

Množice

Množice so podobne seznamom, a s to razliko, da lahko vsebujejo vsak element samo enkrat.

Po drugi strani (in ne le po drugi strani, tudi tehnično) pa so podobne slovarjem. Vsebujejo lahko le elemente, ki so nespremenljivi, poleg tega pa lahko zelo hitro ugotovimo, ali množica vsebuje določen element ali ne, na podoben način kot pri slovarjih hitro ugotovimo, ali vsebujejo določen ključ ali ne.

Množice so podobne slovarjem tudi zato, ker nimajo vrstnega reda. In to še v veliko večji meri kot slovarji: elementi slovarja so urejeni po vrstnem redu dodajanja (od Pythona 3.6 naprej; podobne strukture v drugih jezikih pa navadno niti to), elementi množice pa ne, poleg tega pa se lahko njihov vrstni red, kakor ga pokaže zanka `for`, popolnoma premeša, ko iz nje dodamo ali odvzamemo kak element.

Končno, množice so podobne slovarjem po tem, da jih zapišemo z zavitimi oklepaji, tako kot smo vajeni pri matematiki.

```
danasnji_klici = {"Ana", "Cilka", "Eva"}
```

Tako lahko sestavimo le neprazno množico. Če napišemo le oklepaj in zaklepaj, `{}`, pa dobimo slovar. (Čemu so se odločili, naj bo to slovar, ne množica? Slovar je bil prej, množice je Python dobil kasneje. Zato. Poleg tega pa slovarje potrebujemo res velikokrat, množice pa precej redkeje.) Če hočemo narediti prazno množico, rečemo

```
prazna = set()
```

"Funkcija" `set` je malo podobna "funkciji" `int`: damo ji lahko različne argumente, pa jih bo spremenila v množico. Damo ji lahko, recimo, seznam, pa bomo dobili množico z vsemi elementi, ki se pojavijo v njem.

```
set([1, 2, 3])
```

```
{1, 2, 3}
```

```
set([6, 42, 1, 3, 1, 1, 6])
```

```
{1, 3, 6, 42}
```

Poleg seznamov lahko množicam podtaknemo karkoli, prek česar bi lahko šli z zanko `for`, recimo niz ali slovar.

```
s = set("Benjamin")
```

```
s
```

```
{'B', 'a', 'e', 'i', 'j', 'm', 'n'}
```

Vprašamo se lahko, ali množica vsebuje določen element.

```
"e" in s
```

```

True
"u" in s
False
s.add("u")
"u" in s
True
s.add("a")
s.add("a")
s.add("a")

s

```

```
{'B', 'a', 'e', 'i', 'j', 'm', 'n', 'u'}
```

Na koncu smo poskušali v množico dodati element, ki ga že vsebuje. Python se ne pritoži, vendar tudi ne stori ničesar: množica vsak element vsebuje le enkrat.

Če imamo dve množici, lahko izračunamo njuno unijo, presek, razliko (elementi, ki se pojavijo v prvi, ne pa tudi v drugi množici) in simetrično razliko (elementi, ki se pojavijo v eni množici in v drugi) ...

```

u = {1, 2, 3}
v = {3, 4, 5}
u | v
{1, 2, 3, 4, 5}

u & v
{3}

u - v
{1, 2}

u ^ v
{1, 2, 4, 5}

```

Tako kot lahko pri številih uporabimo += za prištevanje (in namesto `x = x + a` pišemo `x += a`), in, podobno, odštejemo, primnožimo in razdelimo z -=, *= in /=, lahko tudi priunijamo, odsekamo ali odštejemo množico z &=, |= in -=. Tako je, na primer `u &= v` isto kot `u = u & v`.

Preverimo lahko tudi, ali je neka množica podmnožica (ali nadmnožica druge). To najpreprosteje storimo kar z operatorji za primerjanje.

```

u = {1, 2, 3}
{1, 2} <= u

```

True

$\{1, 2, 3, 4\} \leq u$

False

$\{1, 2, 3\} \leq u$

True

$\{1, 2, 3\} < u$

False

$\{1, 2, 3\}$, je podmnožica u -ja, ni pa njegove *prava podmnožica*, saj vsebuje kar cel u .

Z množicami je mogoče početi še marsikaj zanimivega - vendar bodi dovolj. Več primerov pa si bomo ogledali sproti.