04 seznami

January 28, 2024

0.1 Podatki v pomnilniku

Pri vsem, kar smo počeli doslej, so bili podatki (pretežno) v datoteki. Samo *pretežno* zato, ker to ni bilo čisto res. Najprej so tu spremenljivke, kot so bile razne temperature, imena predmetov in ljudi na dražbah in tako naprej. A to so bile le posamezne vrednosti, ki so se nanašale na, recimo, trenutno prebrano vrstico, najvišjo ponujeno ceno in tako naprej. Poleg teh od prejšnjega tedna poznamo tudi slovarje. Ti so shranjevali bolj "masovne" podatke; ključi so bile lahko imena oseb in pripadajoče vrednosti so bile lahko skupna količina denarja, ki ga je dotična oseba zapravila na dražbi. Slovar ima lahko zelo veliko, tudi na tisoče ključev in vrednosti.

Ne, pravzaprav ne, da ni bilo *čisto res*, sploh ni bilo res. Res pa je, recimo, da podatkov iz datoteke nikoli nismo v celoti prebrali v pomnilnik in jih obdelovali tam. Ko smo reševali vaje in domačo nalogo, smo jih za vsako nalogo sproti prebrali iz datoteke. Poleg tega so slovarji le ena - in morda niti ne najpogostejša *podatkovna struktura* za shranjevanje podatkov.

Zakaj bi želeli prebrati podatke, recimo celo datoteko v pomnilnik? Izmislimo si lahko kup razlogov.

- Prvi je, preprosto, da ne želimo vedno znova brati datoteke. Zgodilo se bo tudi, da bodo podatki v datoteki zapisani na kak zoprn, zapleten način, ki bo zahteval veliko dela in (računalnikovega) časa za branje. V tem primeru jih gotovo želimo prebrati le enkrat, jih ustrezno obdelati in potem delati z njimi v takšni, predelani obliki.
- Drugi razlog je, da je s podatki v pomnilniku lažje manipulirati, jih zlagati v primernejšo obliko. Za trivialen primer vzemimo datoteko z imeni udeležencev dražbe. Recimo, da je videti tako.

  ```
  Berta
  Cilka
  Ema
  Dani
  Ana
  Fanči
  Greta
  ```

  Izpisati jih želimo po abecedi. To (skoraj) ne bo šlo drugače, kot da jih naložimo v pomnilnik, tam uredimo po abecedi (kakorkoli se že to naredi) in potem izpišemo.
- Tretji: sekvenčni dostop. (Ta je malo podoben prejšnji točki.) Pri nalogah z dražbe nas je večkrat pestilo, da smo pri branju neke vrstice potrebovali tudi podatek o tem, kaj je pisalo v prejšnji. To smo rešili tako, da smo si podatke shranjevali na koncu zanke (tipičen stavek je bil v slogu `prejsnja_temperatura = temperatura`). To gre, dokler potrebujemo le en prejšnji podatek. Kadar jih potrebujemo prejšnjih pet, bo to zoprno.

Skratka: podatke želimo sem ter tja naložiti v pomnilnik.

Edina nam znana *podatkovna struktura*, ki to omogoča, je slovar. Spomnimo le na dve njegovi lastnosti.

- Slovar je sestavljen iz parov ključ-vrednost. Za vsak ključ lahko izvemo (ali nastavimo) njegovo vrednost.
- Na vrstni red elementov v slovarju ne moremo vplivati. "Slučajno" je vrstni red kar vrstni red dodajanja; kasneje ga ne moremo spremeniti. Zategadelj v okviru slovarjev tudi nima smisla zares govoriti o vrstnem redu, se pravi, o tem, kako si sledijo elementi (ta element je pred tem in za onim). Zanka `for` prehodi slovar v nekem bolj ali manj naključnem vrstnem redu. Povedano še drugače: elementi slovarja niso *oštevilčeni*: nesmiselno je govoriti o "prvem", "drugem" in tako naprej elementu slovarja.

Oboje nas lahko zmoti. Zgodilo se bo, da bomo imeli le neke vrednosti in nobenih pripadajočih ključev. Recimo najvišje dnevne temperature v Ljubljani za vsak dan v juliju 2023. Ali pa zapisnik dražbe. Pač - same številke. Kako to shraniti v slovar? Ne bo šlo. Razen na kakšen ekstra neumen način, recimo tako, da so ključi kar zaporedne številke dni oziroma ponujenih cen. A še v tem primeru nam bo šlo na živce, da nismo gospodarji vrstnega reda; temperatur, recimo ne bomo mogli urediti po velikosti, da bi izpisali tri najtoplejše julijske dni. Ker - kot smo napisali v drugi gornji točki, slovarjem ta koncept pač ni lasten.

0.2 Seznami

Seznam je *podatkovna struktura*, ki …

Uh, zdaj sem že tretjič zapisal ta termin, podatkovna struktura, in ga vedno znova dal v poševni tisk, kot da ga omenjam prvič. To pa zato, ker ga ne vprvo ne vdrugo nisem razložil. Ob tretji omembi pa bo že čas.

Wikipedija definira podatkovno strukturo kot "*a collection of data values, the relationships among them, and the functions or operations that can be applied to the data*".

- Kot prvo, podatkovna struktura, očitno shranjuje neke podatke. Tu nimamo kaj filozofirati.
- Nadalje, med njimi obstaja neka zveza. V primeru slovarjev gre za zvezo ključ-vrednost. Za seznam se bo izkazalo, da ima koncept vrstnega reda. Ta element je prvi, ta drugi, ta tretji. Nekatere zveze so lahko bolj tehnične in se nanašajo na to, kako so urejeni podatki znotraj strukture.
- Končno - in najpomembnejše - za podatkovno strukturo so operacije, ki jih lahko izvajamo s temi podatki in - predvsem, če bi šlo za predmet Algoritmi in podatkovne strukture - kako učinkovite, hitre so te operacije. Za slovar smo povedali, da lahko za vsak ključ zelo hitro izvemo pripadajočo vrednost (povedali smo, da so Pythonovi slovarji tako učinkoviti, da je čas, ki ga potrebuje za to, neodvisen od števila elementov, shranjenih v slovarju!), enako hitro lahko tudi izvemo, ali določen ključ obstaja; tudi dodajanje in brisanje elementov je enako hitro. V kontekstu seznamov bo o tem nesmiselno govoriti, saj nimajo ključev.

Ob vsaki podatkovni strukturi bomo torej povedali, kaj lahko vsebuje in kaj zna (učinkovito) početi s temi stvarmi.

Seznam, torej, je podatkovna struktura, v katero lahko shranimo poljubno število elementov poljubnega tipa (v Pythonu; drugod navadno velja pravilo, da morajo biti vsi elementi istega tipa).

Naredimo lahko, recimo, seznam imen.

[1]:
```
imena = ["Ana", "Berta", "Cilka", "Dani", "Ema", "Fanči"]
```

Takoj vidimo: slovarji so imeli zavite oklepaje, seznami oglate. Po tem jih ločimo. Slovarji imajo pare ključ-vrednost, seznami pač le … elemente. (Ne ključ ne vrednost jim ne bomo rekli. V smislu prevoda iz angleščine bi bilo bolj prav govoriti o stvareh, saj je v angleščini to *item*. Pa *element* tudi ni ravno čisto slovenska beseda. :) A kar je, je; tako se je ustalilo.)

V slovar lahko dajem tudi kaj drugega, recimo števila,

[2]:
```
teze = [57, 66, 58, 52, 65, 68]
```

ali mešanico nizov in števil (česar ne bom pokazal, ker je grdo). Lahko naredimo tudi seznam, ki vsebuje več slovarjev, kar bi bilo malo nenavadno, vendar nam bo prav pri tem predmetu nemara prišlo še prav. Lahko pa naredimo tudi prazen seznam

[11]:
```
temperature =[]
```

in vanj dodajamo elemente, recimo iz datoteke.

[12]:
```
for vrstica in open("december.txt"):
    temperature.append(int(vrstica))
```

Kar tako, mimogrede smo spoznali prvo metodo seznamov: `append`. Kot argument podamo kar hočemo - v tem primeru očitno neko število, namreč iz datoteke prebrano temperaturo - in metoda `append` doda ta element na konec seznama.

Tako dobimo seznam napovedanih decembrskih temperatur v Radovljici.

[9]:
```
temperature
```

[9]:
```
[5, 4, -1, -6, -8, 2, 5, -6, -8, -12, -15, 6, 7, -20, 2, 3]
```

Nasvet: uprite se skušnjavi in praznega seznama nikoli ne poimenujte prazen, čeprav bo v začetku morda res takšen.

[10]:
```
prazen = []
for vrstica in open("december.txt"):
    prazen.append(int(vrstica))

print(prazen)
```

```
[5, 4, -1, -6, -8, 2, 5, -6, -8, -12, -15, 6, 7, -20, 2, 3]
```

Očitno ni prazen, ne?

Zdaj poznamo prvi del: kaj lahko podatkovna struktura seznam shranjuje. Drugi je, kaj lahko s temi rečmi počnemo.

0.2.1 Sprehod prek seznama

Najprej nekaj kar že znamo: z zanko `for` lahko gremo prek elementov seznama.

[14]:
```
for temp in temperature:
    print(temp)
```

```
5
4
-1
-6
-8
2
5
-6
-8
-12
-15
6
7
-20
2
3
```

Z datotekami doslej ne znamo početi nič drugega (in tudi v prihodnosti ne bomo znali početi veliko drugega) kot to, da smo prek njih nagnali zanko `for`. Ker lahko isto storimo s seznami, lahko čisto vse, kar smo doslej počeli z datotekami, počnemo tudi s seznami. S to prednostjo, da bodo seznami temperatur tipično že števila (ali karkoli bo treba) in ne bo treba stalno klicati zoprnega `int` ali `float`. No, pa `open` očitno tudi preskočimo. Seznam je pač že tu, prebran, pripravljen in ga je potrebno le pomoliti `for`-u.

Tako, za vajo, poiščimo najmanjši element seznama.

[17]:
```
najm = 100000

for temp in temperature:
    if temp < najm:
        najm = temp

najm
```

[17]:
```
-20
```

Zanko `for` smo že tako natrenirali, da tu ni več česa ponavljati. Vse že znamo.

0.2.2 Funkcije, ki se sprehajajo namesto nas

Tole je četrti teden predmeta in v prvih treh smo ničkolikokrat izračunali največji element, najmanjši element in vsoto. Drugače ni šlo. Python sicer ima funkcije `max`, `min` in `sum`, pa še veliko drugih podobnih, vendar so bili v datotekah nizi in če bi jih prepustili tem funkcijam, bi dobili niz, ki

je zadnji po abecedi (`max`), prvi po abecedi (`min`) ali pa vsoto vseh nizov (`sum` + malo truda z dodatnimi argumenti).

Zdaj, ko imamo sezname *števil*, bodo vse takšne funkcije delale, kar je treba.

[27]:
```
max(temperature)
```

[27]:
```
7
```

[28]:
```
min(teme)
```

[28]:
```
-20
```

Mimogrede povejmo, da imajo seznami (tako kot slovarji in nizi, ne pa tudi datoteke) dolžino. "Dolžina" pomeni preprosto "število elementov". Izvemo ga s funkcijo `len`, ki ji kot argument podamo slovar, niz ali kakršnokoli-že podatkovno strukturo, ki pozna idejo "dolžine".)

[34]:
```
print("Povprečna temperatura decembra bo", sum(temperature) / len(temperature))
```

```
Povprečna temperatura decembra bo -2.625
```

0.2.3 Dolžina seznama

Mimogrede povejmo, da imajo seznami (tako kot slovarji in nizi, ne pa tudi datoteke) dolžino. "Dolžina" pomeni preprosto "število elementov". Izvemo ga s funkcijo `len`, ki ji kot argument podamo slovar, niz ali kakršnokoli-že podatkovno strukturo, ki pozna idejo "dolžine".)

[35]:
```
len(temperature)
```

[35]:
```
16
```

0.2.4 Vsebovanost

Pogosto nas bo zanimalo, ali slovar vsebuje tak in tak element ali ne. Maloprejle smo si pripravili seznam imena, zamolčavši, da so to imena članic knjižnega kluba Molj.

[46]:
```
imena
```

[46]:
```
['Ana', 'Berta', 'Cilka', 'Dani', 'Ema', 'Fanči']
```

Operator `in` poznamo iz prejšnjega tedna, ko nam je povedal, ali slovar vsebuje določen ključ ali ne. Če mu namesto slovarja pomolimo seznam, nam bo povedal, ali vsebuje ta element.

Če nas torej zanima, ali je Cilka v klubu, nam operator `in` pove, da je.

[47]:
```
"Cilka" in imena
```

[47]:
```
True
```

Za Greto nam pove, da ni.

[48]:
```
"Greta" in imena
```

[48]:
```
False
```

0.2.5 Razpakiranje seznama in okostnjak iz splitske omare

Imamo seznam, za katerega smo popolnoma, absolutno, neomajno prepričani, da vsebuje dva elementa.

[33]:
```
s = [42, 13]
```

Njegova elementa lahko priredimo dvema spremenljivkama.

[36]:
```
odgovor, nesreča = s
```

[37]:
```
odgovor
```

[37]:
```
42
```

[38]:
```
nesreča
```

[38]:
```
13
```

Tudi to za nas ni nekaj povsem novega. Razpakiranje smo že vzeli. Imeli smo niz, ki je vseboval, recimo, ime kraja ter največjo in najmanjšo temperaturo; tri podatke torej, ločene z vejicami. Razbili smo ga s `split` in priredili trem spremenljivkam.

[40]:
```
vrstica = "Ljubljana,19,15"

kraj, najvisja, najnizja = vrstica.split(",")
```

To sem vam povedal, razumeli ste, da `split` vrne tri stvari in jih zato priredimo trem spremenljivkam. Argument avtoritete (namreč moje) je zmagal, več niste spraševali.

Pa bi morali. Vsaka funkcija vrne neko reč - in natančno eno reč! Kakšno *reč* vrne `split`? V resnici vrne seznam.

[41]:
```
vrstica.split(",")
```

[41]:
```
['Ljubljana', '19', '15']
```

Sezname smo torej ponevedoma uporabljali že prejšnjo uro, vendar sem jih zamolčal, ker še ni bil čas zanje. Zdaj, ko je, končno veste tudi, kaj dela `split`.

0.2.6 Dolžina seznama

Mimogrede povejmo, da imajo seznami (tako kot slovarji in nizi, ne pa tudi datoteke) dolžino. "Dolžina" pomeni preprosto "število elementov". Izvemo ga s funkcijo `len`, ki ji kot argument podamo slovar, niz ali kakršnokoli-že podatkovno strukturo, ki pozna idejo "dolžine".)

[35]:
```
len(temperature)
```

[35]: 16

0.2.7 Primer: štetje mašil

[55]:
```
ana_je_rekla = """
danes sem am zjutraj like vstala pa ampak sem še napol spala pa like nisem vedla
a čem prec poklicat Julijo pa sem bla pač like ne bom še pa je pač nisem pa sem␣
 ↪pač
like še naprej kar ležala pa am čakal če me bo pač ona"
"""
```

Dajmo like prešteti, kolikokrat je uspela v enem stavku reči *am*, *like* ali *pač*.

[56]:
```
masila = ["like", "pač", "am"]

masil = 0
for beseda in ana_je_rekla.split():
    if beseda in masila:
        masil += 1

print(masil)
```

```
10
```

`ana_je_rekla.split()` nam vrne seznam Aninih besed (tu nam malo pomaga, da je stavek zapisan brez ločil, sicer bi se morali znebiti še teh. Pa vse je zapisano z malimi črkami in tako naprej. Najstlovenščina ima tudi prednosti.

[50]:
```
ana_je_rekla.split()
```

[50]:
```
['Danes',
 'sem',
 'am',
 'zjutraj',
 'like',
 'vstala',
 'pa',
 'ampak',
 'sem',
 'še',
 'napol',
 'spala',
 'pa',
 'like',
 'nisem',
 'vedla',
```

```
 'a',
 'čem',
 'prec',
 'poklicat',
 'Julijo',
 'pa',
 'sem',
 'bla',
 'pač',
 'like',
 'ne',
 'bom',
 'še',
 'pa',
 'je',
 'pač',
 'nisem']
```

Za vsako besedo s tega seznama (`for beseda in ana_je_rekla.split()`) preverimo ali se nahaja tudi v seznamu mašil, ki smo ga definirali na vrhu. Če, potem prištejemo še eno mašilo.

Mogoče nas zanima še delež mašil v njenem govoru? Potem potrebujemo še število besed. Da ne razbijamo dvakrat, kar takoj pokličimo `split`, shranimo seznam v besede in ga uporabimo dvakrat.

[54]:
```
masila = ["like", "pač", "am"]

besede = ana_je_rekla.split()
masil = 0
for beseda in besede:
    if beseda in masila:
        masil += 1

print(masil / len(besede))
```

```
0.20408163265306123
```

Dvajset odstotkov, vsaka peta beseda, je mašilo.

Primer je seveda umeten. V dejanskem govoru je mašil pač seveda več. Like.

0.2.8 Seznami seznamov in razpakiranje v zanki `for`

Izvedši, da `split` v resnici vrača sezname, bi morda prišli na idejo, da datoteko "kolesa.txt", ki vsebuje kolo, s katerim je šel nekdo na izlet, ter prevoženo število kilometrov in skupen vzpon v metrih,

```
Nakamura,16,24
Nakamura,11,80
Nakamura,14,15
Cube,50,1888
```

```
... in tako naprej ...
```

preberemo v seznam, ki bi bil videti tako

```
voznje = [["Nakamura", 16, 24], ["Nakamura", 11, 80], ["Nakamura", 14, 15], ["Cube", 50, 1888]]
```

Seznam bo torej vseboval sezname. Seznam lahko vsebuje karkoli in ko sem nazadnje preverjal, je kategorija "karkoli" pokrivala tudi sezname.

Preden nadaljujemo, omenimo, da se sme seznam razvleči tudi v več vrstic.

```
voznje = [
    ["Nakamura", 16, 24],
    ["Nakamura", 11, 80],
    ["Nakamura", 14, 15],
    ["Cube", 50, 1888]]
```

Nato posvarimo: ideja je pravilna, ampak pred seznami, ki vsebujejo različne stvari smo že posvarili. Tole nam ne bo všeč; ko bomo znali malo več, seznam `voznje` ne bo vseboval seznamov s tremi elementi (nizom in dvema številoma) temveč nekaj drugega.

Ampak za zdaj naj bo: sestavili bomo seznam seznamov.

Naivnež se kar takoj loti.

[57]:
```
voznje = []
for vrstica in open("kolesa.txt"):
    voznje.append(vrstica.split(","))
```

Končavši program je zadovoljen s svojo učinkovitostjo, saj je rezultat `split`-a - seznam podatkov za posamezno vožnjo - kar takoj odposredoval `append`-u.

Izpisovali ne bomo, seznam je predolg, saj vsebuje.

[59]:
```
len(voznje)
```

[59]:
```
100
```

voženj. Zato mi boste morali verjeti na besedo, da se mu ni čisto posrečilo. `split` vrne nize. Drugo in tretjo reč, razdaljo in višino, je pozabil pretvoriti v `int`. Ni kaj, treba bo narediti počasi.

[62]:
```
voznje = []
for vrstica in open("kolesa.txt"):
    kolo, razdalja, visina = vrstica.split(",")
    voznje.append([kolo, int(razdalja), int(visina)])
```

V zapiskih ne bomo izpisovali teh 100 vrstic, na predavanjih in doma pa le. Da smo naredili prav, bomo preverili tako, da bomo izračunali, koliko je prevozil s katerim kolesom. Za vajo in za ponovitev slovarjev.

[63]:
```
razdalje = {}

for podatek in voznje:
```

```
    kolo, razdalja, visina = podatek

    if kolo not in razdalje:
        razdalje[kolo] = 0
    razdalje[kolo] += razdalja

print(razdalje)
```

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

Takšno reč smo napisali prvič in zadnjič. Namreč

```
for podatek in voznje:
    kolo, razdalja, visina = podatek
```

Druga vrstica je namreč nepotrebna. Če želimo razpakirati element seznama, nam ga ni potrebno najprej dati v spremenljivko (`podatek`) in jo potem razvleči. Ne, seznama ne razpakira le prirejanje, `=`, temveč zna to tudi zanka `for`. Pišemo lahko - in odslej tudi bomo - kar tako:

[64]:
```
razdalje = {}

for kolo, razdalja, visina in voznje:
    if kolo not in razdalje:
        razdalje[kolo] = 0
    razdalje[kolo] += razdalja

print(razdalje)
```

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

Kar smo pisali prej, deluje, vendar je grdo in, kot se bo kmalu izkazalo, povzroča tudi druge sitnosti in dolgovezja. Da se pa tudi grše; če vam bodo pri programiranju pomagali prijatelji, ki so bolj vešči drugih jezikov kot Pythona, vam bodo pokazali nekaj, kar bo še grše in bo za vaše prihodnje programiranje še bolj nerodno, dolgovezno in okorno. Zato: če spustimo zanko `for` čez seznam (ali kako drugo reč), ki vsebuje elemente, ki jih je potrebno razpakirati, jih razpakiramo že v glavi zanke.

0.3 Prva prijateljica seznamov: funkcija `zip`

Zgodilo se nam bo, da bodo podatki, s katerimi bomo delali, v dveh seznamih. Situacijo smo nakazali zgoraj: imamo seznam imen oseb in seznam njihovih tež.

[87]:
```
imena = ["Ana", "Berta", "Cilka", "Dani"]
teze = [57, 66, 58, 52]
```

Zdaj nam je izpisati imena in teže, v slogu

```
Ana: 57
Berta: 66
Cilka: 58
Dani: 52
```

```
Ema: 65
Fanči: 68
```

Morda bi kdo prišel na idejo narediti tole:

[88]:
```
for ime in imena:
    for teza in teze:
        print(ime, teza)
```

```
Ana 57
Ana 66
Ana 58
Ana 52
Berta 57
Berta 66
Berta 58
Berta 52
Cilka 57
Cilka 66
Cilka 58
Cilka 52
Dani 57
Dani 66
Dani 58
Dani 52
```

Izpis razodeva, da bi takšen prišel na slabo idejo. Kar bi storil je, da bi šel prek vseh imen in *za vsako ime* prek vseh tež. To očitno ni to.

Kdo drug bi prišel na idejo napisati

[89]:
```
for ime in imena:
for teza in teze:
    print(ime, teza)
```

```
  Cell In[89], line 2
    for teza in teze:
    ^
IndentationError: expected an indented block after 'for' statement on line 1
```

V slepem upanju, da bosta tidve zanki potem tekli nekako … vzporedno.

Ta ideja je očitno še slabša od prve. Prej je Python vsaj nekaj naredil, zdaj pa se že v izhodišču upre. (Pravzaprav je to morda boljše. Stvar okusa. V vsakem primeru je enako neuporabno kot prej.)

Razčistimo: potrebujemo eno zanko. Ne dveh. Torej, gotovo,

```
for ime, teza in ...
```

Zanka naj bi torej dobivala pare, vendar iz dveh seznamov hkrati.

Deus ex machina se imenuje `zip`. Funkciji damo dva seznama, pa ju bo spakirala v seznam parov … ki ga spet odpakiramo v `for`. (Če koga skrbi, da bo to pakiranje in razpakiranje počasno, ali da bo kakorkoli "obremenjevalo" računalnik, je njegova skrb odveč. Te reči so dobro domišljene in narejene tako, da so hitre. Kar počnemo tu, je zelo osnovna in pogosta reč.)

[90]:
```
for ime, teza in zip(imena, teze):
    print(ime + ":", teza)
```

```
Ana: 57
Berta: 66
Cilka: 58
Dani: 52
```

Funkcija `zip` lahko prejme tudi več kot eno reč. Če ji damo tri sezname, bo sestavljala trojke.

[91]:
```
imena = ["Ana", "Berta", "Cilka", "Dani"]
teze = [57, 66, 58, 52]
visine = [1.56, 1.75, 1.70, 1.68]

for ime, teza, visina in zip(imena, teze, visine):
    print(ime + ":", teza / visina ** 2)
```

```
Ana: 23.422090729783037
Berta: 21.551020408163264
Cilka: 20.06920415224914
Dani: 18.42403628117914
```

Ali, lepše:

[92]:
```
imena = ["Ana", "Berta", "Cilka", "Dani"]
teze = [57, 66, 58, 52]
visine = [1.56, 1.75, 1.70, 1.68]

for ime, teza, visina in zip(imena, teze, visine):
    print(ime + ":", round(teza / visina ** 2, 2))
```

```
Ana: 23.42
Berta: 21.55
Cilka: 20.07
Dani: 18.42
```

0.4 Drugi prijatelj seznamov: funkcija `enumerate`

Imamo torej seznam decembrskih temperatur.

[65]:
```
temperature
```

[65]: `[5, 4, -1, -6, -8, 2, 5, -6, -8, -12, -15, 6, 7, -20, 2, 3]`

in hrepenimo po tem, da bi jih izpisali takole:

```
1. december: 5
2. december: 4
3. december: -1
```

in tako naprej.

Z zanko bomo šli prek temperatur, vedeti pa moramo tudi zaporedno številko dneva. To bi lahko naredili tako:

[66]:
```
dan = 0

for temp in temperature:
    dan += 1
    print(dan, ". december:", temp)
```

```
1 . december: 5
2 . december: 4
3 . december: -1
4 . december: -6
5 . december: -8
6 . december: 2
7 . december: 5
8 . december: -6
9 . december: -8
10 . december: -12
11 . december: -15
12 . december: 6
13 . december: 7
14 . december: -20
15 . december: 2
16 . december: 3
```

Presledek pred piko se kar blešči, ampak potrpimo. Ko bo čas, se ga bomo temeljito znebili. Ampak zdaj je čas za nekaj drugega: situacija, ko potrebujemo neko stvar iz slovarja, hkrati pa tudi njeno "zaporedno številko", je tako pogosta, da nam Python v ta namen ponuja posebno funkcijo: `enumerate`.

Funkcija `enumerate` kot argument prejme seznam ali poljubno drugo stvar, prek katere je možno nagnati zanko `for`, recimo datoteko, slovar, niz ali nam še neznane skrivnostne reči. `enumerate` spremeni tak seznam elementov v seznam parov (zaporedna številka, element). Par preprosto razpakiramo v glavi zanke.

Če je bil ta opis zapleten: primer bo preprost.

[67]:
```
for dan, temp in enumerate(temperature):
    print(dan, ". december:", temp)
```

```
0 . december: 5
1 . december: 4
```

```
2 . december: -1
3 . december: -6
4 . december: -8
5 . december: 2
6 . december: 5
7 . december: -6
8 . december: -8
9 . december: -12
10 . december: -15
11 . december: 6
12 . december: 7
13 . december: -20
14 . december: 2
15 . december: 3
```

Odlično, ni? Prihranilo nam je, da bi morali sami šteti dneve. Šteje jih kar zanka, oziroma, točneje, `enumerate`. Le `for temp in temperature` zamenjamo s `for dan, temp in enumerate(temperature)`. Kot rečeno zgoraj: `enumerate` poskrbi, da namesto elementov dobimo pare zaporednih številk in elementov.

Aja, ničti december? Hmnoja, računalnikarji pač štejemo od 0. Večina programskih jezikov šteje od 0. Med osamelci je, žal, tudi R, s katerim bodo nekatere od vas - tiste, ki se boste v svojem študiju srečali s kako bolj statistično analizo podatkov - maltretirali pri drugih predmetih. Štetje od 1 zveni bolj naravno, v resnici pa je štetje od 0, kot se bo sproti izkazalo tudi pri tem predmetu, bolj praktično. (Istočasno pa ima tudi zgodovinskotehnične razloge v neki zvezi med tabelami in kazalci v C-ju. Karkoli že so tabele in kazalci. In C.)

Če komu ni prav, ga ne bomo obsojali; raje mu pokažimo, kaj popraviti. Lahko pač prišteje 1 ob izpisu.

[68]:
```
for dan, temp in enumerate(temperature):
    print(dan + 1, ". december:", temp)
```

```
1 . december: 5
2 . december: 4
3 . december: -1
4 . december: -6
5 . december: -8
6 . december: 2
7 . december: 5
8 . december: -6
9 . december: -8
10 . december: -12
11 . december: -15
12 . december: 6
13 . december: 7
14 . december: -20
15 . december: 2
16 . december: 3
```

Lepše in bolj prav je prositi `enumerate`, naj šteje od 1. To naredimo z dodatnim argumentom `start`, ki ga podamo poimensko.

[69]:
```
for dan, temp in enumerate(temperature, start=1):
    print(dan, ". december:", temp)
```

```
1 . december: 5
2 . december: 4
3 . december: -1
4 . december: -6
5 . december: -8
6 . december: 2
7 . december: 5
8 . december: -6
9 . december: -8
10 . december: -12
11 . december: -15
12 . december: 6
13 . december: 7
14 . december: -20
15 . december: 2
16 . december: 3
```

Namesto `1` lahko podamo tudi drugo številko. Vendar bomo najbrž vedno začeli z 1 (kadar ne bomo z `0`, kar bo večkrat, kot si predstavljate).

0.5 Indeksiranje

Do elementov slovarja smo prišli tako, da smo v oglatih oklepajih navedli ključ, pa smo dobili pripadajočo vrednost.

Seznami nimajo ključev, pač pa imajo vrstni red. Ker lahko tako govorimo o prvem drugem in tako naprej elementu, jih lahko primemo kar za njihove zaporedne številke.

[18]:
```
temperature
```

[18]:
```
[5, 4, -1, -6, -8, 2, 5, -6, -8, -12, -15, 6, 7, -20, 2, 3]
```

Četrto temperaturo dobimo preprosto tako, da v oklepaje zapišemo `4`.

[19]:
```
temperature[4]
```

[19]:
```
-8
```

Da, res je, četrta temperatura v seznamu je .. ups, -6?! -8 je peta!

Ja, spet. Šteti začnemo pri 0. Začetni (da se izognem besedi “prvi”) element slovarja ima *indeks* 0.

[20]:
```
temperature[0]
```

[20]: 5

Spomnimo se, da ima seznam `temperature` šestnajst elementov.

[29]:
```
len(temperature)
```

[29]: 16

Zadnja temperatura je potemtakem šestnajsta.

[22]:
```
temperature[16]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[22], line 1
----> 1 temperature[16]

IndexError: list index out of range
```

To sem naredil iz treh razlogov. :)

Prvi je, da pomislimo: če ima prvi element indeks 0, ima šestnajsti element indeks 15.

[23]:
```
temperature[15]
```

[23]: 3

Drži, zadnje število je 15.

Drugi je, da vidite, kaj se zgodi, če poskusimo uporabiti prevelik indeks. Opis napake bi moral zadoščati, *index out of range*, vendar nič hudega, če ga vidite in če povemo: takrat, ko vidite to napako, preverite indekse.

Tretji poučni nauk gornjega primera je, da se pri indeksiranju od konca lahko zmotimo in da ima Python zato priročen trik: z desne indeksiramo z negativnimi indeksi. Zadnji element ima indeks -1, predzadnji -2 in tako nazaj.

[24]:
```
temperature
```

[24]: [5, 4, -1, -6, -8, 2, 5, -6, -8, -12, -15, 6, 7, -20, 2, 3]

[25]:
```
temperature[-1]
```

[25]: 3

[26]:
```
temperature[-2]
```

[26]: 2

0.5.1 Primer: skupna vrednost prodanih izdelkov na dražbi

Naučivši se, kako dostopati do elementov z njihovim indeksom, opravimo z drugim grdavšem iz preteklosti, okostnjakom z dražbe: kako priti do prejšnjega elementa? Za zdaj uporabimo, kar znamo (čeprav bomo kmalu znali več in lahko naredili preprosteje): `enumerate`.

Spomnimo se: prvi zapisnik dražbe je bil videti kot seznam ponujenih cen, "cena" `-1` pa je pomenila, da je bil izdelek prodan za ceno iz prejšnje vrstice.

Po novem so cene lahko v seznamu.

[73]:
```
cene = []
for vrstica in open("../domace-naloge/02-drazba/drazba.txt"):
    cene.append(int(vrstica))

cene
```

[73]:
```
[11,
 17,
 24,
 30,
 -1,
 13,
 27,
 33,
 -1,
 12,
 27,
 34,
 40,
 -1,
 9,
 -1,
 8,
 20,
 30,
 31,
 -1]
```

Gornji kos kode na vašem računalniku najbrž ne deluje. Odlično: to pomeni, da se boste naučili popraviti (relativno) pot do datoteke drazba.txt. Pri meni je en direktorij višje, potem pa v poddirektoriju domace-naloge in znotraj tega v 02-drazba. Pri vas pa … poiščite.

Zdaj lahko gremo prek *oštevilčenega* seznama cen. Ko naletimo na `-1`, k vsoti prištejemo vrednost *prejšnjega* elementa.

[74]:
```
vsota = 0
for i, cena in enumerate(cene):
    if cena == -1:
        vsota += cene[i - 1]
```

```
vsota
```

[74]: 143

Na moč imenitno.

Če se hočemo malo pošaliti, štejemo od `-1`, pa nam ne bo potrebno odštevati `-1`. `i` bo namreč vseboval indeks *prejšnjega* elementa. :)

[75]:
```
vsota = 0
for i, cena in enumerate(cene, start=-1):
    if cena == -1:
        vsota += cene[i]

vsota
```

[75]: 143

0.5.2 Rezine

Rado se zgodi, da ne potrebujemo celotnega seznama, temveč le del. Recimo prvih pet elementov. Ali zadnjih pet. Ali vse elemente od dvainpetdesetega od sedeminpetdesetega. Z številko, od 52 do 57.

Najprej razčistiom tole: koliko elementov je od 52. do 57.? To navadno preštejemo na prste; potrebovali bomo eno dlan in še prvi prst druge. Šest. To se, klasično, zgodi, ko gremo na počitnice od 17. do 22. julija in bi radi vedeli, koliko dni bo to trajalo. Nekako vemo, da en dan več, kot je razlika, ampak za vsak slučaj raje preštejemo, ne? :)

Python je programski jezik in v programskih jezikih si raje ne privoščimo napak, ki bi izvirale iz tega, da se je nekdo uštel. Stvari naj bi bile karseda preproste (po tem, ko jih obladamo; čas, ki je potreben za to, da jih obvladamo, naj bi bil seveda čim krajši, vendar je pomembneje to, da po tem, ko znamo, delamo čim manj napak).

Zato je Python preprost: če zahtevamo vse elemente od 52. do 57. bo vrnil pet elementov. Če hočemo vse od 8. do 12., bodo štirje. Kateri štirje? Pač: osmi, deveti, deseti in enajsti. Pa dvanajsti? Ne. Ta bi bil peti.

Python ima torej preprosto pravilo: spodnja meja je všteta, zgornja ne. Ko se navadiš, se ti zdi, da je tako edino logično. In vidiš, da tudi odlično deluje.

Kako zahtevamo takšen podseznam? Da bomo imeli s čim delati, malo podaljšajmo seznam imen.

[78]:
```
imena = ['Ana', 'Berta', 'Cilka', 'Dani', 'Ema', 'Fanči', 'Greta', 'Helga',␣
↪'Iva']
```

Imena od petega do osmega dobimo z

[79]:
```
imena[5:8]
```

[79]:
```
['Fanči', 'Greta', 'Helga']
```

Fanči je peta,

[80]:
```
imena[5]
```

[80]:
```
'Fanči'
```

in Helga sedma,

[81]:
```
imena[7]
```

[81]:
```
'Helga'
```

Od petega do osmega pomeni torej brez osmega, tako da so tri.

Vzemimo prvih pet imen.

[82]:
```
imena[0:5]
```

[82]:
```
['Ana', 'Berta', 'Cilka', 'Dani', 'Ema']
```

Ker pogosto potrebujemo ravno prvih toliko in toliko, smemo spodnjo mejo kar izpustiti.

[83]:
```
imena[:5]
```

[83]:
```
['Ana', 'Berta', 'Cilka', 'Dani', 'Ema']
```

Izpustiti smemo tudi zgornjo mejo.

[84]:
```
imena[5:]
```

[84]:
```
['Fanči', 'Greta', 'Helga', 'Iva']
```

Tole se kar lepo bere in razume:

- `:5` pomeni *prvih pet*, torej prvih pet (ničti, prvi, drugi, tretji in četrti, ampak brez petega).
- `5:` pomeni *brez prvih pet*, namreč vse od petega naprej (brez ničtega, prvega, drugega, tretjega in četrtega).

To, da štejemo od 0, se imenitno ujame s pravilom, da je zadnji indeks izvzet.

Imenitnosti še ni konec. Kaj bi bilo tole?

[85]:
```
imena[-3:]
```

[85]:
```
['Greta', 'Helga', 'Iva']
```

Podali smo spodnjo mejo in izpustili zgornjo. Vendar je bila spodnja meja negativna, se pravi z desne. Dobimo torej minus tretji, minus drugi in minus prvi element. Kar so slučajno ravno trije.

Kaj pa to?

[86]:
```
imena[:-3]
```

[86]:
```
['Ana', 'Berta', 'Cilka', 'Dani', 'Ema', 'Fanči']
```

To so vsa od začetka (spodnjo mejo, 0, smo izpustili) do minus tretjega - ampak brez njega. Torej manjkajo -3, -2 in -1. Ravno trije.

Dopolnimo, kar smo pisali prej:

- `:5` pomeni *prvih pet*,
- `5:` pomeni *brez prvih pet*,
- `-5:` pomeni *zadnjih pet*,
- `:-5` pomeni *brez zadnjih pet*.

Tega si ne boste zapomnili. Ni treba. Samo spomnite se, da obstaja in vedno znova poskusite rekonstruirati iz pravil. Sčasoma vam bo prišlo v kri - če boste uporabljali.

0.5.3 Rezine in zip in prejšnji element

Zdaj znamo izračunati vsoto prodanih elementov še boljše kot prej. Razumemo tole?

[94]:
```
vsota = 0
for cena, naslednji in zip(cene, cene[1:]):
    if naslednji == -1:
        vsota += cena

vsota
```

[94]:
```
143
```

Kaj se dogaja? V bistvu imamo dva seznama, `cene` in `cene` brez prve.

[95]:
```
print(cene)
print(cene[1:])
```

```
[11, 17, 24, 30, -1, 13, 27, 33, -1, 12, 27, 34, 40, -1, 9, -1, 8, 20, 30, 31,
-1]
[17, 24, 30, -1, 13, 27, 33, -1, 12, 27, 34, 40, -1, 9, -1, 8, 20, 30, 31, -1]
```

`zip` ju popari: 11 in 17, 17 in 24, 24 in 30, 30 in -1 ... Ker je drugi seznam le za ena zamaknjen prvi, bodo pari, ki jih vrne `zip`, preprosto zaporedni elementi. Razpakiramo ju v `cena` in `naslednji`. Če je `naslednji` enak `-1`, je `cena` zadnja ponujena cena za "ta" predmet.