

Wizualizacja danych

- Wykład 3

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 .

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

```
1 print(list(range(5)))
2 print(list(range(1, 11)))
3 print(list(range(0, 30, 5)))
4 print(list(range(0, 10, 3)))
5 print(list(range(0, -10, -1)))
6 print(list(range(0)))
7 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

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

Przykład:

```
1 i = 0  
2 while i < 5:  
3     print(i)  
4     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)>
```

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

0 * 0 = 0

0 * 1 = 0

0 * 2 = 0

1 * 0 = 0

1 * 1 = 1

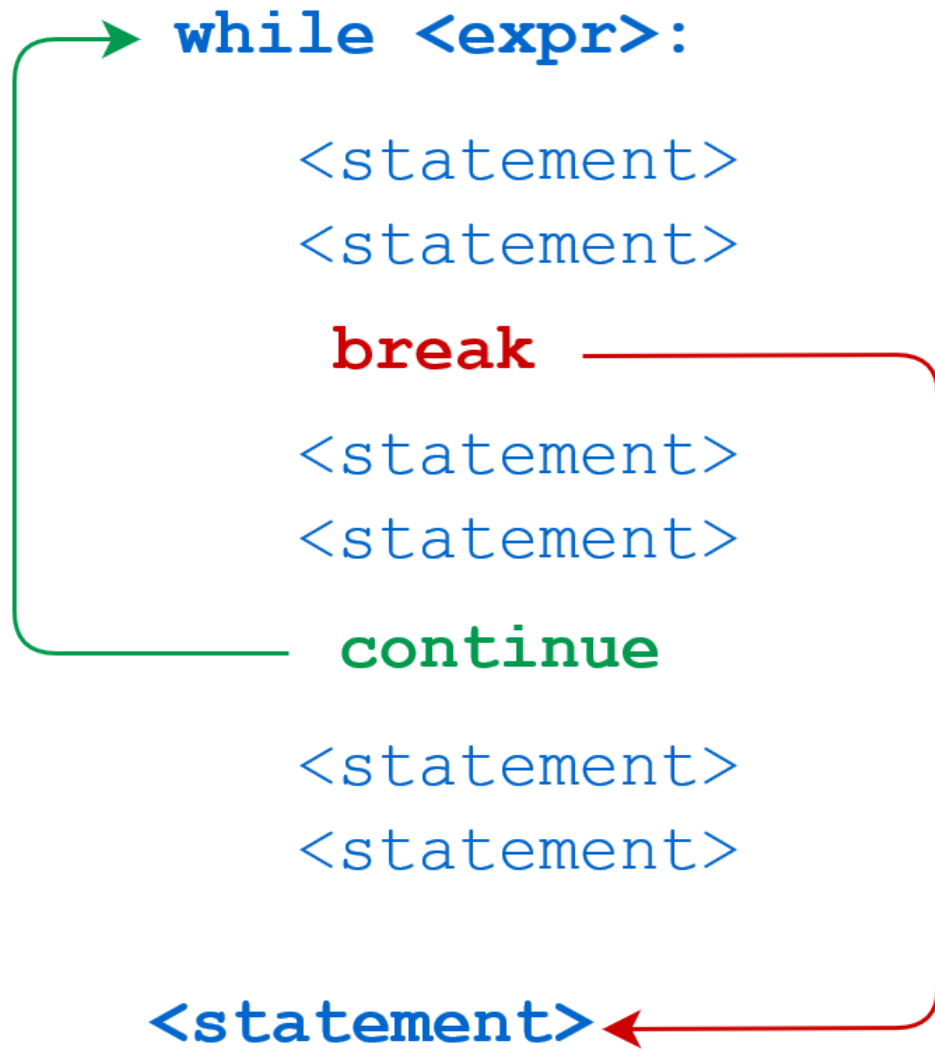
1 * 2 = 2

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

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

```
1 a = [1, 3, 'abc', False, -2.3, 'XYZ', 9.3]
2 print(a[6:0:-2])
3 print(a[::-1])
4 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 \leq
- napisy - porządek leksykograficzny (omówimy przy napisach)

```
1 a = [4,-5,3.4,-11.2]
2 print(min(a))
3 print(max(a))
4 b = ['abc', 'ABcd', 'krt', 'abcd']
5 print(min(b))
6 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 przeczytania

- <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)` - wstawi 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

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

1

6

1

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

0

2

2

`list.count(x)` - zwraca ile razy występuje 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://pytlearn.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

