N, L = None):  
    if not L:  
        L = []  
    L.append(N)  
    return L  
  
print(update(1)) # z każdym wywołaniem funkcji  
print(update(2)) # tworzona jest nowa lista  
print(update(3))
```

```
[1]  
[2]  
[3]
```

## Dowolna liczba argumentów (pozycyjnych)

```
# funkcja przyjmuje dwa "normalne" argumenty  
# z pozostałych powstanie krotka  
def funkcja(arg1, arg2, *args):  
    print(arg1, arg2, args)  
  
funkcja('a', 'b', 'c', 'd', 'e', 'f')  
  
a b ('c', 'd', 'e', 'f')
```

### \*args i key

```
# po *args mogą występować tylko argumenty kluczowe  
# mogą (ale nie muszą) mieć wartości domyślne  
def funkcja(arg1, *args, kwarg1, kwarg2="key"):  
    print("arg1 =", arg1)  
    print("*args =", args)  
    print("kwarg1 =", kwarg1)  
    print("kwarg2 =", kwarg2)  
  
funkcja('a', 'b', 'c', 'd', kwarg1="argument kluczowy")  
  
arg1 = a  
*args = ('b', 'c', 'd')  
kwarg1 = argument kluczowy  
kwarg2 = key  
  
funkcja('a', 'b', 'c', 'd')
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-53-aa278ac540f2> in <module>()  
----> 1 funkcja('a', 'b', 'c', 'd')  
  
TypeError: funkcja() missing 1 required keyword-only argument: 'kwarg1'
```

## Przykład

```
# średnia z dowolnej liczby argumentów  
def srednia(*args):  
    if not args: # brak argumentów -> 0  
        return 0  
    return sum(args) / len(args)  
  
srednia()
```

```
0
```

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

```
2.5
```

```
srednia(*range(100))
```

```
49.5
```

## Dowolna liczba argumentów (kluczowych)

```
# *args -> krotka argumentów pozycyjnych  
# **kwargs -> słownik argumentów kluczowych  
def funkcja(**kwargs):  
    for key, value in kwargs.items():  
        print(key, value, type(value))
```

```
funkcja(imie="Jan", nazwisko="Kowalski", wiek=25)
```

```
imie Jan <class 'str'>  
wiek 25 <class 'int'>  
nazwisko Kowalski <class 'str'>
```

```
funkcja("Jan", "Kowalski", 25)
```

```
-----  
TypeError                                Traceback (most recent call last)  
  
<ipython-input-60-1c93f25ce1db> in <module>()  
----> 1 funkcja("Jan", "Kowalski", 25)  
  
TypeError: funkcja() takes 0 positional arguments but 3 were given
```

## Forsowanie argumentów kluczowych

```
# zwróć sumę a i b  
# jeśli flaga==True zwróć 0  
def moja_suma(a, b, flaga=False):  
    if flaga:  
        return 0  
    else:  
        return a + b  
  
moja_suma(1, 2) # dla dwóch argumentów działa OK  
  
3  
  
moja_suma(1, 2, 3) # flaga == 3  
  
0
```

## Przykład

```
def create_car(marka, model, **options):  
    print(marka, model)  
    for opcja, wartosc in options.items():  
        print(" {}: {}".format(opcja, wartosc))  
  
create_car("Fiat", "126p", kolor="czarny", rocznik=1980)  
create_car("Ford", "Mustang", kolor="różowy", moc="200KM", skrzynia="manualna")  
  
Fiat 126p  
kolor: czarny  
rocznik: 1980  
Ford Mustang  
moc: 200KM  
kolor: różowy  
skrzynia: manualna
```

## Argumenty ze słownika

```
def create_car(marka, model, **options):
    print(marka, model)
    for opcja, wartosc in options.items():
        print(" {}: {}".format(opcja, wartosc))

ford_mustang = {"kolor": "różowy", "moc": "200KM", "skrzynia": "manualna"}

# **dict jako argument -> rozpakuj słownik
create_car("Ford", "Mustang", **ford_mustang)
```

```
Ford Mustang
moc: 200KM
kolor: różowy
skrzynia: manualna
```

## Funkcja jako rezultat funkcji

---

```
def kwadrat(x):
    print("kwadrat({})".format(x))
    return x*x

def kwadrat_sumy(a, b):
    print("kwadrat_sumy({}, {})".format(a, b))
    return kwadrat(a + b) # return może zwrócić inną funkcję

wynik = kwadrat_sumy(2, 3)

print("wynik =", wynik)

kwadrat_sumy(2, 3)
kwadrat(5)
wynik = 25
```

## Wynik funkcji jako argument funkcji

---

```
def kwadrat(x):
    print("kwadrat({})".format(x))
    return x*x

def suma(a, b):
    print("suma({}, {})".format(a, b))
    return a + b

def kwadrat_sumy(a, b):
    print("kwadrat_sumy({}, {})".format(a, b))
    return kwadrat(suma(a, b)) # wynik funkcji jest argumentem

wynik = kwadrat_sumy(2, 3)

print("wynik =", wynik)

kwadrat_sumy(2, 3)
suma(2, 3)
kwadrat(5)
```

```
wynik = 25
```

## Funkcja jako argument funkcji

```
def kwadrat(x):
    print("kwadrat({})".format(x))
    return x*x

def suma(a, b):
    print("suma({}, {})".format(a, b))
    return a + b

def kwadrat_sumy(a, b, f1, f2):
    print("kwadrat_sumy({}, {}, {}, {})".format(a, b, f1, f2))
    return f2(f1(a, b))

wynik = kwadrat_sumy(2, 3, suma, kwadrat) # funkcja jest argumentem

print("wynik =", wynik)

kwadrat_sumy(2, 3, <function suma at 0x7f9254388d08>, <function kwadrat at 0x7f925438ed90>)
suma(2, 3)
kwadrat(5)
wynik = 25
```

## Zmienne globalne

```
zmienna_globalna = "Hello World!"

print("Zmienna globalna przed =", zmienna_globalna)

def zmien():
    # print("Wewnatrz funkcji przed =", zmienna_globalna) # not assigned yet
    zmienna_globalna = "Bye World!" # zmienna_globalna jest zmienną lokalną
    print("Wewnatrz funkcji po =", zmienna_globalna)

zmien()

print("Zmienna globalna po =", zmienna_globalna)
```

```
Zmienna globalna przed = Hello World!
Wewnatrz funkcji po = Bye World!
Zmienna globalna po = Hello World!
```

## Zmienne globalne

```
zmienna_globalna = "Hello World!"

print("Zmienna globalna przed =", zmienna_globalna)

def zmien():
    global zmienna_globalna # sygnalizuje, że operujemy na zmiennej globalnej
```

```
print("Wewnatrz funkcji przed =", zmienna_globalna)
zmienna_globalna = "Bye World!"
print("Wewnatrz funkcji po =", zmienna_globalna)

zmien()

print("Zmienna globalna po =", zmienna_globalna)
```

```
Zmienna globalna przed = Hello World!
Wewnatrz funkcji przed = Hello World!
Wewnatrz funkcji po = Bye World!
Zmienna globalna po = Bye World!
```

## Rekurencja

---

```
i = 0

def funkcja():
    global i # uzywaj zmiennej globalnej i
    i += 1
    print(i, end=' ')
    if i < 10:
        return funkcja() # wywołuje sama siebie

funkcja()
```

```
1 2 3 4 5 6 7 8 9 10
```

## Rekurencja - silnia

---

```
def silnia(n):
    result = 1
    while n > 1:
        result *= n
        n -= 1
    return result

print("Klasycznie silnia(5) =", silnia(5))
```

```
Klasycznie silnia(5) = 120
```

```
def silnia(n):
    if n < 2:
        return 1 # 0! = 1, 1! = 1
    return n*silnia(n - 1)

print("Rekurencyjnie silnia(5) =", silnia(5))
```

```
Rekurencyjnie silnia(5) = 120
```

## Dokumentacja

```
help(help)
```

```
Help on _Helper in module _sitebuiltins object:
```

```
class _Helper(builtins.object)
| Define the builtin 'help'.
|
| This is a wrapper around pydoc.help that provides a helpful message
| when 'help' is typed at the Python interactive prompt.
|
| Calling help() at the Python prompt starts an interactive help session.
| Calling help(thing) prints help for the python object 'thing'.
|
| Methods defined here:
|
| __call__(self, *args, **kwds)
|     Call self as a function.
|
| __repr__(self)
|     Return repr(self).
|
| -----
|
| Data descriptors defined here:
|
| __dict__
|     dictionary for instance variables (if defined)
|
| __weakref__
|     list of weak references to the object (if defined)
```

## Dokumentowanie własnych funkcji

```
def funkcja():
    pass
```

```
help(funkcja)
```

```
Help on function funkcja in module __main__:
```

```
funkcja()
```

```
def funkcja():
    """To jest moja funkcja, która nic nie robi"""
    pass
```

```
help(funkcja)
```

```
Help on function funkcja in module __main__:
```

```
funkcja()
To jest moja funkcja, która nic nie robi
```

## Dokumentacja (PEP 257)

---

- dokumentacja (czyli tzw. *docstring*) wg standardów PEP 257:
  - jednoliniowy opis

```
"""Mój opis"""
```

- wieloliniowy opis

```
"""Mój opis
```

```
więcej opisu
"""

```

## Opis

---

- opis powinien krótki i jasny (i mieć sens), np:

```
# taki opis nie ma sensu
def moja_funkcja(a, b):
    """moja_funkcja(a, b) -> a + b"""
    return a + b

# taki jest lepszy
def moja_funkcja(a, b):
    """Zwraca sumę podanych liczb"""
    return a + b
```

## Dokumentowanie zmiennych

---

```
def reszta(a = 0, b = 0):
    """Zwraca resztę z dzielenia

    Keyword arguments:
    a -- licznik
    b -- mianownik
    """
    if not b:
        return 0
    return a%b

help(reszta)
```

```
Help on function reszta in module __main__:
```

```
reszta(a=0, b=0)
```

Zwraca resztę z dzielenia

Keyword arguments:  
a -- licznik  
b -- mianownik

## Wyrażenie *lambda*

---

```
def suma(a, b): # "klasyczna" funkcja
    return a + b

lsuma = lambda a, b: a + b # "anonimowa" funkcja

suma(1, 2) # wywołujemy funkcję suma
```

3

```
lsuma(1, 2) # wywołujemy wyrażenia lambda
```

3

## Wyrażenie *lambda*

---

```
def wrapper(funkcja):
    print("Wywouję funkcję:", funkcja)
    funkcja()

def f():
    print("Hello World!")

wrapper(f)
```

```
Wywouję funkcję: <function f at 0x7f92543518c8>
Hello World!
```

```
wrapper(lambda: print("Hello World!"))
```

```
Wywouję funkcję: <function <lambda> at 0x7f92543517b8>
Hello World!
```

## Wyrażenie *lambda*

---

```
def increase(n):
```

```
print("Teraz tworzę lambdę")
return lambda x: x + n

inc2 = increase(2) # powiększa o 2
inc4 = increase(4) # inc4 = lambda x: x + 4
```

Teraz tworzę lambdę  
Teraz tworzę lambdę

```
inc2(10)
```

12

```
inc4(10)
```

14

## Funkcja *main*

---

- najczęściej program podzielony jest na pojedyncze funkcje
- zwiększa to czytelność kodu i ułatwia jego debugowanie
- funkcja main "steruje" programem - uruchamiana jest przy wywołaniu skryptu

## Funkcja *main*

---

```
# skrypt.py

def start():
    print("Zaczynam program")

def operations():
    print("Wykonuje obliczenia")

def end():
    print("Kończę program")

# taki skrypt nic nie zrobi
# bo żadna funkcja nie jest wywołana
```

## Funkcja *main*

---

```
# skrypt.py

def start():
    print("Zaczynam program")

def operations():
    print("Wykonuje obliczenia")
```

```
def end():
    print("Kończę program")

def main():
    start()
    operations()
    end()

if __name__ == "__main__":
    main()
```

Zaczynam program  
Wykonuje obliczenia  
Kończę program

If sprawia że funkcja main() jest wywołana jedynie w momencie kiedy uruchomimy skrypt. Jeśli zaimportujemy skrypt wtedy funkcja main() nie zostanie wywołana.

[Dismiss](#)

## Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)Branch: master ▾ [Python](#) / [4\\_Teoria](#) / [python\\_teoria\\_05.md](#)[Find file](#) [Copy path](#) Stach Poprawa struktury katalogow

4493d4e on Feb 15, 2019

0 contributors

199 lines (117 sloc) 3.12 KB

[Raw](#) [Blame](#) [History](#)   

# Python

## Teoria 5

- *timeit*, czyli pythonowy stoper

### Dygresja o *main*

```
import math

print(__name__)      # nazwa skryptu
print(math.__name__) # nazwa zainportowanego modułu
```

```
__main__
math
```

```
# instrukcje z main
# z zainportowanych modułów
# nie zostaną wywołane
```

```
if __name__ == "__main__":
    print("Hello World!")
```

```
Hello World!
```

### *timeit*

```
def silnia(n):
```

```
"""Zwraca silnię liczbę n."""
if n < 2:
    return 1 # 0! = 1, 1! = 1
return n*silnia(n - 1)

from timeit import timeit

# timeit(funkcja, setup, number) -> czas wykonania funkcji w s
# funkcja - funkcja do wykonania
# setup - konfiguracja
# number - liczba powtórzeń
timeit("silnia(10)", setup="from __main__ import silnia", number=10)

3.7365999560279306e-05

timeit("silnia(100)", setup="from __main__ import silnia", number=10)

0.0004746449994854629
```

## timeit z linii komend

```
# silnia.py

def silnia(n):
    """Zwraca silnię liczbę n."""
    if n < 2:
        return 1 # 0! = 1, 1! = 1
    return n*silnia(n - 1)

• możemy przetestować naszą funkcję z linii komend

$ python -m timeit -n10 'from silnia import silnia; silnia(10)'
10 loops, best of 3: 9.11 usec per loop
```

## Koty Ali

- Pierwszego dnia Ala dostała jednego kota. Każdego kolejnego dnia dostaje o jednego więcej niż dnia poprzedniego. Ile kotów ma Ala po  $n$  dniach?
  - pierwszy dzień: 1
  - drugi dzień: 1 + 1
  - trzeci dzień: 1 + 1 + 1 ...

## Koty Ali - algorytm I

```
def licz_koty_v1(n):
    """Zwraca liczbę kotów po n dniach."""
    n_cats = 0 # Ala nie ma kota
```

```
for dzien in range(1, n + 1): # pętla po dniach
    for koty in range(dzien): # liczba kotów = nr dnia
        n_cats += 1           # dodaj kota
return n_cats                 # dwie pętle -> n*n operacji

%timeit -n3 licz_koty_v1(10000) # liczba kotów po 10 dniach

3 loops, best of 3: 4.43 s per loop
```

## Koty Ali - algorytm II

---

```
def licz_koty_v2(n):
    """Zwraca liczbę kotów po n dniach."""
    n_cats = 0                  # Ala nie ma kota
    for dzien in range(1, n + 1): # pętla po dniach
        n_cats += dzien          # dodaj koty (= nr dnia)
    return n_cats                # jedna pętla -> n operacji

%timeit -n3 licz_koty_v2(10000) # liczba kotów po 10 dniach

3 loops, best of 3: 904 µs per loop
```

## Koty Ali - algorytm III

---

```
def licz_koty_v3(n):
    """Zwraca liczbę kotów po n dniach."""
    return n * (n + 1) // 2 # jedna operacja

%timeit -n3 licz_koty_v3(10000) # liczba kotów po 10 dniach

3 loops, best of 3: 535 ns per loop
```

## Koty Ali - algorytm IV

---

```
def licz_koty_v4(n):
    """Zwraca liczbę kotów po n dniach."""
    koty = list(range(n + 1)) # n zmiennych w pamięci
    return sum(koty)          # jedna pętla -> n operacji

%timeit -n3 licz_koty_v4(10000) # liczba kotów po 10 dniach

3 loops, best of 3: 411 µs per loop
```



[Dismiss](#)

## Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)Branch: master ▾ [Python](#) / [4\\_Teoria](#) / [python\\_teoria\\_06.md](#)[Find file](#) [Copy path](#) Stach Poprawa struktury katalogow

4493d4e on Feb 15, 2019

0 contributors

1099 lines (626 sloc) 18.1 KB

[Raw](#) [Blame](#) [History](#)   

# Python

## Teoria 6

- przestrzenie nazw
- moduły

## Nazwy zmiennych

```
zmienna = 10 # zmienna wskazuje na int
print(zmienna)

zmienna = "Hello World!" # zmienna wskazuje na str
print(zmienna)

def funkcja():
    print("Jestem funkcją.")

zmienna = funkcja # zmienna wskazuje na funkcje
zmienna()

10
Hello World!
Jestem funkcją.
```

## Niebezpieczne nadpisania

```
def moja_nazwa(): # moja_nazwa wskazuje na funkcję
```

```
print("Jestem funkcją.")

def moja_nazwa(): # moja_nazwa wskazuje inną funkcję
    print("Jestem nową funkcją.")

moja_nazwa()
```

Jestem nową funkcją.

```
moja_nazwa = 10 # moja_nazwa wskazuje na int

moja_nazwa() # TypeError: 'int' object is not callable
```

```
-----
TypeError                                     Traceback (most recent call last)

<ipython-input-3-d535b594ed2f> in <module>()
      1 moja_nazwa = 10 # moja_nazwa wskazuje na int
      2
----> 3 moja_nazwa() # TypeError: 'int' object is not callable

TypeError: 'int' object is not callable
```

## Moduły - *import ...*

```
import math

def sin(x):
    """Zwraca x."""
    return x

math.sin(math.pi / 2) # wywołanie [moduł].[funkcja]
```

1.0

```
sin(math.pi / 2) # wywołanie [funkcja]
```

1.5707963267948966

## Moduły - *from ... import ...*

```
from math import sin

def sin(x): # nadpisuje sin z math
    """Zwraca x."""
    return x
```

```
sin(math.pi / 2) # wywołanie funkcji sin
```

```
1.5707963267948966
```

```
def sin(x):
    """Zwraca x."""
    return x

from math import sin # nadpisuje sin

sin(math.pi / 2) # wywołanie math.sin
```

```
1.0
```

## Przestrzeń nazw

---

- abstrakcyjna przestrzeń przechowująca nazwy
- np. przestrzeń nazw wbudowanych

```
print(dir(__builtin__)[-72:])
```

```
['abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'dreload', 'enumerate', 'eval', 'exec', 'filter', 'float', 'format', 'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

## Uwaga: nadpisać można wszystko

---

```
slownik = dict(x=1, y=2)
```

```
print(slownik)
```

```
{'y': 2, 'x': 1}
```

```
def dict(x, y): # nazwy wbudowane też można nadpisać
    return x, y
```

```
slownik = dict(x=1, y=2)
```

```
print(slownik)
```

```
(1, 2)
```

## Przestrzeń nazw lokalnych

---

- zmienne zdefiniowane wewnątrz funkcji (niedostępne poza nią)

```
def funkcja():          # zmienne lokalne dostępne
    zmienna_lokalna = 1  # są tylko wewnątrz funkcji
    return zmienna_lokalna # w której zostały zdefiniowane

print("zmienna_lokalna =", zmienna_lokalna)
```

---

```
NameError                                Traceback (most recent call last)

<ipython-input-13-7a69ba8766dd> in <module>()
      3     return zmienna_lokalna # w której zostały zdefiniowane
      4
----> 5 print("zmienna_lokalna =", zmienna_lokalna)

NameError: name 'zmienna_lokalna' is not defined
```

## Przestrzeń nazw globalnych

---

- dostępne w całym module (pliku)

```
zmienna_globalna = "Python"

def funkcja():
    return zmienna_globalna

# wewnątrz funkcji zmienne globalne są dostępne
funkcja()

'Python'
```

## Zmienne lokalne nadpisują globalne

---

```
zmienna_globalna = "Python"

def funkcja():
    zmienna_globalna = "Nowy Python" # lokalna zmienna_globalna
    print("in funkcja:", zmienna_globalna)

funkcja()

print("outside funkcja:", zmienna_globalna) # globalna bez zmian

in funkcja: Nowy Python
outside funkcja: Python
```

## Albo *global* albo *local*

---

```
zmienna_globalna = "Python"

def funkcja():
    print("in funkcja:", zmienna_globalna) # globalna?
    zmienna_globalna = "Nowy Python"
    print("in funkcja:", zmienna_globalna) # lokalna?

funkcja() # UnboundLocalError: local variable 'zmienna_globalna'
           # referenced before assignment

-----
UnboundLocalError                               Traceback (most recent call last)

<ipython-input-16-c9f400fd03ff> in <module>()
      6     print("in funkcja:", zmienna_globalna) # lokalna?
      7
----> 8 funkcja() # UnboundLocalError: local variable 'zmienna_globalna'
      9         # referenced before assignment

<ipython-input-16-c9f400fd03ff> in funkcja()
      2
      3 def funkcja():
----> 4     print("in funkcja:", zmienna_globalna) # globalna?
      5     zmienna_globalna = "Nowy Python"
      6     print("in funkcja:", zmienna_globalna) # lokalna?

UnboundLocalError: local variable 'zmienna_globalna' referenced before assignment
```

## Kolejność przestrzeni

- nazwy lokalne, potem globalne, na końcu wbudowane

```
list = tuple # zmienna globalna nadpisuje wbudowane list
dict = float # zmienna globalna nadpisuje wbudowane dict

print("type(list()) =", type(list()))
print("type(dict()) =", type(dict()))

def funkcja():
    dict = int # zmienna lokalna nadpisuje globalne dict
    print("In funkcja:")
    print("\ttype(dict()) =", type(dict())) # lokalna
    print("\ttype(list()) =", type(list())) # globalna
    print("\ttype(int()) =", type(int())) # wbudowana

funkcja()

type(list()) = <class 'tuple'>
type(dict()) = <class 'float'>
In funkcja:
    type(dict()) = <class 'int'>
    type(list()) = <class 'tuple'>
    type(int()) = <class 'int'>
```

## Wielokrotne zagnieżdżenie

```
a = "global a" # zasięg zmiennych a, b, c
b = "global b" # jest globalny
c = "global c"

def funkcja():
    a = "local a" # lokalne a, b dostępne w funkcji
    b = "local b" # i w każdym kolejnym zagnieżdzeniu

    def funkcja_w_funkcji():
        a = "local local a" # dostępna tylko w funkcja_w_funkcji
        print(a, b, c, sep='\n')

    funkcja_w_funkcji()

funkcja() # od "najlokalniejszej" do "najglobalniejszej"

local local a
local b
global c
```

## Uwaga na globalne mutowalne

---

```
x = [1, 2, 3]
y = ['a', 'b', 'c']

def funkcja():
    x = [1, 2, 3, 4] # przypisanie -> zmienna lokalna
    y.append('d')     # modyfikacja -> ciągle globalna

funkcja()

print(x, y, sep='\n')

[1, 2, 3]
['a', 'b', 'c', 'd']
```

## Wymuszanie zmiennej globalnej

---

```
zmienna = "globalna"

def f():
    global zmienna # przypisanie nie tworzy zmiennej lokalnej
    zmienna = "nowa globalna" # ale modyfikuje globalną

f()

print(zmienna)

nowa globalna
```

## Zasięg

---

```
def funkcja(flag=True):
    if flag:    # zmienna zdefiniowana w bloku
        x = 10 # jest dostępna poza tym blokiem
    else:
        x = 20

    print(x)
```

```
funkcja()
```

```
10
```

```
funkcja(False)
```

```
20
```

## Zasięg

- obszar dostępności danej przestrzeni nazw

```
import math
from math import cos

a = 1 # zasięg -> cały plik

def f():
    # uwaga: zaleca się wszystkie importy robić na początku
    from math import log # zasięg log(...) -> funkcja
    b = 2 # zasięg -> funkcja
    c = log(3)

d = math.sin(4) # sin(...) poza zasięgiem [moduł].[funkcja]
e = cos(5)      # zasięg cos(...) -> cały plik
# f = log(6)    # NameError: name 'log' is not defined
```

## Własny moduł - my\_module.py

```
"""To jest mój pierwszy moduł."""

zmienna_globalna = "Unikaj zmiennych globalnych!"

def moja_funkcja():
    """Drukuje zmienną globalną."""
    print(zmienna_globalna)

def inna_funkcja(word="", n=0):
    """Drukuje word n razy."""
    print(word * n)
```

## Importowanie własnego modułu

```
import my_module

my_module.zmienna_globalna

'Unikaj zmiennych globalnych!'

my_module.moja_funkcja()
```

Unikaj zmiennych globalnych!

```
my_module.inna_funkcja("-", 10)
```

-----

## Dokumentacja modułu

---

```
import my_module

help(my_module)

Help on module my_module:

NAME
    my_module - To jest mój pierwszy moduł.

FUNCTIONS
    inna_funkcja(word='', n=0)
        Drukuje word n razy.

    moja_funkcja()
        Drukuje zmienną globalną.

DATA
    zmienna_globalna = 'Unikaj zmiennych globalnych!'

FILE
    /doc/insync/scratch/zajęcia/2016/języki skryptowe - python/js-python/my_module.py
```

## Zawartość modułu

---

```
import my_module

dir(my_module)

['__builtins__',
 '__cached__',
 '__doc__',
```

```
'__file__',  
'__loader__',  
'__name__',  
'__package__',  
'__spec__',  
'inna_funkcja',  
'moja_funkcja',  
'zmienna_globalna']
```

## *doc, file, name, package*

---

```
print(my_module.__doc__) # dokumentacja modułu
```

To jest mój pierwszy moduł.

```
print(my_module.__file__) # ścieżka do pliku źródłowego
```

/doc/insync/scratch/zajęcia/2016/języki skryptowe - python/js-python/my\_module.py

```
print(my_module.__name__) # nazwa modułu
```

my\_module

```
print(my_module.__package__) # paczka to zbiór modułów
```

## *loader, spec, cached*

---

```
# informacja o "loaderze", który został wykorzystany  
# do zaimportowania modułu  
print(my_module.__loader__)
```

```
<_frozen_importlib_external.SourceFileLoader object at 0x7f134aded6a0>
```

```
print(my_module.__spec__) # ustalone w momencie importowania modułu
```

```
ModuleSpec(name='my_module', loader=<_frozen_importlib_external.SourceFileLoader object at  
0x7f134aded6a0>, origin='/doc/insync/scratch/zajęcia/2016/języki skryptowe - python/js-python  
/my_module.py')
```

## **Skrypt wykonujący operacje**

---

```
"""To jest mój pierwszy moduł."""
```

```
zmienna_globalna = "Unikaj zmiennych globalnych!"  
  
def moja_funkcja():  
    """Drukuje zmienną globalną."""  
    print(zmienna_globalna)  
  
def inna_funkcja(word="", n=0):  
    """Drukuje word n razy."""  
    print(word * n)  
  
inna_funkcja('-', 10)  
moja_funkcja()  
inna_funkcja('-', 10)
```

## Wykonywanie podczas importowania

---

```
Python 2.7.15  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import my_module  
-----  
Unikaj zmiennych globalnych!  
-----  
>>>
```

## "Funkcja main"

---

```
"""To jest mój pierwszy moduł."""  
  
zmienna_globalna = "Unikaj zmiennych globalnych!"  
  
def moja_funkcja():  
    """Drukuje zmienną globalną."""  
    print(zmienna_globalna)  
  
def inna_funkcja(word="", n=0):  
    """Drukuje word n razy."""  
    print(word * n)  
  
def main():  
    inna_funkcja('-', 10)  
    moja_funkcja()  
    inna_funkcja('-', 10)  
  
if __name__ == "__main__":  
    main()
```

## Importowanie a main

---

- instrukcje w *main* nie zostaną wykonane podczas importowania (bo `__name__ = nazwa modułu`)

```
Python 2.7.15  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import my_module
```

&gt;&gt;&gt;

## Prywatność

```
public = "public"
# _ przed nazwą -> from ... import * pomija
/internal = "internal use"

from private import * # importuj wszystko

public

'public'

/internal

-----
NameError                               Traceback (most recent call last)

<ipython-input-52-69fdfd559f34> in <module>()
      1 _internal

NameError: name '_internal' is not defined
```

## Gdzie interpreter szuka modułów? - linux

1. bieżący katalog
2. w katalogach określonych w zmiennej środowiskowej PYTHONPATH
3. w katalogach określonych w trakcie instalacji (np. /usr/lib/python )
4. w katalogach określonych w zmiennej sys.path

## Zmienne środowiskowe - linux

- zmienne powłoki systemowej

```
goran@goran-ift:~$ export MOJA_ZMIENNA="wartość mojej zmiennej"
goran@goran-ift:~$ echo $MOJA_ZMIENNA
wartość mojej zmiennej
```

- część zmiennych jest inicjowana przy starcie powłoki, np

```
goran@goran-ift:~$ echo $PATH
/home/goran/soft/anaconda3/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr,
```

## Zmienne środowiskowe - linux

---

- zmienne można aktualizować

```
goran@goran-ift:~$ export PATH=/newpath/:$PATH
goran@goran-ift:~$ echo $PATH
/newpath:/home/goran/soft/anaconda3/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
```

## *sys.path*

---

```
import sys

sys.path

['',
 '/home/goran/soft/anaconda3/lib/python35.zip',
 '/home/goran/soft/anaconda3/lib/python3.5',
 '/home/goran/soft/anaconda3/lib/python3.5/plat-linux',
 '/home/goran/soft/anaconda3/lib/python3.5/lib-dynload',
 '/home/goran/soft/anaconda3/lib/python3.5/site-packages',
 '/home/goran/soft/anaconda3/lib/python3.5/site-packages/Sphinx-1.4.6-py3.5.egg',
 '/home/goran/soft/anaconda3/lib/python3.5/site-packages/setuptools-27.2.0-py3.5.egg',
 '/home/goran/soft/anaconda3/lib/python3.5/site-packages/IPython/extensions',
 '/home/goran/.ipython']
```

## *sys.path*

---

```
sys.path.append("/moja/sciezka/do/modulow")

sys.path

['',
 '/home/goran/soft/anaconda3/lib/python35.zip',
 '/home/goran/soft/anaconda3/lib/python3.5',
 '/home/goran/soft/anaconda3/lib/python3.5/plat-linux',
 '/home/goran/soft/anaconda3/lib/python3.5/lib-dynload',
 '/home/goran/soft/anaconda3/lib/python3.5/site-packages',
 '/home/goran/soft/anaconda3/lib/python3.5/site-packages/Sphinx-1.4.6-py3.5.egg',
 '/home/goran/soft/anaconda3/lib/python3.5/site-packages/setuptools-27.2.0-py3.5.egg',
 '/home/goran/soft/anaconda3/lib/python3.5/site-packages/IPython/extensions',
 '/home/goran/.ipython',
 '/moja/sciezka/do/modulow']
```

## Moduł poza katalogiem roboczym

---

```
%%writefile /home/goran/trojmian.py
"""Moduł do obsługi trójmianu kwadratowego."""
```

```
from math import sqrt

def _delta(a, b, c):
    """Liczy wyróżnik trójmianu."""
    return b**2 - 4*a*c

def _solve(a, b, d):
    """Liczy miejsca zerowe."""
    d = sqrt(d)
    return (-b + d) / 2 / a, (-b - d) / 2 / a

def solution(a, b, c):
    """Zwraca miejsca zerowe."""
    d = _delta(a, b, c)

    if d < 0: return None
    elif d == 0: return -b / 2 / a
    else: return _solve(a, b, d)
```

Overwriting /home/goran/trojmian.py

## Moduł poza katalogiem roboczym

```
# nie ma w katalogu roboczym ani w PYTHONPATH itd
import trojmian
```

```
-----
ImportError                                     Traceback (most recent call last)

<ipython-input-56-e2dca95631fe> in <module>()
      1 # nie ma w katalogu roboczym ani w PYTHONPATH itd
----> 2 import trojmian

ImportError: No module named 'trojmian'
```

```
import sys
sys.path.append("/home/goran/") # dodajemy ścieżkę do sys.path
import trojmian # teraz działa
```

## Trójmian w akcji

```
help(trojmian) # nie ma _delta, ale jest sqrt!
```

```
Help on module trojmian:

NAME
    trojmian - Moduł do obsługi trójmianu kwadratowego.

FUNCTIONS
    solution(a, b, c)
        Zwraca miejsca zerowe.
```

```
sqrt(...)  
sqrt(x)  
  
Return the square root of x.  
  
FILE  
/home/goran/trojmian.py
```

## Trójmian w akcji

---

```
from trojmian import * # importuj wszystko  
  
solution(1, 2, 1) # dostęp do solution  
  
-1.0  
  
sqrt(4) # dostęp do sqrt z math!  
  
2.0  
  
_delta(1, 2, 1) # ale _delta nie została zimportowana  
  
-----  
NameError Traceback (most recent call last)  
<ipython-input-62-a5950f05b403> in <module>()  
----> 1 _delta(1, 2, 1) # ale _delta nie została zimportowana  
  
NameError: name '_delta' is not defined
```

## Paczka

---

- uporządkowany zbiór modułów

```
package_name/      # top-level  
    __init__.py   # wymagane (może być puste)  
    subpackage1/  
        __init__.py  
        module1.py  
        module2.py  
        ...  
    subpackage2/  
        __init__.py  
        ...
```

## init.py

---

```
%%writefile listy_zadan/__init__.py
"""Inicjalizacja paczki listy_zadan"""
import math # można np. importować moduły
```

Overwriting listy\_zadan/\_\_init\_\_.py

## Lista 4

---

```
%%writefile listy_zadan/lista4/__init__.py
"""Inicjalizacja podpaczki lista4"""


```

Overwriting listy\_zadan/lista4/\_\_init\_\_.py

## Lista 4 - zadanie 2

---

```
%%writefile listy_zadan/lista4/zad2.py
"""Napisz funkcję, która znajduje mniejszą liczbę z dwóch podanych."""

def min2(a, b):
    """Zwraca mniejszą z dwóch podanych liczb."""
    if a > b:
        return b
    return a
```

Overwriting listy\_zadan/lista4/zad2.py

## Lista 4 - zadanie 3

---

```
%%writefile listy_zadan/lista4/zad3.py
"""Napisz funkcję, która z podanych liczb (ilość dowolna) znajduje najmniejszą."""

from listy_zadan.lista4.zad2 import min2

def min(*a):
    """Zwraca najmniejszą z podanych liczb."""
    current_min = a[0]

    for x in a:
        current_min = min2(current_min, x)

    return current_min
```

Overwriting listy\_zadan/lista4/zad3.py

## Importowanie

---

```
import listy_zadan.lista4.zad2
import listy_zadan.lista4.zad3 as zad3

print("Zad2 =", listy_zadan.lista4.zad2.min2(1, 5))
print("Zad3 =", zad3.min(6, 2, 6, 3, 5))
```

```
Zad2 = 1
Zad3 = 2
```

[Dismiss](#)

## Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)Branch: master ▾ [Python](#) / [4\\_Teoria](#) / [python\\_teoria\\_07.md](#)[Find file](#) [Copy path](#) Stach Poprawa struktury katalogow

4493d4e on Feb 15, 2019

[0 contributors](#)

1139 lines (595 sloc) 15.9 KB

[Raw](#) [Blame](#) [History](#)   

# Python

## Teoria 7

- Bilioteka standardowa
  - os
  - sys
  - time / datetime
  - random
- Liczby pseudolosowe

## Moduł os

- przenośne używanie funkcjonalności zależnych od systemu operacyjnego
  - operacje na plikach (o tym za tydzień)
  - operacje na ścieżkach
  - uruchamianie poleceń systemowych
  - zarządzanie zmiennymi środowiskowymi
  - ...

## System operacyjny

```
import os

# POSIX = Portable Operating System Interface for Unix
os.name # posix, nt, ce, java

'posix'
```

```
if os.name == "posix": # jeśli unix
    cmd = "ls"          # użyj ls
elif os.name == "nt": # jeśli windows
    cmd = "dir"         # użyj dir

print(cmd)
```

ls

## Katalog roboczy

---

```
current_path = os.getcwd() # pwd

print("Katalog roboczy:", current_path)

new_path = "/home/goran"

os.chdir(new_path) # zmień katalog roboczy

print("Nowy katalog roboczy:", os.getcwd())

os.chdir(current_path) # wracamy
```

Katalog roboczy: /doc/insync/scratch/zajęcia/2016/języki skryptowe - python/js-python  
Nowy katalog roboczy: /home/goran

## Zawartość katalogu

---

```
os.listdir("/usr") # lista plików i katalogów

['src', 'local', 'bin', 'include', 'sbin', 'lib', 'games', 'share', 'locale']

os.listdir() # lista plików i katalogów w pwd

['my_script.py',
 'usos.py',
 '.ipynb_checkpoints',
 'js-python_w05.html',
 'js-python_w01.html',
 'js-python_l02.md',
 'test.2016_11_11_175921.log',
 'js-python_w04.ipynb',
 'js-python_w06.html',
 'README.md',
 'js-python_w03.html',
 'test.2016_11_11_175922.log',
 'js-python_l00.md',
 'iloczyn.py',
 'js-python_w04.html',
 'temp.py',
 'js-python_w03.ipynb',
```

```
'js-python_l01.pdf',
'js-python_w05.ipynb',
'print_args.py',
'src',
'js-python_l01.md',
'middle2.py',
'js-python_w01.ipynb',
'listy_zadan',
'__pycache__',
'dodaj_studentow.py',
'js-python_w07.ipynb',
'test.2016_11_11_175942.log',
'js-python_w02.ipynb',
'js-python_w02.html',
'temp',
'my_module.py',
'js-python_w06.ipynb',
'private.py']
```

## Tworzenie / usuwanie katalogów

---

```
def check(ls, folder):
    """Sprawdza czy folder znajduje się na liście."""
    if folder in ls:
        print("{} is found.".format(folder))
    else:
        print("{} not found.".format(folder))

my_folder = "test"

check(os.listdir(), my_folder)

os.mkdir(my_folder) # stwórz katalog test

check(os.listdir(), my_folder)

os.rmdir(my_folder) # usuń katalog test

check(os.listdir(), my_folder)

test not found.
test is found.
test not found.
```

## Tworzenie / usuwanie plików

---

```
os.mkdir("my_dir") # stwórz katalog "my_dir"

# otwórz plik w trybie zapisu (więcej za tydzień)
file = open("my_dir/my_file", 'w')
file.close()

os.listdir("my_dir")

['my_file']
```

```
# nie można usunąć niepustego katalogu
os.rmdir("my_dir") # OSError: [Errno 39] Directory not empty: 'my_dir'

-----
OSError                                Traceback (most recent call last)

<ipython-input-8-35adb4ad65f2> in <module>()
      1 # nie można usunąć niepustego katalogu
----> 2 os.rmdir("my_dir") # OSError: [Errno 39] Directory not empty: 'my_dir'

OSError: [Errno 39] Directory not empty: 'my_dir'

os.remove("my_dir/my_file") # usuń plik
os.rmdir("my_dir")          # usuń katalog
```

## Drzewo katalogów

---

```
# os.makedirs(..., exist_ok=True) -> działa jak mkdir -p
os.makedirs("level0/level1/level2") # utwórz "ciąg" katalogów

os.listdir("level0")

['level1']

os.listdir("level0/level1")

['level2']

# os.rmdir("level0/level1/level2")    # usunie tylko level2
os.removedirs("level0/level1/level2") # usunie wszystko
```

## Moduł *os.path*

---

```
import os

path = "/my/path" # katalog
file = "file.py" # plik

os.path.join(path, file) # tworzy pełną ścieżkę do pliku

'/my/path/file.py'

os.path.split("/my/path/file.py") # zwraca (head, tail)
```

```
('/my/path', 'file.py')

os.path.dirname("/my/path/file.py") # zwraca split()[0]

'/my/path'

os.path.basename("/my/path/file.py") # zwraca split()[1]

'file.py'
```

## Więcej o *os.path*

---

```
import os.path as path

path.splitext("/my/path/file.py") # wydziela rozszerzenie pliku

('/my/path/file', '.py')

my_path, my_file = path.splitext("/my/path/file.py")
my_base, my_extn = path.splitext(my_file)

print("my_path =", my_path)
print("my_file =", my_file)
print("my_base =", my_base)
print("my_extn =", my_extn)

my_path = /my/path
my_file = file.py
my_base = file
my_extn = .py
```

## Zmienne środowiskowe

---

```
os.getenv("HOME") # pobierz wartość zmiennej środowiskowej

'/home/goran'

os.environ["HOME"] # environ -> słownik z s

'/home/goran'

os.environ["MOJA_ZMIENNA"] = "To musi być string"
```

## os.walk

---

```
# os.walk "podróżuje po drzewie katalogów"
# na każdym kroku zwracając krotkę
# (obecny katalog, lista podkatalogów, lista plików)
for root, dirs, files in os.walk("/home/goran/usos"):
    print(root, dirs, files, sep="\n", end="\n\n")

/home/goran/usos
['Fizyka Komputerowa', 'ISSP']
[]

/home/goran/usos/Fizyka Komputerowa
['2016', '2015', '2017']
[]

/home/goran/usos/Fizyka Komputerowa/2016
[]
['Kasia.Nowak', 'Basia.Marian', 'Marek.Python']

/home/goran/usos/Fizyka Komputerowa/2015
[]
['Marek.Python']

/home/goran/usos/Fizyka Komputerowa/2017
[]
['Basia.Marian']

/home/goran/usos/ISSP
['2016', '2015', '2017']
[]

/home/goran/usos/ISSP/2016
[]
['Kasia.Python', 'Kasia.Nowak', 'Basia.Marian']

/home/goran/usos/ISSP/2015
[]
['Marek.Marian']

/home/goran/usos/ISSP/2017
[]
['Basia.Nowak']
```

## os.walk

---

```
for root, dirs, files in os.walk("/home/goran/usos"):
    for file in files: # pętla po plikach w danym katalogu root
        print(os.path.join(root, file))

/home/goran/usos/Fizyka Komputerowa/2016/Kasia.Nowak
/home/goran/usos/Fizyka Komputerowa/2016/Basia.Marian
/home/goran/usos/Fizyka Komputerowa/2016/Marek.Python
/home/goran/usos/Fizyka Komputerowa/2015/Marek.Python
/home/goran/usos/Fizyka Komputerowa/2017/Basia.Marian
/home/goran/usos/ISSP/2016/Kasia.Python
/home/goran/usos/ISSP/2016/Kasia.Nowak
```

```
/home/goran/usos/ISSP/2016/Basia.Marian  
/home/goran/usos/ISSP/2015/Marek.Marian  
/home/goran/usos/ISSP/2017/Basia.Nowak
```

## Moduł *glob*

---

- umożliwia wykorzystanie uniksowych *dzikich kart*: ?, \*, []

```
import glob

glob.glob("*.ipynb") # lista plików z rozszerzeniem ipynb

['js-python_w04.ipynb',
 'js-python_w03.ipynb',
 'js-python_w05.ipynb',
 'js-python_w01.ipynb',
 'js-python_w07.ipynb',
 'js-python_w02.ipynb',
 'js-python_w06.ipynb']
```

## Więcej o *glob*

---

```
# pliki z rozszerzeniem html oraz
# pasujące do wzorca (coś)(cyfra od 0 do 3)
glob.glob("*[0-3].html")

['js-python_w01.html', 'js-python_w03.html', 'js-python_w02.html']

glob.glob("js-python_?02.*") # js-python_[znak]02.[cokolwiek]

['js-python_102.md', 'js-python_w02.ipynb', 'js-python_w02.html']

glob.glob("js-python_?02.[mi]*") # js-python_[znak]02.[m lub i]cokolwiek

['js-python_102.md', 'js-python_w02.ipynb']
```

## Moduł *sys*

---

- funkcje i parametry specyficzne dla systemu operacyjnego
- ostatnio: ścieżka dostępu do modułów *sys.path*
- referencje do obiektów
- argumenty wywołania skryptu

## Referencje do obiektu

---

```
import sys

x = 1234
y = 1234
z = "Mój tekst"

sys.getrefcount(x) # x, y i ?
```

3

```
sys.getrefcount(z) # getrefcount pracuje na kopii z
```

2

## Referencje do obiektów *mutowalnych*

```
x = [1, 2, 3]
y = x
z = x.copy()

sys.getrefcount(x) # x, y i getrefcount
```

3

```
x is y # x i y wskazują to samo
```

True

```
x is z # x i z wskazują na inne listy
```

False

## Argumenty

```
"""Drukuję argumenty podane w linii komend."""

import sys
```

```
# sys.argv - lista argumentów linii komend
for i, arg in enumerate(sys.argv):
    print("{}: {}".format(i, arg))

!python print_args.py arg1 arg2 33 "hello world"
```

```
0. print_args.py  
1. arg1  
2. arg2  
3. 33  
4. hello world
```

## Przykład

---

```
"""Liczy iloczyn podanych argumentów."""  
  
import sys  
  
if len(sys.argv) < 2: # sys.argv[0] = iloczyn.py  
    print("Musisz podać co najmniej jedną liczbę .")  
else:  
    wyrazenie = "*".join(sys.argv[1:])  
    print(wyrazenie, "=", eval(wyrazenie))
```

Overwriting iloczyn.py

```
python iloczyn.py
```

Musisz podać co najmniej jedną liczbę .

```
python iloczyn.py 1 2 5 4 10 0.5
```

1\*2\*5\*4\*10\*0.5 = 200.0

## Moduł *time*

---

- operacje na czasie
- nie wszystkie funkcjonalności są dostępne na wszystkich platformach
- najczęściej wywoływane są funkcje biblioteki C o tej samej nazwie

## Czas uniksowy

---

```
import time  
  
time.gmtime() # Greenwich Mean Time / czas uniwersalny  
  
time.struct_time(tm_year=2016, tm_mon=11, tm_mday=14, tm_hour=17, tm_min=43, tm_sec=9, tm_wday=0,  
tm_yday=319, tm_isdst=0)  
  
time.time() # ile minęło sekund od
```

```
1479145391.2862186
```

```
time.gmtime(0) # eoki Unixa

time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3, tm_yday=1,
tm_isdst=0)
```

## Przybliżony test

```
time.time() / 60 / 60 / 24 / 365 # lat
```

```
46.90339274137039
```

```
time.gmtime()[0] - time.gmtime(0)[0] # rok teraz - rok epoki linuksa
```

```
46
```

## Czas lokalny

```
time.localtime() # czas lokalny
```

```
time.struct_time(tm_year=2016, tm_mon=11, tm_mday=14, tm_hour=18, tm_min=43, tm_sec=19, tm_wday=0,
tm_yday=319, tm_isdst=0)
```

```
time.localtime()[3] - time.gmtime()[3] # różnica Polska - GMT
```

```
1
```

## Czas w formie czytelnej

```
teraz = time.localtime()

time.strftime("%c", teraz) # lub time.asctime(teraz)

'Mon Nov 14 18:43:22 2016'

dzien_tygodnia = time.strftime("%A", teraz) # %a - skrót

print("Dzisiaj jest", dzien_tygodnia)
```

Dzisiaj jest Monday

```
time.strftime("%Y-%m-%d %H:%M", teraz) # ROK-MIESIĄC-DZIEŃ GODZINA:MINUTA
```

'2016-11-14 18:43'

- pełna lista dyrektyw [tutaj](#)

## Pomiar czasu (*wall time*)

```
import time

def poczekalnia(n):
    """Czeka n sekund."""
    time.sleep(n)

przed = time.time() # czas przed wywołaniem funkcji
poczekalnia(2)

po = time.time() # czas po wywołaniu funkcji
print("Czekałem {} sekund.".format(po - przed))
```

Czekałem 2.002260446548462 sekund.

## Pomiar czasu procesora (*process time*)

```
import time

def poczekalnia(n):
    """Czeka n sekund."""
    time.sleep(n)

przed = time.clock() # czas przed wywołaniem funkcji
poczekalnia(2)

po = time.clock() # czas po wywołaniu funkcji
print("Czekałem {} sekund.".format(po - przed))
```

Czekałem 0.001589000000000626 sekund.

## Moduł *datetime*

- wygodniejszy w użyciu do zarządzania czasem i datą
- arytmetyka czasu i daty, np.

```
import datetime

t0 = datetime.datetime.now()
t1 = datetime.datetime(2017, 2, 4) # początek sesji

print(t1 - t0)

81 days, 5:16:24.258938
```

## Przykład

---

```
teraz = datetime.datetime.now()
uniks_end = datetime.datetime(2038, 1, 19, 3, 14, 7)

print("teraz =", teraz)
print("rok =", teraz.year)
print("miesiąc =", teraz.month)
print("dzień =", teraz.day)
print("Do końca uniksa zostało", uniks_end - teraz)

teraz = 2016-11-14 18:43:37.185897
rok = 2016
miesiąc = 11
dzień = 14
Do końca uniksa zostało 7735 days, 8:30:29.814103
```

## Przykład - logi

---

```
import os
from datetime import datetime

def create_log(log_path, process):
    """Tworzy unikatowy log."""
    timestamp = datetime.now().strftime('%Y_%m_%d_%H%M%S')
    filename = ".".join([process, timestamp, "log"])
    with open(os.path.join(log_path, filename), 'w+') as f:
        f.write("Uruchomiono proces.")

create_log(os.getcwd(), "test")

%%bash
ls -l *.log

-rw-rw-r-- 1 goran goran 19 lis 11 17:59 test.2016_11_11_175921.log
-rw-rw-r-- 1 goran goran 19 lis 11 17:59 test.2016_11_11_175922.log
-rw-rw-r-- 1 goran goran 19 lis 11 17:59 test.2016_11_11_175942.log
-rw-rw-r-- 1 goran goran 19 lis 14 18:43 test.2016_11_14_184342.log
```

## Moduł random

---

- generator Mersenne Twister
- zaniedbywalna korelacja między kolejnymi liczbami (dobry do symulacji Monte Carlo)
- przewidywalność (nie nadaje się do kryptografii)
- szybki, ale "nieelegancki"

## Losowe liczby zmiennoprzecinkowe

---

```
import random

# random.random() -> losowa liczba z [0, 1)

losowe = [random.random() for _ in range(10)]

print(losowe)

[0.2698846730753022, 0.39646893011652196, 0.3569316622351202, 0.7422356288778353, 0.9872672338526717,
0.8037923911064603, 0.8702317548225692, 0.4956627050929412, 0.3254081126694237, 0.3929978024317329]

# random.uniform(a, b) -> losowa z przedziału [a, b]

losowe = [random.uniform(99, 100) for _ in range(10)]

print(losowe)

[99.99533728109901, 99.59952026049403, 99.42567811786209, 99.78233290994207, 99.84681728147966,
99.53305937003043, 99.47972787766165, 99.94064273021912, 99.8788743089142, 99.76186755697995]
```

## Losowe liczby całkowite

---

```
import random

# random.randint(a, b) - losowa całkowita z [a, b]

losowe = [random.randint(1, 10) for _ in range(10)]

print(losowe)

[4, 3, 7, 2, 5, 3, 6, 6, 5, 1]

# random.randrange(stop) -> losowa < stop
# random.randrange(start, stop[, step])

losowe = [random.randrange(10) for _ in range(10)]

print(losowe)

[7, 5, 0, 2, 4, 5, 1, 1, 1, 4]
```

## Losowe z sekwencji

---

```
import random

x = "Python"
y = ['a', 'b', 'c', 1, 2, 3]

random.choice(x) # losowa litera z str

'o'

random.choice(y) # losowy element listy

2

random.sample(y, 3) # losowy podzbiór

[1, 3, 'b']
```

## Seed

---

```
import random

for _ in range(5):
    print(random.random())

0.5943111012611626
0.7799061974116444
0.9688698425524694
0.09623949574155088
0.7030765207216848

for _ in range(5):
    random.seed(1234) # stały seed -> stała wartość
    print(random.random())

0.9664535356921388
0.9664535356921388
0.9664535356921388
0.9664535356921388
0.9664535356921388
```

## Totolotek

---

```
from random import randint

def losowanie():
    """Losuje 6 liczb od 1 do 49."""
    return sorted([randint(1, 49) for _ in range(6)])

def check(a, b):
    """Sprawdza ilość takich samych elementów."""
    return len([n for n in a if n in b])

def play():
    """Gra w lotka."""
    lotto = losowanie() # losowanie lotto
    kupon = losowanie() # kupon chybilek-trafił
    return check(kupon, lotto)
```

## Totolotek - symulacja

---

```
import matplotlib.pyplot as plt

n = 10000 # ilość gier

wyniki = [play() for _ in range(n)]

plt.hist(wyniki, 6)

(array([ 4.80300000e+03,   3.70800000e+03,   1.26200000e+03,
       1.96000000e+02,   2.90000000e+01,   2.00000000e+00]),
 array([ 0.        ,  0.83333333,  1.66666667,  2.5        ,  3.33333333,
        4.16666667,  5.        ]),
 <a list of 6 Patch objects>)
```

[Dismiss](#)

## Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)Branch: master ▾ [Python](#) / [4\\_Teoria](#) / [python\\_teoria\\_08.md](#)[Find file](#) [Copy path](#) Stach Poprawa struktury katalogow

4493d4e on Feb 15, 2019

0 contributors

1255 lines (727 sloc) 18.6 KB

[Raw](#) [Blame](#) [History](#)   

# Python

## Teoria 8

- Błędy i wyjątki
- Operacje na plikach

### Błędy (bug)

Without requirements or design, programming is the art of adding bugs to an empty text file.

*Louis Srygley*

- błędy leksykalne i składniowe
- błędy typowania
- błędy semantyczne i logiczne
- błędy działania
- nieskończone obliczenia

### Błędy leksykalne (syntax error)

- pojedyncza jednostka leksykalna, której nie przewiduje definicja języka

```
x = 1
x++ # operator ++ nie istnieje
```

```
File "<ipython-input-1-c2b85b799f03>", line 2
      x++ # operator ++ nie istnieje
               ^
SyntaxError: invalid syntax
```

## Błędy składniowe (*syntax error*)

- niepoprawnie zestawione poprawne jednostki leksykalne

```
if True          # brakuje :
    print("Hello World")) # dodatkowy )

File "<ipython-input-2-3cc3ffe6806b>", line 1
    if True          # brakuje :
                    ^
SyntaxError: invalid syntax
```

## Błędy typowania

- wyrażenie nieadekwatne do typu

```
x = 1

x[0] = 2 # x nie jest sekwencyjnym typem danych

-----
TypeError                                     Traceback (most recent call last)

<ipython-input-3-4bf6b631916d> in <module>()
      1 x = 1
      2
----> 3 x[0] = 2 # x nie jest sekwencyjnym typem danych

TypeError: 'int' object does not support item assignment
```

## Błędy działania (*runtime error*)

- pojawiają się w trakcie działania programu (np. odczyt z pliku, który nie istnieje)

```
def iloraz(a, b):
    """Zwraca a / b"""
    return a / b

iloraz(10, 0) # dzielenie przez 0
```

```
-----
ZeroDivisionError                           Traceback (most recent call last)

<ipython-input-4-976842d0fc36> in <module>()
      3     return a / b
      4
----> 5 iloraz(10, 0) # dzielenie przez 0
```

```
<ipython-input-4-976842d0fc36> in iloraz(a, b)
    1 def iloraz(a, b):
    2     """Zwraca a / b"""
----> 3     return a / b
        4
      5 iloraz(10, 0) # dzielenie przez 0

ZeroDivisionError: division by zero
```

## Błędy semantyczne (*semantic error*)

---

- niezgodność oczekiwania ze stanem faktycznym

```
def dzialanie(a, b, c):
    """Zwraca iloraz a przez sumę b i c."""
    return a / b + c # zamiast a / (b + c)
```

## Błędy logiczne

---

- program liczy nie to co trzeba (w tym też błędy semantyczne)
- najtrudniejsze do znalezienia

```
def delta(a, b, c):
    """Liczy wyróżnik trójmianu kwadratowego."""
    return b - 4*a*c # zamiast b*b - 4*a*c
```

## Nieskończone pętle

---

```
def loop(i = 0):
    while i < 10:
        i -= 1 # i zawsze będzie mniejsze od 10
```

## Therac-25

---

- maszyna do radioterapii nowotworów
- na skutek błędów programistycznych kilka osób zmarło na skutek napromieniowania
- błąd typu *race condition* - przy zbyt szybkim wprowadzaniu danych (przez operatora) parametry zabiegu nie były prawidłowo inicjowane

## Zapobieganie błędom

---

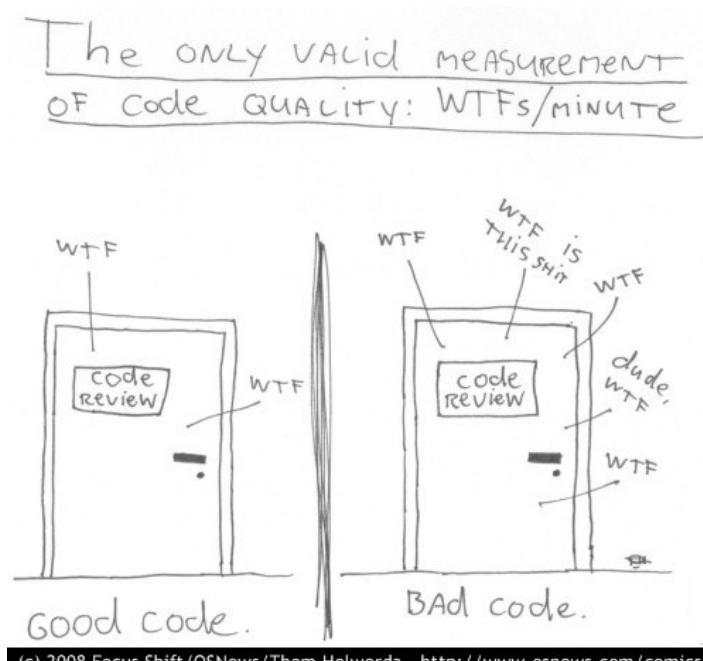
- pisanie czytelnego kodu
- code review*
- debugowanie

## Czytelność kodu

- zrozumiałe nazwy zmiennych (nawet kosztem długości)
- komentowanie kodu źródłowego, który nie jest zrozumiały od razu
- tworzenie dokumentacji w trakcie pisania programu
- opisywanie przyjętych założeń (w komentarzach i/lub dokumentacji)

## Code review

- sprawdzenie kodu przez inną osobę - poniższy link przedstawia typową sytuację w pracy, zrozumiecie jak zobaczycie na własne oczy ;)



(c) 2008 Focus Shift/OSNews/Thom Holwerda - <http://www.osnews.com/comics>

## Debugowanie

If debugging is the process of removing software bugs, then programming must be the process of putting them in.

*Edsger Dijkstra*

- systematyczne redukowanie błędów w kodzie
- kontrolowane wykonanie programu
- debugger

## \_\_debug\_\_

```
# __debug__ - wbudowana stała  
# równa True - jeśli uruchomione bez -O (optimize)
```

```
if __debug__:
    print("Jestem w trybie debugowania.")
else:
    print("Jestem w trybie normalnym.")
```

Overwriting debug.py

```
python debug.py
```

Jestem w trybie debugowania.

```
python -O debug.py
```

Jestem w trybie normalnym.

## Przykład

---

```
"""Wyznacza pierwsze wyrazy ciągu Fibonacciego."""
fib = [0, 1]

for i in range(10):
    if not __debug__":
        print("i =", i)
        print("fib =", fib)
        print("fib[i-2] =", fib[i-2])
        print("fib[i-1] =", fib[i-1])
        print()

    fib.append(fib[i-2] + fib[i-1])

print(fib) # [0, 1, 1, 2, 3, 5, 8, 13, ...]
```

```
python fib.py
```

[0, 1, 1, 2, 3, 5, 8, 13, ...]

## Przykład - *debug mode*

---

```
python -O fib.py
```

```
i = 0
fib = [0, 1]
fib[i-2] = 0
fib[i-1] = 1

i = 1
```

```
fib = [0, 1, 1]
fib[i-2] = 1
fib[i-1] = 0

i = 2
fib = [0, 1, 1, 1]
fib[i-2] = 0
fib[i-1] = 1

i = 3
fib = [0, 1, 1, 1, 1]
fib[i-2] = 1
fib[i-1] = 1

i = 4
fib = [0, 1, 1, 1, 1, 2]
fib[i-2] = 1
fib[i-1] = 1

i = 5
fib = [0, 1, 1, 1, 1, 2, 2]
fib[i-2] = 1
fib[i-1] = 1

i = 6
fib = [0, 1, 1, 1, 1, 2, 2, 2]
fib[i-2] = 1
fib[i-1] = 2

i = 7
fib = [0, 1, 1, 1, 1, 2, 2, 2, 3]
fib[i-2] = 2
fib[i-1] = 2

i = 8
fib = [0, 1, 1, 1, 1, 2, 2, 2, 3, 4]
fib[i-2] = 2
fib[i-1] = 2

i = 9
fib = [0, 1, 1, 1, 1, 2, 2, 2, 3, 4, 4]
fib[i-2] = 2
fib[i-1] = 3

[0, 1, 1, 1, 1, 2, 2, 2, 3, 4, 4, 5]
```

## Wbudowany debugger

---

```
import pdb # wbudowany debugger

pdb.set_trace() # zacznię debugować

fib = [0, 1]

for i in range(10):
    fib.append(fib[i-2] + fib[i-1])

print(fib) # [0, 1, 1, 2, 3, 5, 8, 13, ...]
```

## pdb

---

- *n* - następna instrukcja
- *s* - wejdź w funkcję
- *r* - wyjdź z funkcji
- *p* - wydrukuj zmienną
- *q* - przerwij
- *enter* - powtórz ostatnią komendę
- ...

## Wyjątki (*exceptions*)

- błędy działania (*runtime error*)
- wykryte podczas wykonywania są nazywane wyjątkami
- programista może decydować co robić z wyjątkami

## Przykład - iloraz

```
def iloraz(a, b):
    """Zwraca a / b"""
    return a / b

iloraz(10, 2)

5.0

iloraz(10, 0)

-----
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-17-0a93c896245e> in <module>()
----> 1 iloraz(10, 0)

<ipython-input-15-a5a78f530c06> in iloraz(a, b)
      1 def iloraz(a, b):
      2     """Zwraca a / b"""
----> 3     return a / b

ZeroDivisionError: division by zero
```

## Iloraz - zabezpiecznie *if*

```
def iloraz(a, b):
    """Zwraca a / b lub zero, jeśli b = 0."""
    if b == 0: # można tak, ale lepiej korzystać z wyjątków
```

```
    return 0
    return a / b

iloraz(10, 2)

5.0

iloraz(10, 0)

0
```

## Iloraz - wyjątek

---

```
def iloraz(a, b):
    """Zwraca a / b lub zero, jeśli b = 0."""
    try:    # spróbuj
        return a / b
    except: # jeśli error to
        return 0
```

```
iloraz(10, 2)

5.0

iloraz(10, 0)

0
```

## Przykład - pobierz liczbę całkowitą

---

```
while True:
    try:    # spróbuj rzutować na int
        x = int(input("Podaj liczbę: "))
        break
    except: # jeśli się nie uda to
        print("Spróbuj jeszcze raz.")
```

```
Podaj liczbę: a
Spróbuj jeszcze raz.
Podaj liczbę: 1.0
Spróbuj jeszcze raz.
Podaj liczbę: 1
```

## Typy wyjątków

---

- pełna lista wbudowanych wyjątków [link](#)

```
while True:  
    try:  
        x = int(input("Podaj liczbę: "))  
        break  
    except ValueError: # jeśli błąd wartości  
        print("Spróbuj jeszcze raz.")
```

```
Podaj liczbę: a  
Spróbuj jeszcze raz.  
Podaj liczbę: 1.0  
Spróbuj jeszcze raz.  
Podaj liczbę: 1
```

## Komunikat wyjątku

```
while True:  
    try:  
        x = int(input("Podaj liczbę: "))  
        break  
    except ValueError as err: # err = komunikat błędu  
        print("Spróbuj jeszcze raz, bo", err)
```

```
Podaj liczbę: a  
Spróbuj jeszcze raz, bo invalid literal for int() with base 10: 'a'  
Podaj liczbę: 1.0  
Spróbuj jeszcze raz, bo invalid literal for int() with base 10: '1.0'  
Podaj liczbę: 1
```

## Przykład

```
# więcej o plikach za chwilę  
  
import sys  
  
def read_data():  
    try:  
        plik = open('data.txt') # otwórz plik  
        linia = plik.readline() # wczytaj linię  
        dane = int(linia)      # rzutuj linię na int  
    except IOError as err:    # IOError  
        print("Błąd I/O:", err)  
    except ValueError as err: # ValueError  
        print("Złe dane:", err)  
    except:                  # InnyError  
        print("Coś poszło nie tak...")  
        sys.exit(0)
```

## Przykład - test

```
read_data()
```

Błąd I/O: [Errno 2] No such file or directory: 'data.txt'

Python

Writing data.txt

```
read_data()
```

Złe dane: invalid literal for int() with base 10: 'Python'

## *try...finally*

```
def iloraz(a, b):
    """Zwraca a / b."""
    try:
        return a / b
    finally: # wykonaj mimo zgłoszonego wyjątku
        print("Posprzątam bez względu na wyjątki.")
```

```
iloraz(10, 5)
```

Posprzątam bez względu na wyjątki.

2.0

```
iloraz(10, 0)
```

Posprzątam bez względu na wyjątki.

---

```
ZeroDivisionError                                Traceback (most recent call last)

<ipython-input-33-0a93c896245e> in <module>()
----> 1 iloraz(10, 0)

<ipython-input-31-2179acde9974> in iloraz(a, b)
      2     """Zwraca a / b."""
      3     try:
----> 4         return a / b
```

```
5     finally: # wykonaj mimo zgłoszonego wyjątku
6         print("Posprzątam bez względu na wyjątki.")
```

ZeroDivisionError: division by zero

## try...except...finally

---

```
def iloraz(a, b):
    """Zwraca a / b lub zero, jeśli b = 0."""
    try:
        return a / b
    except:
        print("Użytkownik nie zna matematyki.")
    finally: # wykona zawsze bez względu na wyni try
        print("Posprzątam bez względu na wyjątki.")

iloraz(10, 5)
```

Posprzątam bez względu na wyjątki.

2.0

```
iloraz(10, 0)
```

Użytkownik nie zna matematyki.  
Posprzątam bez względu na wyjątki.

## Zgłaszanie wyjątków

---

```
def iloraz(a, b):
    """Zwraca a / b."""
    if b == 0:
        raise NameError("Dzielenie przez zero.")
    return a / b

iloraz(10, 5)
```

2.0

```
iloraz(10, 0)
```

```
NameError                                 Traceback (most recent call last)

<ipython-input-39-0a93c896245e> in <module>()
----> 1 iloraz(10, 0)

<ipython-input-37-0c4a0fa13db9> in iloraz(a, b)
      2     """Zwraca a / b."""
      3     if b == 0:
----> 4         raise NameError("Dzielenie przez zero.")
      5     return a / b

NameError: Dzielenie przez zero.
```

## Zgłaszanie wyjątków test

---

```
try:
    iloraz(10, 0)
except NameError as err:
    print("Błąd:", err)
```

```
Błąd: Dzielenie przez zero.
```

## Własne wyjątki

---

- możliwe jest definiowanie własnych wyjątków
- o tym w przyszłości, jak już poznamy klasy

## assert

---

```
# assert expression jest równoważne
if __debug__:
    if not expression: raise AssertionError

import sys

assert sys.argv[1] != "Python"

print("Assert test.")
```

## assert - test

---

```
python assert.py 1
```

```
Assert test.
```

```
python assert.py Python
```

```
Traceback (most recent call last):
  File "assert.py", line 4, in <module>
    assert sys.argv[1] != "Python"
AssertionError
```

```
python -O assert.py Python
```

```
Assert test.
```

## Operacje na plikach

- do otwierania plików służy funkcja wbudowana *open*
- przyjmuje wiele argumentów, przy czym dwa najważniejsze to: *file* i *mode*

```
open(file, mode)
```

- *file* - nazwa pliku (lub pełna ścieżka, jeśli nie w katalogu roboczym)
- *mode* - tryb

## Tryby pracy nad plikiem

Tryb	Opis
r	tylko do odczytu (domyślnie)
w	tylko do zapisu (istniejący plik zostanie nadpisany)
x	tylko do zapisu (plik nie może istnieć)
a	tylko do zapisu (od końca pliku)
+	aktualizowanie pliku (odczyt i zapis)
t	tryb tekstowy (domyślnie)
b	tryb binarny

- np. `open(file, r+b)` otwiera plik do odczytu, z możliwością zapisu, w trybie binarnym

## Zapis do pliku

```
# otwórz plik do zapisu
# usuń zawartość jeśli plik istnieje
file = open("test", 'w')

file.write("0123456789") # zapisz do pliku

file.close() # zamknij plik
```

```
file.write("jeszcze coś") # plik już jest zamknięty

-----
ValueError                                Traceback (most recent call last)

<ipython-input-46-456f9e28996a> in <module>()
----> 1 file.write("jeszcze coś") # plik już jest zamknięty

ValueError: I/O operation on closed file.
```

## Dopisywanie do pliku

---

```
# otwórz plik do zapisu
# ustaw się na końcu pliku
file = open("test", 'a')

file.write("abcdefghijkl") # zapisz do pliku

file.close() # zamknij plik
```

## Odczyt pliku

---

```
file = open("test", "r") # otwórz tylko do odczytu

zawartosc = file.read() # wczytaj całą zawartość pliku

file.close()

print(zawartosc)
```

0123456789abcdefghijkl

## *r+* vs *a*

---

```
# a (append) - zaczyna dopisywać na koniec pliku
# r+ - zaczyna zapisywać od początku (nadpisując dane)

file = open("test", "r+") # odczyt z możliwością zapisu

file.write("." * 5) # zapisz 5 kropek

file.close()

# Bash
cat test
```

.....56789abcdefghijklj

### ***W+ VS r+***

---

```
# r+ - zaczyna zapisywać od początku (nadpisując dane)
# w+ - najpierw czyści plik (jeśli istnieje)

file = open("test", "w+") # zapis z możliwością odczytu

file.write("." * 5) # zapisz 5 kropek

file.close()

# Bash
cat test
```

.....

### ***W VS W+***

---

```
file = open("test", "w") # tylko zapis

file.write("." * 5) # zapisz 5 kropek

zawartosc = file.read()

file.close()

print(zawartosc)
```

```
-----  
UnsupportedOperation                               Traceback (most recent call last)  
  
<ipython-input-53-6fb740cd3fea> in <module>()  
      3     file.write("." * 5) # zapisz 5 kropek  
      4  
----> 5     zawartosc = file.read()  
      6  
      7     file.close()
```

UnsupportedOperation: not readable

### ***W VS W+***

---

```
file = open("test", "w+") # zapis z możliwością odczytu

file.write("." * 5) # zapisz 5 kropek
```

```
zawartosc = file.read()

file.close()

print(zawartosc) # nie wydrukuje bo jesteśmy na końcu pliku
```

## *seek*

---

```
file = open("test", "w+") # zapis z możliwością odczytu

file.write("." * 5) # zapisz 5 kropek

file.seek(0) # ustaw położenie

zawartosc = file.read()

file.close()

print(zawartosc) # nie wydrukuje bo jesteśmy na końcu pliku
```

.....

## *tell*

---

```
# Bash
cat test

.....
```

```
file = open("test", "a+") # dodawanie z możliwością odczytu

file.tell() # pozycja w pliku
```

5

```
file.write("12345") # dopisz 12345

file.tell()
```

10

```
# Bash
cat test
```

.....12345

## seek and read

```
# Bash
cat test

.....12345

# seek(offset, punkt_odniesienia)
# po = 0, 1, 2 (początek pliku, bieżąca pozycja, koniec pliku)
# w trybie tekstowym tylko 0 jest dozwolone

file.seek(6, 0) # o 6 znaków od początku

file.read(1)

'2'

file.tell() # read(n) przesuwa o n
```

7

## Uwaga

- wygodnie jest wczytać cały plik do pamięci
  - *read()* - zawartość jako pojedynczy string
  - *readlines()* - zawartość jako lista (linia -> element)
- jednak w przypadku dużych plików może to być katastrofalne, wtedy lepiej
  - *read(n)* - wczytaj *n* bajtów
  - *readline()* - wczytaj linię

## Otwieranie plików a wyjątki

```
try:
    file = open("złe_dane")
    data = file.read() # zgłasza wyjątek
finally: # porządek nawet w przypadku wyjątku
    print("Czyszczę śmieci.")
    file.close()
```

## with statement

- gwarantuje, że jeśli wywołane zostało *\_\_enter\_\_()* (np. otwarcie pliku)
- to zostanie wywołane *\_\_exit\_\_()* (np. zamknięcie pliku)
- nawet jeśli po drodze wystąpi wyjątek

## with open

```
with open("plik_z_danymi") as file:  
    data = file.read()
```

- gwarantuje, że plik zostanie zawsze poprawnie zamknięty
- wygodniejsze niż *try...finally...*

## Comma-separated values (CSV)

- forma przechowywania danych w plikach tekstowych
- każde pole oddzielone jest przecinkiem
- plik csv

```
imię, nazwisko, ocena  
Kasia, Kowalska, 4  
Jan, Nowak, 4
```

- tabela

imię	nazwisko	ocena
Kasia	Kowalska	4
Jan	Nowak	4

## Moduł csv

```
import csv  
  
data = [["imie", "nazwisko", "ocena"],  
        ["Kasia", "Kowalska", 4],  
        ["Jan", "Nowak", 4]]  
  
with open("oceny.csv", "w") as csvfile:  
    writer = csv.writer(csvfile) # tworzymy "pisarza" csv  
    for wpis in data:  
        writer.writerow(wpis)  
  
  
# Bash  
cat oceny.csv
```

```
imie,nazwisko,ocena  
Kasia,Kowalska,4  
Jan,Nowak,4
```

## Wczytywanie danych

```
import csv

with open('oceny.csv', 'r') as csvfile:
    reader = csv.reader(csvfile) # tworzymy "czytelnika" csv
    for wpis in reader:
        print(wpis)

['imie', 'nazwisko', 'ocena']
['Kasia', 'Kowalska', '4']
['Jan', 'Nowak', '4']
```

[Dismiss](#)

## Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)Branch: master ▾ [Python](#) / [4\\_Teoria](#) / [python\\_teoria\\_09.md](#)[Find file](#) [Copy path](#)

Stach Poprawa struktury katalogow

4493d4e on Feb 15, 2019

0 contributors

648 lines (347 sloc) 8.7 KB

[Raw](#) [Blame](#) [History](#)

# Python

## Teoria 9

- typy sekwencyjne: *set* i *frozenset*
- generatory
- omówienie zadań z listy 6

### Typy sekwencyjne

- do tej pory poznaliśmy cztery typy sekwencyjne:
  - *str* - typ tekstowy
  - *list* - *mutowalna* sekwencja
  - *tuple* - *niemutowalna* sekwencja
  - *range* - *niemutowalna* sekwencja liczb

### Sekwencja *set*

- *set* jest *mutowalną* sekwencją
- różnice względem *list*
  - nie może zawierać duplikatów
  - jest nieuporządkowana
  - może zawierać tylko *hashowalne* obiekty

### Definiowanie *set*

```
zbior = {1, 4, 6, 2, 1} # nawiasy klamrowe
```

```
type(zbior)

set

print(zbior)

{1, 2, 4, 6}

zbior[0] # nie ma indeksowania zbiorów

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-3-119e4f5506a9> in <module>()
----> 1 zbior[0] # nie ma indeksowania zbiorów

TypeError: 'set' object does not support indexing
```

## *set z list*

```
lista = [1, 4, 6, 2, 1]

zbior = set(lista) # stwórz zbiór na bazie listy

print(zbior)

{1, 2, 4, 6}
```

## Dodawanie elementów do zbioru

```
zbior = {1, 2, 3}

print(zbior)

{1, 2, 3}

zbior.add(4) # dodaj 4

print(zbior)

{1, 2, 3, 4}
```

```
zbior.add(4) # jeszcze raz dodaj 4  
  
print(zbior) # nie ma duplikatów  
  
{1, 2, 3, 4}
```

## Elementy *niehashowalne*

```
zbior = {1, 2, 3}
```

```
lista = [4, 5] # lista jest niehashowalna  
  
# zbiór nie może przechowywać  
# elementów niehashowalnych  
zbior.add(lista)
```

```
-----  
TypeError Traceback (most recent call last)  
  
<ipython-input-8-c1da28ad9f1d> in <module>()  
      5 # zbiór nie może przechowywać  
      6 # elementów niehashowalnych  
----> 7 zbior.add(lista)  
  
TypeError: unhashable type: 'list'
```

## Sekwencje hashowalne

```
zbior = {1, 2, 3}  
  
krotka = (4, 5) # krotka jest hashowalna  
  
# więc można dodać krotkę  
# do zbioru  
zbior.add(krotka)  
  
print(zbior)
```

```
{(4, 5), 1, 2, 3}
```

## Usuwanie elementów ze zbioru

```
zbior = {1, 2, 3}  
  
zbior.remove(1) # usuń 1  
  
print(zbior)
```

```
{2, 3}
```

```
# zwróci błąd jeśli element nie istnieje
zbior.remove(1)
```

```
-----  
KeyError                               Traceback (most recent call last)  
  
<ipython-input-11-15d16e079b36> in <module>()
      1 # zwróci błąd jeśli element nie istnieje
----> 2 zbior.remove(1)  
  
KeyError: 1
```

## Bezpieczne usuwanie?

```
zbior = {1, 2, 3}  
  
# aby uniknąć błędów i przerwania programu
# można sprawdzić, czy na pewno element
# znajduje się w zbiorze
if 1 in zbior:
    zbior.remove(1)  
  
# lub skorzystać z try...except
try:
    zbior.remove(1)
except:
    pass  
  
# lub skorzystać z discard
zbior.discard(1)
```

## Zastosowanie zbiorów

```
# usuwanie duplikatów z listy
lista = [2, 1, 2, 3, 4, 3, 2, 1, 5, 5, 2]
lista = list(set(lista))
# uwaga - tracimy kolejność
print(lista)  
  
[1, 2, 3, 4, 5]
```

## Część wspólna zbiorów

```
A = {1, 2, 3, 4, 5}
B = {3, 4, 5, 6, 7}

A.intersection(B) # część wspólna A i B
```

```
{3, 4, 5}
```

```
# lub krócej
A & B
```

```
{3, 4, 5}
```

## Część wspólna list

---

```
A = [1, 2, 3, 4, 5]
B = [3, 4, 5, 6, 7]

# rzutuje A i B na zbiory
# wyznacza ich część wspólną
# rzutuje na listę
wspolna = list(set(A) & set(B))

print(wspolna)
```

```
[3, 4, 5]
```

## frozense

---

- jak *set* ale *niemutowalny*, czyli nie można modyfikować zawartości

```
zamrozony_zbior = frozenset([1, 2, 3, 4, 5])
```

```
type(zamrozony_zbior)
```

```
frozenset
```

```
zamrozony_zbior.add(1) # nie można dodawać / usuwać
```

---

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-20-c3b7844219a4> in <module>()
----> 1 zamrozony_zbior.add(1) # nie można dodawać / usuwać

AttributeError: 'frozenset' object has no attribute 'add'
```

## Generatory

---

- funkcja, która zachowuje się jak iterator (czyli można np. wykorzystać ją w pętli)
- zamiast tworzyć całą listę obiektów, które będą przechowywane w pamięci
- oszczędza czas i pamięć

## Ciąg geometryczny

---

```
def geometryczny(a1, q, n):
    """Generuje n wyrazów ciągu geometrycznego."""

    ciag = [a1]

    for _ in range(n-1): # n-1 bo pierwszy już jest
        ciag.append(ciag[-1]*q) # następny = poprzedni * iloraz

    return ciag

ciag = geometryczny(1, 3, 10)

print(ciag)

[1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683]
```

## Ciąg geometryczny - generator

---

```
def gen_geometryczny(a1, q, n):
    """Generuje n wyrazów ciągu geometrycznego."""

    for _ in range(n):
        yield a1 # zwróć obecną wartość a1
        a1 *= q # i czekaj na kolejną iterację
```

## Generator a lista

---

```
ciag1 = geometryczny(1, 3, 10)      # zwraca listę
ciag2 = gen_geometryczny(1, 3, 10) # zwraca generator

print(ciag1)
```

```
[1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683]
```

```
print(ciag2)
```

```
<generator object gen_geometryczny at 0x7f9bac1ec678>
```

## Generator jak iterator

```
next(ciąg2) # pierwszy element
```

```
1
```

```
next(ciąg2) # kolejny element itd
```

```
3
```

```
# a najczęściej w pętli
for i in gen_geometryczny(1, 3, 10):
    print(i, end=', ')
```

```
1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683,
```

## Lista czy generator

- generator będzie z reguły szybszy
- generator może być nieskończony
- jednak nie można wykonywać operacji na "elementach", np. sortowania
- lista może być efektywniejsza, jeśli planujemy więcej pętli

## "Wyczerpanie" generatora

```
ciąg = gen_geometryczny(1, 3, 20) # generuje 20 wyrazów
```

```
for i in range(10): # wydrukuj 10 pierwszych
    print(next(ciąg), end=' ')
```

```
1 3 9 27 81 243 729 2187 6561 19683
```

```
for element in ciąg: # nie zaczyna od pierwszego!
    print(element, end=' ')
```

```
59049 177147 531441 1594323 4782969 14348907 43046721 129140163 387420489 1162261467
```

## Powtórka: listy składane (*list comprehensions*)

```
# chcemy stworzyć listę zawierającą
# kwadraty wszystkich cyfr
```

```
kwadraty = []
```

```
for x in range(10):
    kwadraty.append(x**2)
```

```
print(kwadraty)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
# a korzystając z listy składanej
```

```
kwadraty = [x**2 for x in range(10)]
```

```
print(kwadraty)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

## Append vs lista składana

---

```
def create_list(n=100000):
    """Tworzy listę kwadratów pierwszych n liczb naturalnych"""

    kwadraty = []

    for x in range(n):
        kwadraty.append(x**2)

    return kwadraty

def list_comprehension(n=100000):
    """Tworzy listę kwadratów pierwszych n liczb naturalnych"""

    return [x**2 for x in range(n)]
```

## Czas

---

```
timeit -n3 x = create_list() # lista tworzona przez append
```

```
3 loops, best of 3: 48.3 ms per loop
```

```
timeit -n3 y = list_comprehension() # lista składana
```

```
3 loops, best of 3: 42.5 ms per loop
```

```
create_list() == list_comprehension() # wynik ten sam
```

```
True
```

## Przykład

```
def force(m, a):
    """Zwraca wartość siły [N].
    Liczy siłę jaką należy zadziałać na ciało
    o masie m [kg], aby nadać mu przyspieszenie a [m/s^2].
    """
    return m*a

wagi = [10, 20, 30, 40, 50] # kg
przyspieszenia = [1, 2, 3, 4, 5] # m/s^2

# stwórz tablicę z wartościami sił

sily = [force(m, a) for m, a in zip(wagi, przyspieszenia)]

print(sily)
```

```
[10, 40, 90, 160, 250]
```

## Generator "jak lista składana"

```
# lista składana
list_comprehension = [x**2 for x in range(10)]

# generator expression
generator = (x**2 for x in range(10))

for x in generator:
    print(x, end=' ')
```

```
0 1 4 9 16 25 36 49 64 81
```

```
# lub prościej

for x in (x**2 for x in range(10)):
    print(x, end=' ')
```

```
0 1 4 9 16 25 36 49 64 81
```

## Uwagi na koniec

- unikaj tworzenia list przez *append*, jeśli można wykorzystać *list comprehension*
- unikaj tworzenia list jeśli jest to zbędne (użyj generatorów)
- unikaj *generator function* na rzecz *generator expression*