

Terke

S seznami je v navidez tesni žlahti podatkovni tip *tuple*. Ime je skovano iz angleških besed *quintuple*, *sextuple*, *septuple*, *octuple* ... ki jih je skupni *tuple*. V slovenščini imamo peterko, šesterko, sedm(t)erko, osem(t)erko, deveterko, skratka, *terko*.

Kako jih sestavimo

Na prvi pogled je razlika v tem, da ima terka okrogle oklepaje namesto seznamovih oglatih. (Se pravi, terke so lepše od seznamov. :)

```
imena = ("Ana", "Berta", "Cilka", "Dani", "Ema", "Fanči")
```

Naredimo lahko tudi prazno terko.

```
prazna = ()
```

Ali terko z enim elementom.

```
ena = ("Ana", )
```

Ne spreglejte vejice. ("Ana",) je terka z enim elementom, ("Ana") pa je samo niz "Ana", ki smo ga iz Pythonu neznanega razloga dali v oklepaj. Ker se stvari sme dajati v nepotrebne oklepaje, so stvari v oklepajih samo stvari, ne terke. Da povemo, da gre za terko, moramo dodati še vejico.

Še ena čudna reč: terke ne potrebujejo oklepajev!

```
imena = "Ana", "Berta", "Cilka", "Dani", "Ema", "Fanči"
```

imena so zdaj terka, točno takšna kot prej. Vendar je to tako čudno, da celo Python, ko ga vprašamo po vsebini spremenljivke `imena`, odgovori z zapisom v oklepajih.

```
imena
```

```
('Ana', 'Berta', 'Cilka', 'Dani', 'Ema', 'Fanči')
```

Za terko dolžine 0 seveda potrebujemo oklepaje.

Sitno je, da za terko dolžine 1 ne potrebujemo oklepajev.

```
ena = "Ana",
```

ena je zdaj terka,

```
ena
```

```
('Ana',)
```

in to samo zato, ker smo na koncu vrstice - morda čisto ponesreči - napisali vejico.

S terkami brez oklepajev so potemtakem same sitnosti? Zakaj je to potem sploh dovoljeno? Zato, ker obstajajo situacije, ko je to videti boljše. Takrat - in le takrat - jih pišemo brez. Primer je tik pred koncem tega poglavja, še en pa na koncu poglavja o funkcijah.

Kaj počnemo z njimi - in česa ne?

S terkami lahko počnemo marsikaj, kar lahko počnemo s seznamami.

```
for ime in imena:
    print(ime)
```

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

```
imena[2]
```

```
'Cilka'
```

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

```
('Dani', 'Ema', 'Fanči')
```

Ne moremo pa jih spreminjati.

```
imena.append("Greta")
```

```
-----
AttributeError                                Traceback (most recent call last)
Cell In[7], line 1
----> 1 imena.append("Greta")
```

```
AttributeError: 'tuple' object has no attribute 'append'
```

```
del imena[2]
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[8], line 1
----> 1 del imena[2]
```

```
TypeError: 'tuple' object doesn't support item deletion
```

```
imena[2] = "Cecilija"
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[9], line 1
```

```
----> 1 imena[2] = "Cecilija"
```

```
TypeError: 'tuple' object does not support item assignment
```

Ne in ne. Ne gre.

Terke so zanimiva reč: **ko enkrat naredimo terko, bo vedno ostala takšna, kot je bila ob rojstvu**. Pravzaprav pa to ni nič nepričakovanega: isto lastnost imajo tudi `int`, `float`, `bool`, `str`. Razen seznamov in slovarjev torej vse, kar poznamo.

Zakaj potem sploh obstajajo?

Zakaj, pravzaprav bi si kdo želel podatkovni tip, katerega vrednosti se ne da nikoli spremeniti?

Ker je jih ne da spreminjati

Včasih potrebujemo točno to: nekaj, kar se gotovo ne bo spremenilo. Se spomnite ključev slovarjev? Takrat smo povedali, da so ključi lahko samo stvari, ki so nespremenljive. To so, bolj ali manj, številke in nizi (pa `True`, `False` in `None`, ki pa niso prav pogosti gosti med ključi slovarja, in, recimo, funkcije in moduli, ki so tam še redkeje). Včasih bi koga zamikalo kot ključ uporabiti seznam. To ne gre: sezname očitno lahko spreminjamo, torej ne morejo biti ključi. Če taiste vrednosti vržemo v terko, pa gre.

```
d = {}  
s = [1, 2, 3]  
d[s] = 0
```

```
-----  
TypeError                                Traceback (most recent call last)  
Cell In[13], line 1  
----> 1 d[s] = 0
```

```
TypeError: unhashable type: 'list'
```

```
t = tuple(s)  
t  
(1, 2, 3)  
d[t] = 0  
d  
{(1, 2, 3): 0}
```

Ker so tako preproste

Drugič, Python terke veliko uporablja interno. Ko, recimo, kličemo funkcijo z več argumenti, jih funkcija dejansko (a za programerja neopazno) dobi zapakirane v terko. Veliko stvari znotraj Pythona je v resnici terka. Seznami bi bili za to vlogo manj primerni, ker se lahko spreminjajo, ker se morajo znati podaljševati in skrajševati ... Preprosto prezapleteni so, preveč stvari omogočajo, pretežko jih je nadzorovati. Vendar nas ta, drugi razlog pri tem predmetu še najmanj zanima.

Tudi nam se bo zgodilo, da bomo kdaj na hitro naredili terko, ker bo tako praktično. Recimo, da imamo dve spremenljivki, `x` in `y`.

```
x = 6
y = 3
```

Morda je za nadaljevanje programa potrebno, da je v `x` manjša v `y` pa večja vrednost. Po potrebi ju torej zamenjamo.

```
x, y = y, x
```

Razumemo? Na desni strani je terka (samo brez oklepajev - to je eden od primerov, ko si jo upamo napisati tako, zato to tudi storimo). Na levi strani pa to terko razpakiramo. V bistvu torej `x, y = (y, x)`, le z manj navlake.

Kaj pa, če ne vemo, ali je `x` dejansko manjši od `y` in nismo prepričani, da ju je treba zamenjati? Začetnik piše

```
if x > y:
    x, y = y, x
```

Stari maček Pythona pa

```
x, y = min(x, y), max(x, y)
```

Za hranjenje "zapisov"

Recimo, da bi radi shranili podatke iz ene vrstice datoteke o, na primer, dražbi. Zapisali bi radi torej ime predmeta, ponudnika in ponujeno ceno. Lahko bi uporabili seznam,

```
ponudba = ["slika", "Ana", 30]
```

S tem ni nič narobe, vendar to ni tipično delo za sezname. Seznami so nekaj, kar se spreminja, vanj dodajamo elemente, jih brišemo. Takšne stvari tipično shranjujemo v terke.

```
ponudba = ("slika", "Ana", 30)
```

Razlika niti ni velika, v praksi bo relativno vseeno. Vsaj za vas. Pravi programer v Pythonu pa se v tem vidi, če ne drugega, neobvezno pravilo, in se ob prvem, seznamu, kar malo zdrzne. Priporočam, da se tudi sami navadite takega reda.