numpy

January 28, 2024

0.1 Indeksiranje s seznamom

Imejmo dve tabeli.

[1]:
```
import numpy as np

a = np.array([5, 1, 8, 7, 3, 12, 5, 9, 4])
b = np.array([2, 0, 3, 4])
```

Sestaviti želimo tabelo, ki vsebuje drugi, ničti, tretji in četrti element tabele `a`. To dobimo tako, da namesto običajnega indeksa podamo seznam indeksov.

[2]:
```
a[[2, 0, 3, 4]]
```

[2]:
```
array([8, 5, 7, 3])
```

Namesto seznama lahko uporabimo kar tabelo. Pravzaprav je to celo bolj običajno. In očitno smo si prav zato pripravili `b`.

[3]:
```
a[b]
```

[3]:
```
array([8, 5, 7, 3])
```

Tudi s takimi indeksi je možno prirejati.

[4]:
```
a[b] = 42

a
```

[4]:
```
array([42,  1, 42, 42, 42, 12,  5,  9,  4])
```

Takšno indeksiranje deluje tudi pri večdimenzionalnih tabelah.

[5]:
```
a = np.array([[5, 8, 7, 1, 3],
              [2, 5, 1, 1, 4],
              [7, 6, 1, 9, 2]])
```

Če je tabela dvodimenzionalna, lahko kot indekse uporabimo dva seznama enakih dolžin. Pare elementov iz teh seznamov bo `numpy` obravnaval kot indekse.

[6]:
```
t = a[[0, 2, 2], [1, 3, 4]]

t
```

[6]:
```
array([8, 9, 2])
```

Tole je tabela, ki vsebuje elemente z indeksi `(0, 1)`, `(2, 3)` in `(2, 4)`.

Kateri so, bomo lažje videli, če jim kaj priredimo.

[7]:
```
a[[0, 2, 2], [1, 3, 4]] = 100

a
```

[7]:
```
array([[  5, 100,   7,   1,   3],
       [  2,   5,   1,   1,   4],
       [  7,   6,   1, 100, 100]])
```

Za to nalogo bodi dovolj. Še nekaj na to temo pa pri naslednji nalogi.

0.2 Tabele samih enakih vrednosti

Še tri funkcije za sestavljanje tabel spoznajmo.

Prva sestavi tabelo ničel. Kot argument moramo podati obliko tabele.

[8]:
```
a = np.zeros((3, 5))

a
```

[8]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

Tip elementov te tabele je `float`. Če bi radi `int`-e, moramo podati še argument `dtype`.

[9]:
```
a = np.zeros((3, 5), dtype=int)

a
```

[9]:
```
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
```

Kaj preostali dve, je očitno.

[10]:
```
np.ones((3, 5), dtype=int)
```

[10]:
```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
```

```
       [1, 1, 1, 1, 1]])
```

[11]:
```
np.full((3, 5), 42)
```

[11]:
```
array([[42, 42, 42, 42, 42],
       [42, 42, 42, 42, 42],
       [42, 42, 42, 42, 42]])
```

0.3 Naloga

Lotite se naloge 13, Transparent Origami.

Izgled "origamija" bomo očitno shranjevali v tabeli ničel in enk. Možno bi bilo tudi zgolj shranjevati koordinate točk, vendar moramo na koncu naloge izrisati sliko, tako da bo prej ko slej potrebno sestaviti takšno tabelo.

Če boste origami shranili kot `bool`, boste lahko pri prepogibanju uporabljali seštevanje: `True + True` bo `True` in ne `2`. Lahko pa ga shranjujete kot `int` in uporabljate bitni `|`. Ali pa vzamete `int` in seštevate, potem pa ste previdni pri interpretaciji številk, ki jih imate.

Najbolj zanimivo bo branje podatkov. Nanj se nanaša vsa "snov", ki smo si jo ogledali zgoraj.

Jaz bi se lotil tako, da bi vsebino datoteke najprej razkosal v koordinate točk in navodila za prepogibanje,

[12]:
```
xy, instructions = open("example.txt").read().split("\n\n")
```

Koordinate so takle niz:

[13]:
```
xy
```

[13]:
```
'6,10\n0,14\n9,10\n0,3\n10,4\n4,11\n6,0\n6,12\n4,1\n0,13\n10,12\n3,4\n3,0\n8,4\n
1,10\n2,14\n8,10\n9,0'
```

Vse številke zložite v tabelo. Iz nje bo lahko ugotoviti dimenzije origamija; uporabite `np.max` po ustrezni osi, potem pa pri klicu `np.zeros` pazite na obliko, to je, kje bo `x` in kje `y`. Potem to tabelico koordinat - oziroma njena stolpca, saj menda znamo dobiti ničti in prvi stolpec? - uporabite za to, da v enem zamahu postavite vse enice, kamor je treba.

Sledi prepogibanje. Prepogibamo vedno na sredi, zato je dovolj preveriti, ali vrstica z navodilom vsebuje `x` (ali `y`). Številko lahko ignorirate, saj jo poznate. Prepogib izvedete tako, da tabelico, ki predstavlja papir, v vsakem koraku zamenjate z novo, ki jo dobite tako, da seštejete levo in prezrcaljeno desno (ali zgornjo in prezrcaljeno spodnjo) polovico tabele. Za prepogib tako ne potrebujemo nobene zanke, le preprosto indeksiranje (vrstic ali stolpcev) in seštevanje v numpyju. Edina zanka v programu bo

```
for instruction in instructions.splitlines()
```

ali kaj podobnega.