Wprowadzenie do języka Python - Wykład 3

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

Typowanie w krotce

```python
x: tuple[int, str, float] = (3, "yes", 7.5)
a: tuple[int, ...] = (1, 2, 3)
a = (3, 4, 9, -2)
```

po staremu:

```python
from typing import Tuple

x: Tuple[int, str, float] = (3, "yes", 7.5)
y: Tuple[int, ...] = (1, 2, 3)
```

Zbiór - set

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

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

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

Zbiory a typowanie

```
1  x: set[int] = {6, 7}
```

po staremu:

```
1  from typing import Set
2
3  x: Set[int] = {6, 7}
```

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

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

```
x = {2, 3, 4, -3, 5, 2}
y = {5, 6, 7}
z = {'a', 'b'}
w = {3, 4}
print(x.isdisjoint(y))
print(x.isdisjoint(z))
print(w.issubset(x))
print(x.issubset(w))
print(w <= x)
print(w < x)
print(w.issuperset(x))
print(x.issuperset(w))
print(w >= x)
print(w > x)
```

```
False
True
True
False
True
True
False
True
False
False
```

```python
x = {2, 3, 4, -3, 5, 2}
y = {5, 6, 7}
z = {'a', 'b'}
w = {3, 4}
print(x.union(y))
print(x | y)
print(x.intersection(y))
print(x & y)
print(x.difference(y))
print(x - y)
print(x.symmetric_difference(y))
print(x ^ y)
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}
```

```
x = {2, 3, 4, -3, 5, 2}
x.update({11})
print(x)
x |= {12}
print(x)
x = {2, 3, 4, -3, 5, 2}
x.intersection_update({5})
print(x)
x = {2, 3, 4, -3, 5, 2}
x &= {5}
print(x)
x = {2, 3, 4, -3, 5, 2}
```

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

```
x = {2, 3, 4, -3, 5, 2}
x.difference_update({4})
print(x)
x = {2, 3, 4, -3, 5, 2}
x -= {4}
print(x)
x = {2, 3, 4, -3, 5, 2}
x.symmetric_difference_update({4, 11})
print(x)
x = {2, 3, 4, -3, 5, 2}
x ^= {4, 11}
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 KeyErroe
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}
```

```
x = {2, 3, 4, -3, 5, 2}
print(x)
x.discard(2)
print(x)
x = {2, 3, 4, -3, 5, 2}
print(x.pop())
print(x)
x.clear()
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

Słownik a typowanie

```
1  x: dict[str, float] = {"field": 2.0}
```

po staremu:

```
1  from typing import Dict
2
3  x: Dict[str, float] = {"field": 2.0}
```

```python
d = {"one": 1, "two": 2, "three": 3, "four": 4}
print(d)
print(list(d))
print(list(d.values()))
d["one"] = 42
print(d)
del d["two"]
print(d)
d["two"] = None
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)]
```

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

```
True
False
False
True
<dict_keyiterator object at 0x0000022389963600>
one
two
three
four
```

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

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

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

```
{'key1': 0, 'key2': 0, 'key3': 0}
{'key1': (3, 4, 5), 'key2': (3, 4, 5), 'key3': (3, 4, 5)}
```

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

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

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

```
[3, 4, 5]
<generator object <genexpr> at 0x00000223899B0510>
9
16
25
```

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

```
[3, 4, 'w', 5]
[9, 16, 25]
```

```python
a = [3, 4, "w", 5]
print(a)
w = [i*i if isinstance(i, int) else i for i in a]
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 $\preccurlyeq$ między ciągami $\alpha, \beta \in X^*$ ustala się następująco:

- jeśli istnieje wskaźnik j taki, że $\alpha(j) \neq \beta(j)$, to znajdujemy najmniejszy i o tej własności. Wówczas
  - $\alpha \preccurlyeq \beta$ gdy $\alpha(i) \preccurlyeq \beta(i)$ lub $\beta \preccurlyeq \alpha$ gdy $\beta(i) \preccurlyeq \alpha(i)$ (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 $\alpha = \beta$
  - jeśli są różnej długość np. β jest dłuższy od α (w szczególności β może być nieskończony), to $\alpha \prec \beta$

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 każej 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/cbe8418eb4798b92d7fcba4f48d328
https://gist.github.com/pjastr/e7d5fcbebd578c1df122d307e00517
https://gist.github.com/pjastr/00c1df223918f975d678d8455b4f5b

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
```

```
b = 4.2
c = 5
str2 = f"{b:f} + {c:d} = {b+c:e}"
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.