

Slovarji

Vzemimo, spet, podatke o kolesarjenju, tiste, s štirimi stolpci. Za zdaj smo ugotovili le, kolikšni sta skupna prevožena razdalja in višina, podatek o kolesu pa kar zanemarili. Lahko bi izračunali, recimo, koliko je prevozil s Canyonom.

```
vozenj_canyon = 0
razdalja_canyon = 0
visina_canyon = 0

for vrstica in open("kolesa.txt"):
    kolo, razdalja, visina = vrstica.split(",")
    if kolo == "Canyon":
        vozenj_canyon += 1
        razdalja_canyon += int(razdalja)
        visina_canyon += int(visina)

print("Voženj:", vozenj_canyon)
print("Razdalja:", razdalja_canyon)
print("Višina:", visina_canyon)
```

```
Voženj: 28
Razdalja: 2932
Višina: 25630
```

Zdaj pa isto še za Cube, Nakamura in Stevens. Če damo to v isti program, bomo imeli štirikar po tri spremenljivke.

Primeri pa se lahko še kaj hujšega: v družini nas je pet in imamo osem koles. (Štirje po enega in eden štiri.)

Kako lepo bi bilo, če bi lahko imeli eno samo spremenljivko `vozenj`, ki bi imela "predalčke" Canyon, Cube, Nakamura in Stevens. In, seveda, spremenljivki `razdalja` in `visina`, vsako s štirimi predalčki.

Očitno to obstaja, sicer si tega ne bi želel, ne? Takšnim spremenljivkam - točneje, taki vrsti spremenljivk rečemo slovar - po angleško *dictionary*, ime Pythonovega podatkovnega tipa pa je `dict`.

Za začetek pripravimo nekaj preprostejšega: slovar z višinami hribov. Takšen bi bil:

```
hribi = {"Triglav": 2864, "Storžič": 2132, "Grintovec": 2558}
```

Slovar hribi ima predalčke "Triglav", "Storžič" in "Grintovec", vrednosti, shranjene v teh predalčkih pa so 2864, 2132 in 2558. Kot vidimo, slovar pripravimo tako, da med zavite oklepaje zapišemo pare z imenom predalčka (ki mu učeno rečemo *ključ*, *key*) in pripadajočimi *vrednostmi* (*value*), ločimo pa jih z dvopičji.

Python nam seveda lahko izpiše celotno vsebino slovarja.

```
hribi
```

```
{'Triglav': 2864, 'Storžič': 2132, 'Grintavec': 2558}
```

Do posameznih vrednosti v slovarju pridemo tako, da "predalček" podamo v oglatih oklepajih.

```
hribi["Triglav"]
```

```
2864
```

```
hribi["Storžič"]
```

```
2132
```

Če kak ključ ne obstaja, Python to jasno pove.

```
hribi["Stol"]
```

```
-----  
KeyError                                Traceback (most recent call last)  
Cell In[6], line 1  
----> 1 hribi["Stol"]
```

```
KeyError: 'Stol'
```

Predalčki se vedejo kot spremenljivke. Triglavu lahko priredimo drugo višino

```
hribi["Triglav"] = 2800
```

```
hribi
```

```
{'Triglav': 2800, 'Storžič': 2132, 'Grintavec': 2558}
```

Lahko mu tudi prištejemo (zdaj manjkajočih) 64 metrov.

```
hribi["Triglav"] += 64
```

```
hribi
```

```
{'Triglav': 2992, 'Storžič': 2132, 'Grintavec': 2558}
```

Skratka, kot katerakoli druga spremenljivka.

Oboroženi z novim znanjem izračunajmo kolesarsko statistiko.

```
vozenj = {"Canyon": 0, "Cube": 0, "Nakamura": 0, "Stevens": 0}  
razdalje = {"Canyon": 0, "Cube": 0, "Nakamura": 0, "Stevens": 0}  
visine = {"Canyon": 0, "Cube": 0, "Nakamura": 0, "Stevens": 0}
```

```
for vrstica in open("kolesa.txt"):  
    kolo, razdalja, visina = vrstica.split(",")  
    vozenj[kolo] += 1  
    razdalje[kolo] += int(razdalja)  
    visine[kolo] += int(visina)
```

Pa zdaj? No, to kar potrebujemo je v slovarjih.

vozenj

```
{'Canyon': 28, 'Cube': 43, 'Nakamura': 29, 'Stevens': 0}
```

razdalje

```
{'Canyon': 2932, 'Cube': 2939, 'Nakamura': 488, 'Stevens': 0}
```

visine

```
{'Canyon': 25630, 'Cube': 70947, 'Nakamura': 1499, 'Stevens': 0}
```

Kaj pa, če bi radi to *lepo* izpisali? Recimo podatke za *vsako kolo*?

Čim rečemo "za vsako" nas to spomni na ... zanko `for`. Zanko `for` smo doslej naganjali prek datotekm, ko smo morali ponoviti kos programa za *vsako vrstico datoteke*. Kaj se zgodi, če jo poskusimo spustiti čez slovar?

```
for bogvekajboto in vozenj:
    print(bogvekajboto)
```

Canyon

Cube

Nakamura

Stevens

Zanka `for` daje ključne slovarja. Prav, če imamo ključ, potem že znamo priti do vrednosti.

```
for kolo in vozenj:
    print(kolo, "-", vozenj[kolo], "vozenj")
```

Canyon - 28 vozenj

Cube - 43 vozenj

Nakamura - 29 vozenj

Stevens - 0 vozenj

Ker imajo vsi trije slovarji tako ali tako iste ključne, lahko mimogrede izpišemo še skupno prevoženo višino in razdaljo.

```
for kolo in vozenj:
    print(kolo, "-", vozenj[kolo], "vozenj,", razdalje[kolo], "km,", visine[kolo], "m.")
```

Canyon - 26 vozenj, 2766 km, 26392 m.

Cube - 43 vozenj, 3174 km, 66705 m.

Nakamura - 22 vozenj, 439 km, 1119 m.

Stevens - 9 vozenj, 607 km, 3395 m.

Nadvse imenitno. No, kar imenitno. Nad vse(m) imenitno pa še ni, ker bo postalo še imenitnejše. :)

Dodajanje v slovar

Program se začne z

```
vozenj = {"Canyon": 0, "Cube": 0, "Nakamura": 0, "Stevens": 0}
razdalje = {"Canyon": 0, "Cube": 0, "Nakamura": 0, "Stevens": 0}
visine = {"Canyon": 0, "Cube": 0, "Nakamura": 0, "Stevens": 0}
```

Tule je videti nekaj prostora za izboljšave. Kot prvo: moramo res tipkati vsa ta imena koles in ničle? Ne bi šlo brez?

Kot drugo: da to deluje, moramo vnaprej vedeti, katere znamke koles se bodo pojavile v datoteki. Kaj, če to ni vnaprej znano? Kako bi programirali v tem primeru?

Naučiti se bomo morali tri stvari:

1. ustvariti prazen slovar,
2. preveriti, ali slovar ima določen predalček in
3. v primeru, da ga nima, dodati nov predalček.

Vse troje se bo izkazalo za čisto preprosto.

Kako sestaviti prazen slovar, zlahka uganemo. Pač ničesar ne zapišemo med oklepaje.

```
vozenj = {}
vozenj
{}

```

Kako preveriti, ali slovar vsebuje določen ključ? Besedo `in` smo doslej uporabljali v zanki `for`, ko smo pisali, recimo `for kolo in vozenj`. Lahko jo tudi v `if`, oziroma, brez `if`, v logičnem izrazu, kot operator. Operator `a in c` vrne `True`, če `c` vsebuje `a`.

Še vedno imamo pri roki

```
razdalje
{'Canyon': 2766, 'Cube': 3174, 'Nakamura': 439, 'Stevens': 607}
```

Vidimo, da slovar vsebuje `"Canyon"`, ne pa `"Focus"`. Operator `in` se s tem videnjem strinja.

```
"Canyon" in razdalje
```

```
True
```

```
"Focus" in razdalje
```

```
False
```

Odlično, znamo preveriti, ali predalček obstaja. Kljukica pod točko 2.

Zdaj pa še dodajanje predalčka.

```
razdalje["Focus"] = 0

razdalje
{'Canyon': 2766, 'Cube': 3174, 'Nakamura': 439, 'Stevens': 607, 'Focus': 0}

hribi["Šmarna gora"] = 667

hribi
{'Triglav': 2992, 'Storžič': 2132, 'Grintavec': 2558, 'Šmarna gora': 667}
```

Pa imamo boljši program za kolesarsko statistiko.

```
vozenj = {}
razdalje = {}
visine = {}

for vrstica in open("kolesa.txt"):
    kolo, razdalja, visina = vrstica.split(",")
    if not kolo in vozenj:
        vozenj[kolo] = razdalje[kolo] = visine[kolo] = 0
    vozenj[kolo] += 1
    razdalje[kolo] += int(razdalja)
    visine[kolo] += int(visina)

for kolo in vozenj:
    print(kolo, "-", vozenj[kolo], "vozenj,", razdalje[kolo], "km,", visine[kolo], "m.")

Nakamura - 22 vozenj, 439 km, 1119 m.
Cube - 43 vozenj, 3174 km, 66705 m.
Canyon - 26 vozenj, 2766 km, 26392 m.
Stevens - 9 vozenj, 607 km, 3395 m.
```

Ker vsi trije slovarji bodisi vsebujejo bodisi ne vsebujejo kolesa, preverjamo le, ali je kolo že v slovarju `vozenj`. Če ga ni tam, ga tudi v ostalih ni, torej ga dodamo v vse.

Ker se `not kolo in vozenj` bere čudno, obstaja poleg operatorja `in` tudi operator `not in`.

```
"Scott" not in vozenj
```

True

Slovarji - v splošnem

Ključni gornjega slovarja ("imena predalčkov") so bili nizi, vrednosti pa števila, `int`. Slovarji niso omejeni zgolj na te tipe.

Vrednosti so lahko karkoli: ne le `int` in `float`, temveč tudi nizi, logične vrednosti in vsi raznorazni tipi, ki jih bomo še spoznali pri predmetu. Slovar lahko vsebuje celo druge slovarje (uh, celo samega sebe!) kot vrednost.

Ključni so bolj omejeni. Ključni morajo biti nespremenljivi. Za naše potrebe in naše dosedanje znanje: ključ je lahko karkoli razen slovarja. "Prepovedanim" tipom ključev se bosta kmalu pridružila še seznam in množica, drugih prepovedanih tipov ključev pa pri tem predmetu ne boste spoznali. Vsaj ne takšnih, ki bi vas jih zamikalo uporabiti kot ključ.

Pri tem predmetu bodo ključni skoraj vedno nizi, včasih `int`-i in skoraj nikoli nič drugega.

Isti slovar ima lahko različne tipe ključev. In različne tipe vrednosti.

```
{"Ana": 129415, 4: True, 3.14: {}}
```

```
{'Ana': 129415, 4: True, 3.14: {}}
```

Če se komu zdi to slaba ideja, bo to zato, ker je dejansko slaba. Navadno bodo vsi ključni nekega slovarja in vse vrednosti nekega slovarja enakega tipa. Sicer ... no, kaj pravzaprav shranjujemo v slovarju, če se v njem pojavlja tako raznovrstna šara?!

Slovarji imajo tudi metode

Vsaka stvar v Pythonu ima metode. Niz, kot smo rekli, ducate. Slovarji jih imajo malo manj, a vseeno jih ne bomo našteali vseh, saj si jih tako ali tako ne bi zapomnili. Bomo že sproti videli, kar bomo potrebovali. (Kdor je neučakan, pa lahko pogleda dokumentacijo).

Vrednosti

Vzemimo slovar `visine`. Kakšna je skupna višina, ki jo je prevozil kolesar?

```
visine
```

```
{'Nakamura': 1119, 'Cube': 66705, 'Canyon': 26392, 'Stevens': 3395}
```

```
1119 + 66705 + 26392 + 3395
```

```
97611
```

Ne, nisem mislil tako. Ne bomo seštevali ročno. Kar program naj sešteje.

```
skupna = 0
```

```
for kolo in visine:
    skupna += visine[kolo]
```

```
skupna
```

```
97611
```

Če razmislimo, nas ključi tu pravzaprav ne zanimajo. Zanimajo nas le vrednosti. Tule bi nam prišlo bolj prav, če bi šla zanka `for` prek vrednosti, ne prek ključev.

Ni problema. Slovarji imajo metodo `values`, ki vrne vrednosti.

```
for visina in visine.values():
    print(visina)
```

```
1119
66705
26392
3395
```

In to seveda znamo sešteti.

```
skupna = 0
for visina in visine.values():
    skupna += visina
```

```
skupna
```

```
97611
```

Prej ko slej boste odkrili, da obstaja funkcija `sum`, ki lahko sešteje, kar ji podamo kot argument. Če ji podamo `visine.values()` bo seštela vse vrednosti v slovarju.

```
sum(visine.values())
```

```
97611
```

Torej:

```
print("Število voženj:", sum(vozenj.values()))
print("Skupna razdalja:", sum(razdalje.values()))
print("Skupna višina:", sum(visine.values()))
```

```
Število voženj: 100
```

```
Skupna razdalja: 6986
```

```
Skupna višina: 97611
```

Privzete vrednosti

Slovarji imajo metodo `setdefault(kljuc, vrednost)`, ki za podani ključ nastavi podano vrednost; če ključ že obstaja, pa ne stori ničesar. V vsakem primeru pa vrne vrednost, ki pripada temu ključu - bodisi obstoječo (in torej nespremenjeno) vrednost, bodisi pravkar nastavljeno novo vrednost.

```
vozenj
```

```
{'Nakamura': 22, 'Cube': 43, 'Canyon': 26, 'Stevens': 9}
```

```
vozenj.setdefault("Scott", 0)
```

```

0
vozenj
{'Nakamura': 22, 'Cube': 43, 'Canyon': 26, 'Stevens': 9, 'Scott': 0}
vozenj.setdefault("Nakamura", 0)
22
vozenj
{'Nakamura': 22, 'Cube': 43, 'Canyon': 26, 'Stevens': 9, 'Scott': 0}
To poenostavi naše računanje.
vozenj = {}
razdalje = {}
visine = {}

for vrstica in open("kolesa.txt"):
    kolo, razdalja, visina = vrstica.split(",")
    vozenj.setdefault(kolo, 0)
    razdalje.setdefault(kolo, 0)
    visine.setdefault(kolo, 0)
    vozenj[kolo] += 1
    razdalje[kolo] += int(razdalja)
    visine[kolo] += int(visina)

for kolo in vozenj:
    print(kolo, "-", vozenj[kolo], "vozenj,", razdalje[kolo], "km,", visine[kolo], "m.")
Nakamura - 22 vozenj, 439 km, 1119 m.
Cube - 43 vozenj, 3174 km, 66705 m.
Canyon - 26 vozenj, 2766 km, 26392 m.
Stevens - 9 vozenj, 607 km, 3395 m.
Meh, ali pa tudi ne. Stavek if in malo prirejanja,
    if kolo not in vozenj:
        vozenj[kolo] = razdalje[kolo] = visine[kolo] = 0
smo zamenjali s tremi vrsticami setdefault-ov. No, če bi imeli en sam slovar,
pa je
    d.setdefault(x, 0)
gotovo krajše in preprostejše kot
    if x not in d:
        d[x] = 0

```


Get

Za metodo `get` vam moram povedati preprosto zato, ker je to ena od stvari, za katere je boljše, da o njej slišite od modrih, starejših ljudi, kot od vrstnikov.

Metoda `get` vrne vrednost, ki pripada podanemu ključu.

```
visine
```

```
{'Nakamura': 1119, 'Cube': 66705, 'Canyon': 26392, 'Stevens': 3395}
```

```
visine.get("Cube")
```

```
66705
```

Ne počnite tega. V Pythonu se reče

```
visine["Cube"]
```

```
66705
```

V vaših programih bom sicer žalostno opazoval takšno uporabo `get`-a. Do nje bo mnogokrat prišlo, ker vam bodo pri reševanju nalog "pomagali" znanci, ki sicer programirajo v Javi ali podobno okornih jezikih, ki zaradi določenih omejitev ne morajo do vsebine slovarjev z oglatimi oklepaji.

Zakaj Python potem sploh ima `get`, če naj bi ga ne uporabljali?

Metodi `get` lahko podamo privzeto vrednost, ki naj jo vrne, če ključ ne obstaja.

```
visine.get("Cube", 0)
```

```
66705
```

```
visine.get("Focus", 0)
```

```
0
```

Zato in samo zato.

items

Videli smo, da gre zanka `for`, ko jo spustimo čez slovar, prek njegovih ključev. Če jo naženemo prek tega, kar vrne `values`, bo šla prek vrednosti. Slovarji imajo še metodo `items`, ki vrača pare ključev in vrednosti.

Vrnimo se malenkost nazaj. Spomnimo se, kako smo uporabljali rezultat `split`: ker je vračal več stvari, smo ga prirejali več spremenljivkam. Recimo, da imamo niz z dvema besedama, "Ana Benjamin" in ga razdelimo s `split`. Dobimo dve besedi, torej ju moramo shraniti v dve spremenljivki.

```
ona, on = "Ana Benjamin".split()
```

```
ona
```

```
'Ana'
```

on

'Benjamin'

Če metoda `items` vrača pare, bo tudi zanka `for` potrebovala dve spremenljivki, da ga shrani.

```
for kolo, visina in visine.items():  
    print(kolo, " - ", visina)
```

Nakamura - 1119

Cube - 66705

Canyon - 26392

Stevens - 3395

Zdaj pa kar brž napišimo program, ki bo povedal, katero kolo se je največ vzpenjalo.

```
naj_visina = 0
```

```
for kolo, visina in visine.items():  
    if visina > naj_visina:  
        naj_visina = visina  
        naj_kolo = kolo
```

```
print("Največ se je vzpenjal", naj_kolo, "in sicer", naj_visina, "metrov.")
```

Največ se je vzpenjal Cube in sicer 66705 metrov.

keys

Tudi ta je iz poglavja "boljše, da vam za to povem jaz".

`keys` vrne vse ključe slovarja.

```
for kolo in visine.keys():  
    print(kolo)
```

Nakamura

Cube

Canyon

Stevens

To je seveda čisto drugače kot

```
for kolo in visine:  
    print(kolo)
```

Nakamura

Cube

Canyon

Stevens

Aja, ni. Isto je.

Glede metode `keys` bi vam rad povedal le to: pozabite, da obstaja.

Prav, zakaj potem Python sploh ima to metodo?! Pojma nimam. Za hec sem vprašal za mnenje chatGPT. :) Takole me je podučil. (Kogar ne zanima krajši komentar chatGPTja, lahko z branjem tu zaključim.)

In Python, dictionaries (`dict`) have a `keys()` method that returns a view object containing the keys of the dictionary. While it might seem redundant since you can iterate over the dictionary itself to access its keys, there are some important distinctions and use cases for using `dict.keys()`:

1. **View Object:** `dict.keys()` returns a view object that is a dynamic view on the keys of the dictionary. This means that any changes made to the dictionary are immediately reflected in the view, and vice versa. This is not the case when you directly iterate over the dictionary.
2. **Compatibility:** The `.keys()` method is available in Python 2 and Python 3, which makes it more backward-compatible for code that needs to run on both versions. Direct iteration over the dictionary may be less compatible.
3. **Explicitness:** Using `.keys()` can make your code more explicit and easier to understand. It clearly signals your intention to work with the keys of the dictionary.

Nadaljeval je s primerom kode, ki je bila nesmiselna. Zaključil jo je s komentarjem:

In this example, the view object `key_view` reflects changes made to the dictionary, while iterating directly over the dictionary does not provide the same dynamic behavior. Depending on your use case and specific needs, you may choose to use `dict.keys()` or direct iteration, but understanding the differences and advantages of using `dict.keys()` can help you make informed decisions in your code.

Vem, da ne razumete, vendar: napisano nima smisla. Če spremenimo slovar je slovar spremenjen. Prav tako je spremenjen "key_view". Oboje je "dinamično". "*does not provide the same dynamic behavior*" je tipičen primer chatGPTjevega flanca. Zadnji stavek pa sploh.

Tiste tri točke so pa zanimive. Tretja je mogoče smiselna in je stvar okusa.

Druga točka je popolnoma mimo. Res je, da imata oba metodo `keys`, vendar ne vrača iste stvari. Ena od največjih razlik med Pythonom 2 in 3 je prav v tem, kaj vračajo `keys` in podobne metode.

Prva točka je pa zanimiva. S `keys` lahko ustvarimo nek objekt, ki vsebuje le ključne, ne pa tudi vrednosti. Zakaj bi to hoteli, sicer ne vem, vendar ... no, tu je napisal nekaj smiselnega.