Tartalomjegyzék

Beépített függvények

Összetett adattípusokon általánosan értelmezett függvények

	tuple	set	list	dict
új elem	-	add	insert, append	(update)
elem törlése	-	remove	remove	(pop)
törlés mind	-	clear	-	clear
elem kivétele	-	pop	pop	pop
elem indexe	index	-	index	-
hányszor szerepel	count	-	count	-
másolás	-	copy	-	copy

Reference

- `add(<item>)` ? új elem hozzáadása
- `insert(<index>,<item>)` ? új elem beillesztése
- `append(<item>)` ? új elem hozzáf?zése
- `remove(<item>)` ? elem törlése
- `clear()` ? gy?jtemény kiürítése
- S: `pop()`
L: `pop([<index>])`
D: `pop(<key>[, dv])` ? egy elem visszaadása és törlése a gyüjteményb?l
- `count(<item>)` ? elem indexe, ha tartalmazza
? megszámolja és visszatér, hogy hányszor tartalmazza a gy?jtemény az adott elemet

- `copy()` ? gy?jtemény lemásolása
- `len(<coll>)` ? gy?jtemény hossza

Halmazok függvényei

```
S = set([pi, 'abc', 35, pi])
```

- `union(<coll>[, <coll>]*)`
  egyesíti a halmazokat, az eredménnyel visszatér
- `update(<coll>[, <coll>]*)`
  union - az eredmény bekerül az "http://wiki.math.bme.huels?"http://wiki.math.bme.hu halmazba (|=)
- `intersection(<coll>[, <coll>]*)`, `intersection_update(<coll>[, <coll>]*)`
  visszaadja a közös részt ill. halmaz felülírása az eredménnyel (&, &=)
- `difference(<coll>[, <coll>]*)`, `difference_update<coll>[, <coll>]*)`
  különbség visszaadása ill. halmaz felülírása az eredménnyel (-, -=)
- `discard(<item>)`, `remove(<item>)`
  elem törlése a halmazból, remove esetén hiba, ha nem létezik
- `isdisjoint(coll)`
  igaz, ha nincs közös elemük
- `issubset(<coll>)`
  részhalmaza-e a paraméter (<, <=)
- `issuperset(<coll>)`
  a halmaz részhalmaza-e a paraméternek (>, >=)

Lista függvényei

```
L = ['a', 'orange', 1, 1.56]
```

- `sort([cmp=None,] [key=None,] [reverse=False])`
  rendezi a lista elemeit
- `extend(<coll>)`
  hozzáadja a paraméterként átadott gy?jtemény elemeit a listához (+)
- `reverse()`
  megfordítja a lista elemeinek sorrendjét
- * operator
  lemásolja és összef?zi újra a lista elemeit

Szótár függvényei

```
D = {'a':4, 'orange':5, 1:5, 1.56:5.67}
```

- `get(<key>[, <dv>])`
  a kulcshoz tartozó értékkel tér vissza, ha létezik;
  egyébként `None` ill. `dv`, ha megadtuk
- `has_key(<key>)`
  igaz, ha a szótár eleme a kulcs
- `items()`
  a szótár elemeit adja vissza egy listában (kulcs, érték) tuple formájában

- `keys()`
  a kulcsokat adja vissza egy listában
- `iterkeys()`
  szótár kulcsain lépked? iterátorral tér vissza
- `iteritems()`
  szótár értékein lépked? iterátorral tér vissza
- `popitem()`
  szótár egy elemét eltávolítja és visszaadja (kulcs, érték) tuple formájában
- `update(<coll>)`
  dict vagy tuple lista elemeit illeszti be a szótárba
- `setdefault(<key>[, value])`
  beilleszti az elemet és visszatér az értékkel
- `values()`
  visszatér az értékekb?l képzett listával

Függvények számításokhoz

- `var(<string>)`
  szimbolikus számításokhoz hozhatunk létre változókat
- `subs(<args>)`
  kifejezés függvénye, amivel behelyettesíthetjük a változókat a paraméterben felsorolt értékekkel és visszakapjuk a kifejezés numerikus eredményét
- `float(<kif|num|string>)`
  lebeg?pontos számmá próbálja alakítani a megadott paramétert (lsd. RR, RDF)
- `n()`
  kifejezések függvénye, a kiértékelt kifejezést adja vissza
- `exp(3)`
  e3
- `pow(x,z)`
  xz
- `log(exp(3))` ? természetes alapú log
  3
- `sqrt(16)` ? négyzetgyök
  4
- `sin(pi)`, `cos(pi)`, `tan(pi)` ? szögfüggvények
  `arcsin()`, ... ? inverzeik
- `floor(4.5)` ? alsó egészrész
  4
- `ceil(4.6)` ? fels? egészrész
  5

Egyenl?ség vizsgálata lebeg?pontos számok esetén

Fontos megjegyezni, hogy a lebeg?pontos számok mindig közelít? értékek, így két lebeg?pontos szám, ami pl. számítás eredménye, valószín?leg nem egyezik meg egymással!

```
sage: sqrt(4.0)^2 == 4.0
True
```

DE

```
sage: sqrt(5.0)^2 == 5.0
```

```
False
```

Megoldás:

```
sage: abs(sqrt(5.0)^2 == 5.0) < 10^(-10)
True
```

Kifejezeséken használt függvények

Az algebrai kifejezeséket összeggé alakíthatjuk az `expand()` függvénnyel vagy `.expand()` metódussal, ill. szorzattá a `.factor()`-ral:

```
sage: a,b = var('a','b')
sage: expand((a+b)^2)
a^2+2*a*b+b^2

sage: x = var('x')
sage: (x^2-1).factor()
(x-1)*(x+1)
```

A kifejezéseket a `simplify()` vagy a `full_simplify()` metódussal tudjuk egyszer?síteni:

```
sage: p,x = var('p','x')
sage: t = p/pow(p,x)
sage: t.simplify()
p^(-x + 1)

sage: x = var('x')
sage: t = (x + 1)**3-x**3-1
sage: t.full_simplify()
3*x^2 + 3*x
```

Egyenletek megoldása

`solve()` ? megpróbálja szimbolikus átalakításokkal meghatározni az egyenlet gyökét

```
sage: (x + 1/x == 4).solve(x)
[x == -sqrt(3) + 2, x == sqrt(3) + 2]
```

`roots()` ? mint a solve, csak az eredményt numerikusan és a gyök multiplicitását adja vissza

```
sage: (x**2 + 2*x + 1 == 0).roots(x)
[(-1, 2)]

sage: (x**3 - x**2 - x + 1 == 0).roots(x)
[(-1, 1),(1, 2)]
```

Az el?z? metódusok néha nem tudnak explicit alakban megoldani egy egyenletet, vagy nem ad meg minden gyököt, vagy hamis gyököket talál. Ekkor a `find_root()` segíthet egy numerikus megoldást találni a megadott intervallumon
(legyen pl: 0 < y < /2):

```
sage: y = var(?y?)
sage: find_root(cos(y) == sin(y), 0, pi/2)
0.78539816339744839
```

Az algoritmus mindig csak egy gyököt talál meg és az is egy közelít? érték.