

Wprowadzenie do języka Python - wykład 2

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

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

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

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

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

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

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

```
## xyzabc
```

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

```
## a b
```

```
## c
```

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

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

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

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	$x ** y$

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

```
## 8
```

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

```
## 20.8
```

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

```
## 2
```

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

```
## 4
```

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

```
## 0.8
```

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

```
## 0
```

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

```
## 0.8
```

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

```
## 0.0
```

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

```
## 1
```

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

```
## 1
```

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

```
ZeroDivisionError: division by zero
```


Przypisanie z operacją arytmetyczną

Lista zawiera wybrane operacje.

Inna nazwa to złożone operatory przypisania.

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

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

6

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

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

Instrukcje warunkowe

Składnia

```
if <expr>:  
    <statement>
```

else

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

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

```
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')
```


elif

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

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

Zagnieżdżone instrukcje warunkowe:

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

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

for - pętla

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

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

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)))
```

While - pętla

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

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

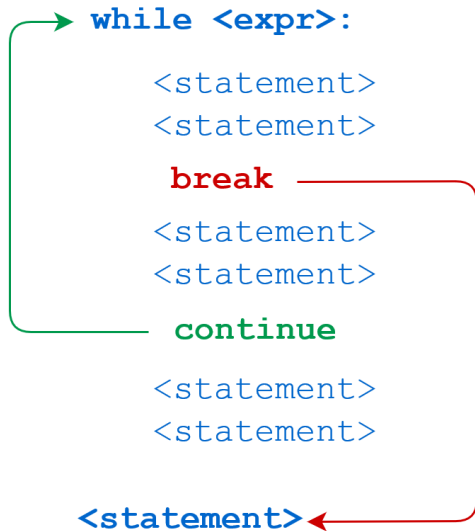

Zagnieżdżone pętle

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

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

```
for i in range(5):  
    for j in range(5):  
        print(i, "*", j, "=", i * j)
```

break/continue



break

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

continue

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

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 na kolejny wykład

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

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

```
## 5
```

```
print(b)
```

```
## 7
```


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.

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

Pusta lista:

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

Lista z liczbami:

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

Lista mieszana:

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

Kolejność ma znaczenie

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

```
## False
```

Elementy na liście nie muszą być unikalne

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

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

```
print(a == b)
```

```
## False
```

Indeks - dostęp do elementów listy (od zera)

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

```
## 3
```

```
print(a[4])
```

```
## -2.3
```

```
print(a[0])
#print(a[7])
```

```
## 1
```

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

```
## 9.3
```

```
print(a[-5])
```

```
## abc
```

```
print(a[-7])
```

```
## 1
```

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

```
## [3, 'abc', False]
```

```
print(a[-5:-2])
```

```
## ['abc', False, -2.3]
```

```
print(a[:4])
```

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



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

```
## ['abc', False, -2.3, 'XYZ', 9.3]
```

```
print(a[0:6:2])
```

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

```
print(a[1:6:2])
```

```
## [3, False, 'XYZ']
```

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

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

```
print(a[::-1])
```

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

```
print(a[:])
```

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

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

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

```
print(a[::-2])
```

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

Specjalne funkcje

Długość (rozmiar listy)

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

7

Implementacja samodzielna długości:

```
def dlugosc(lista):  
    x = 0  
    for i in lista:  
        x += 1  
  
    return x
```

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

```
## 7
```

Maksimum i minimum?

Działa wtedy gdy mamy porządek

- ▶ liczby \leq
- ▶ napisy - porządek leksykograficzny (omówimy przy napisach)

```
a = [4, -5, 3.4, -11.2]
print(min(a))
```

```
## -11.2
```

```
print(max(a))
```

```
## 4
```

```
b = ['abc', 'ABcd', 'krt', 'abcd']
print(min(b))
```

```
## ABCd
```

```
print(max(b))
```

```
## krt
```

Modyfikacja i wstawianie

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

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



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

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

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

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

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

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

Dodawanie list

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

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

Mnożenie listy przez liczbę całkowitą (int)

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

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

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

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

Usuwanie elementów z listy

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

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

```
del a[:]
print(a)
```

```
## []
```

Do poczytania

- ▶ <https://docs.python.org/3.9/tutorial/introduction.html#lists>
- ▶ <https://docs.python.org/3.9/tutorial/datastructures.html#more-on-lists>

`list.append(x)` - dodaje element na końcu listy. Równoważnie
`a[len(a):] = [x]`

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

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


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

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

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

Różnice?

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

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

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

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

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

```
a.insert(4, 9.0)
print(a)
```

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

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

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

```
## [1, 3, 'abc']
```

```
b = [3, 4, 5, 3]
b.remove(3)
print(b)
```

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

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

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

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

```
## False
```

```
print(a)
```

```
## [1, 3, 'abc']
```

```
b = [3, -4, 6.2, 7]
print(b.pop(3))
```

```
## 7
```

```
print(b)
```

```
## [3, -4, 6.2]
```

`list.clear()` - usuwa wszystkie elementy z listy. Równoważnie:
`del a[:]`

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

```
## 1
```

```
print(a.index(3, 5))
```

```
## 6
```

```
print(a.index(3, 1, 4))
```

```
## 1
```



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

```
## 0
```

```
print(a.index('abc', 2))
```

```
## 2
```

```
print(a.index('abc', 1, 4))
```

```
## 2
```

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

```
a = ['abc', 'xyz', 'abc', 'efg']  
print(a.count('abc'))
```

```
## 2
```

```
print(a.count(4))
```

```
## 0
```

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

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

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

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

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

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

`list.copy()` - tworzy kopię listy

Spójrzmy na przykład jak działa operator przypisania dla list.

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

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

```
print(a)
```

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

Różnica z użyciem copy.

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

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

```
print(a)
```

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

Bibliografia

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

Bibliografia - cd2

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

Bibliografia - cd3

- ▶ <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
- ▶ <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