Slovar

Osnove programiranja

Nejc Ilc

Iz množice v slovar

Prejšnji teden smo spoznali množico. Slovar je njen bratranec.

lastnost	množica	slovar
je spremenljiv objekt (iz plastelina)	✓	✓
elementi [1] morajo biti edinstveni (unikatni)	✓	✓
elementi so lahko samo nespremenljivi objekti	✓	✓
elementi niso urejeni, njihov vrstni red je poljuben	✓	✘
elementov ne označujemo z indeksi	✓	✓

1. Temu, čemur pri množici rečemo element množice, je pri slovarju ključ.

Primer `shramba`

Slovar je malo bolj napredna množica.

V svoji shrambi živil (pri nas temu lepo rečemo *špajz*) pogrešamo malo sistematičnosti (in precej čokolade). Začnimo voditi evidenco. Najprej popišimo množico živil:

```
shramba = {'banana', 'kruh', 'mleko'}
```

Zdaj pa bi radi zabeležili tudi količino živil. Hm, z množico to ne gre zlahka. Gre pa s slovarjem:

```
shramba = {'banana': 5, 'kruh': 1, 'mleko': 12}
```

Kaj pove zgornji zapis? Enostavno, imamo 5 banan, en hleb kruha in 12 litrov mleka. Nič čokolade 😭 Treba bo v trgovino …

Fotografija: Maria Lin Kim [vir]

Izrazoslovje

Element, ključ, vrednost

Slovar je zbirka elementov (ang. *items*), ki so oblike ključ: vrednost (ang. *key: value*).

Za *ključ* veljajo isti zakoni kot za element množice - biti mora edinstven in nespremenljiv ("kamen").

Vrednost je lahko katerikoli objekt.

Slovar si torej lahko predstavljamo takole:

```
slovar = {  ključ_1: vred_1, ključ_2: vred_2, ...}
          └── element 1 ──┘ └── element 2 ──┘
```

```
shramba = {
  'banana': 5, # element 1
  'kruh':   1, # element 2
  'mleko': 12  # element 3
  }
```

Naslavljanje

Iz slovarja potegnemo vrednost tako, da uporabimo ključ. Slovar nima številčnih indeksov, tako kot niz, seznam, terka in ostala zaporedja. Njegovi indeksi so kar ključi.

```
>>> shramba['banana']
5
```

Pravkar smo postali hudo lačni, privoščimo si banano. Sedaj moramo popraviti vrednost v shrambi. Zopet uporabimo ključ:

```
>>> shramba['banana'] = 4
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12}
```

Ustvarimo slovar

Prazen slovar

Če napišemo

```
a = {}
```

s tem ne ustvarimo prazne množice, kot bi si mislili matematiki, temveč prazen slovar.

```
>>> type(a)
<class 'dict'>
```

Prav tako lahko pokličemo konstruktor za gradnjo novih objektov tipa slovar (`dict`):

```
>>> a = dict()
>>> a
{}
```

Z začetnimi vrednostmi

To smo že videli:

```
a = {'konj': 4, 'riba': 0, 'človek': 2}
```

Slovar iz seznama terk

Če konstruktorju podamo seznam terk, pri čemer so terke pari, bo iz njega naredil slovar. Poglejmo:

```
seznam_terk = [
    ('konj', 4),
    ('riba', 0),
    ('človek', 2)
    ]
a = dict(seznam_terk)

>>> a
{'konj': 4, 'riba': 0, 'človek': 2}
```

Fotografija: Nathan Dumlao [vir]

Ključi

Kaj vse je lahko ključ

Ključ je lahko objekt, ki ima nespremenljiv podatkovni tip, torej: `int`, `float`, `str`, `tuple`.

Aha, tudi celo število! Potem lahko slovar napišemo tako:

```
polica = {0: 'banana', 1: 'mleko'}
```

Pravkar smo naredili nekaj, kar se obnaša podobno kot seznam.

```
>>> polica[0]
'banana'
>>> polica[1] = 'kruh'
>>> polica
{0: 'banana', 1: 'kruh'}
```

Ko nimaš pravega ključa ...

Dostop do vrednosti v slovarju gre preko ključa

```
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12}
```

```
>>> shramba['sir']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'sir'
```

Ključ `'sir'` ne obstaja, zato dobimo napako tipa `KeyError`.

Da se izognemo napaki, najprej preverimo, ali ključ obstaja. Uporabimo operator vsebovanosti `in`:

```
>>> 'sir' in shramba
False
```

Napišimo funkcijo, ki preveri zalogo živila v shrambi:

```
def preveri_zivilo(shramba, zivilo):
    if zivilo in shramba:
        return shramba[zivilo]
    else:
        return 'Živila ' + zivilo + ' ni v shrambi.'
```

Testirajmo:

```
>>> preveri_zivilo(shramba, 'banana')
4
>>> preveri_zivilo(shramba, 'jabolko')
'Živila jabolko ni v shrambi.'
```

Podobno obnašanje dobimo, če uporabimo metodo `get()`, več o tem čez nekaj strani.

Dodajmo/spremenimo vrednost

Gremo nazaj na naš primer s shrambo.

Šli smo v trgovino in kupili tablico temne čokolade. Radi bi jo dodali v shrambo. Dodati moramo torej nov par *ključ: vrednost*. To storimo preprosto tako, da naslovimo željeni ključ. Če v slovarju še ne obstaja, se bo ustvaril.

```
shramba['čokolada'] = 1
```

Če ključ že obstaja, se bo vrednost na tem ključu spremenila/prepisala.

```
>>> shramba['čokolada'] = 2
>>> shramba['čokolada']
2
>>> shramba['čokolada'] += 1
>>> shramba['čokolada']
3
```

Napišimo funkcijo za dodajanje živil v shrambo. Sprejme tri argumente: slovar `shramba`, niz z nazivom živila `zivilo` in količino živila `kolicina`, ki pa ni obvezen argument - privzeto je 1. Ker je slovar spremenljiv objekt, bodo vse spremembe na njem vidne tudi, ko se funkcija zaključi.

```
def dodaj_zivilo(shramba, zivilo, kolicina=1):
    if zivilo in shramba:
        shramba[zivilo] += kolicina
    else:
        shramba[zivilo] = kolicina
```

```
>>> dodaj_zivilo(shramba, 'čokolada')
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12, 'čokolada': 1}
>>> dodaj_zivilo(shramba, 'čokolada', 2)
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12, 'čokolada': 3}
```

Izbrišimo element slovarja

Uporabimo že poznani stavek `del`

Element lahko izbrišemo iz slovarja na tri načine. Prvi je z uporabo stavka `del`, druga dva pa sta metodi slovarja `popitem()` oziroma `pop()`. Metodi si bomo ogledali pozneje.

```
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12, 'čokolada': 3}
>>> del shramba['mleko']
>>> shramba
{'banana': 4, 'kruh': 1, 'čokolada': 3}
```

V naš program za upravljanje s shrambo dodajmo še eno funkcijo, in sicer `porabi_zivilo()`, ki bo pogledala, ali neko živilo je v shrambi in če je, mu bo zmanjšala količino. Če porabimo vso količino živila, ga izbrišemo iz shrambe.

```
def porabi_zivilo(shramba, zivilo, kolicina=1):
    # Preverimo zalogo živila. Če ga ni, dobimo niz.
    zaloga = preveri_zivilo(shramba, zivilo)
    if type(zaloga) is not str:
        # Če smo želeli porabiti več kot imamo,
        # se omejimo na toliko, kolikor imamo.
        if kolicina > zaloga:
            kolicina = zaloga
        # Če porabimo vso zalogo, izbrišemo živilo.
        if kolicina == zaloga:
            del shramba[zivilo]
        # Sicer pa zgolj zmanjšamo količino živila.
        else:
            shramba[zivilo] -= kolicina
```

```
>>> dodaj_zivilo(shramba, 'čokolada')
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12, 'čokolada': 1}
>>> dodaj_zivilo(shramba, 'čokolada', 2)
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12, 'čokolada': 3}
```

Metode slovarja

Najprej si oglejmo metode, ki vračajo *pogled* (*view*) na slovar - to so objekti, ki se dinamično spreminjajo ob spremembi slovarja.

`slovar.keys()`

Metoda vrne ključe slovarja.

```
>>> kljuci = shramba.keys()
>>> kljuci
dict_keys(['banana', 'kruh', 'mleko', 'čokolada'])
```

Vrnjeni objekt `kljuci` se posodobi, ko se posodobi slovar. Izbrišimo ključ `'čokolada'` in preverimo:

```
>>> del shramba['čokolada']
>>> kljuci
dict_keys(['banana', 'kruh', 'mleko'])
```

`slovar.values()`

Vrne vrednosti slovarja, zopet kot *pogled*, ki se ob posodobitvah spreminja.

```
>>> shramba.values()
dict_values([4, 1, 12])
```

`slovar.items()`

Vrne elemente slovarja, ki so organizirani v seznam terk (nam že poznana oblika).

```
>>> shramba.items()
dict_items([('banana', 4), ('kruh', 1), ('mleko', 12)])
```

Metode slovarja

Nadaljujmo s pregledom uporabnih metod.

`slovar.copy()`

Naredi plitvo kopijo slovarja.

```
>>> klet = shramba.copy()
>>> klet['vino'] = 3
>>> klet
{'banana': 4, 'kruh': 1, 'mleko': 12, 'vino': 3}
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12}
```

`slovar.clear()`

```
>>> klet.clear()
>>> klet
{}
```

`slovar.get(k, d=None)`

Vrne vrednost pod ključem `k`, če ta obstaja. Če ne, nam vrne vrednost `d`, ki je privzeto `None`. Podobno funkciji `preveri_zivilo()`, kajne?

```
>>> shramba.get('mleko', 'Živila ni v shrambi.')
12
>>> shramba.get('sok', 'Živila ni v shrambi.')
'Živila ni v shrambi.'
```

`slovar.update(slovar2)`

Združi skupaj slovarja `slovar` in `slovar2`.

```
>>> shramba.update({'sir': 3})
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12, 'sir': 3}
```

Metode slovarja

`dict.fromkeys(z, v=None)`

Zgradi slovar iz zaporedja `z` (niz, seznam, terka, ...), pri čemer vsi elementi dobijo vrednost `v`.

```
>>> ocene = dict.fromkeys(['Janko', 'Metka'], 5)
>>> ocene
{'Janko': 5, 'Metka': 5}
```

`slovar.popitem()`

Odstrani in vrne zadnji element slovarja (terko):

```
>>> shramba.popitem()
('sir', 3)
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12}
```

`slovar.pop(k, d)`

Če ključ `k` obstaja v slovarju, ta metoda odstrani ključ in vrne vrednost elementa. Če ključ ne obstaja, vrne `d`. Če `d` ni podan in ključ ne obstaja, vrne `KeyError`.

```
>>> shramba
{'banana': 4, 'kruh': 1, 'mleko': 12}
>>> shramba.pop('kruh')
1
>>> shramba.pop('kruh')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'kruh'
```

```
>>> shramba.pop('kruh', 'Ključ ne obstaja')
'Ključ ne obstaja'
```

Sprehod čez slovar

Slovar je zaporedje, preko katerega se lahko sprehajamo z zanko `for`

Zanka po ključih

Če čisto naivno napišemo zanko `for` takole:

```
for zivilo in shramba:
    print(zivilo)
```

dobimo sledeči izpis vseh ključev slovarja:

```
banana
mleko
```

Očitno si lahko privoščimo bananin frapé. Izpišimo še količine:

```
for zivilo in shramba:
    print(zivilo, ': ', shramba[zivilo], sep='')
```

Zanka po elementih

Zanko na koncu levega stolpca lahko elegantneje napišemo z uporabo metode `items()` in razpakiranja terke:

```
for zivilo, kolicina in shramba.items():
    print(zivilo, ': ', kolicina, sep='')
```

Dobimo:

```
banana: 4
mleko: 12
```

Urejanje elementov slovarja - po ključih

Pri seznamih smo spoznali metodo `sort()`, ki uredi objekt, ki je klical to metodo. Obstaja tudi funkcija `sorted()`, ki vrne nov (urejen) objekt. Če mu kot argument podamo slovar, to razume kot seznam ključev. Pred urejanjem v shrambo dodajmo ananas.

```
>>> shramba['ananas'] = 5
>>> shramba
{'banana': 4, 'mleko': 12, 'ananas': 5}
>>> sorted(shramba)
['ananas', 'banana', 'mleko']
>>> shramba
{'banana': 4, 'mleko': 12, 'ananas': 5}
```

`sorted` je vrnil nov objekt (urejen seznam ključev). Objekt `shramba` se ni spremenil.

Napišimo funkcijo `uredi()`, ki bo vrnila urejene ključe slovarja. Način urejanja podamo z argumentom `padajoce`.

```
def uredi(shramba, po_kolicini=False, padajoce=False):
    if po_kolicini:
        # To bomo naredili na naslednji strani
        pass
    else:
        s = sorted(shramba.items(), reverse=padajoce)
    return s
```

```
>>> uredi(shramba)
[('ananas', 5), ('banana', 4), ('mleko', 12)]
>>> uredi(shramba, padajoce=True)
[('mleko', 12), ('banana', 4), ('ananas', 5)]
```

Urejanje elementov slovarja - po vrednostih

Če želimo shrambo urediti po količinah živil (po vrednosti), moramo biti malo zviti:

1. funkciji `sorted` moramo kot argument podati seznam terk oblike `(kljuc, vrednost)` in
2. definirati moramo funkcijo, ki bo vrnila element terke, po katerem urejamo.

Začnimo s točko 2 in definirajmo preprosto funkcijo, ki sprejme zaporedje in vrne drugi element tega zaporedja:

```
def dobi_vrednost(item):
    return item[1]
```

Sedaj dopolnimo funkcijo `uredi()` s prejšnje strani:

```
def uredi(shramba, po_kolicini=False, padajoce=False):
    if po_kolicini:
        s = sorted(
            shramba.items(),
            reverse=padajoce,
            key=dobi_vrednost)
    else:
        s = sorted(
            shramba.items(),
            reverse=padajoce)
    return s
```

```
>>> uredi(shramba, po_kolicini=True, padajoce=True)
[('mleko', 12), ('ananas', 5), ('banana', 4)]
```

Pa smo!

Napredna stopnja `glasba`

Radi bi si uredili svojo glasbeno zbirko.

Napravimo zgled večnivojske zgradbe slovarja. V slovarju bomo kot vrednosti hranili slovarje. V zbirki hranimo izvajalce in za vsakega od njih še albume. Za vsak album imamo tudi podatek o letu izdaje.

```
# Naredimo najprej prazen slovar
zbirka = {}
# Dodamo prvega izvajalca,
# njegovi albumi so zaenkrat prazen slovar
zbirka['Vlado Kreslin'] = {}
# Dodamo prvi album in letnico izdaje
zbirka['Vlado Kreslin']['Kreslinčice'] = 2002
print(zbirka)
```

Izpis:

```
{'Vlado Kreslin': {'Kreslinčice': 2002}}
```

Sedaj dodajmo podporne funkcije, ki bodo omogočale:

- dodajanje izvajalca v zbirko,
- dodajanje albuma izvajalcu, leto izdaje ni obvezno,
- popravke v letu izdaje za nek album,
- izbris izvajalca,
- izpis celotne zbirke,
- izpis vseh izvajalcev,
- izpis vseh albumov, pri čemer lahko določimo, od katerega leta izdaje naprej,
- izpis vseh albumov določenega izvajalca.

Glej program `glasba.py` z izdelano rešitvijo.