resitev

January 28, 2024

0.1 Rezalnik

Napiši razred `Rezalnik`, katerega osnovna funkcija je, da ima metodo `razrezi(s)`, ki razreže podani seznam `s` na sezname določene dolžine. Privzeta dolžina je 2, lahko pa jo spremenimo z metodo `nastavi_dolzino(dolzina)`.

```
>>> s = ["Ana", "Berta", "Cilka", "Donald", "Ema", "Fanči", "Greta", "Helga", "Iva"]
>>>
>>> r = Rezalnik()
>>> r.razrezi(s)
[['Ana', 'Berta'],
 ['Cilka', 'Donald'],
 ['Ema', 'Fanči'],
 ['Greta', 'Helga'],
 ['Iva']]
>>>
>>> r.nastavi_dolzino(4)
>>> r.razrezi(s)
[['Ana', 'Berta', 'Cilka', 'Donald'],
 ['Ema', 'Fanči', 'Greta', 'Helga'],
 ['Iva']]
>>>
>>> r.nastavi_dolzino(3)
>>> r.razrezi(s)
[['Ana', 'Berta', 'Cilka'],
 ['Donald', 'Ema', 'Fanči'],
 ['Greta', 'Helga', 'Iva']]
```

Razred naj ima torej metode:

- `__init__`, ki dela, kar mora,
- `razrezi`, ki prejme nek seznam `s` in vrne seznam podseznamov, ki določene dolžine,
- `nastavi_dolzino`, ki določi dolžino podseznamov. Privzeta dolžina, ki se uporablja, če ne pokličemo te metode, je 2.

Dodatne naloge ni.

0.1.1 Testi

Testi: testi.py

0.1.2 Rešitev

Rezalnik si bo moral zapomniti eno stvar: širino. Konstruktor jo nastavi na 2, `nastavi_dolzino` jo spremeni.

`razrezi(s)` gre prek vseh možnih začetkov koščkov, torej `for i in range(0, len(s), self.dolzina)` in za vsakega sestavi košček, `s[i:i + self.dolzina]`.

[1]:
```
class Rezalnik:
    def __init__(self):
        self.dolzina = 2

    def nastavi_dolzino(self, dolzina):
        self.dolzina = dolzina

    def razrezi(self, s):
        return [s[i:i + self.dolzina]
                for i in range(0, len(s), self.dolzina)]
```

[2]:
```
s = ["Ana", "Berta", "Cilka", "Donald", "Ema", "Fanči", "Greta", "Helga", "Iva"]

rezalnik = Rezalnik()
```

[3]:
```
rezalnik.razrezi(s)
```

[3]:
```
[['Ana', 'Berta'],
 ['Cilka', 'Donald'],
 ['Ema', 'Fanči'],
 ['Greta', 'Helga'],
 ['Iva']]
```

[4]:
```
rezalnik.nastavi_dolzino(3)
```

[5]:
```
rezalnik.razrezi(s)
```

[5]:
```
[['Ana', 'Berta', 'Cilka'],
 ['Donald', 'Ema', 'Fanči'],
 ['Greta', 'Helga', 'Iva']]
```

0.1.3 Rezanje na koščke

Da reč naredimo malo naprednejšo, napišimo funkcijo `chunked(s, i)`, ki ji podamo nekaj, kar je potrebno razrezati in dolžino, vrne pa, kar vrača gornji `razrezi`.

[6]:
```
def chunked(s, dolzina):
    return [s[i:i + dolzina]
            for i in range(0, len(s), dolzina)]
```

Funkcija deluje tudi na nizih.

[7]:
```
chunked("Benjamin", 2)
```

[7]:
```
['Be', 'nj', 'am', 'in']
```

Ne deluje pa na generatorjih.

Na hitro sestavimo generator praštevil do 100 in jih razsekajmo v trojke.

[8]:
```
g = (x for x in range(2, 100) if all(x % n != 0 for n in range(2, x)))

chunked(g, 3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-f22d355f7e54> in <module>
      1 g = (x for x in range(2, 100) if all(x % n != 0 for n in range(2, x)))
      2
----> 3 chunked(g, 3)

<ipython-input-6-4ca9b12daeac> in chunked(s, dolzina)
      1 def chunked(s, dolzina):
      2     return [s[i:i + dolzina]
----> 3             for i in range(0, len(s), dolzina)]


TypeError: object of type 'generator' has no len()
```

To ne deluje, ker generatorji nimajo dolžine (nihče ne ve vnaprej, koliko stvari bodo zgenerirali), poleg tega pa jih ni mogoče indeksirati.

Šlo bi tako.

[9]:
```
def chunked(s, dolzina):
    kosi = []
    for i, x in enumerate(s):
        if i % dolzina == 0:  # smo na meji, začenjamo nov kos
            kosi.append([])
        kosi[-1].append(x)  # dodamo v zadnji kos
    return kosi

g = (x for x in range(2, 100) if all(x % n != 0 for n in range(2, x)))

chunked(g, 3)
```

[9]:
```
[[2, 3, 5],
 [7, 11, 13],
 [17, 19, 23],
 [29, 31, 37],
 [41, 43, 47],
```

```
 [53, 59, 61],
 [67, 71, 73],
 [79, 83, 89],
 [97]]
```

Deluje, ni pa posebej zabavno. Sprogramirajmo tako, da bo.

[10]:
```
def chunked(s, size):
    s = iter(s)
    while True:
        chunk = [x for _, x in zip(range(size), s)]
        if not chunk:
            break
        yield chunk
```

Najprej spremenimo `s` v iterator. To storimo zato, da bo kasneje, ko jo kličemo znotraj `zip`, dajala vedno nove elemente, ne pa začenjala vedno od začetka.

Glavni trik je `zip(range(size), s)`. Ker se `zip` konča takrat, ko zmanjka elementov krajšega od seznamov, bo iz `s` pobral le `size` elementov. To, kar vrača `range`, nas v resnici ne zanima, torej `x for _, x in ...`. Tako sestavimo trenutni kos. Če je prazen, je veselja konec. Če ni, ga vrnemo - z `yield`, ker je dodatno imenitno, da je naš `chunked` generator.

[11]:
```
g = (x for x in range(2, 100) if all(x % n != 0 for n in range(2, x)))

chunked(g, 3)
```

[11]:
```
<generator object chunked at 0x7fc747e4b4d0>
```

[12]:
```
list(chunked(g, 3))
```

[12]:
```
[[2, 3, 5],
 [7, 11, 13],
 [17, 19, 23],
 [29, 31, 37],
 [41, 43, 47],
 [53, 59, 61],
 [67, 71, 73],
 [79, 83, 89],
 [97]]
```

Razred je potem takšen.

[13]:
```
class Rezalnik:
    def __init__(self):
        self.dolzina = 2

    def nastavi_dolzino(self, dolzina):
```

```
        self.dolzina = dolzina

    def razrezi(self, s):
        return list(chunked(s, self.dolzina))
```