

Wizualizacja danych

- Wykład 4

Lista jako stos

```
1 stack = [3, 4, 5, 8, 9]
2 stack.append(6)
3 stack.append(7)
4 print(stack)
5 print(stack.pop())
6 print(stack)
```

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

```
7
```

```
[3, 4, 5, 8, 9, 6]
```

Lista jako kolejka

```
1 from collections import deque
2
3 queue = deque(["aw", "tg", "kj"])
4 queue.append("gg")
5 print(queue)
6 print(queue.popleft())
7 print(queue)
```

```
deque(['aw', 'tg', 'kj', 'gg'])
```

```
aw
```

```
deque(['tg', 'kj', 'gg'])
```

List Comprehensions

```
1 squares = []  
2 for x in range(5):  
3     squares.append(x ** 2)  
4  
5 print(squares)
```

```
[0, 1, 4, 9, 16]
```

```
1 squares = [x**2 for x in range(5)]  
2 print(squares)
```

```
[0, 1, 4, 9, 16]
```

Krotka - tuple

```
1 krotka = 123, 'abc', True
2 krotka2 = (123, 'abc', True)
3 print(krotka[2])
```

True

```
1 krotka[0] = 1
```

TypeError: 'tuple' object does not support item assignment

<https://docs.python.org/3.10/library/stdtypes.html#tuple>

Zbiór - set

```
1 cyfry = {'raz', 'dwa', 'raz', 'trzy', 'raz', 'osiem'}
2 print(cyfry)
```

```
{'dwa', 'trzy', 'osiem', 'raz'}
```

<https://docs.python.org/3.10/library/stdtypes.html#set>

```
1 x = {2, 3, 4, -3, 5, 2}
2 y = {5, 6, 7}
3 z = {'a', 'b'}
4 w = {3, 4}
5 print(x)
6 print(len(x))
7 print(0 in x)
8 print(0 not in x)
```

```
{2, 3, 4, 5, -3}
```

```
5
```

```
False
```

```
True
```

```
1 x = {2, 3, 4, -3, 5, 2}
2 y = {5, 6, 7}
3 z = {'a', 'b'}
4 w = {3, 4}
5 print(x.isdisjoint(y))
6 print(x.isdisjoint(z))
7 print(w.issubset(x))
8 print(x.issubset(w))
9 print(w <= x)
10 print(w < x)
11 print(w.issuperset(x))
12 print(x.issuperset(w))
13 print(w >= x)
14 print(w > x)
```

False
True
True
False
True
True
False
True
False
False

```
1 x = {2, 3, 4, -3, 5, 2}
2 y = {5, 6, 7}
3 z = {'a', 'b'}
4 w = {3, 4}
5 print(x.union(y))
6 print(x | y)
7 print(x.intersection(y))
8 print(x & y)
9 print(x.difference(y))
10 print(x - y)
11 print(x.symmetric_difference(y))
12 print(x ^ y)
13 print(x.copy()) # płytka kopia
```

```
{2, 3, 4, 5, 6, 7, -3}
{2, 3, 4, 5, 6, 7, -3}
{5}
{5}
{2, 3, 4, -3}
{2, 3, 4, -3}
{2, 3, 4, 6, 7, -3}
{2, 3, 4, 6, 7, -3}
{2, 3, 4, 5, -3}
```



```
1 x = {2, 3, 4, -3, 5, 2}
2 x.update({11})
3 print(x)
4 x |= {12}
5 print(x)
6 x = {2, 3, 4, -3, 5, 2}
7 x.intersection_update({5})
8 print(x)
9 x = {2, 3, 4, -3, 5, 2}
10 x &= {5}
11 print(x)
12 x = {2, 3, 4, -3, 5, 2}
```

{2, 3, 4, 5, 11, -3}

{2, 3, 4, 5, 11, 12, -3}

{5}

{5}

```
1 x = {2, 3, 4, -3, 5, 2}
2 x.difference_update({4})
3 print(x)
4 x = {2, 3, 4, -3, 5, 2}
5 x -= {4}
6 print(x)
7 x = {2, 3, 4, -3, 5, 2}
8 x.symmetric_difference_update({4, 11})
9 print(x)
10 x = {2, 3, 4, -3, 5, 2}
11 x ^= {4, 11}
12 print(x)
```

{2, 3, 5, -3}

{2, 3, 5, -3}

{2, 3, 5, 11, -3}

{2, 3, 5, 11, -3}

```
1 x = {2, 3, 4, -3, 5, 2}
2 print(x)
3 x.add(11)
4 print(x)
5 x.remove(-3) # usuwa gdy jest, inaczej KeyError
6 print(x)
7 x.discard(12)
8 print(x)
```

{2, 3, 4, 5, -3}

{2, 3, 4, 5, 11, -3}

{2, 3, 4, 5, 11}

{2, 3, 4, 5, 11}

```
1 x = {2, 3, 4, -3, 5, 2}
2 print(x)
3 x.discard(2)
4 print(x)
5 x = {2, 3, 4, -3, 5, 2}
6 print(x.pop())
7 print(x)
8 x.clear()
9 print(x)
```

```
{2, 3, 4, 5, -3}
```

```
{3, 4, 5, -3}
```

```
2
```

```
{3, 4, 5, -3}
```

```
set()
```

Słownik

```
1 tel = {'jack': 4098, 'sape': 4139}
2 tel['guido'] = 4127
3 print(tel)
4 tel['jack']
5 del tel['sape']
6 tel['irv'] = 4127
7 print(tel)
```

```
{'jack': 4098, 'sape': 4139, 'guido': 4127}
```

```
{'jack': 4098, 'guido': 4127, 'irv': 4127}
```

<https://docs.python.org/3.10/library/stdtypes.html#mapping-types-dict>

```
1 d = {"one": 1, "two": 2, "three": 3, "four": 4}
2 print(d)
3 print(list(d))
4 print(list(d.values()))
5 d["one"] = 42
6 print(d)
7 del d["two"]
8 print(d)
9 d["two"] = None
10 print(d)
```

```
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
['one', 'two', 'three', 'four']
[1, 2, 3, 4]
{'one': 42, 'two': 2, 'three': 3, 'four': 4}
{'one': 42, 'three': 3, 'four': 4}
{'one': 42, 'three': 3, 'four': 4, 'two': None}
```

```
1 d = {"one": 1, "two": 2, "three": 3, "four": 4}
2 print(d)
3 print(list(reversed(d)))
4 print(list(reversed(d.values())))
5 print(list(reversed(d.items())))
```

```
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

```
['four', 'three', 'two', 'one']
```

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

```
[('four', 4), ('three', 3), ('two', 2), ('one', 1)]
```

```
1 d = {"one": 1, "two": 2, "three": 3, "four": 4}
2 print("one" in d)
3 print(1 in d)
4 print("one" not in d)
5 print(1 not in d)
6 print(iter(d))
7 for elem in iter(d):
8     print(elem)
```

True

False

False

True

<dict_keyiterator object at 0x000001DD3E81B600>

one

two

three

four


```
1 d = {"one": 1, "two": 2, "three": 3, "four": 4}
2 d.clear()
3 print(d)
4 d = {"one": 1, "two": 2, "three": 3, "four": 4}
5 e = d.copy() # płyta kopia
6 print(e)
```

```
{}
```

```
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

```
1 x = ('key1', 'key2', 'key3')
2 y = 0
3 d1 = dict.fromkeys(x, y)
4 print(d1)
5 z = (3, 4, 5)
6 d2 = dict.fromkeys(x, z)
7 print(d2)
```

```
{'key1': 0, 'key2': 0, 'key3': 0}
```

```
{'key1': (3, 4, 5), 'key2': (3, 4, 5), 'key3': (3, 4, 5)}
```

```
1 d = {"one": 1, "two": 2, "three": 3, "four": 4}
2 print(d.get("two"))
3 print(d.items())
4 print(d.keys())
5 print(d.pop("three"))
6 print(d)
7 print(d.popitem())
8 print(d)
```

```
2
dict_items([('one', 1), ('two', 2), ('three', 3), ('four', 4)])
dict_keys(['one', 'two', 'three', 'four'])
3
{'one': 1, 'two': 2, 'four': 4}
('four', 4)
{'one': 1, 'two': 2}
```

```
1 d = {"one": 1, "two": 2, "three": 3, "four": 4}
2 d.update(red=1, blue=2)
3 print(d)
4 print(d.values())
```

```
{'one': 1, 'two': 2, 'three': 3, 'four': 4, 'red': 1, 'blue': 2}
dict_values([1, 2, 3, 4, 1, 2])
```

```
1 d = {"one": 1, "two": 2, "three": 3, "four": 4}
2 d2 = {"a": 11, "b": 12}
3 d3 = d | d2
4 print(d3)
5 print(d)
6 d |= d2
7 print(d)
```

```
{'one': 1, 'two': 2, 'three': 3, 'four': 4, 'a': 11, 'b': 12}
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
{'one': 1, 'two': 2, 'three': 3, 'four': 4, 'a': 11, 'b': 12}
```

Python comprehension

```
1 a = [3, 4, 5]
2 print(a)
3 a2 = [i**2 for i in a]
4 print(a2)
5 a3 = {i**2 for i in a}
6 print(a3)
7 a4 = {i: i**2 for i in a}
8 print(a4)
```

[3, 4, 5]

[9, 16, 25]

{16, 9, 25}

{3: 9, 4: 16, 5: 25}

```
1 a = [3, 4, 5]
2 print(a)
3 a2 = (i**2 for i in a)
4 print(a2)
5 for i in a2:
6     print(i)
```

[3, 4, 5]

<generator object <genexpr> at 0x000001DD3E84C350>

9

16

25

```
1 a = [3, 4, "w", 5]
2 print(a)
3 w = [i*i for i in a if isinstance(i, int)]
4 print(w)
```

```
[3, 4, 'w', 5]
```

```
[9, 16, 25]
```



```
1 a = [3, 4, "w", 5]
2 print(a)
3 w = [i*i if isinstance(i, int) else i for i in a]
4 print(w)
```

```
[3, 4, 'w', 5]
```

```
[9, 16, 'w', 25]
```

Napisy

- trochę podobne do listy
- typ sekwencyjny do przechowywania znaków, ale w odróżnieniu od listy jest niezmienny
- w języku Python nie ma oddzielnego typu znakowego
- apostrofy i cudzysłów można stosować zamiennie, ale konsekwentnie

Inne nazwy: - string, napisy, łańcuchy znaków

Abstrakcyjnie:

- na końcu każdego napisu jest znak “zerowy” - będzie widać lepiej w C/C++

Tablica znaków ASCII <https://upload.wikimedia.org/wikipedia/commons/5/5c/ASCII-Table-wide.pdf>

```
1 a = "Olsztyn"  
2 print(a)  
3 print(a[3])  
4 #a[2] = 'w'
```

Olsztyn

z

```
1 a = "Olsztyn"  
2 b = "Gdańsk"  
3 print(a + b)  
4 print(a * 2)  
5 print(2 * a)
```

OlsztynGdańsk

OlsztynOlsztyn

OlsztynOlsztyn

Specjalne funkcje

- `chr()` zamienia liczbę całkowitą na znak
- `ord()` zamienia znak na liczbę całkowitą odpowiadającą pozycji w tabeli znaków
- `len()` - długość napisu
- `str()` - rzutuje argument na napis

Porządek leksykograficzny

Mądra definicja z wikipedii:

Relację leksykograficzną między ciągami ustala się następująco:

$$\alpha, \beta \in X^*$$

- jeśli istnieje wskaźnik j taki, że $\alpha(j) \neq \beta(j)$, to znajdujemy najmniejszy j o tej własności. Wówczas

- gdy $\alpha(j) < \beta(j)$ lub gdy $\beta(j) < \alpha(j)$ (tzn. relacja między ciągami jest zgodna z relacją między odpowiednimi elementami)

- jeśli taki j nie istnieje, to

- jeśli oba są skończone i tej samej długości, to

$$\alpha = \beta$$

- jeśli oba ciągi są nieskończone, to

- jeśli są różnej długości np. α jest dłuższy od β (w szczególności może być nieskończony), to

$$\alpha \preceq \beta$$

Przykłady:

```
1 print("A" < "a")
2 print("Abc" < "aTw")
3 print("vccx" < "123")
4 print("ABC" < "AB")
5 print("AB" < "ABC")
```

True

True

False

False

True

Fomatowanie napisów

- trzy różne konwencje
- niektóre rzeczy nie działają w każdej wersji 3.x
- warto zastanowić się czy warto używać tych konstrukcji?
czasem może lepiej skorzystać z funkcji print?

styl printf

Zaczerpnięty z języka C - stare.

<https://docs.python.org/3.10/library/stdtypes.html#old-string-formatting>

```
1 a = "abc"
2 str = "a to %s" % a
3 print(str)
4 b = 4
5 c = 5
6 str2 = "%d + %d = %d" % (b, c, b + c)
7 print(str2)
```

a to abc

4 + 5 = 9

Dodatkowe:

<https://gist.github.com/pjastr/02d01dba3d5f5c3e60ed74cb32c913>

<https://gist.github.com/pjastr/cbe8418eb4798b92d7fcba4f48d3284>

<https://gist.github.com/pjastr/e7d5fcbebd578c1df122d307e005170>

<https://gist.github.com/pjastr/00c1df223918f975d678d8455b4f5b0>

styl format

<https://docs.python.org/3.10/library/string.html#formatstrings>

```
1 a = "abc"  
2 str = "a to {}".format(a)  
3 print(str)  
4 b = 4.2  
5 c = 5  
6 str2 = "{0} + {1} = {2}".format(b, c, b + c)  
7 print(str2)
```

a to abc

4.2 + 5 = 9.2

```
1 b = 4.2
2 c = 5
3 str2 = "{0:f} + {1:d} = {2:e}".format(b, c, b + c)
4 print(str2)
```

4.200000 + 5 = 9.200000e+00

<https://pyformat.info/>

<https://gist.github.com/pjastr/27fe6c13cd8bcba63be561a05af030a0>

<https://gist.github.com/pjastr/d40fe6eaf21a9595bcf6b43e7b020fbd>

<https://gist.github.com/pjastr/8a630b4a575e6a389a7dc3c5e8dc65a0>

<https://gist.github.com/pjastr/d42d937c7b00b80b5dfe309b4ac0e854>

<https://gist.github.com/pjastr/dbc9a0c1e8bf612b68e0bc7789daf53b>

<https://gist.github.com/pjastr/adfd7371dcbe4e4034bfb12dcfe30129>

<https://gist.github.com/pjastr/2980fb68a484dcb5ff595774eec4195c>

Dodatkowe przykłady:

<https://docs.python.org/3.9/library/string.html#format-examples>

<https://gist.github.com/pjastr/d42d937c7b00b80b5dfe309b4ac0e8>

<https://gist.github.com/pjastr/90f1accf7f54d8c74ac036d59a24a9d>

<https://gist.github.com/pjastr/adfd7371dcbe4e4034bfb12dcfe3012>

<https://gist.github.com/pjastr/2980fb68a484dcb5ff595774eec4195>

f-Strings

https://docs.python.org/3.10/reference/lexical_analysis.html#f-strings

```
1 a = "abc"
2 str = f"a to {a}"
3 print(str)
4 b = 4.2
5 c = 5
6 str2 = f"{b} + {c} = {b+c}"
7 print(str2)
```

a to abc

4.2 + 5 = 9.2

```
1 b = 4.2
2 c = 5
3 str2 = f"{b:f} + {c:d} = {b+c:e}"
4 print(str2)
```

4.200000 + 5 = 9.200000e+00

Dodatkowe

- podział stałych

<https://docs.python.org/3.10/library/string.html?highlight=string#module-string>

- funkcje wbudowane dot. napisów

<https://docs.python.org/3.10/library/stdtypes.html#string-methods>

Bibliografia

- <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
- <https://towardsdatascience.com/a-guide-to-python-comprehensions-4d16af68c97e>,
dostęp online 20.02.2023.

