

Recimo, da delamo nekaj s kraji, opisanimi z imenom, zemljepisnimi koordinatami in številom prebivalcev. Če je teh krajev več, jih nočemo spravljati v posamične spremenljivke, temveč bo vsak kraj opisan s terko.

```
kraj1 = ("Kremenik", 46.091158, 14.201702, 19)
kraj2 = ("Vogrsko", 45.905166, 13.708147, 908)
```

To je neprikladno: če želimo izračunati razdaljo med tema krajema (kot vemo, je Zemlja plosčata, zato smemo v ta namen uporabiti Pitagorov izrek), moramo pisati

```
from math import sqrt

sqrt((kraj1[1] - kraj2[1]) ** 2 + (kraj1[2] - kraj2[2]) ** 2)
0.527436784922135
```

Do imena kraja pridemo s `kraj1[0]`, do števila prebivalcev s `kraj1[3]`.

```
kraj1[0]
'Kremenik'
kraj1[3]
19
```

Neugledno, nerodno, nepraktično, nevarno, ... skratka ne. Veliko raje bi pisali - kot v vseh normalnih jezikih - `kraj1.ime`, `kraj1.sirina`, `kraj1.dolzina` in `kraj1.prebivalcev`.

Način, na katerega to storimo, se je v Pythonu sčasoma izboljševal. Pokazal bom aktualnega, čeprav bo vključeval uporabo razredov, ki se jih bomo učili šele v zadnjih tednih predavanj.

```
from typing import NamedTuple
```

```
class Kraj(NamedTuple):
    ime: str
    sirina: float
    dolzina: float
    prebivalcev: int
```

Po novem shranimo podatke o krajih tako:

```
kraj1 = Kraj("Kremenik", 46.091158, 14.201702, 19)
kraj2 = Kraj("Vogrsko", 45.905166, 13.708147, 908)
```

Zdaj imamo, kot smo želeli,

```
kraj1.ime
'Kremenik'
kraj1.prebivalcev
```

19

```
sqrt((kraj1.sirina - kraj2.sirina) ** 2 + (kraj1.dolzina - kraj2.dolzina) ** 2)  
0.527436784922135
```

Lepši je tudi izpis

kraj1

```
Kraj(ime='Kremenik', sirina=46.091158, dolzina=14.201702, prebivalcev=19)
```

Če nas pomuči nostalgija, pa sta to še vedno tudi terki:

```
kraj1[0]
```

```
'Kremenik'
```

```
kraj1[3]
```

19

```
ime, sir, vis, preb = kraj1
```

```
ime
```

```
'Kremenik'
```

Tipi atributov

Poleg imen *atributov* smo našeli tudi njihove tipe. Snovalci Pythona vestno poudarjajo - in to obljubo bodo težko požrli - da bo označevanje tipov v Pythonu vedno namenjeno le knjižnicam in okoljem, ki nam pomagajo loviti napake, Python sam pa ujemanja tipov nikoli ne bo preverjal.

```
kraj = Kraj(3.14, True, None, "malo")
```

kraj

```
Kraj(ime=3.14, sirina=True, dolzina=None, prebivalcev='malo')
```

Pythonu je vseeno. Naš problem.

Vseeno je lepo, da se tudi mi držimo obljub; če obljubimo, da bo nekaj `int`, naj bo `int`.

Kaj pa, če je lahko `int` ali pa je neznano, kar bi radi označili z `None`?

```
from typing import NamedTuple, Optional
```

```
class Kraj(NamedTuple):  
    ime: str  
    sirina: float  
    dolzina: float  
    prebivalcev: Optional[int]
```

```
kraj = Kraj("Stara Gora", 46.656897, 15.637519, None)
```

```
kraj
```

```
Kraj(ime='Stara Gora', sirina=46.656897, dolzina=15.637519, prebivalcev=None)
```

Kako povemo, da bo neka vrednost seznam nizov? Ali slovar, katerega ključi bodo cela števila, vrednosti pa, huh, seznamami nizov? Označevanje tipov je lahko predmet čisto ločene "lekcije". Na hitro pa le povejmo, kako je s tem. Kot prvo, povemo lahko, da bo nekaj seznam, ne da bi se izjavnili glede tipa elementov.

```
imena: list
```

Če hočemo povedati, da bo vseboval nize, bomo rekli

```
imena: list[str]
```

Slovar, katerega ključi so `int`, vrednosti pa `str`, bi bil

```
po_starosti: dict[int, str]
```

Če so vrednosti seznamami nizov, pa

```
po_starosti: dict[int, list[str]]
```

Samo toliko, na hitro.

Privzete vrednosti

Atributi imajo lahko tudi privzete vrednosti.

```
from typing import NamedTuple, Optional
```

```
class Kraj(NamedTuple):
    ime: str
    sirina: float
    dolzina: float
    prebivalcev: Optional[int] = None
```

```
kraj = Kraj("Stara Gora", 46.656897, 15.637519)
```

```
kraj
```

```
Kraj(ime='Stara Gora', sirina=46.656897, dolzina=15.637519, prebivalcev=None)
```

```
from typing import NamedTuple, Optional
```

```
class Kraj(NamedTuple):
    ime: str
    sirina: float = 46.119552
    dolzina: float = 14.838005
    prebivalcev: Optional[int] = None
```

```
kraj = Kraj("Sreda")
kraj
Kraj(ime='Sreda', sirina=46.119552, dolzina=14.838005, prebivalcev=None)
```

Spreminjanje vrednosti

Če imamo nek kraj

```
kraj
Kraj(ime='Sreda', sirina=46.119552, dolzina=14.838005, prebivalcev=None)
```

in želimo spremeniti katero od njegovih lastnosti, nas bo to spomnilo, da imamo opravka s terkami.

```
kraj.prebivalcev = 368
```

```
-----
AttributeError                                Traceback (most recent call last)
Cell In[30], line 1
----> 1 kraj.prebivalcev = 368
```

```
AttributeError: can't set attribute
```

Terke so nespremenljive. Pač pa

```
kraj._replace(prebivalcev=367)
Kraj(ime='Sreda', sirina=46.119552, dolzina=14.838005, prebivalcev=367)
kraj._replace(ime="Vače", prebivalcev=368)
Kraj(ime='Vače', sirina=46.119552, dolzina=14.838005, prebivalcev=368)
```

Terka je še vedno nespremenljiva in zato nespremenjena: kraj je še vedno Sreda.

```
kraj
Kraj(ime='Sreda', sirina=46.119552, dolzina=14.838005, prebivalcev=None)
```

Metoda `_replace` ima nekoliko zavajajoče ime, saj ničesar ne spremeni, temveč sestavi novo terko. Če bi torej v resnici hoteli "spremeniti" terko `kraj`, bi morali -- podobno kot pri nizih, kjer smo pisali `ime = ime.replace("n", "r")` -- napisati

```
kraj = kraj._replace(ime="Vače", prebivalcev=368)
kraj
Kraj(ime='Vače', sirina=46.119552, dolzina=14.838005, prebivalcev=368)
```

Dokumentacija

NamedTuple in namedtuple.