

## Prirejanje rezinam

Spomnimo se, da je rezinam v Pythonu možno tudi prirejati.

```
s = ["Ana", "Berta", "Cilka", "Dani", "Ema"]

s[1:3] = ["Benjamin", "Cene", "Damjan"]

s
['Ana', 'Benjamin', 'Cene', 'Damjan', 'Dani', 'Ema']
```

Tudi v numpyju gre, in še bolj imenitno je, le da prirejanje rezinam nikoli ne bo spremenilo velikosti tabele: tisto, kar prirejamo, mora imeti enake dimenzije kot rezina, ki ji prirejamo.

Imejmo dve tabeli:

```
import numpy as np

a = np.array([5, 1, 8, 7, 3, 12, 5, 9, 4])
b = np.array([42, 0, 13, 66])

b ima štiri elemente. Rezini a[3:7], ki ima prav tako štiri, torej lahko priredimo b.
```

```
a[3:7] = b

a
array([ 5,  1,  8, 42,  0, 13, 66,  9,  4])
```

Seveda lahko režemo tudi b. Priredimo a[3:7] elemente b-ja v obratnem vrstnem redu.

Rezini a[:2] priredimo zadnja dva elementa b.

```
a[:2] = b[-2:]

a
array([13, 66,  8, 42,  0, 13, 66,  9,  4])
```

Vidimo: vse je dovoljeno, dokler je na obeh koncih enako število elementov.

Rezinam lahko prirejamo skalarje (fancy ime za število).

```
a[2:5] = -8

a
array([13, 66, -8, -8, -8, 13, 66,  9,  4])
```

```
a[:,2] = 100
```

```
a
```

```
array([100, 66, 100, -8, 100, 13, 100, 9, 100])
```

Rezinam je seveda možno tudi prištevati, jih s čim pomnožiti...

```
a[:,2] += [1, 2, 3, 4, 5]
```

```
a
```

```
array([101, 66, 102, -8, 103, 13, 104, 9, 105])
```

```
a[:,2] /= 2
```

```
a
```

```
array([50, 66, 51, -8, 51, 13, 52, 9, 52])
```

```
a[:,4] += b
```

```
a
```

```
array([92, 66, 64, 58, 51, 13, 52, 9, 52])
```

In vse to seveda deluje tudi z večdimenzionalnimi tabelami in rezinami.

## Mapiranje - in indeksiranje z večdimenzionalnimi tabelami

Že pri prejšnji nalogi smo videli, da lahko za indekse uporabljamo tudi sezname. To lahko uporabljamo tudi za mapiranje: če vsaka številka ustreza neki osebi, lahko tabelo števil na ta način pretvorimo v tabelo imen oseb.

```
imena = np.array(["Ana", "Berta", "Cilka", "Dani", "Ema"])
```

```
a = np.array([3, 1, 4, 1, 2, 0])
```

```
imena[a]
```

```
array(['Dani', 'Berta', 'Ema', 'Berta', 'Cilka', 'Ana'], dtype='<U5')
```

To ni v bistvu nič novega, le nov pogled na nekaj, kar že znamo. (Novo pa je to, da lahko numpyjeve tabele vsebujejo tudi nize.)

Imenitno pa je, da tole deluje tudi, če je tabela večdimenzionalna.

```
a = np.array([[3, 1, 4], [1, 2, 0]])
```

```
imena[a]
```

```
array([[ 'Dani', 'Berta', 'Ema'],  
       [ 'Berta', 'Cilka', 'Ana']], dtype='<U5')
```

Tabela, ki jo dobimo z `imena[a]`, je enake oblike kot `a`, le da so elementi `a`-ja zamenjani z ustreznimi elementi tabele `imena`.

## Naloga

S tem, kar smo spoznali zdaj, bomo reševali nalogo 20: Trench Map.

Zame je bila to ena najbolj zoprnih nalog - tudi zato, ker sem se je lotil narobe in potem še naredil eno napačno predpostavko. Zato tokrat (še) malo več navodil, kako se je je pametno lotiti. Če vam bo preveč nadležna, pač preskočite reševanje in pogledjte mojo rešitev. Boljše to, kot da se vse skupaj zagabi. Ne bom zameril. :)

Kdor noče spoilerjev, pa lahko tule neha brati.

V začetku preberite podatke v tabelo, ki je velika toliko, kot so pač veliki podatki. Tabela je lahko tipa `bool`; `#` je lahko `True` in `.` bo `False`.

Potem bo potrebno petdesetkrat ponoviti tole.

- Pri meni - in najbrž pri vseh ostalih - je bila transformacija takšna, da se v enem koraku vsi `.` na neskončni površini zamenjajo z `#`, v naslednjem nazaj v `.`, potem spet v `#` ...

Zato pripravite tabelo, ki je za štiri znake večja od trenutnega stanja in jo inicializirajte na `False` ali `True`, odvisno od tega, ali ste v lihem ali sodem koraku. V sredino te tabele - se pravi tako, da okrog ostane rob širine 2 - vnesite trenutno stanje.

- Potem pripravite tabelo vsot. Ta bo za dva znaka (torej rob 1 na vsaki stranici) večja od trenutnega stanja. Vanjo boste devetkrat prišteli "okno" iz tabele iz prejšnje točke, tako da bo to okno drselo 0, 1 in 2 elementa od levega in od zgornjega roba. Pri vsakem prištevanju je potrebno tabelo množiti z ustreznim faktorjem, potenco dvojke. Nič hudega, če zdajle še ne razumete: ko boste brali nalogo, boste razumeli).
- Vsote potem premapirate čez transformacijo in tako dobite novo stanje.

V bistvu ni težko, je pa polno zafrkavanja z drobnimi napakami. Če ne bo šlo, pa samo pogledjte rešitev.