03 množice

January 28, 2024

0.1 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 zapišemo z zavitimi oklepaji, tako kot smo vajeni pri matematiki.

[1]:
```
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

[2]:
```
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.

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

[3]:
```
{1, 2, 3}
```

[4]:
```
set(range(5))
```

[4]:
```
{0, 1, 2, 3, 4}
```

[5]:
```
set([6, 42, 1, 3, 1, 1, 6])
```

[5]:
```
{1, 3, 6, 42}
```

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

[6]:
```
set("Benjamin")
```

[6]:
```
{'B', 'a', 'e', 'i', 'j', 'm', 'n'}
```

V množico lahko dodajamo elemente in vprašamo se lahko, ali množica vsebuje določen element.

[7]:
```
s = set("Benjamin")
```

[8]:
```
"e" in s
```

[8]:
```
True
```

[9]:
```
"u" in s
```

[9]:
```
False
```

[10]:
```
s.add("u")
```

[11]:
```
"u" in s
```

[11]:
```
True
```

[12]:
```
s.add("a")
s.add("a")
s.add("a")
s
```

[12]:
```
{'B', 'a', 'e', 'i', 'j', 'm', 'n', 'u'}
```

Na koncu smo poskušali v množico dodati element, ki ga že vsebuje. To seveda ne gre, množica vsak element vsebuje le enkrat.

Za brisanje elementov iz množice imamo dve metodi: `remove` in `discard`. Razlikujeta se potem, kaj storita, če elementa ni v množici: prva javi napako, druga ne naredi ničesar.

0.1.1 Presek in unija

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) ...

[13]:
```
u = {1, 2, 3}
v = {3, 4, 5}
u | v
```

[13]:
```
{1, 2, 3, 4, 5}
```

[14]:
```
u & v
```

[14]:
```
{3}
```

[15]:
```
u - v
```

[15]:
```
{1, 2}
```

[16]:
```
u ^ v
```

[16]:
```
{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`.

0.1.2 Podmnožice

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

[17]:
```
u = {1, 2, 3}
```

[18]:
```
{1, 2} <= u
```

[18]:
```
True
```

[19]:
```
{1, 2, 3, 4} <= u
```

[19]:
```
False
```

[20]:
```
{1, 2, 3} <= u
```

[20]:
```
True
```

[21]:
```
{1, 2, 3} < u
```

[21]:
```
False
```

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

0.1.3 Kdaj uporabiti množice?

Množice so tipičen primer podatkovne strukture (uh, učen izraz! za prvi letnik bi bilo čisto dovolj govoriti o podatkovnih tipih), ki nam mora priti na misel v pravem trenutku. Vse, kar počnemo z množicami, bi lahko naredili tudi s seznami, le počasneje in daljše bi bilo. Zdaj, ko veste za množice, jih morate imeti pač stalno malo v mislih.