Wizualizacja danych
semestr letni 2024

Dr Anna Muranova
UWM w Olsztynie

Wykład 4

Złożone typu danych w Python

- **List (Lista)** jest kolekcją uporządkowaną i zmienną. Zezwala na duplikowanych członków.
- **Tuple (Krotka)** to kolekcja uporządkowana i niezmienna. Zezwala na duplikowanych członków.
- **Set (Zbiór)** to kolekcja, która jest nieuporządkowana, niezmienna (ale można dodawać i usuwać elementy) i nieindeksowana. Brak duplikatów członków.
- **Dictionary (Słownik)** jest kolekcją zmienną. Brak duplikatów członków. Uporządkowany po Python 3.7 i wyżej, nie uporządkowany w Python 3.6 i niżej.

Zbiór

Cechy:

- niezmienny, ale umożliwia dodawanie i usuwanie elementów,
- poszczególne elementy rozdzielamy przecinkami, piszemy klamry,
- nie są uporządkowane,
- są dynamiczne (mogą mieć różną długość),
- elementy nie powtarzają się.

Zbiory: pisownia

```
nazwa = {element1, element2, ..., elementN}
```

- ```
  cyfry = {'raz', 'dwa', 'raz', 'trzy', 'raz', 'osiem'}
  print(cyfry)
  ```
- ```
  x = {2, 3, 4, -3, 5, 2}
  y = {5, 6, 7}
  z = {'a', 'b'}
  w = {3, 4}
  print(x)
  #print(x[0])
  print(len(x))
  print(0 in x)
  print(0 not in x)
  ```

Zbiory: dodawanie i usuwanie elementów

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

Zbiory: funkcje dla sprawdzania

```
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)
```

Zbiory: funkcje na zbiorach

```
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())
```

Zbiory: funkcje dla zmian 1

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

Zbiory: funkcje dla zmian 2

```
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)
```

Zbiory: kopiowanie vs. przypisanie

```
x = {2, 3, 4, -3, 5, 2}
y = x
z = x.copy()
x.update({15})
print(x)
print(y)
print(z)
```

Zbiory: inne funkcje

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

print(len(test_set))
print(max(test_set))
print(min(test_set))
```

Zbiory: iteracja

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

for i in test_set:
    print(i)

print([i**2 for i in test_set])
```

Słownik

Cechy:

- zmienny, umożliwia dodawanie , usuwanie elementów i zamianą wartości,
- brak duplikatów członków,
- od Python 3.7 są uporządkowane,
- są dynamiczne (mogą mieć różną długość),

Słownik: pisownia

```
tel = {'jack': 4098, 'sape': 4139}
tel['guido'] = 4127
print(tel)
tel['jack']
del tel['sape']
tel['irv'] = 4127
print(tel)

test_dict = {1: 4098, 4: 4139, 5: 'slowo','slowo':623}
test_dict[4] = 'trrr'
print(test_dict)
```

Słownik: używanie

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

d["one"] = 42
print(d)
del d["two"]
print(d)
d["two"] = None
print(d)
```

Słownik: sprawdzanie

```
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)
```

Słownik: modyfikacja

```
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)
```

Słownik: tworzenie 1

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

Słownik: tworzenie 2

```
# initializing lists
test_keys = ["Rash", "Kil", "Varsha"]
test_values = [1, 4, 5]

# using dictionary comprehension
# to convert lists to dictionary
res = {test_keys[i]: test_values[i] for i in range(len(test_keys))}

# Printing resultant dictionary
print("Resultant dictionary is : " + str(res))
```

Słownik: tworzenie 2

- ```
  test_keys = ["Rash", "Kil", "Varsha"]
  test_values = [1, 4, 5]

  res = dict(zip(test_keys, test_values))

  print("Resultant dictionary is : " + str(res))
  ```
- https://www.geeksforgeeks.org/python-convert-two-lists-into-a-dictionary/
  ```

Słownik: inne funkcje 1

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

Słownik: inne funkcje 2

```
d = {"one": 1, "two": 2, "three": 3, "four": 4}

print(max(d))
print(min(d))
print(min(d.values()))
print(len(d))
```

Słownik: inne funkcje 3

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

Słownik: iteracja

```
d = {"one": 1, "two": 2, "three": 3, "four": 4}
for i in d:
    print(i)

d = {"one": 1, "two": 2, "three": 3, "four": 4}
for i, j in d.items():
    print(i,':',j)

d = {"one": 1, "two": 2, "three": 3, "four": 4}
for x in d.values():
    print(x)

print([x**2 for x in d.values()])
print([i*2 for i in d])
```

Funkcje: domyślny argument

```
def sumsub(a, b, c=0, d=0):
    return a - b + c - d

print(sumsub(12, 4))
print(sumsub(3, 4, 5, 7))
```

Funkcje: *args

- ```
  def srednia(*values):
      return (sum(values)) / (len(values))

  print(srednia(2, 3, 4, 6))
  print(srednia(45))
  ```
- ```
  def srednia(coeff, *values):
      return (sum(values*coeff)) / (len(values))

  print(srednia(2, 3, 4, 6))
  print(srednia(2,4))
  ```

Funkcje: **kwargs

- ```
  def f(**kwargs):
  print(kwargs)

  f()
  f(pl="Polish", en="English")
  ```
- ```
  def f(**kwargs):
  print(kwargs)

  f()
  f(**{'pl':"Polish", 'en':"English"})
  ```

Funkcje: import math

Link do dokumentacji https://docs.python.org/3/library/math.html

- ```
  import math

  a=0
  b=math.sin(2*math.pi)
  print(b)
  print(math.isclose(a,b, rel_tol=1e-09, abs_tol=1e-09))
  ```
- ```
  import math

  print(math.e)
  print(math.exp(1))
  print(math.ceil(math.e))
  print(math.floor(math.e))
  print(math.factorial(10))
  ```

Funkcje: import math

Link do dokumentacji https://docs.python.org/3/library/math.html

- ```
  import math

  print(math.gcd())
  print(math.gcd(20,16))
  print(math.gcd(20,15,35,25))
  ```
- ```
  import math

  #print(math.prod())
  #print(math.prod(0,2))
  print(math.prod([0,2]))
  print(math.prod([12,3,21]))
  print(math.prod([12,3,21], start=2))
  ```

Funkcje: potęga

```
import math

print(math.pow(2,100))
print(pow(2,100))
print(2**100)

#print(math.pow(-2,0.5))
print(pow((-2),0.5))
print((-2)**0.5)

print(math.exp(1e-5) - 1)  # gives result accurate to 11 places
print(math.expm1(1e-5) )
```

Oprócz tego `math.exp(x)` dokładniej niż `math.e ** x` oraz `pow(math.e, x)`.

Funkcje: reszta

```
import math

print(math.remainder(8,3))
print(math.fmod(8,3))
print(8%3)

print(math.remainder(8.5,3))
print(math.fmod(8.5,3))
print(8.5%3)

print(math.remainder(-1e-100,1e100))
print(math.fmod(-1e-100,1e100))
print(-1e-100 % 1e100)
```