Wizualizacja danych

Wykład 1

Sprawy organizacyjne

Sprawy organizacyjne

- Sylabus jest dostępny w systemie USOS.
- Regulamin zajęć dostępny jest na stronie prowadzącego zajęcia http://wmii.uwm.edu.pl/~piojas/.
- Forma zaliczenia: egzamin.
- Wykład - 10 godzin.
- https://github.com/pjastr/WizualizacjaDanychNStac2023

Termin egzaminów

- pierwszy – 17.06.2023,
- drugi – 9.09.2023,
- trzeci – 16.09.2023.

Wymagania wstępne

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

Ewentualne braki należy opanować w samodzielnym zakresie.

W razie problemów zapraszam na konsultacje.

Wstęp do języka Python

Język Python

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

Przełomowy rok - 2008

- Utworzenie drugiej gałęzi rozwoju 3.x. Początkowe obie gałęzie były rozwijane niezależnie, lecz wsparcie Pythona 2.x zostało zakończone w roku 2020.
- Python 2.x cały czas jest wykorzystywany np. w ArcGis Desktop https://support.esri.com/en/technical-article/000013224

Podstawowe różnice między 2.x a 3.x

- funkcja `print`

Python 2:

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

Python 3

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

Dzielenie zmiennych typu `int`

Python 2:

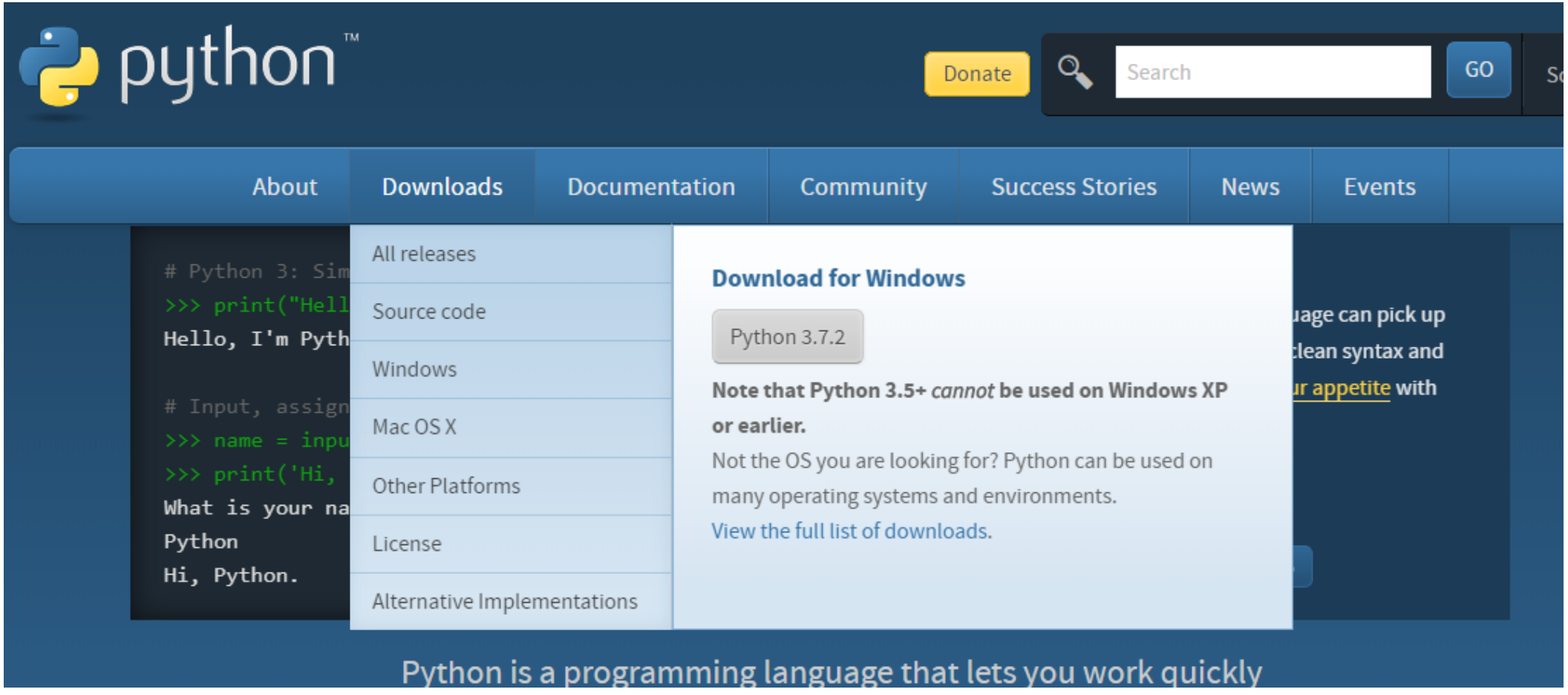
```
1 print '3 / 2 =', 3 / 2
2 print '3 // 2 =', 3 // 2
3 print '3 / 2.0 =', 3 / 2.0
4 print '3 // 2.0 =', 3 // 2.0
```

Python 3

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

Instalacja - Windows

- https://python.org/



Linux

Sprawdzenie wersji na Ubuntu 18.04:

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

Ubuntu

Ręczna instalacja:

```
1  sudo apt install python3
```

Wybór IDE do Pythona

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

Styl PEP8

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

Znaki odstępu:

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

Puste linie:

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

Kodowanie:

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

Stringi:

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

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

Jak sprawdzać styl PEP8?

- Pycharm - Code - Inspect Code
- moduł flake8

Różnice - Editor/Code Style/Python

Cechy języka Python

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

Zen

```
1 import this
```

```
The Zen of Python, by Tim Peters

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

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

Zmienna

- najprościej: przechowuje pewną wartość:

```
1  x = 5
2  y = "John"
```

Wbudowane typy danych

- numeryczne (liczbowe): int, float, complex
- tekstowe: str
- sekwencji: list, tuple
- odwzorowania (mapping type): dict
- zestawów (set types): set, frozenset
- logiczne: bool
- binarne: bytes, bytearray

`type hinting` - wskazywanie typów

Nie będzie używane na wykładzie.

Przykład:

```
1 a: int = 5
2 print(a)
3 a = "Olsztyn"
4 print(a)
```

```
5
Olsztyn
```

Int - typ całkowity

- bez kropki dziesiętnej
- może być dowolnie długi (ograniczenie ilość pamięci)

```
1 print(123123123123123123123123123123123123123 + 1)
```

```
123123123123123123123123123123123123124
```

Jaki system liczbowy?

Domyślnie dziesiętny. Więcej za kilka zajęć przy algorytmach liczbowych.

```
1 print(101)
2 print(101)
3 print(0x101) # zero-x
4 print(0o101) # zero-litera o
5 print(0b101) # zero-b
6 print(0X101) # zero-x
7 print(0O101) # zero-litera o
8 print(0B101) # zero-b
```

```
101
101
257
65
5
257
65
5
```

Sprawdzenie typu

```
1 print(type(234))
```

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

`<class 'int'>` - wszystko jest obiektem

Float - typ zmiennoprzecinkowy

```
 1 print(4.2)
 2 print(4.)
 3 print(.5)
 4 print(.3e4)
 5 print(3e4)
 6 print(2e-2)
 7 print(1.79e308)
 8 print(1.8e308)
 9 print(5e-324)
10 print(1e-325)
```

```
4.2
4.0
0.5
3000.0
30000.0
0.02
1.79e+308
inf
5e-324
0.0
```

Complex - typ zespolony (dot. liczb zespolonych)

```
1  print(2+3j)
2  print(2+5*1j)
```

```
(2+3j)
(2+5j)
```

Uwaga: wyrażenie

```
1  print(4+5*j)
```

skutkuje błędem w większości wypadków.

str - string, napisy, łańcuchy znaków

Obecnie odchodzi się od określenia “tablica znaków”

```
1 print("ABC")
2 print('abc')
```

```
ABC
abc
```

bool - typ logiczny

```
1 print(True)
2 print(False)
```

```
True
False
```

Operator przypisania

= przypisuje prawą stronę do lewej (!), często mylony z operatorem logicznym równa się ==

```
1  x = 5
2  x = "Piotr"
3  a = 4.5
4  A = 56
5  x, y, z = "Orange", "Banana", "Cherry"
6  x = y = z = "Orange"
```

input - operacja wejścia

```
1 num = input ("Wprowadź liczbę :")
2 print(num)
3 name1 = input("Wprowadź imię : ")
4 print(name1)
```

- zawsze przyjmuje napis, w razie potrzeby trzeba zrzutować

```
1 x = str(num)
2 y = int(num)
3 z = float(num)
```

print - instrukcja wyjścia

```
1 print(4.2)
2 # print('Mój wiek to' + 36)
3 print('Mój wiek to', 36)
4 a = 36
5 print('Mój wiek to', a)
6 print('hello', 'world')
7 print('hello', 'world', sep='')
8 print('hello', 'world', sep='\n')
```

```
4.2
Mój wiek to 36
Mój wiek to 36
hello world
helloworld
hello
world
```

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

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

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

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

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

```
xyzabc
```

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

```
a b
 c
```

\t - przesunięcie do następnego “tab”=8 spacji

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

```
sdf 3456    -2
```

Formatowanie napisów będzie później.

Operacje arytmetyczne

Operator	Opis	Składnia
+	Dodawanie	x + y
–	Odejmowanie	x – y
*	Mnożenie	x * y
/	Dzielenie	x / y
//	Dzielenie całkowite	x // y
%	Dzielenie modulo	x % y
**	Potęgowanie	x ** y

```
1 print(5+3)
2 print(4*5.2)
3 print(9-7)
4 print(25%7)
```

```
8
20.8
2
4
```

```
1 print(4/5)
2 print(4//5)
3 print(4/5.0)
4 print(4//5.0)
```

```
0.8
0
0.8
0.0
```

```
1 print(3**0)
2 print(0**0)
```

```
1
1
```

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

Daje info: `ZeroDivisionError: division by zero`

Przypisanie z operacją arytmetyczną

Lista zawiera wybrane operacje.

Inna nazwa to złożone operatory przypisania.

Operator	Zapis	Dłuższa wersja
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5

```
a = 5
a += 1
print(a)
a **= 2
print(a)
```

```
6
36
```

Operatory porównania

Operator	Znaczenie	Przykład
>	Większe niż	x > y
<	Mniejsze niż	x < y
==	Równe	x == y
!=	Nie równa się	x != y
>=	Większe lub równe	x >= y
<=	Mniejsze lub równe	x <= y

Operatory logiczne

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

Operatory bitowe

Operator	Znaczenie	Przykład
&	i - logiczne	x & y
|	lub - logiczne	x | y
^	albo - logiczne	x ^ y
~	negacja - logiczne	~x
<<	przesunięcie w lewo	x << y
>>	przesunięcie w prawo	x >> y

Operator &

```
1 print(4&5)
```

```
4
```

x	100	4
y	101	5
x&y	100	4

Operator |

```
1 print(4|5)
```

```
5
```

x	100	4
y	101	5
x|y	101	5

Operator ^

```
1 print(4^5)
```

```
1
```

x	100	4
y	101	5
x^y	001	1

Operator ~

- równoważnie -(x+1)

```
1 print(~4)
```

```
-5
```

x	100	4
~x	?	-5

Operator <<

a<<b - równoważnie a*pow(2,b)

```
1 print(3<<2)
```

```
12
```

x	0011	3
x<<2	1100	12

Operator >>

a>>b - równoważnie a//pow(2,b)

```
1 print(13>>2)
```

```
3
```

x	1101	13
x>>2	0011	3

Instrukcje warunkowe

Składnia

```
1 if <expr>:
2     <statement>
```

else

```
1 if <expr>:
2     <statement(s)>
3 else:
4     <statement(s)>
```

```
1 a = 5
2 if a > 0:
3     print("Liczba dodatnia")
4 else:
5     print("Liczna ujemna lub zero")
```

```
Liczba dodatnia
```

```
x = 0
y = 5
if x < y:
    print('yes1')

if y < x:
    print('yes2')

if x:
    print('yes3')

if y:
    print('yes4')

if x or y:
    print('yes5')

if x and y:
    print('yes6')
```

```
yes1
yes4
yes5
```

elif

```
if <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
    ...
else:
    <statement(s)>
```

```python
a = 5
if a > 0:
    print("Liczba dodatnia")
elif a == 0:
    print("Zero")
else:
    print("Liczna ujemna")
```

```
Liczba dodatnia
```

Zagnieżdżone instrukcje warunkowe:

```
1 if <expr>:
2     <statement(s)>
3     if <expr>:
4         <statement(s)>
```

```
1  i = 21
2  if i > 0:
3      print("liczba jest dodatnia")
4      if i % 2 == 0:
5          print("Liczba jest dodatkowo parzysta")
```

```
liczba jest dodatnia
```

for - pętla

```
1 for i in <collection>:
2     <loop body>
```

Przykład:

```
1 for i in range(5):
2     print(i)
```

```
0
1
2
3
4
```

range

Generuje nam ciąg liczb (dedykowany typ `range`). Trzeba zamienić na listę “by podejrzeć”.

Uwaga: wszystkie argumenty muszą być w typie całkowitym.

Jeden argument - to “koniec” - ciąg tworzą liczby naturalne od 0 do $n - 1$. Krok domyślny to 1.

Dwa argumenty - to “początek” i “koniec”. Krok domyślny to 1. Wtedy wyświetlone są liczby całkowite z przedziału lewostronnie domkniętego $[początek, koniec)$.

Trzy argumenty - to “początek”, “koniec” oraz krok.

```
print(list(range(5)))
print(list(range(1, 11)))
print(list(range(0, 30, 5)))
print(list(range(0, 10, 3)))
print(list(range(0, -10, -1)))
print(list(range(0)))
print(list(range(1, 0)))
```

```
[0, 1, 2, 3, 4]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[0, 5, 10, 15, 20, 25]
[0, 3, 6, 9]
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
[]
[]
```

While - pętla

```
while <expr>:
    <statement(s)>
```

Przykład:

```
i = 0
while i < 5:
    print(i)
    i = i + 1
```

```
0
1
2
3
4
```

Zagnieżdżone pętle

```
1 for i in <collection>:
2     <loop body>
3     for i in <collection>:
4         <loop body>
```

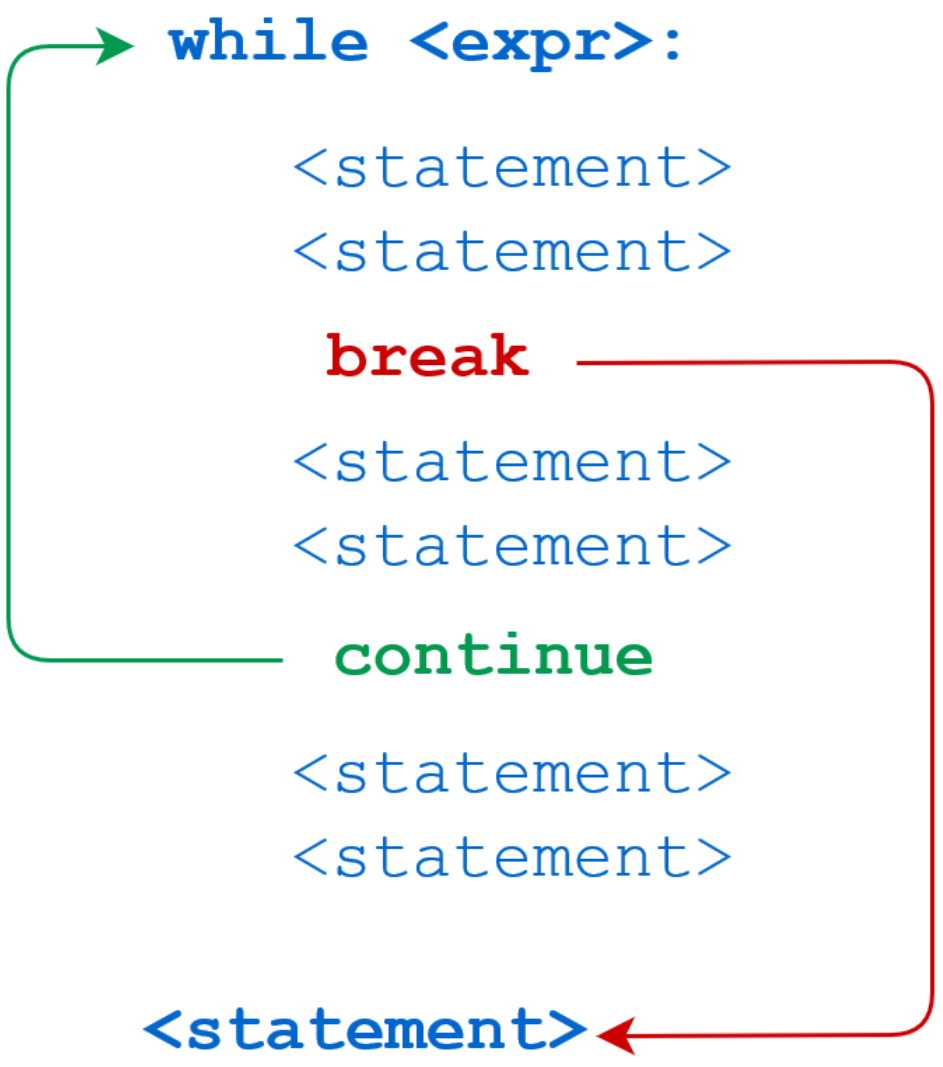
inna opcja:

```
1 while <expr>:
2     <statement(s)>
3     while <expr>:
4         <statement(s)>
```

```python
for i in range(3):
    for j in range(3):
        print(i, "*", j, "=", i * j)
```

```
0 * 0 = 0
0 * 1 = 0
0 * 2 = 0
1 * 0 = 0
1 * 1 = 1
1 * 2 = 2
2 * 0 = 0
2 * 1 = 2
2 * 2 = 4
```

break/continue



break

```
1 n = 5
2 while n > 0:
3     n -= 1
4     if n == 2:
5         break
6     print(n)
```

```
4
3
```

continue

```
1 n = 5
2 while n > 0:
3     n -= 1
4     if n == 2:
5         continue
6     print(n)
```

```
4
3
1
0
```

Kolejność operatorów

Od ostatniego:

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

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

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

Pytanie do przemyślenia

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

```
1 a = 5
2 b = a
3 b += 2
4 print(a)
5 print(b)
```

```
5
7
```

Listy

Lista w Pythonie to tzw. typ sekwencyjny umożliwia przechowywanie elementów różnych typów.

Cechy:

- zmienny (`mutable`) - umożliwia przypisanie wartości pojedynczym elementom
- do zapisu używamy nawiasów kwadratowych
- poszczególne elementy rozdzielamy przecinkami
- każdy element listy ma przyporządkowany indeks
- elementy listy są numerowane od zera
- listy są uporządkowane
- listy są dynamiczne (mogą mieć różną długość)
- listy mogą być zagnieżdżone

Uwaga!

Listy w języku Python są specyficzną strukturą danych nie zawsze dostępną w innych językach programowania. Pojęcie listy w całej informatyce “szersze”. Wyróżnia się np. listy jednokierunkowe, które nie muszą mieć indeksu. Nie będziemy takich przypadków analizować.

```
1 nazwa = [element1, element2, ..., elementN]
```

Pusta lista:

```
1 a = []
2 b = list()
```

Lista z liczbami:

```
1 a = [2, 3, 4.5, 5, -3.2]
```

Lista mieszana:

```
1 b = ['abcd', 25+3j, True, 1]
```

Kolejność ma znaczenie

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

```
False
```

Elementy na liście nie muszą być unikalne

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

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

Indeks - dostęp do elementów listy (od zera)

```
1 a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
2 print(a[1])
3 print(a[4])
4 print(a[0])
5 #print(a[7])
```

```
3
-2.3
1
```

```
1 a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
2 print(a[-1])
3 print(a[-5])
4 print(a[-7])
```

```
9.3
abc
1
```

```python
a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
print(a[1:4])
print(a[-5:-2])
print(a[:4])
```

```
[3, 'abc', False]
['abc', False, -2.3]
[1, 3, 'abc', False]
```

```
1 a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
2 print(a[2:])
3 print(a[0:6:2])
4 print(a[1:6:2])
```

```
['abc', False, -2.3, 'XYZ', 9.3]
[1, 'abc', -2.3]
[3, False, 'XYZ']
```

```
a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
print(a[6:0:-2])
print(a[::-1])
print(a[:])
```

```
[9.3, -2.3, 'abc']
[9.3, 'XYZ', -2.3, False, 'abc', 3, 1]
[1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
```

```
1 a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
2 print(a[::2])
3 print(a[::-2])
```

```
[1, 'abc', -2.3, 9.3]
[9.3, -2.3, 'abc', 1]
```

Specjalne funkcje

Długość (rozmiar listy)

```
1 a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
2 print(len(a))
```

```
7
```

Implementacja samodzielna długości:

```
 1 def dlugosc(lista):
 2     x = 0
 3     for i in lista:
 4         x += 1
 5 
 6     return x
 7 
 8 
 9 a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
10 print(dlugosc(a))
```

```
7
```

Maksimum i minimum?

Działa wtedy gdy mamy porządek

- liczby `<=`
- napisy - porządek leksykograficzny (omówimy przy napisach)

```
a = [4,-5,3.4,-11.2]
print(min(a))
print(max(a))
b = ['abc', 'ABCd', 'krt', 'abcd']
print(min(b))
print(max(b))
```

```
-11.2
4
ABCd
krt
```

Modyfikacja i wstawianie

```
1 a = [4,-5,3.4,-11.2]
2 a[2] = 'a'
3 print(a)
```

```
[4, -5, 'a', -11.2]
```

```
1 a = [4,-5,3.4,-11.2]
2 a[2] = ['a','b']
3 print(a)
```

```
[4, -5, ['a', 'b'], -11.2]
```

```
1 a = [4,-5,3.4,-11.2]
2 a[1:2] = ['a','b']
3 print(a)
```

```
[4, 'a', 'b', 3.4, -11.2]
```

```
1 a = [4,-5,3.4,-11.2]
2 a[1:3] = ['a','b']
3 print(a)
```

```
[4, 'a', 'b', -11.2]
```

Dodawanie list

```
1 a = [4,-5,3.4,-11.2]
2 b = ['a','b','c']
3 print(a+b)
```

```
[4, -5, 3.4, -11.2, 'a', 'b', 'c']
```

Mnożenie listy przez liczbę całkowitą (`int`)

```
1 a = [4,-5,3.4,-11.2]
2 print(a*2)
3 print(2*a)
```

```
[4, -5, 3.4, -11.2, 4, -5, 3.4, -11.2]
[4, -5, 3.4, -11.2, 4, -5, 3.4, -11.2]
```

Usuwanie elementów z listy

```
1 a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
2 del a[2]
3 print(a)
4 del a[:]
5 print(a)
```

```
[1, 3, False, -2.3, 'XYZ', 9.3]
[]
```

Do poczytania

- https://docs.python.org/3.10/tutorial/introduction.html#lists
- https://docs.python.org/3.10/tutorial/datastructures.html#more-on-lists

`list.append(x)` - dodaje element na końcu listy.

Równoważnie `a[len(a):] = [x]`

```
1 a = [1, 3, 'abc', False]
2 a.append(5.3)
3 print(a)
```

```
[1, 3, 'abc', False, 5.3]
```

`list.extend(iterable)` - dodaje elementy z argumenty na koniec listy. Równoważnie: `a[len(a):] = iterable`

```
1 a = [1, 3, 'abc', False]
2 b = [3, -2]
3 a.extend(b)
4 print(a)
```

```
[1, 3, 'abc', False, 3, -2]
```

Różnice?

```
1 a = [1, 3, 'abc', False]
2 b = [3, -2]
3 a.append(b)
4 print(a)
```

```
[1, 3, 'abc', False, [3, -2]]
```

`list.insert(i, x)` - wstawia element `x` na pozycji `i`

```
1 a = [1, 3, 'abc', False]
2 a.insert(0,'w')
3 print(a)
4 a.insert(4,9.0)
5 print(a)
```

```
['w', 1, 3, 'abc', False]
['w', 1, 3, 'abc', 9.0, False]
```

`list.remove(x)` - usuwa element z listy (pierwszy od początku)

```
1 a = [1, 3, 'abc', False]
2 a.remove(False)
3 print(a)
4 b = [3, 4, 5, 3]
5 b.remove(3)
6 print(b)
```

```
[1, 3, 'abc']
[4, 5, 3]
```

`list.pop()` - usuwa i zwraca ostatni element

`list.pop(i)` - usuwa i zwraca element na pozycji `i`

```
1 a = [1, 3, 'abc', False]
2 print(a.pop())
3 print(a)
4 b = [3, -4, 6.2, 7]
5 print(b.pop(3))
6 print(b)
```

```
False
[1, 3, 'abc']
7
[3, -4, 6.2]
```

`list.clear()` - usuwa wszystkie elementy z listy.

Równoważnie: `del a[:]`

```
1 a = [1, 3, 'abc', False]
2 a.clear()
3 print(a)
```

```
[]
```

`list.index(x)` - zwraca indeks elementu x (o ile istnieje, inaczej błąd), w przypadku duplikatów pierwszy z lewej
`list.index(x,start)` - zwraca indeks elementu x (o ile istnieje, inaczej błąd) zaczynając od pozycji `start`, w przypadku duplikatów pierwszy z lewej
`list.index(x,start,end)` - zwraca indeks elementu x (o ile istnieje, inaczej błąd) zaczynając od pozycji `start` a kończąc na `end-1`, w przypadku duplikatów pierwszy z lewej

```
a = [1, 3, 1, 4, 5, 2, 3]
print(a.index(3))
print(a.index(3, 5))
print(a.index(3, 1, 4))
```

```
1
6
1
```

```
a = ['abc','xyz','abc','efg']
print(a.index('abc'))
print(a.index('abc', 2))
print(a.index('abc', 1, 4))
```

```
0
2
2
```

`list.count(x)` - zwraca ile razy występuję element `x` na liście

```
1 a = ['abc','xyz','abc','efg']
2 print(a.count('abc'))
3 print(a.count(4))
```

```
2
0
```

`list.sort()` - sortuje listę (o ile elementy można posortować)

```
1 a = ['abc','xyz','abc','efg']
2 a.sort()
3 print(a)
```

```
['abc', 'abc', 'efg', 'xyz']
```

`list.reverse()` - odwraca kolejność elementów na liście (nie ma nic związku z sortowaniem!)

```
1 a = [4, 5, -2, 7.3, 9, -22, 23]
2 a.reverse()
3 print(a)
```

```
[23, -22, 9, 7.3, -2, 5, 4]
```

`list.copy()` - tworzy kopię listy

Spójrzmy na przykład jak działa operator przypisania dla list.

```
1 a = [4, 5, -2, 7.3, 9, -22, 23]
2 b = a
3 b[2] = 100
4 print(b)
5 print(a)
```

```
[4, 5, 100, 7.3, 9, -22, 23]
[4, 5, 100, 7.3, 9, -22, 23]
```

Różnica z użyciem copy.

```
1 a = [4, 5, -2, 7.3, 9, -22, 23]
2 b = a.copy()
3 b[2] = 100
4 print(b)
5 print(a)
```

```
[4, 5, 100, 7.3, 9, -22, 23]
[4, 5, -2, 7.3, 9, -22, 23]
```

Bibliografia

- https://pl.wikipedia.org/wiki/Python, dostęp online 20.02.2023.
- https://bulldogjob.pl/news/264-java-php-ruby-jak-wlasciwie-wymawiac-nazwy-technologii. dostęp online 12.02.2019.
- https://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html, dostęp online 14.02.2019.
- K. Ropiak, Wprowadzenie do języka Python, http://wmii.uwm.edu.pl/\~kropiak/wd/Wprowadzenie%20do%20j%C4%99zyka%20Python.p dostęp online 14.02.2019.
- B. Slatkin, Efektywny Python. 59 sposobów na lepszy kod, Helion 2015.
- https://www.python.org/dev/peps/pep-0008/, dostęp online 20.02.2023.

Bibliografia - cd2

- https://www.flynerd.pl/2017/05/python-4-typy-i-zmienne.html, dostęp online 14.02.2019.
- http://pytolearn.csd.auth.gr/p0-py/01/print.html, dostęp online 15.02.2019.
- https://www.tutorialspoint.com/python3/python_lists.htm, dostęp online 17.02.2019.
- https://realpython.com/python-data-types/, dostęp online 5.01.2022
- https://www.w3schools.com/python/python_variables.asp, dostęp online 5.01.2022
- https://www.w3schools.com/python/python_variables_multiple.asp, dostęp online 5.01.2022

Bibliografia - cd3

- https://realpython.com/python-print/, dostęp online 5.01.2022
- https://www.programiz.com/python-programming/operators, dostęp online 5.01.2022
- https://realpython.com/python-conditional-statements/, dostęp online 5.01.2022
- https://realpython.com/python-for-loop/, dostęp online 5.01.2022
- https://realpython.com/python-while-loop/, dostęp online 5.01.2022