Seznam

Osnove programiranja

Nejc Ilc

Met krompirja

Bodoča olimpijska disciplina, Praskež se uri za zapisnikarja. Svetovni rekord je 101,9 m.

- Tekmovalci lučajo krompir. Kdor ga vrže najdlje, je zmagovalec.
- Praskež mora beležiti dolžine metov in nato razglasiti najdaljši met.
- Organizatorji želijo od njega tudi statistiko - povprečje vseh metov.

Pomagajmo mu pri vodenju seznama metov in izračunih.

Fotografija: Łukasz Rawa [vir]

Programski paket za podporo tekmovanja

Zastonj

`met_krompirja_v1.sb3`

Neorganizirani Praskež, ki vse dolžine metov krompirja nosi v svoji kosmati pameti. Zatakne se mu že pri drugi seriji tekmovanja.

Premium

`met_krompirja_v2.sb3`

Praskež si omisli seznam, v katerega beleži dolžine. Ko se mu zahoče, izvede del programa za izračun najdaljšega meta oziroma povprečja vseh metov.

Seznam

Zaporedje vrednosti

Python ima precej vgrajenih podatkovnih tipov. Do sedaj smo spoznali:

- ničnI tip `NoneType`, ki ima samo vrednost `None` in pomeni "nič",
- Booleov tip `bool`, ki zavzame vrednosti `True` ali `False`,
- številčna tipa `int` in `float` ter
- niz znakov `str`.

Slednji, niz znakov torej, je že prvi predstavnik podatkovnih tipov, ki opisujejo zaporedje podatkov, denimo:

`abeceda = 'ABCČDEFGHIJKLMNOPRSŠTUVZŽ'`.

Spoznajmo sedaj seznam (v pythonščini je to `list`).

Gre za zaporedje elementov, ki so lahko poljubnega podatkovnega tipa.

Kako definiramo seznam

Uporabimo oglate oklepaje `[ ]`

- znak `[` dobimo, če na tipkovnici pritisnemo `Alt Gr` + `F` oziroma `Ctrl` + `Alt` + `F`
- znak `]` dobimo, če na tipkovnici pritisnemo `Alt Gr` + `G` oziroma `Ctrl` + `Alt` + `G`

Elemente seznama ločimo z vejico

```
[57, 32, 99, 98, 57]
```

Seznamu seveda lahko damo ime z uporabo operatorja prirejanja `=`

```
moja_najljubsa_stevila = [7, 12, 22, 42, 3.14]
```

Aha, zgornji seznam vsebuje tako cela kot tudi necela števila. Lahko notri stlačimo še kaj?

Kaj lahko seznam vsebuje

Vsebuje lahko vrednosti ali imena (spremenljivke) poljubnih podatkovnih tipov.

Pazi tole:

```
import math
a = 22
moja_najljubsa_stevila = [7, 'dvanajst', a, 7*6, math.pi]
```

Seznam lahko vsebuje tudi sezname (in še veliko drugega, kot bomo videli do poletja).

```
vreca = [None, True, False, 42, 3.14, 3+2j, 'niz', ['in', ['seznam','v'], 'seznamu']]
```

PS. Mimogrede sem notri vrgel še kompleksno število $3 + 2i$, ki je tipa `complex`, imaginarna enota pa je `j`.

Hišne številke

Vsak element v seznamu ima svoj naslov

Praskež in navadni smrtniki začenjajo štetje z 1.

Programerji začnemo z 0. Vsaj v Pythonu[1].

Naslovu elementa v seznamu pravimo indeks. Seznami so torej oštevilčeni oziroma indeksirani.

Koliko je hiš v kraju?

Dolžino seznama dobimo s funkcijo `len()`.

1. Nekateri, redki, programski jeziki začenjajo sezname z naslovom 1, npr. Fortran, Julia, Lua, MATLAB, R, Scratch.

Fotografija: Elena Mozhvilo [vir]

Indeksi in dostop do elementov

Vsak element seznama ima svoj indeks, preko katerega ga lahko dosežemo.

Imamo seznam imen študentov. Nad elemente seznama napišimo njihove indekse.

```
# indeksi:     0         1         2         3
studenti = ['Sofija', 'Lenart', 'Patrik', 'Erazem']
```

- Do elementov dostopamo z oglatimi oklepaji. Študentka Sofija je prva v seznamu in ima indeks 0. Njeno ime dobimo tako:

  `studenti[0]`

- Drugi element ima indeks 1.

  `>>> studenti[1]`

  `Lenart`

- Zadnji element ima indeks 3, ki ga lahko izračunamo tako: "dolžina seznama minus ena":

  `len(studenti) - 1`

  torej

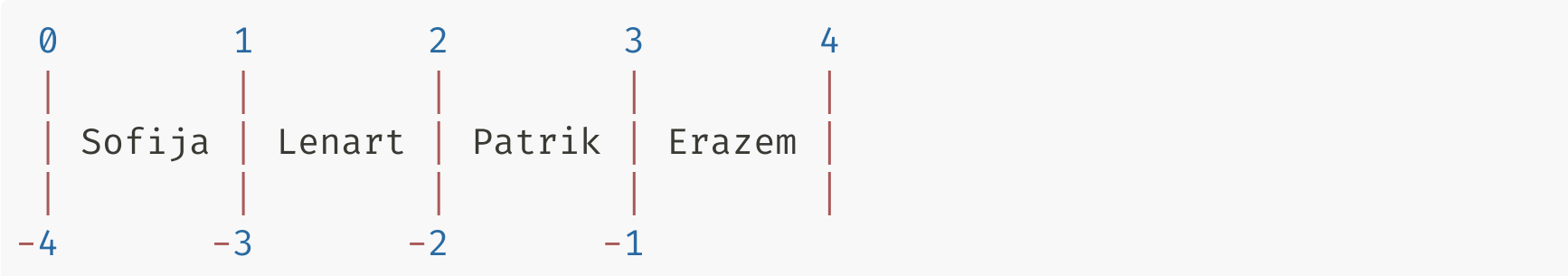
  `studenti[len(studenti) - 1]`

  V Pythonu obstaja tudi lažji način, da pridemo do zadnjega, predzadnjega itd. Obrni stran.

Negativni indeksi

Olajšajo nam dostop do elementov na koncu seznama.

Da se bomo v nadaljevanju lažje razumeli, si predstavljajmo, da so indeksi elementov seznama napisani pred samim elementom. Indeksom od 0 do `len(studenti)-1`, ki so napisani nad elementi, dodajmo pod seznam še negativne indekse. Poleg tega smo dodali še indeks 4, razlog povemo na čez nekaj strani 😄




```
>>> studenti[3]
'Erazem'
>>> studenti[len(studenti)-1]
'Erazem'
>>> studenti[-1]
'Erazem'
```

```
>>> studenti[0]
'Sofija'
>>> studenti[-4]
'Sofija'
>>> studenti[-len(studenti)]
'Sofija'
```

```
>>> studenti[4]
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Vožnja čez seznam `postaje`

Uporabimo zanko, da se premikamo po indeksih seznama

Kot primer vzemimo postajališča LPP linije 18 (od centra proti Stožicam):

```
postaje = ['Konzorcij', 'Cankarjev dom', 'Svetčeva', 'Večna pot', 'Živalski vrt']
```

Vožnja tja ...

Prvo postajališče ima indeks 0, drugo 1,..., do `len(postaje)-1`. Potrebujemo števec, ki se vsako iteracijo zanke poveča za 1, s pogojem v glavi zanke pa pazimo, da se pravočasno ustavimo. Števec v zanki oz. indeks seznama pogosto označimo z `i`.

```
i = 0
while i < len(postaje):
    postaja = postaje[i]
    print(postaja)
    i += 1
```

... in spet nazaj

```
i = len(postaje)-1
while i >= 0:
    print(postaje[i])
    i -= 1
```

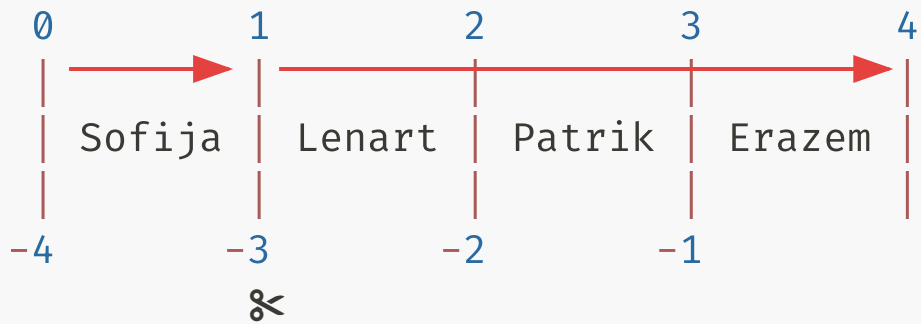
Za hec uporabimo še negativne indekse:

```
i = 1
while i <= len(postaje):
    print(postaje[-i])
    i += 1
```

Rezanje seznamov

Seznam je kot čajna klobasa - lahko ga režemo na rezine.

Navpične črtice na spodnji sliki označujejo mesta, kjer lahko zarežemo v seznam. Recimo, da želimo razdeliti seznam študentov na dve rezini (manjša seznama) tako, da bo v prvem samo prvi element, v drugem pa vsi ostali. Kje bomo zarezali? Zakaj smo dodali indeks 4?



Rezino v Pythonu določimo tako, da povemo njen začetni in končni indeks (lahko tudi korak, počakajmo).

```
>>> studenti[0:1]
['Sofija']
```

```
>>> studenti[1:4]
['Lenart', 'Patrik', 'Erazem']
```

```
>>> studenti[1:-1]
['Lenart', 'Patrik']
```

Rezine

Pravila za oblikovanje rezin

Splošna oblika rezine je:

```
seznam[zacetek:konec:korak]
```

Iz seznama `seznam` izrežemo rezino, ki vsebuje elemente z indeksi od vključno `zacetek` do ne vključno `konec` (torej do `konec-1`), pri čemer se indeksi povečujejo s korakom `korak`.

Števila `zacetek`, `konec` in `korak` so opcijski - lahko katerega od njih tudi izpustimo. Pri tem velja:

- če izpustimo `zacetek`, je isto, kot bi rekli, da je `zacetek` enak 0. Torej se rezina začne na začetku seznama;
- če izpustimo `konec` pomeni, da se rezina konča na koncu seznama, torej je `konec` enak `len(seznam)`;
- če izpustimo `korak`, se razume, da je korak 1.

Rezine - primeri

Spomnimo se spet na Praskeža in prvenstvo v metu krompirja. Praskež je v svoj seznam zabeležil tole:

```
meti = [57, 32, 99, 98, 57, 101, 48]
```

```
>>> meti[0:3]           # Od ničtega do ne vključno tretjega, torej od 0 do 2. To so prvi trije meti.
[57, 32, 99]
>>> meti[:3]            # Če začnemo rezino na začetku, lahko indeks 0 izpustimo. Dobimo isto kot prej.
[57, 32, 99]
>>> meti[3:len(meti)]   # Meti od vključno četrtega (ki ima indeks 3) do zadnjega, torej do ne vključno len(meti).
[98, 57, 101, 48]
>>> meti[3:]            # Indeks konca lahko v tem primeru izpustimo.
[98, 57, 101, 48]
>>> meti[:]             # Če zamolčimo tako začetek kot konec, dobimo cel seznam (od 0 do len(meti)).
[57, 32, 99, 98, 57, 101, 48]
>>> meti[0:len(meti):2] # Do sedaj je bil korak privzeto 1. Spremenimo ga na 2 in sedaj skačemo po 2 naprej.
[57, 99, 57, 48]
>>> meti[::2]           # Izpustimo začetek in konec, korak pustimo na 2.
[57, 99, 57, 48]
>>> meti[1::2]          # Začnemo pri drugem elementu in skačemo po 2 naprej vse do konca
[32, 98, 101]
>>> meti[4::-1]         # Korak je lahko negativen - gremo od desne proti levi!
[57, 98, 99, 32]        # Konec smo tukaj izpustili, torej gremo do začetka.
```

`range()`

Funkcija, ki generira sezname števil v podanem obsegu

Izkoristimo zagon, ki nam ga je dalo spoznanje rezin, in odkrijmo generator obsegov. Obnaša se namreč zelo podobno kot to določajo pravila rezin.

Splošni klic funkcije `range()` je:

```
range(zacetek, konec, korak)
```

Funkcija vrne seznam zaporednih celih števil (`int`) od vključno `zacetek` do ne vključno `konec` s korakom `korak`. Če izpustimo argument `korak`, se privzame korak 1.

Funkcija `range()` vrne nekaj, čemur rečemo generator - to pomeni, da nam izroči elemente seznama šele tedaj, ko jih zares potrebujemo.

```
>>> range(1, 20, 2)
range(1, 20, 2)
```

Če želimo vse elemente zdaj in takoj, to zahtevamo tako, da generator pretvorimo v seznam. Uporabimo funkcijo `list()`:

```
>>> list(range(1, 20, 2))
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

`range()` - primeri

Prvih 10 naravnih števil, vključimo tudi 0:

```
>>> list(range(10))   # Izpustili smo konec in korak
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Prvih 10 pozitivnih števil:

```
>>> list(range(1, 11))  # Izpustili smo korak
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Liha števila: izdelajmo seznam vseh lihih števil od `a` do `b`, vendar brez slednjega. Uporabnik zagotovi, da je `a` lih. Primer za `a = 1` in `b = 20`:

```
>>> list(range(a, b, 2))
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

Seznam večkratnikov, ki vsebuje večkratnike števila `a` od `a` do vključno `b`. Primer za `a = 3` in `b = 21`:

```
>>> list(range(a, b+1, a))
[3, 6, 9, 12, 15, 18, 21]
```

Soda števila: izdelajmo seznam vseh sodih števil od `a` do vključno `b`. Ni nujno, da je `a` sod. Primer za `a = 1` in `b = 20`:

```
>>> list(range(a+a%2, b+1, 2)) # a%2 je 1, če je a lih
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Odštevalnik od 5 do vključno 0:

```
>>> list(range(5, -1, -1))  # korak je lahko negativen
[5, 4, 3, 2, 1, 0]
```

Operatorji nad seznami

Vsebovanost

Z operatorjema `in` in `not in` lahko preverimo, ali seznam vsebuje določen element ali ne.

```
>>> 101 in [57, 32, 99, 98, 57, 101, 48]
True
>>> 'Ana' not in ['Sofija', 'Lara', 'Marija']
True
```

Seštevanje in množenje

Operator `+` lepi skupaj dva seznama, operator `*` naredi več kopij vsebine seznama.

```
>>> [1, 2, 3, 4] + [5, 6]
[1, 2, 3, 4, 5, 6]
>>> [0]*5
[0, 0, 0, 0, 0]
```

Primerjanje

Kdaj je nek seznam večji/manjši od drugega? Gledamo istoležne elemente in jih primerjamo. Uporabimo lahko tudi `>=` in `<=`.

```
>>> [1, 2, 3] > [5, 6]     # Ni res, 1 ni večja od 5
False
>>> [1, 2, 3] > [1, 2, 2]  # Res je, 3 je več kot 2
True
```

Enakost?

```
>>> [1, 2, 3] == [3, 2, 1]
False
>>> ['jabolko', 'kivi'] != ['jabolko', 'kivi']
False
```

Funkcije nad seznami

Vgrajene funkcije, ki sprejmejo seznam kot argument (razen prve)

`list()`

```
>>> list()      # Ustvarimo prazen seznam, isto kot []
[]
>>> list('ABC') # Niz znakov pretvorimo v seznam znakov
['A', 'B', 'C']
```

`bool()`

Vrne `True`, če seznam ni prazen, sicer `False`.

```
>>> bool([])
False
>>> bool([1, 2])
True
```

`len()`, `sum()`, `min()`, `max()`

```
>>> meti = [57, 32, 99, 98, 57, 101, 48]
>>> len(meti) # Že poznamo: dolžina seznama
7
>>> sum(meti) # Vsota vseh elementov seznama
492
>>> min(meti) # Najmanjši element
32
>>> max(meti) # Največji element
101
>>> max(['Rob', 'Rok']) # Isto kot max('Rob', 'Rok')
'Rok'
```

Znamo izračunati povprečno dolžino meta?

```
>>> sum(meti) / len(meti) # vsota / dolžina
70.28571428571429
```

Objekti - zelo na hitro

V Pythonu so vse *stvari* v resnici objekti. Objekti imajo svoje lastnosti in funkcije, ki jim pravimo metode. Do njih pridemo s piko, recimo:

```
ime_objekta.ime_metode(argumenti)
```

Seznam vseh lastnosti/funkcij objekta dobimo s funkcijo `dir()`. Lahko pa v Thonnyju odpremo `Pogled` → `Pregledovalnik objektov`. Nato izberemo spremenljivko v oknu *Spremenljivke*.

To nam je znano od prejšnjič, ko smo uvažali module. Uvoženi modul je tudi objekt in do njegovih konstant in funkcij pridemo s piko.

```
>>> import math as m
>>> m.pi          # Dostopamo do konstante
3.141592653589793
>>> m.sin(m.pi)  # Dostopamo do funkcije
1.2246467991473532e-16  # To je praktično 0
```

Tudi nizi so objekti. Ko uporabimo funkcijo `str()`, v resnici izdelamo nov objekt tipa `str`.

```
>>> a = str(42)
>>> a
'42'
>>> a.isnumeric()  # Pokličemo metodo niza, ki pove,
True               # ali niz vsebuje samo števke.
```

Metode seznama

Seznam je objekt … poglejmo njegove najbolj uporabne metode

metoda	opis
`seznam.clear()`	čiščenje seznama
`seznam.copy()`	kopiranje seznama
`seznam.sort()`	urejanje seznama
`seznam.append(x)`	na konec dodamo element `x`
`seznam.insert(i,x)`	na indeks `i` vrinemo element `x`
`s1.extend(s2)`	Seznam `s1` razširimo s `s2` (to naredi tudi `+`)

metoda	opis
`seznam.reverse()`	obračanje seznama
`seznam.count(x)`	štetje pojavitev elementa `x`
`seznam.index(x)`	indeks elementa `x`
`seznam.pop()`	ven vzamemo zadnji element
`seznam.pop(i)`	ven vzamemo element z indeksom `i`
`seznam.remove(x)`	izbrišemo element `x`

seznam.clear()

```
# Počistimo vsebino seznama, postane prazen
>>> seznam = [1, 2, 3]
>>> seznam
[1, 2, 3]
>>> seznam.clear()
>>> seznam
[]
```

seznam.copy()

```
# Naredimo "pravo" kopijo seznama, ne zgolj "povezave"
>>> seznam = [1, 2, 3]
>>> ni_prava_kopija = seznam
>>> ni_prava_kopija[0] = 0 # Spremenimo prvi element
>>> seznam  # Ups, s tem smo spremenili tudi seznam!
[0, 2, 3]
>>> kopija = seznam.copy()
>>> kopija[0] = 1
>>> kopija
[1, 2, 3]
>>> seznam
[0, 2, 3]
```

seznam.sort()

```
# Urejanje seznama po velikosti / abecedi ...
>>> seznam = [3, 1, 2, 3]
>>> seznam.sort()
>>> seznam
[1, 2, 3, 3]
>>> seznam.sort(reverse=True)
>>> seznam
[3, 3, 2, 1]
>>> seznam = ['Robnik', 'Rok', 'Rob']
>>> seznam.sort()
>>> seznam
['Rob', 'Robnik', 'Rok']
>>>
```

seznam.append(x)

```
# Dodamo element `x` na konec seznama.
>>> seznam = [1, 2, 3]
>>> seznam.append(4)
>>> seznam
[1, 2, 3, 4]
>>> seznam.append([5, 6]) # Če dodamo seznam,
>>> seznam                # dobimo seznam v seznamu.
[1, 2, 3, 4, [5, 6]]
```

seznam.insert(i, x)

```
# V seznam na indeks `i` vstavimo nov element `x`.
>>> seznam = [1, 2, 3]
>>> seznam.insert(1, 0)
>>> seznam
[1, 0, 2, 3]
>>> seznam.insert(1000, 4)
>>> seznam
[1, 0, 2, 3, 4]
```

seznam1.extend(seznam2)

```
# Seznam `seznam1` razširimo s seznamom `seznam2`.
# Zlepimo ju skupaj. To dela tudi `+` oziroma `+=`.
>>> seznam1 = [1, 2, 3]
>>> seznam2 = [4, 5, 6]
>>> seznam1.extend(seznam2)
>>> seznam1
[1, 2, 3, 4, 5, 6]
>>> seznam1 = seznam1 + [7, 8]
>>> seznam1
[1, 2, 3, 4, 5, 6, 7, 8]
>>> seznam1 += [9]
>>> seznam1
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> seznam1[:3] + [0, 0, 0] + seznam1[3:]
[1, 2, 3, 0, 0, 0, 4, 5, 6, 7, 8, 9]
```

seznam.reverse()

```
# Obrnemo seznam, od desne proti levi
>>> seznam = [1, 2, 3]
>>> seznam.reverse()
>>> seznam
[3, 2, 1]
```

seznam.count(x)

```
# Preštejemo, kolikokrat se pojavi element `x`.
>>> seznam = [1, 2, 3, 3]
>>> seznam.count(2)
1
>>> seznam.count(3)
2
>>> seznam.count(4)
0
```

seznam.index(x)

```
# Vrne indeks prve pojavitve elementa `x`
>>> seznam = [1, 2, 3, 3]
>>> seznam.index(2)
1
>>> seznam.index(3)
2
>>> seznam.index(4)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 4 is not in list
```

seznam.pop()

```
# Iz seznama brišemo zadnji element in ga vrnemo
>>> seznam = [1, 2, 3]
>>> seznam.pop()
3
>>> seznam
[1, 2]
```

seznam.pop(i)

```
# Iz seznama vzamemo element z indeksom `i`
>>> seznam = [1, 2, 3]
>>> seznam.pop(1)
2
>>> seznam
[1, 3]
```

seznam.remove(x)

```
# Iz seznama brišemo element `x` (prvo pojavitev)
>>> seznam = [1, 3, 2, 3]
>>> seznam.remove(3)
>>> seznam
[1, 2, 3]
>>> seznam.remove(3)
>>> seznam
[1, 2]
>>> seznam.remove(3)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

Brisanje elementov je mogoče tudi s stavkom `del`:

```
>>> seznam = [1, 3, 2, 3]
>>> del seznam[1]
>>> seznam
[1, 2, 3]
>>> del seznam[0:2]
>>> seznam
[3]
```

Program za beleženje metov krompirja

Pomagajmo zapisnikarju Praskežu, tokrat v Pythonu

Beležka `met_krompirja_v1`

- uporabnik vnaša dolžine metov enega po enega
- konča tako, da vpiše znak `'X'`
- če vpiše `'I'`, program izpiše seznam dolžin

Tablica `met_krompirja_v2`

- če uporabnik vpiše `'S'`, program izpiše padajoče urejen seznam
- če uporabnik vpiše `'N'`, program izpiše najdaljšo dolžino meta
- če uporabnik vpiše `'P'`, program izpiše povprečno dolžino meta

Superračunalnik `met_krompirja_v3`

- če uporabnik vpiše `'M'`, program izpiše mediano dolžine meta
- uporabnik vpiše -1, če je bil met neveljaven; ta met se ne upošteva pri izračunih povprečja in mediane, tudi se ne izpiše pri opciji `'S'`

`eval()`

Funkcija, ki ovrednoti (ang. evaluate) podani niz

Rezultat funkcije `eval()` je ovrednoten vhodni niz: kot če bi ga vnesli v interaktivno lupino Pyhona.

```
>>> eval('"2"*2+str(0//3)+"1"')
'2201'
>>> vnos = input('Vnesi izraz: ') # Vnesemo: 2**8
>>> eval(vnos)
256
>>> eval('[1, 2, 3]')
[1, 2, 3]
>>> vnos = input('Vnesi seznam: ') # Vnesemo: [1, 2, 3]
>>> eval(vnos)
[1, 2, 3]
>>> vnos = input('Vnesi seznam števil, ločenih z vejico: ') # Vnesemo: 57, 32, 99
>>> eval('[' + vnos + ']') # Dodamo oglate oklepaje, da je to res seznam
[57, 32, 99]
```

Nizi so v sorodu s seznami

Oba podatkovna tipa opisujeta zaporedje

Kar smo tu povedali za sezname, velja tudi za nize. No, precej metod sicer manjka … Lahko pa na enak način indeksiramo elemente, delamo rezine, seštevamo in množimo nize, uporabljamo funkcijo `len`, operatorja `in` in `not in`, …

```
>>> niz = 'ABC'
>>> len(niz)
3
>>> niz[0]
'A'
>>> niz[len(niz)-1]
'C'
>>> niz[-1]
'C'
>>> niz[0:2]
'AB'
>>> niz[1:]
'BC'
>>> niz[::-1]
'CBA'
```

```
>>> 'Rad imam '+ niz + ' sirček'
'Rad imam ABC sirček'
>>> niz * 3
'ABCABCABC'
>>> niz += ' sirček'
>>> niz
'ABC sirček'
>>> 'sir' in niz
True
>>> niz.count('s')
1
>>> niz.index('s')
4
```

Vgnezdeni seznam *aka.* seznam v seznamu

Element seznama je lahko drug seznam

Poleg dolžine meta krompirja si zapomnimo tudi ime tekmovalca

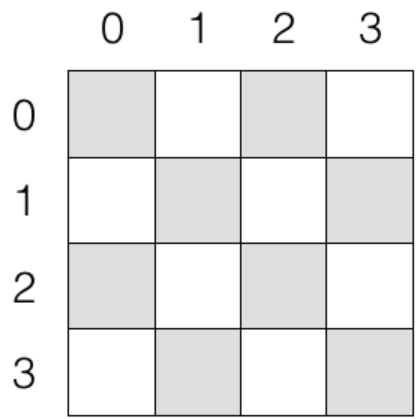
```
>>> rezultati = [[101.9, 'Peter'], [99, 'Janez'], [95, 'Metka']]
>>> rezultati[0]    # Dobimo podatke prvega meta - to je seznam oblike [dolžina, ime]
[101.9, 'Peter']
>>> rezultati[0][0] # Uporabimo še en nivo naslavljanja
101.9
>>> rezultati[0][1]
'Peter'
>>> prvi = rezultati[0] # Lahko pa si vgnezdeni seznam zapomnimo v novi spremenljivki
>>> prvi[0]             # in dostopamo do vsebine z enonivojskim indeksom
101.9
>>> rezultati.sort(reverse=True) # Kaj se zgodi, če seznam uredimo?
>>> rezultati
[[101.9, 'Peter'], [99, 'Janez'], [95, 'Metka']]
```

Sprehod čez seznam seznamov

sahovnica

Tokrat potrebujemo dve zanki, za vsak nivo eno

Za primer vzemimo majhno šahovnico 4 x 4 polj. Ima torej 4 vrstice in 4 stolpce. Vsaka vrstica naj bo seznam štirih polj. Imamo tudi seznam vseh vrstic, ki predstavlja celo šahovnico.




```
plosca = [
    ['▉','▒','▉','▒'],
    ['▒','▉','▒','▉'],
    ['▉','▒','▉','▒'],
    ['▒','▉','▒','▉'],
    ]

st_vrstic = len(plosca)
st_stolpcev = len(plosca[0])

vrstica = 0
while vrstica < st_vrstic:
    stolpec = 0
    while stolpec < st_stolpcev:
        # Izpišemo vsebino polja brez nove vrstice
        print(plosca[vrstica][stolpec], end='')
        stolpec += 1
    print() # Izpišemo novo vrstico
    vrstica += 1
```