Notatki ze szkolenia w formie warsztatów w siedzibie firmy Sii we Wrocławiu

- Poziom podstawowy, 8h
- 26.05.2018
- Przygotował: Grzegorz Gwóźdź

O Pythonie

- Stworzony przez Guido van Rossum (https://en.wikipedia.org/wiki/Guido_van_Rossum)
- Nazwa od “Monty Python’s Flying Circus” (https://en.wikipedia.org/wiki/Monty_Python%27s_Flying_Circus)

Do czego można użyć?

- Automatyzacja zadań
- Prosty sposób na skomplikowane narzędzia
  - Przeszukiwanie wielu plików
  - Własna baza danych
  - Prototyp gierki
  - Aplikacja z interfejsem graficznym
- Rozszerzenia dla innych języków

Warto?

- Bogata biblioteka standardowa (I/O, sieć, obsługa plików, Tk)
- Posiada wiele wysoko poziomowych struktur danych
- Składnia Pythona jest czytelna, zasady PEP88
- Interpretowany - szybki development

Interpreter

Jest to program, który wykonuje napisany przez nas program instrukcja po instrukcji.

REPL

> **R** - read
> **E** - eval
> **P** - print
> **L** - loop

Uruchomienie

```
python3.6
```

```
Python 3.6.5 (default, Mar 30 2018, 15:54:24)
[GCC 4.2.1 Compatible Apple LLVM 9.0.0 (clang-900.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

UWAGA W wielu systemach domyślnym Pythonem jest 2.7 i komenda ```python``` uruchomi właśnie taką wersję.

UWAGA W Windowsie może zajść potrzeba ustawienia zmiennej *path*
``` set path=%path%;C:\python36 ```
```

Wyłączenie

```
Linux/Mac: Control+D
Windows: Control+Z

quit()
exit()
```

Znaki zachęty

primary prompt

```
>>> print('Hello')
Hello
>>>
```

secondary prompt

Kontynuacja

```
>>> print(
... 'Hello'
... )
Hello
>>>
```

In [1]:

```
print('Hello')
```

```
Hello
```

In [2]:

```
print()
```

Komenda z linii poleceń

```
python3 -c "print('Hello')"
```

Kod źródłowy

Domyślnie traktowany jako UTF-8.

Biblioteka standardowa używa jednak tylko ASCII.

Jeśli chcemy zastosować inne kodowanie niż domyślne, to pierwsza linia powinna zawierać

```
# -*- coding: encoding -*-
```

gdzie **encoding** to jeden ze standardów (https://docs.python.org/3/library/codecs.html#module-codecs) wspieranych przez Pythona.

Na przykład dla Windwos-1252:

```
# -*- coding: cp-1252 -*-
```

Pod Linuxem może być w drugiej linii, ze względu na "shebang"

```
#!/usr/bin/env python3
# -*- coding: cp-1252 -*-
```

Najważniejsze funkcje

`print()` i `help()`

PEP8

- 4 spacje na wcięcie - bez tabów
- 79 znaków
- puste linie do separacji klas i większych bloków kodu
- jeśli to możliwe, to komentarz w tej samej linii co kod
- używaj docstringów
- spacje dookoła operatorów i po przecinkach
- `CamelCase` dla klas, `lower_case_with_underscore` dla funkcji i metod. Zawsze używaj `self` jako pierwszego argumenty metody
- Kod źródłowy w UTF-8 lub ASCII

Komentarze

Jednolinijkowe

In [3]:

```
# komentarz
```

In [4]:

```
print('Hello')  # Wypisze Hello
```

```
Hello
```

Wielolinijkowe

In [5]:

```
'''

'''
```

Out[5]:

```
'\n\n'
```

Python jako kalkulator

Liczby

In [6]:

```
2 + 2
```

Out[6]:

```
4
```

In [7]:

```
2.0 + 2
```

Out[7]:

```
4.0
```

In [8]:

```
10 / 2
```

Out[8]:

```
5.0
```

In [9]:

```
10 // 2
```

Out[9]:

```
5
```

In [10]:

```
7 // 3
```

Out[10]:

```
2
```

In [11]:

```
7 % 3
```

Out[11]:

```
1
```

In [12]:

```
5.7 % 2.4
```

Out[12]:

```
0.9000000000000004
```

In [13]:

```
from decimal import Decimal
Decimal(5.7) % Decimal(2.4)
```

Out[13]:

```
Decimal('0.900000000000000035527136788 01')
```

In [14]:

```
2 ** 8
```

Out[14]:

```
256
```

In [15]:

```
width = 200
height = 5 * 9
width * height
```

Out[15]:

```
9000
```

In [16]:

```
4 * 3.75 - 1
```

Out[16]:

```
14.0
```

Tabelka z kolejnością operatorów (https://docs.python.org/3/reference/expressions.html#operator-precedence)

Ostatnie wyrażenie jest przypisane do zmiennej _, która jest tylko do odczytu (chociaż może zostać nadpisana). Ten znak ma też inne zastosowania:

- Ignorowanie pewnych wartości ("I don't care")
- Nadawanie funkcjom specjalnego znaczenia
- Do oddzielania cyfr w liczbach (np: `100_000_000`)

In [17]:

```
10 * 20
```

Out[17]:

```
200
```

In [18]:

```
_ / 5
```

Out[18]:

```
40.0
```

Oprócz `int` oraz `float` jest również `complex`

In [19]:

```
(1 + 2j) * (3 + 4J)
```

Out[19]:

```
(-5+10j)
```

Napisy

Dwa różne sposoby zapisu

- `"..."`
- `'...'`

In [20]:

```
"Hello"
```

Out[20]:

```
'Hello'
```

In [21]:

```
'World'
```

Out[21]:

```
'World'
```

In [22]:

```
"Hello 'World'"
```

Out[22]:

```
"Hello 'World'"
```

In [23]:

```
'Ala ma "kota"'
```

Out[23]:

```
'Ala ma "kota"'
```

In [24]:

```
'Pojedyncze \'apostrofy\''
```

Out[24]:

```
"Pojedyncze 'apostrofy'"
```

In [25]:

```
"Pierwsza linia\nDruga linia"
```

Out[25]:

```
'Pierwsza linia\nDruga linia'
```

In [26]:

```
print('C:\folder\nazwa\podkatalog')
```

```
C:older
azwa\podkatalog
```

In [27]:

```
a = '''To jest tekst wielolinijkowy
        białe znaki
zostają automatycznie załączone
'''
print(a)
```

```
To jest tekst wielolinijkowy
        białe znaki
zostają automatycznie załączone
```

In [28]:

```
'Ala' + ' ma ' + 'kota'
```

Out[28]:

```
'Ala ma kota'
```

In [29]:

```
'-' * 20
```

Out[29]:

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

In [30]:

```
'Napisy ' "obok " 'siebie ' 'są ' 'automatycznie ' 'łączone'
```

Out[30]:

```
'Napisy obok siebie są automatycznie łączone'
```

In [31]:

```
text = ('Napisy '
        "w kolumnie "
        'muszą zostać '
        'ujęte w niawiasy '
        'by mogły być '
        'automatycznie '
        'złączone')
print(text)
```

```
Napisy w kolumnie muszą zostać ujęte w niawiasy by mogły być automatycznie złączone
```

Indeksowanie

In [32]:

```
word = 'Python'
print(word[0])
print(word[-1])
```

```
P
n
```

In [33]:

```
word[0:2]
```

Out[33]:

```
'Py'
```

In [34]:

```
word[2:5]
```

Out[34]:

```
'tho'
```

In [35]:

```
word[0:2] + word[2:6]
```

Out[35]:

```
'Python'
```

In [36]:

```
word[:2] + word[2:]
```

Out[36]:

```
'Python'
```

In [37]:

```
word[-2:]
```

Out[37]:

```
'on'
```

In [38]:

```
word[-1:-20:-2]
```

Out[38]:

```
'nhy'
```

In [39]:

```
word[120:123]
```

Out[39]:

```
''
```

In [40]:

```
word[::-1]
```

Out[40]:

```
'nohtyP'
```

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

ZADANIE Napisz funkcję, która wypisze tabelkę jak powyżej dla zadanego tekstu w argumencie.

In [41]:

```
word[4:28]
```

Out[41]:

```
'on'
```

In [42]:

```
word[37:]
```

Out[42]:

```
''
```

In [44]:

```
word[0] = 'J'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-91a956888ca7> in <module>()
----> 1 word[0] = 'J'

TypeError: 'str' object does not support item assignment
```

Napisy w Pythonie są **immutable**

> [**immutable**](https://docs.python.org/3/glossary.html#term-immutable) An object with a fixed value. Immutable objects include numbers, strings and tuples. Such an object cannot be altered. A new object has to be created if a different value has to be stored. They play an important role in places where a constant hash value is needed, for example as a key in a dictionary.
> [**mutable**](https://docs.python.org/3/glossary.html#term-mutable) Mutable objects can change their value but keep their id()

In [45]:

```
'J' + word[1:]
```

Out[45]:

```
'Jython'
```

In [46]:

```
len(word)
```

Out[46]:

```
6
```

in, not in

In [47]:

```
'H' in 'Hello'
```

Out[47]:

```
True
```

In [48]:

```
'H' not in 'Hello'
```

Out[48]:

```
False
```

In [49]:

```
'Hell' in 'Hello'
```

Out[49]:

```
True
```

len, min, max

In [50]:

```
max('Hellasdasd')
```

Out[50]:

```
's'
```

In [51]:

```
len('Ala ma kota')
```

Out[51]:

```
11
```

index, count

In [52]:

```
'Hello'.index('l')
```

Out[52]:

```
2
```

In [53]:

```
w = 'Ala Ala ma kota Ala Al'
w.count('Ala')
```

Out[53]:

```
3
```

Operacje na stringach

capitalize, title

In [54]:

```
'hello world'.capitalize()
```

Out[54]:

```
'Hello world'
```

In [55]:

```
'hello world'.title()
```

Out[55]:

```
'Hello World'
```

lower, upper, casefold, swapcase

In [56]:

```
'HeLLS'.lower()
```

Out[56]:

```
'hells'
```

In [57]:

```
'HeLLS'.upper()
```

Out[57]:

```
'HELLS'
```

In [58]:

```
'Dreißig'.lower()
```

Out[58]:

```
'dreißig'
```

In [59]:

```
'Dreißig'.casefold()
```

Out[59]:

```
'dreissig'
```

In [60]:

```
'aaaBBBccc'.swapcase()
```

Out[60]:

```
'AAAbbbCCC'
```

`center`, `ljust`, `rjust`

In [61]:

```
'Ala'.center(20)
```

Out[61]:

```
'        Ala         '
```

In [62]:

```
'Ala'.ljust(20)
```

Out[62]:

```
'Ala                 '
```

In [63]:

```
'Ala'.rjust(20)
```

Out[63]:

```
'                 Ala'
```

`find`, `rfind`

In [64]:

```
'Ala ma kota'.find('ma')
```

Out[64]:

```
4
```

In [65]:

```
'Ala ma kota'.find('masz')
```

Out[65]:

```
-1
```

In [66]:

```
'ala ala ala'.find('ala')
```

Out[66]:

```
0
```

In [67]:

```
'ala ala ala'.rfind('ala')
```

Out[67]:

```
8
```

UWAGA Metoda ```find``` powinna być używana tylko do uzyskania pozycji szukanego napisu. Zawieranie natomiast sprawdza się za pomocą ```in```.

Formatowanie stringów - *formatted string literals* (https://docs.python.org/3/reference/lexical_analysis.html#formatted-string-literals)

- str.format
- f'Hello{}'

format, {{, {:b}, {1:<30}, {0:*^30}, {0:#x}

In [68]:

```
name = 'grzesiek'
age = 34
'Hello: {1!r} {0}'.format(name.title(), str(age))
```

Out[68]:

```
"Hello: '34' Grzesiek"
```

In [69]:

```
name = 'grzesiek'
age = 34
'Hello: {0:<30} {1}'.format(name.title(), str(age))
```

Out[69]:

```
'Hello: Grzesiek                       34'
```

In [70]:

```
name = 'grzesiek'
age = 34
'Hello: {0:*^30} {1}'.format(name.title(), str(age))
```

Out[70]:

```
'Hello: ***********Grzesiek********** 34'
```

In [71]:

```
name = 'grzesiek'
age = 34
'Hello: {0} {1:#x}'.format(name.title(), age)
```

Out[71]:

```
'Hello: Grzesiek 0x22'
```

In [72]:

```
name = 'grzesiek'
age = 34
'Hello: {1!r} {0}    {{     }}'.format(name.title(), str(age))
```

Out[72]:

```
"Hello: '34' Grzesiek    {     }"
```

In [73]:

```
name = 'grzesiek'
age = 34
'Hello: {0} {1:b}'.format(name.title(), age)
```

Out[73]:

```
'Hello: Grzesiek 100010'
```

In [74]:

```
name = 'grzesiek'
age = 34
greeting = f'Hello: {name} {age}'
print(greeting)
```

```
Hello: grzesiek 34
```

In [75]:

```
name = 'grzesiek'
age = 34
greeting = f'Hello: {name.title()} {age}'
print(greeting)
```

```
Hello: Grzesiek 34
```

Parametry formatu mogą być nazwane.

- `!s` - `str()`
- `!r` - `repr()`
- `!a` - `ascii()`

In [76]:

```
from datetime import datetime
today = datetime(year=2018, month=1, day=13)
print(today)
a = f'Dzisiaj jest {today.year}'
print(a)
```

```
2018-01-13 00:00:00
Dzisiaj jest 2018
```

In [77]:

```
data = {'name': 'Ala'}
a = f'Imie = {data["name"]}'
print(a)
```

```
Imie = Ala
```

In [78]:

```
for align, text in zip('<^>', ['left', 'center', 'right']):
    row = '{0:{fill}{align}16}'.format(text, fill=align, align=align)
    print(row)
```

```
left<<<<<<<<<<<<
^^^^^center^^^^^
>>>>>>>>>>>right
```

`islower`, `isalnum`, `istitle`

In [79]:

```
'Ala'.islower()
```

Out[79]:

```
False
```

In [80]:

```
'ala'.islower()
```

Out[80]:

```
True
```

In [81]:

```
'ala'.isalnum()
```

Out[81]:

```
True
```

In [82]:

```
'Park Jurajski'.istitle()
```

Out[82]:

```
True
```

replace

In [83]:

```
text = 'Ala ma kota'
text = text.replace('kota', 'psa')
print(text)
```

```
Ala ma psa
```

startswith, endswith

strip

In [84]:

```
'    Ala ma kota a '.strip()
```

Out[84]:

```
'Ala ma kota a'
```

zfill

In [85]:

```
'1'.zfill(2)
```

Out[85]:

```
'01'
```

In [86]:

```
'Ala'.center(30, '-')
```

Out[86]:

```
'-------------Ala--------------'
```

split, rsplit, splitlines

In [87]:

```
'Ala ma kota'.split()
```

Out[87]:

```
['Ala', 'ma', 'kota']
```

Listy

Może przechowywać różne typy danych.

In [88]:

```
ala = ['Ala', 'ma', 4, 'koty']
print(ala)
```

```
['Ala', 'ma', 4, 'koty']
```

In [89]:

```
len(ala)
```

Out[89]:

```
4
```

Indeksowanie, przekrój.

In [90]:

```
print(ala)
ala = ala[::-1]
print(ala)
```

```
['Ala', 'ma', 4, 'koty']
['koty', 4, 'ma', 'Ala']
```

In [91]:

```
ala[0] = 'magda'
ala[0] = ala[0].title()
print(ala)
```

```
['Magda', 4, 'ma', 'Ala']
```

Kopia, dokładanie, dodawanie, podmiana.

In [92]:

```
a = [1, 2, 3]
b = [1, 2, 3]
print(id(a) == id(b))
print(a == b)
print(a is b)
```

```
False
True
False
```

Usuwanie przekroju. Czyszczenie.

In [93]:

```
a = [1, 2, 3]
a[:] = []
print(a)
```

```
[]
```

In [94]:

```
a = [1, 2, 3]
a = []
print(a)
```

```
[]
```

In [95]:

```
a = [1, 2, 3]
a[0:2] = []
print(a)
```

```
[3]
```

In [96]:

```
a = [1, 2, 3]
a[0:2] = [5, 9, 2, 'sdsd']
print(a)
```

```
[5, 9, 2, 'sdsd', 3]
```

In [97]:

```
name = 'grzesiek'
age = 34
'Hello: {0:*^30} {1}'.format(name.title(), str(age))
```

Out[97]:

```
'Hello: ***********Grzesiek********** 34'
```

Zagnieżdżanie.

In [98]:

```
[[], [ 1, 3, 5], 434, 'asd']
```

Out[98]:

```
[[], [1, 3, 5], 434, 'asd']
```

Dynamiczny typ zmiennej

`type`

Konwersja typu.

Nieobecność wartości.

`str(None)`, a `int`?

In [99]:

```
int('FF', 16)
```

Out[99]:

```
255
```

Przypisywanie

> **UWAGA** Kopiowanie tylko dla prostych typów!

`id()`

Przypisanie kilku zmiennych naraz.

In [100]:

```
a, b = [1, 2, 3], [4, 5, 6]
print(a, b)
```

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

In [101]:

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

```
2 5
5 2
```

In [102]:

```
print('Hello', 'World', end='\n', sep='-')
print('Ala ma kota')
```

```
Hello-World
Ala ma kota
```

```
user_input = input()
```

Usuwanie zmiennych

In [103]:

```
n = 5
print(n)
del n
# Już nie ma n
# print(n)
```

5

Przepływ sterowania

Warunek `if`

Wcięcia na 4 spacje.

Warunkiem może być *coś* co daje w wyniku True lub False, jawnie lub niejawnie.

Łączenie warunków `and`, `or`, `not`.

In [104]:

```
2 == 2 and 2 != 3
```

Out[104]:

True

In [105]:

```
(2 == 2) and (2 != 3)
```

Out[105]:

True

In [106]:

```
False or True
```

Out[106]:

True

Kolejność operatorów (https://docs.python.org/3/reference/expressions.html#operator-precedence)

In [107]:

```
(True and 2) > (4 and True)
```

Out[107]:

True

In [108]:

```
True and 2 > 4 and True
```

Out[108]:

False

```
1 < 2 < 3
```

In [109]:

```
a = 4
2 < a < 5
```

Out[109]:

True

pass

In [110]:

```
age = 45

if age > 90:
    print('bardzo stary')
elif age > 60:
    print('stary')
elif age > 30:
    pass
    # print('prawie stary')
else:
    print('młodziutki')
```

Pętla `for`

string

In [111]:

```
for i in 'Ala ma kota':
    print(i)
```

```
A
l
a

m
a

k
o
t
a
```

list

In [112]:

```
jako_lista = list('Ala ma kota')
print(jako_lista)
for i in jako_lista:
    print(i)
```

```
['A', 'l', 'a', ' ', 'm', 'a', ' ', 'k', 'o', 't', 'a']
A
l
a

m
a

k
o
t
a
```

enumerate()

In [113]:

```
jako_lista = list('Ala ma kota')
print(jako_lista)

for x, i in enumerate(jako_lista):
    print(x, i)
```

```
['A', 'l', 'a', ' ', 'm', 'a', ' ', 'k', 'o', 't', 'a']
0 A
1 l
2 a
3
4 m
5 a
6
7 k
8 o
9 t
10 a
```

In [114]:

```
list(enumerate('Ala'))
```

Out[114]:

```
[(0, 'A'), (1, 'l'), (2, 'a')]
```

`continue`, `break`, `else`

In [115]:

```
for word in 'Ala ma dwa jasne dziwne koty'.split():

    if word == 'dwa':
        continue

    if word == 'dziwne':
        pass
        # break

    print(word)
else:
    print('Wszystko ok!')
    print('Cała pętla przeszła (nie było breaka)')
```

```
Ala
ma
jasne
dziwne
koty
Wszystko ok!
Cała pętla przeszła (nie było breaka)
```

Zakres `range()`

In [116]:

```
for i in range(10, -10, -2):
    print(i)
```

```
10
8
6
4
2
0
-2
-4
-6
-8
```

Range kontra lista

Pętla `while`

Operator potrójny

In [117]:

```
age = 10
name = 'Ala' if age == 0 else 'Karolina'
print(name)
```

```
Karolina
```

In [118]:

```
age = 10
name = 'Ala' if age == 0 else ('Karolina' if age == 1 else 'Magda')
print(name)
```

```
Magda
```

Funkcja

def, nazwa, nawiasy, dwukropek, wcięcie, docstring

In [119]:

```
def Funkcja():
    ''' Ta funkcja nie nie robi'''
    pass

print(Funkcja.__doc__)
```

```
 Ta funkcja nie nie robi
```

`global`, `nonlocal`

In [120]:

```
a_global = 7

def foo():
    a_global = 6  # local
    print('foo:', a_global)

    def inner():
        global a_global
        print('inner:', a_global)

    inner()

foo()
```

```
foo: 6
inner: 7
```

`return`

Bez wartości zwraca None. Funkcja bez returnu domyślnie ma `return None`.

In [121]:

```
def funkcja():
    '''
    Dokumentacja funkcji
    '''
    # return None

def f2():
    return None

print(funkcja.__doc__)

a = funkcja()
print(a)

print(funkcja() == f2())
```

```
    Dokumentacja funkcji

None
True
```

Argumenty funkcji

Domyślne wartości

Parametry nazwane (named parameters, pass-by-name, or keyword arguments).

kwargs

Pakowanie i rozpakowanie listy jako argumentów.

In [122]:

```
def func(a):
    pass

func(0, a=0)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-122-3dc92d657535> in <module>()
      2     pass
      3
----> 4 func(0, a=0)

TypeError: func() got multiple values for argument 'a'
```

```
def foo(a, *args, **kwargs)
```

Pakowanie / odpakowanie argumentów.

In [123]:

```
def func(*args):
    print(args)

a = [4, 5, 6]
func(a)   # func([4, 5, 6])
func(*a)  # func(4, 5, 6)
```

```
([4, 5, 6],)
(4, 5, 6)
```

In [124]:

```
def myprint(*args, **kwargs):
    sep = kwargs['sep'] if 'sep' in kwargs else ' '
    print(sep.join(args))

myprint('Hello', 'World', '!', sep='=')
myprint('Hello', 'World', '!')
```

```
Hello=World=!
Hello World !
```

In [125]:

```
def func(**kwargs):
    print(kwargs)

func(sep='')
```

```
{'sep': ''}
```

UWAGA Domyślna wartość ewaluowana jest tylko raz.

In [126]:

```
def func_bad(lista=[]):
    lista.append(5)
    return lista

numbers = func_bad()
liczby = func_bad()
cyfry = func_bad()

print(numbers)
print(liczby)
print(cyfry)
```

```
[5, 5, 5]
[5, 5, 5]
[5, 5, 5]
```

In [127]:

```
def func_good(lista=None):
    if lista == None:
        lista = []
    lista.append(5)
    return lista

numbers = func_good()
liczby = func_good()
cyfry = func_good()

print(numbers)
print(liczby)
print(cyfry)
```

```
[5]
[5]
[5]
```

In [128]:

```
numbers = func()

print(numbers)

liczby = func()

print(liczby)
```

```
{}
None
{}
None
```

Lambda - funkcja bez nazwy

```
lambda a, b: a + b
```

In [129]:

```
def make_incrementor(n):
    return lambda x: x + n

f = make_incrementor(7)
f(10)
```

Out[129]:

```
17
```

In [130]:

```
def make_incrementor(n):
    return lambda x, y: x + y + n

def incrementor_10(x, y):
    n = 10
    return x + y + n

# incrementor_10 = make_incrementor(10)
incrementor_20 = make_incrementor(20)
incrementor_30 = make_incrementor(30)

print(incrementor_10(3, 5))
print(incrementor_20(3, 5))
print(incrementor_30(3, 5))
```

```
18
28
38
```

In [131]:

```
pairs = [(0, 'zero'), (2, 'two'), (1, 'one'), (3, 'three')]
pairs.sort(key=lambda pair: pair[1])
pairs
```

Out[131]:

```
[(1, 'one'), (3, 'three'), (2, 'two'), (0, 'zero')]
```

UWAGA Poniżej łatwy do przeoczenia błąd.

In [132]:

```
def make_incrementors(n):
    lista = []
    for i in range(n):
        lista.append(lambda x: x + i)
    return lista

incrementors = make_incrementors(5)

for incr in incrementors:
    print(incr(0))
```

```
4
4
4
4
4
```

Naprawiamy

In [133]:

```
def make_incrementors(n):
    lista = []
    for i in range(n):
        def make_lambda(a):
            return lambda x: x + a
        lista.append(make_lambda(i))
    return lista

incrementors = make_incrementors(5)

for incr in incrementors:
    print(incr(0))
```

```
0
1
2
3
4
```

Przypisy - function annotations

In [134]:

```
def dodaj(a: int, b: int = 8) -> int:
    print(dodaj.__annotations__)
    print(dodaj.__doc__)
    return a + b

dodaj(3, 4)
```

```
{'a': <class 'int'>, 'b': <class 'int'>, 'return': <class 'int'>}
None
```

Out[134]:

```
7
```

Struktury danych

Lista - `list` (https://docs.python.org/3/library/stdtypes.html#list)

In [135]:

```
n = [2, 5, 7, 11, 13]
```

list.append(x)

Również `a[len(a):] = [x]`

In [136]:

```
n.append(100)
print(n)

n[len(n):] = [7, 5]
print(n)
```

```
[2, 5, 7, 11, 13, 100]
[2, 5, 7, 11, 13, 100, 7, 5]
```

list.extend(iterable)

In [137]:

```
numbers = [3, 14]
numbers.extend([1, 2, 3])
print(numbers)
```

```
[3, 14, 1, 2, 3]
```

list.insert(i, x)

In [138]:

```
numbers = [1, 2, 3]
numbers.insert(1, 37)
print(numbers)
```

```
[1, 37, 2, 3]
```

list.remove(x)

In [139]:

```
numbers = [1, 2, 3]
numbers.remove(1)
print(numbers)
```

```
[2, 3]
```

list.pop([index])

In [140]:

```
numbers = [1, 2, 3, 4, 5, 6]
print(numbers)
numbers.pop()
print(numbers)
numbers.pop(3)
print(numbers)
```

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

list.clear() lub del a[:]

In [141]:

```
numbers = [1, 2, 3, 4, 5]
numbers.clear()
print(numbers)
```

```
[]
```

In [142]:

```
numbers = [1, 2, 3, 4, 5]
del numbers[:]
print(numbers)
```

```
[]
```

In [143]:

```
numbers = [6, 12, 34, 45, 85]
del numbers[2:3]
print(numbers)
```

```
[6, 12, 45, 85]
```

list.index(x[, start[, end]])

In [144]:

```
numbers = [6, 12, 34, 45, 85]
index = numbers.index(34)
print(index)
```

```
2
```

list.count(x)

In [145]:

```
numbers = [6, 12, 7, 34, 45, 85, 7, 77]
count_7 = numbers.count(7)
print(count_7)
```

```
2
```

list.sort(key=None, reverse=False)

In [146]:

```
numbers = [6, 12, 7, 34, 45, 85, 7, 77]
numbers.sort()  # *IN PLACE*
print(numbers)
```

```
[6, 7, 7, 12, 34, 45, 77, 85]
```

list.reverse()

In [147]:

```
numbers = [6, 12, 7, 34, 45, 85, 7, 77]
numbers.reverse()  # *IN PLACE*
print(numbers)
```

```
[77, 7, 85, 45, 34, 7, 12, 6]
```

list.copy()

In [148]:

```
fruits = ['apple', 'orange', 'dog']
fruits_copy = fruits.copy()
print(fruits == fruits_copy)  # takie same wartosci
print(fruits is fruits_copy)  # rozne obiekty
```

```
True
False
```

Za pomocą listy można zaimplementować stos (```append()```, ```pop()```). Również kolejkę, ale nie warto. **Dlaczego?**

len, sum, min, max

In [149]:

```
numbers = [3, 6, 3, 9, 12, 4, 9, 2, 102, 43, 1,]
print('len(numbers) =', len(numbers))
print('sum(numbers) =', sum(numbers))
print('min(numbers) =', min(numbers))
print('max(numbers) =', max(numbers))
```

```
len(numbers) = 11
sum(numbers) = 194
min(numbers) = 1
max(numbers) = 102
```

Test na zawieranie in.

In [150]:

```
if 10 in [3, 4, 5, 10]:
    print('tak')
```

```
tak
```

List Comprehensions

```
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- Przykład: squares za pomocą for

In [151]:

```
numbers = []
for n in range(10):
    numbers.append(n * n)
print(numbers)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- Przykład: squares za pomocą *list comprehension*

In [152]:

```
numbers = [x * x for x in range(10)]
print(numbers)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Więcej pętli i warunków można stosować.

In [153]:

```
letters = [x for x in 'Python' for n in range(4)]
print(letters)
```

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

Lista krotek.

In [154]:

```
pairs = [(x, n) for x in 'Python' for n in range(2)]
print(pairs)
```

```
[('P', 0), ('P', 1), ('y', 0), ('y', 1), ('t', 0), ('t', 1), ('h', 0), ('h', 1), ('o', 0), ('o'
, 1), ('n', 0), ('n', 1)]
```

Zagnieżdżanie *List Comprehensions*

In [155]:

```
>>> matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12]
]

print(matrix)
```

```
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

Transpozycja matrix

In [156]:

```
[[row[i] for row in matrix] for i in range(4)]
```

Out[156]:

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Analiza transpozycji

In [157]:

```
transposed = []
for i in range(4):
    transposed.append([row[i] for row in matrix])
transposed
```

Out[157]:

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

In [158]:

```
transposed = []
for i in range(4):
    transposed_row = []
    for row in matrix:
        transposed_row.append(row[i])
    transposed.append(transposed_row)

transposed
```

Out[158]:

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

In [159]:

```
list(zip(*matrix))
```

Out[159]:

```
[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```

`str.split()` i `str.join(list)`

In [160]:

```
'-'.join(['Ala', 'ma', 'kota'])
```

Out[160]:

```
'Ala-ma-kota'
```

In [161]:

```
'-'.join([str(n ** 2) for n in range(100) if n % 7 == 0])
```

Out[161]:

```
'0-49-196-441-784-1225-1764-2401-3136-3969-4900-5929-7056-8281-9604'
```

Krotka - [`tuple`](https://docs.python.org/3/library/stdtypes.html#tuple)

Wartości oddzielone przecinkiem.

In [162]:

```
a = 2, 3, 4
print(type(a))
```

```
<class 'tuple'>
```

Zagnieżdżanie.

In [163]:

```
a = (1, 2, 3, (4, 9))
```

Nie można modyfikować.

In [164]:

```
a = (1, 2, 3, (4, 9))
a[2] = 4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-164-687256216588> in <module>()
      1 a = (1, 2, 3, (4, 9))
----> 2 a[2] = 4

TypeError: 'tuple' object does not support item assignment
```

Może zawierać zmienne (modyfikowalne) obiekty.

In [165]:

```
a = ([21, 29], 'Ala ma kota')
a[0][1] = 87
print(a)
```

```
([21, 87], 'Ala ma kota')
```

Konstrukcja z jednym elementem lub bez.

In [166]:

```
a = ()
b =(2,)
print(a, b)
```

```
() (2,)
```

Rozpakowywanie.

In [167]:

```
a, b, c = (5, 3, 1)
print(a, b, c)
```

```
5 3 1
```

In [168]:

```
a, b, *c = (5, 3, 1, 43, 12, 32)
print(a)
print(b)
print(c)
```

```
5
3
[1, 43, 12, 32]
```

Szczególny przypadek rozpakowania - swap.

In [169]:

```
a = 2
b = 7
print(a, b)
a, b = b, a
print(a, b)
```

```
2 7
7 2
```

Zbiór - set (https://docs.python.org/3/library/stdtypes.html#set)

Nieuporządkowany. Bez duplikatów.

- Suma |
- Część wspólna &
- Różnica -
- Różnica symetryczna ^

In [170]:

```
a = set('abracadabra')
b = set('alacazam')
print(a)
print(b)
```

```
{'c', 'a', 'd', 'r', 'b'}
{'c', 'a', 'm', 'z', 'l'}
```

`set()`, ale nie {}

In [171]:

```
zbior = set()
slownik = {}

print(type(zbior))
print(type(slownik))
```

```
<class 'set'>
<class 'dict'>
```

`in`

In [172]:

```
letters = set('Ala ma kota')
'm' in letters
```

Out[172]:

```
True
```

Słownik - [`dict` (https://docs.python.org/3/library/stdtypes.html#mapping-types-dict)](https://docs.python.org/3/library/stdtypes.html#mapping-types-dict)

Tablica asocjacyjna.

Indeksowany kluczami (immutable). Co może być kluczem? Według dokumentacji "*(...) it needs a __hash__() method (...)*"

Nieuporządkowany zbiór **klucz: wartość**.

`list(dict)`

In [173]:

```
people = dict()
people['Grzesiek'] = 34
people['Sheldon'] = 27
people['Morty'] = 16
print(list(people))
```

```
['Grzesiek', 'Sheldon', 'Morty']
```

Krotki mogą być kluczami jeśli zawierają tylko stringi, liczby lub krotki. Jeśli zawierają choć jeden element *mutable* nie może zostać użyte jako klucz.

`del`

In [174]:

```
people = dict()
people['Grzesiek'] = 34
people['Sheldon'] = 27
people['Morty'] = 16
print(people)
del people['Grzesiek']
print(people)
```

```
{'Grzesiek': 34, 'Sheldon': 27, 'Morty': 16}
{'Sheldon': 27, 'Morty': 16}
```

`in` - testuje obecność klucza

Budowanie słownika z sekwencji par **klucz: wartość** lub tak jak *List Comprehensions*.

In [175]:

```
people = (('Grzesiek', 34), ('Sheldon', 27), ('Morty', 16))
dict(people)
```

Out[175]:

```
{'Grzesiek': 34, 'Sheldon': 27, 'Morty': 16}
```

Tworzenie `set`, `dict` oraz `tuple` tak jak `list` za pomocą *List Comprehensions*.

In [176]:

```
lista = [x for x in range(10)]
to_nie_krotka = (x for x in range(10))
zbior = {x for x in range(10)}
slownik = {x: x for x in range(10)}

print('lista         →', lista)
print('to_nie_krotka →', to_nie_krotka)
print('zbior         →', zbior)
print('slownik       →', slownik)
```

```
lista         → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
to_nie_krotka → <generator object <genexpr> at 0x7f59a8272888>
zbior         → {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
slownik       → {0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9}
```

Moduły

- Podział na pliki dla łatwiejszego utrzymania
- Wykorzystanie funkcji w kilku programach
- Można importować moduł w module

`__name__`, `main`

Przykład `>>> import fibo`

```
from xxx import yyy, zzz
```

```
import xxx as yyy
```

Moduł może zawierać wyrażenia, które zostaną wykonanie podczas importu.

Każdy moduł ma swoją tablicę symboli.

`from xxx import *` - importuje wszystko oprócz symboli zaczynających się od _. Nie stosuje się tego ze względu na ryzyko przysłonienia symboli, które zostały już zdefiniowane.

> **UWAGA** Moduł ładowany jest tylko raz.

```
import importlib; importlib.reload(modulename)
```

Moduł jako skrypt: `python3 script.py`

`__pycache___`

`sys.ps1`

`sys.path.append`

Komenda `dir()` nie pokaże wbudowanych funkcji. Trzeba użyć `import builtins; dir(builtins)`.

package - kolekcja modułów.

`from package import item`

- `item` może być submodułem, funkcją, klasą, zmienną

`import item.subitem.subsubitem`

- wszystkie oprócz ostatniego muszą być paczką
- ostatni może być modułem lub paczką, ale nie może być klasą, funkcją lub zmienną z poprzedniej pozycji

Paczka to folder z plikiem `__init__.py`

- `__init__.py` może być pusty
- `__all__`, `from package import *`
- `__path__` - nazwa folderu, w którym jest `__init__.py`. Można modyfikować.

Relatywne importy

- `from . imoprt xxx`
- `from .. import xxx`
- `from ../yyy import xxx`

`__main__.py`

`python3 -m http.server`

`python3 -m json.tool`

Operacje I/O

Odczyt i zapis do pliku

`open`, `read`, `seek`, `tell`, `readline`, `readlines`, `write`, `close`, `with`

In [177]:

```
plik = open('/tmp/plik.txt', 'w')
plik.write('Ala')
plik.write('ma')
plik.write('kota')
plik.close()
```

In [178]:

```
plik = open('/tmp/plik.txt')
print('dane =', plik.read())
print('dane =', plik.read())
plik.seek(0)
print('dane =', plik.read())
plik.seek(2)
print('dane =', plik.read(3))
print('tell =', plik.tell())

plik.close()
```

```
dane = Alamakota
dane =
dane = Alamakota
dane = ama
tell = 5
```

`with` sam zadba o zamknięcie pliku po opuszczeniu jego bloku

In [179]:

```
with open('/tmp/plik.txt') as f:
    print(f.read())
```

```
Alamakota
```

ZADANIE Policz ile razy każdy wyraz występuje w pliku. Który występuje najczęściej?

In [180]:

```
import urllib.request
import os
import itertools
import operator
import json

def get_reading(url):
    response = urllib.request.urlopen(url)
    data = response.read()
    text = data.decode('utf-8')
    return text

def get_winner(url):
    text = get_reading(url)

    words = text.split()
    histogram = dict(zip(set(words), itertools.repeat(0)))
    for word in words:
        histogram[word] += 1

    pairs = histogram.items()
    winner = max(pairs, key=operator.itemgetter(1))
    return winner

url_pan_tadeusz = 'https://wolnelektury.pl/media/book/txt/pan-tadeusz.txt'
url_zemsta = 'https://wolnelektury.pl/media/book/txt/zemsta.txt'
url_w_pustyni = 'https://wolnelektury.pl/media/book/txt/w-pustyni-i-w-puszczy.txt'

print(get_winner(url_pan_tadeusz))
print(get_winner(url_zemsta))
print(get_winner(url_w_pustyni))
```

```
('w', 1496)
('/', 806)
('i', 3827)
```

ZADANIE Różnica symetryczna słów z pięciu wybranych lektur.

ZADANIE Z GWIAZDKĄ Przetłumacz Pana Tadeusza na wybrany język.

Błędy i wyjątki

Błędy parsowania

In [181]:

```
while True print('Hello')
```

```
  File "<ipython-input-181-f796159d33e4>", line 1
    while True print('Hello')
                     ^
SyntaxError: invalid syntax
```

Wyjątki

In [182]:

```
1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-182-42c5125e8846> in <module>()
----> 1 1/0

ZeroDivisionError: division by zero
```

try, except

Wszystkie lub konretne.

In [183]:

```
try:
    1/0
except:
    pass
```

Własne wyjątki

In [184]:

```
class B(Exception):
    pass

class C(B):
    pass

class D(C):
    pass

for cls in [B, C, D]:
    try:
        raise cls()
    except D:
        print("D")
    except C:
        print("C")
    except B:
        print("B")
```

```
B
C
D
```

In [185]:

```
class X(Exception):
    pass

class Y(Exception):
    pass

class Z(Y):
    pass

try:
    raise Z()
except Y:
    print('mam')
except:
    print('nie mam')
```

```
mam
```

Przykład: Zapis do pliku

In [186]:

```
import sys

try:
    f = open('/tmp/asdadeasdf4rfsdfs.txt')
    s = f.readline()
    i = int(s.strip())
    f.close()
except:
    print('Brak pliku')
```

```
Brak pliku
```

Rzucanie wyjątków

In [187]:

```
class MyException(Exception): pass

try:
    1/0
    raise MyException()
except ZeroDivisionError:
    pass
```

else, finally

In [188]:

```
try:
    1/0
except:
    print('except')
else:
    print('else')
finally:
    print('finally! (always)')
```

```
except
finally! (always)
```

In [189]:

```
try:
    1/1
except:
    print('except')
else:
    print('else')
finally:
    print('finally! (always)')
```

```
else
finally! (always)
```

Klasy, obiektowość

- Połączenie danych i funkcji
- Klasa, instancja, obiekt
- Dziedziczenie
- Przeciążanie metod
- Dostęp do metod klasy bazowej
- Mogą być modyfikowane podczas działania programu
- Pierwszy argument metody
- Operatory mogą być przedefiniowane

__init__()

In [190]:

```
class MyClass:

    def __init__(self, value=0):
        self.value = value

    def f(self):
        pass

obj = MyClass()  # MyClass.__init__(self)
print(obj.value)
```

```
0
```

Atrybuty klasy i instancji

Przykład "Dog and tricks"

In [191]:

```
class Dog:
    tricks = []

    def __init__(self, name):
        self.name = name
        self.tricks = Dog.tricks  # Uwaga!
        # self.tricks = Dog.tricks.copy()

    def add_trick(self, trick):
        self.tricks.append(trick)

sheldon = Dog('Sheldon')
stevo = Dog('Stevo')

sheldon.add_trick('Roll')

print(sheldon.tricks)
print(stevo.tricks)
```

```
['Roll']
['Roll']
```

isinstance()

obj.__class__

In [192]:

```
def add(a, b):
    if isinstance(a, int) and isinstance(b, int):
        return a + b
    return 0

print(add('1', 2))
```

```
0
```

Dziedziczenie

In [193]:

```
class Base:
    def hello(self):
        print('Base: Hello')
        self.bye()

    def bye(self):
        print('Base: Bye bye', type(t))

class Derived(Base):
    def bye(self):
        print('Derived: Bye bye')

base = Base()
derived = Derived()

derived.hello()
```

```
Base: Hello
Derived: Bye bye
```

issubclass()

In [194]:

```
class A: pass

class Z: pass

class B(A, Z): pass

print(issubclass(A, Z))
```

```
False
```

Prywatne zmienne

`__var` → `_classname__var`

In [195]:

```
class Dog:
    def __init__(self):
        self.__tricks = ['Roll']

dog = Dog()
# nie zadziala: print(dog.__trickcs)
print(dog._Dog__tricks)
```

```
['Roll']
```

Introspekcja

`dir`, `type`, `id`, `inspect`

In [196]:

```
class Dog:

    def __init__(self):
        self.__tricks = ['Roll']

dog = Dog()
print(dir(dog))
print(str(type(dog)))
```

```
['_Dog__tricks', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__for
mat__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '_
_le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
<class '__main__.Dog'>
```

In [197]:

```
import inspect
inspect.getmembers(int)

class Dog:

    def __init__(self):
        self.__tricks = ['Roll']


dog = Dog()
inspect.getmembers(dog)
```

Out[197]:

```
[('_Dog__tricks', ['Roll']),
 ('__class__', __main__.Dog),
 ('__delattr__',
  <method-wrapper '__delattr__' of Dog object at 0x7f59a816dda0>),
 ('__dict__', {'_Dog__tricks': ['Roll']}),
 ('__dir__', <function Dog.__dir__>),
 ('__doc__', None),
 ('__eq__', <method-wrapper '__eq__' of Dog object at 0x7f59a816dda0>),
 ('__format__', <function Dog.__format__>),
 ('__ge__', <method-wrapper '__ge__' of Dog object at 0x7f59a816dda0>),
 ('__getattribute__',
  <method-wrapper '__getattribute__' of Dog object at 0x7f59a816dda0>),
 ('__gt__', <method-wrapper '__gt__' of Dog object at 0x7f59a816dda0>),
 ('__hash__', <method-wrapper '__hash__' of Dog object at 0x7f59a816dda0>),
 ('__init__',
  <bound method Dog.__init__ of <__main__.Dog object at 0x7f59a816dda0>>),
 ('__init_subclass__', <function Dog.__init_subclass__>),
 ('__le__', <method-wrapper '__le__' of Dog object at 0x7f59a816dda0>),
 ('__lt__', <method-wrapper '__lt__' of Dog object at 0x7f59a816dda0>),
 ('__module__', '__main__'),
 ('__ne__', <method-wrapper '__ne__' of Dog object at 0x7f59a816dda0>),
 ('__new__', <function object.__new__(*args, **kwargs)>),
 ('__reduce__', <function Dog.__reduce__>),
 ('__reduce_ex__', <function Dog.__reduce_ex__>),
 ('__repr__', <method-wrapper '__repr__' of Dog object at 0x7f59a816dda0>),
 ('__setattr__',
  <method-wrapper '__setattr__' of Dog object at 0x7f59a816dda0>),
 ('__sizeof__', <function Dog.__sizeof__>),
 ('__str__', <method-wrapper '__str__' of Dog object at 0x7f59a816dda0>),
 ('__subclasshook__', <function Dog.__subclasshook__>),
 ('__weakref__', None)]
```

Wycieczka po bibliotece standardowej

os

In [198]:

```
import os
os.getcwd()
os.chdir('/tmp')
os.system('echo hello')
```

Out[198]:

```
0
```

pathlib

In [199]:

```
from pathlib import Path
root = Path('/root')
home = Path('home')
user = Path('gregory')

homedir = root / home / user
print(homedir)
```

```
/root/home/gregory
```

shutil

In [200]:

```
import shutil
shutil.copyfile('/tmp/a.txt', '/tmp/b.txt')

# NIE:
# import os
# os.system('cp /tmp/a.txt /tmp/b.txt')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-200-742f3ab3bd08> in <module>()
      1 import shutil
----> 2 shutil.copyfile('/tmp/a.txt', '/tmp/b.txt')
      3
      4 # NIE:
      5 # import os

/usr/lib/python3.6/shutil.py in copyfile(src, dst, follow_symlinks)
    118         os.symlink(os.readlink(src), dst)
    119     else:
--> 120         with open(src, 'rb') as fsrc:
    121             with open(dst, 'wb') as fdst:
    122                 copyfileobj(fsrc, fdst)

FileNotFoundError: [Errno 2] No such file or directory: '/tmp/a.txt'
```

glob

In [201]:

```
import glob
glob.glob('*.py')
```

Out[201]:

```
[]
```

sys

In [202]:

```
import sys

print(sys.argv)
```

```
['/home/gregory/python_workshops/.env/lib/python3.6/site-packages/ipykernel_launcher.py', '-f',
'/run/user/1000/jupyter/kernel-b5a0b05c-9606-4bbc-bf47-3d61c0bd7bc2.json']
```

re

In [203]:

```
import re
re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
```

Out[203]:

```
['foot', 'fell', 'fastest']
```

In [204]:

```
import re
re.sub(r'(\b[a-z]+) \1', r'\1','cat in the the hat')
```

Out[204]:

```
'cat in the hat'
```

commonregex (https://github.com/madisonmay/CommonRegex) - zbiór najczęściej używanych (potrzebnych) wyrażeń regularnych.

math

Sufit, podłoga, silnia, największy wspólny dzielnik.

Stopnie → radiany, radiany → stopnie.

Funkcje trygonometryczne.

In [205]:

```
import math
math.cos(math.pi / 4)
```

Out[205]:

```
0.7071067811865476
```

In [206]:

```
import math
math.sqrt(25)
math.degrees(math.pi / 4)
```

Out[206]:

```
45.0
```

random

In [207]:

```
import random
random.random()
```

Out[207]:

```
0.6440811768135081
```

In [208]:

```
import random
text = 'Ala ma dwa koty'
words = text.split()
word = random.choice(words)
print(word)
```

```
koty
```

statistics

In [209]:

```
import statistics
statistics.mean([1,2,3,4])
```

Out[209]:

```
2.5
```

In [210]:

```
from decimal import Decimal

a = Decimal(5.4)
b = Decimal(2.3)

a % b
```

Out[210]:

```
Decimal('0.80000000000000007105427357601')
```

urllib

In [ ]:

```
import urllib.request
data = urllib.request.urlopen('http://www.python.org').read()
data = data.decode()
print(data)
```

datetime

In [212]:

```
from datetime import date
birthday = date(1984, 8, 16)
now = date.today()
now = date(2018, 6, 4)
print('Mam', (now - birthday).days, 'dni')
```

```
Mam 12345 dni
```

Zadanie: Kiedy będę miał 12345 "dniowe urodziny"?

zlib

In [213]:

```
import zlib

text = 'Ala ma kota Ala ma kota Ala ma kota'
btext = text.encode()
ctext = zlib.compress(btext)
print(len(ctext), ctext)
print(len(text), text)

# zapis
with open('/tmp/text.zip', 'wb') as f:
    f.write(ctext)

# odczyt
with open('/tmp/text.zip', 'rb') as f:
    text = zlib.decompress(f.read()).decode()
    print(text)
```

```
23 b'x\x9cs\xccIT\xc8MT\xc8\xce/ITp\xc4\xce\x06\x00\xcd\xdc\x0b\xa2'
35 Ala ma kota Ala ma kota Ala ma kota
Ala ma kota Ala ma kota Ala ma kota
```

timeit

In [214]:

```
from timeit import Timer
Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
```

Out[214]:

```
0.027080390000264742
```

In [215]:

```
from timeit import Timer
Timer('a,b = b,a', 'a=1; b=2').timeit()
```

Out[215]:

```
0.027105746999950497
```

doctest

In [216]:

```
def suma(a, b):
    """Oblicza sumę dwóch liczb

    >>> suma(4, 5)
    9
    """
    return a + b

import doctest
doctest.testmod()
```

Out[216]:

```
TestResults(failed=0, attempted=1)
```

xmlrpc

By poniższy przykład działał trzeba uruchmić również
`python3 -m xmlrpc.server`

In [227]:

```
from xmlrpc.client import ServerProxy as sp
proxy = sp('http://127.0.0.1:8000')
proxy.getData()
```

Out[227]:

```
'42'
```

In [228]:

```
proxy.pow(2, 3)
```

Out[228]:

```
8
```

In [229]:

```
proxy.add(5, 6)
```

Out[229]:

```
11
```

pprint

In [220]:

```
from pprint import pprint as pp

pp({'A': 4})
```

```
{'A': 4}
```

textwrap

In [221]:

```
import textwrap
doc = """The wrap() method is just like fill() except that it returns
... a list of strings instead of one big string with newlines to separate
... the wrapped lines."""
print(textwrap.fill(doc, width=40))
```

```
The wrap() method is just like fill()
except that it returns ... a list of
strings instead of one big string with
newlines to separate ... the wrapped
lines.
```

time

In [222]:

```
import time
a = time.time()
print('Hello')
print(time.time() - a)
```

```
Hello
0.0001323223114013672
```

logging

In [223]:

```
import logging
logging.debug('Debug')
logging.info('Info')
logging.warning('Warning')
logging.error('Error')
logging.critical('Critical')
```

```
WARNING:root:Warning
ERROR:root:Error
CRITICAL:root:Critical
```

collections

In [224]:

```
from collections import deque
dq = deque()
dq.append(1)
dq.append(2)
dq.append(3)
dq.appendleft(4)
dq
```

Out[224]:

```
deque([4, 1, 2, 3])
```

fractions

In [225]:

```
from fractions import Fraction
a = Fraction(1, 2)
b = Fraction(4, 9)
result = a + b
print(result)
print(Fraction(0.02314))
```

```
17/18
3334825452075305/144115188075855872
```

Wirtualne środowisko

```
python3 -m venv nazwa
```

```
$ source bin/activate
```

Przykład tworzenia projektu razem z *env* (Linuks)

Przykładowy projekt, który korzysta z dwóch dodatkowych zewnętrznych bibliotek

```
$ mkdir projekt
$ cd projekt
$ python3 -m venv .env
$ source .env/bin/activate
$ pip3 install commonregex
$ pip3 install pygame
```

"Mrozimy" zainstalowane zależności

```
$ pip3 freeze > requirements.txt
```

Plik **requirements.txt** trzymamy w gicie razem z projektem. Natomiast **.env** nie potrzebujemy. Gdy potem "na świeżo" sklonujemy projekt z serwera, wystarczy:

```
$ cd projekt
$ python3 -m venv .env
$ source .env/bin/activate
$ pip3 install -r requirements.txt
```

W ten sposób zostaną zainstalowane zależności w odpowiednich wersjach. W takich w jakich zaczęliśmy pisać projekt.