Tartalomjegyzék

- 1 Gy?teményes adatszerkezetek, adatstruktúrák
  - 1.1 Listák
    - 1.1.1 Listák kezelése, metódusai
    - 1.1.2 Listaértelmezés
    - 1.1.3 del utasítás
  - 1.2 Sorozat (tuple)
  - 1.3 Halmaz (set)
  - 1.4 Szótár (dict)
    - 1.4.1 Hasító tábla (hash table)

Gy?teményes adatszerkezetek, adatstruktúrák

Érdemes a Python saját tutoralját is elolvasni.

Listák

Ugyanúgy lehet **szeletelni (slice)**, mint a karakterláncot:

```
>>> x = [1, 2, 3, 4]
>>> x[1]
2
>>> x[2:]
[3, 4]
>>> x[0::2]
[1, 3]
>>> x[:]
[1, 2, 3, 4]
>>> x[-1:-3]
[]
>>> x[-1:-3:-1]
[4, 3]
>>> x[-1::-1]
[4, 3, 2, 1]
```

Ugyanúgy lehet összeadni, többszörözni, mint a karakterláncot, de **a lista változtatható (mutable)**:

```
>>> x = [1, 2, 3, 4]
>>> x[:2]*2 + x[-1:]
[1, 2, 1, 2, 4]
>>> x[:2]*2 + x[-1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
```

Az értékadás (=) csak objektumhivatkozással jár, nem történik adatmásolás. Ennek következtében egy `y = x` parancs után, ahol x lista (vagy valamilyen más gy?jteményes adattípus), az y ugyanarra az objektumra fog mutatni. Így ha megváltoztatjuk x-et vagy y-t, változik a másik is. Az objektumhivatkozások megtörténhetnek egy szinttel mélyebben is, a (`z = x[:]` kód esetén az x elemeire mutató hivatkozások másolódnak, de ekkor sem jön létre a teljes objektumról másolat. Ezt hívjuk **sekély másolásnak (shallow copy)**.

Egy másik példa sekély másolásra:

Mély másolás (deep copy), amikor valóban új példány keletkezik az objektumból:

```
import copy
w = copy.deepcopy(x)
```

Listák kezelése, metódusai

Lista létrehozható értékadással, [] az üres lista. A range parancs is listát ad vissza:

```
>>> range(3)
[0, 1, 2]
>>> range(3, 6)
[3, 4, 5]
>>> range(1,10,3)
[1, 4, 7]
```

A lehetséges metódusok például kiírhatók TAB billenty?vel ipythonban:

```
In [31]: x = range(1,5); x
Out[31]: [1, 2, 3, 4]

In [32]: x.append(99); x
Out[32]: [1, 2, 3, 4, 99]

In [33]: x.
x.append   x.extend   x.insert   x.remove   x.sort
x.count    x.index    x.pop      x.reverse

In [33]: x.extend([1,1,1]); x
Out[33]: [1, 2, 3, 4, 99, 1, 1, 1]

In [34]: x.count(1); x
Out[34]: [1, 2, 3, 4, 99, 1, 1, 1]

In [35]: x.index(2)
Out[35]: 1

In [36]: x.insert(4,5); x
Out[36]: [1, 2, 3, 4, 5, 99, 1, 1, 1]

In [37]: x.pop(5); x
Out[37]: [1, 2, 3, 4, 5, 1, 1, 1]

In [38]: x.re
x.remove   x.reverse

In [38]: x.reverse(); x
Out[38]: [1, 1, 1, 5, 4, 3, 2, 1]

In [39]: x.remove(1); x
Out[39]: [1, 1, 5, 4, 3, 2, 1]
```

```
In [40]: x.sort(); x
Out[40]: [1, 1, 1, 2, 3, 4, 5]
```

Listaértelmezés

Írjuk egy listába a 100-nál kisebb nem negatív négyzetszámokat! Egy megoldás:

```
>>> negyzetek = []
>>> for x in range(10):
        negyzetek.append(x**2)

>>> negyzetek
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Egy egyszer? -- igazi Pythonos -- megoldás:

```
>>> negyzetek = [x**2 for x in range(10)]
```

Rövid listák létrehozásának egyszer? módja a listaértelmezés. Általános alakja:

```
[expression for expr in sequence1
            if condition1
            for expr2 in sequence2
            if condition2
            ...
            for exprN in sequenceN
            if conditionN]
```

Kis programrészek helyettesíthet?k vele. Például ***soroljuk fel az 1890 és 1915 közé es? szök?éveket!***

```
szoko = []
for ev in range(1890, 1922):
    if (ev%4 == 0 and ev%100 != 0) or (ev%400 == 0):
        szoko.append(ev)
print szoko
[1892, 1896, 1904, 1908, 1912, 1916, 1920]
```

Lépésenként egyre összetettebb listaértelmezéssel állítsuk el? ugyanezt:

```
>>> szoko = [ev for ev in range(1890, 1915)]
>>> szoko    # ez még az összes év
[1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1
>>> szoko = [ev for ev in range(1890, 1915) if ev%4 == 0]
>>> szoko    # ez a 4-gyel osztható évek listája
[1892, 1896, 1900, 1904, 1908, 1912]
>>> szoko = [ev for ev in range(1890, 1915)
                if (ev%4 == 0 and ev%100 != 0) or ev%400 == 0]
>>> szoko
[1892, 1896, 1904, 1908, 1912]
```

del utasítás

A del utasítással törölhet?k elemek, a lista tartalma vagy az egész lista:

```
>>> l = range(2,7)
>>> del l[2]; l
[2, 3, 5, 6]
```

```
>>> del l[:]; l
[]
>>> del l
>>> l
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'l' is not defined
```

Sorozat (tuple)

Változtathatatlan (immutable), mint a karakterlánc, m?veletei: szeletelés [], összef?zés (+), szorzás (*), összehasonlítás, tagsági viszony (in, not in)

```
>>> t = ()
>>> t
()
>>> t = 1, 5, 5, 5, 3   # itt elhagyható a zárójel
>>> t
(1, 5, 5, 5, 3)
>>> t[1]
5
>>> t.index(3)
4
>>> t.count(5)
3
>>> t[0] = 9
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Maga nem változtatható, de változtatható elemei lehetnek:

```
>>> t = [1,2], [3,4,5], 6
>>> t
([1, 2], [3, 4, 5], 6)
```

Az üres és az 1-hosszú sorozat kényes dolog, mert a ()-jeleket másra is használjuk:

```
>>> t=()
>>> t
()
>>> t = (3)
>>> type(t)
<type 'int'>
>>> t = 3,
>>> type(t)
<type 'tuple'>
```

A sorozatba való be- és kicsomagolás:

```
>>> t = 2, 3, 4
>>> t
(2, 3, 4)
>>> x, y, z = t
>>> x, y, z
(2, 3, 4)
>>> x
2
```

Halmaz (set)

A halmaz rendezetlen, duplikált elemek nélküli gy?jteményes adattípus. Tagsági viszony gyorsabb ellen?rzésére, duplikált elemek törlésére használható. A matematikai halmazm?veletek használhatók.

```
>>> a = set('matematika'); a
set(['a', 'e', 'i', 'k', 'm', 't'])
>>> b = set('matek'); b
set(['a', 'e', 'k', 'm', 't'])
>>> a - b     # halmazok különbsége
set(['i'])
>>> b - a
set([])
>>> a | b     # halmazok uniója (vagy)
set(['a', 'e', 'i', 'k', 'm', 't'])
>>> a & b     # halmazok metszete (és)
set(['a', 'm', 'k', 'e', 't'])
>>> a ^ b     # halmazok szimmetrikus differenciája (kizáró vagy)
set(['i'])
```

Bevitelnél a kapcsos zárójel is használható, kivéve az üres halmazt:

```
>>> t = {1, 2, 3, 4}
>>> type(t)
<type 'set'>
>>> t
set([1, 2, 3, 4])
>>> t = {}
>>> type(t)
<type 'dict'>
```

A listaértelmezéshez hasonlóan van halmazértelmezés is:

```
>>> {x for x in 'asztrológia' if x not in 'asztronómia'}
set(['l', 'g'])
```

Szótár (dict)

A szótár kulcs-érték párokból álló halmaz. Nem a 0, 1, 2,... számokkal van indexelve, mint a sorozat vagy a karakterlánc, hanem bármilyen nem változtatható objektumokkal. Ezeket nevezzük kulcsnak. A kulcs tehát lehet szám, karakterlánc, sorozat, de nem lehet lista. A kulcshoz tartozó érték bármi lehet. Egy szótárban minden kulcsból csak egy lehet.

```
>>> tel = {'Laci': 1243, 'Ági': 2221, 'Zoli': 3321}
>>> type(tel)
<type 'dict'>
>>> tel['Laci']
1243
>>> tel = dict([('Laci',1243), ('Ági',2221), ('Zoli',3321)])
>>> tel['Laci']
1243
>>> tel = dict(Laci=1243, Ági=2221, Zoli=3321)
  File "<stdin>", line 1
    tel = dict(Laci=1243, Ági=2221, Zoli=3321)
                          ^
SyntaxError: invalid syntax
```

```
>>> tel = dict(Laci=1243, Agi=2221, Zoli=3321)
>>> tel['Agi']
2221
```

A listaértelmezésnek szótárakra is van változata:

```
>>> {x: x**3 for x in (11, 12, 13)}
{11: 1331, 12: 1728, 13: 2197}
```

Hasító tábla (hash table)

Általában a **hash tábla** (hash table, **hasító tábla**) egy olyan adatszerkezet, amely egy hash függvény (hasító függvény) segítségével állapítja meg, hogy melyik kulcshoz milyen érték tartozik. A Python dict adattípusa hash tábla. A hash függvény minden objektumhoz egy véges tartományba es? egész számot rendel. A hash tábla kezelésekor kezelni kell az esetleges ütközéseket, azaz azokat az eseteket, amikor két különböz? kulcshoz azonos hash-érték tartozik. A Python a hash függvény értékkészletébe es? egészeken az identikus leképezés, tehát

```
>>> hash(1)
1
>>> hash(1.0)
1
>>> hash(1123)
1123
>>> hash(-1123)
-1123
>>> hash(111111111111111111111111111)
497554224548382901
```

Így e függvényhez könny? **ütközést** kreálni (a születésnapi paradoxon is besegít), de a gyakorlatban nagyon ritka, és a hash tábla kezelés föl van ilyen esetekre készülve (a kriptográfiai célokra használt hash függvényt úgy kell megkonstruálni, hogy ütközést belátható id?n belül ne lehessen könnyen találni).

```
>>> hash(0.1)
2576882278
>>> hash(2576882278)
2576882278
```

Csak változtathatatlan (immutable) objektumoknak lehet hash értéke:

```
>>> l = [1, 2]      # változtatható (mutable)
>>> c = "hash"      # változtathatatlan (immutable)
>>> hash(c)
7799588877615763652
>>> hash(l)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```