

Day 06: Custom Customs

(Povezava na nalogo)

Tudi ta naloga je zelo preprosta, če:

- znamo prebrati datoteko, pri čemer nam bo pomagalo, kar smo se naučili ob dnevu 4,
- znamo delati z množicami, predvsem hitro računati preseke in unije.

Podatki so videti tako.

abc

a
b
c

ab
ac

a
a
a
a

b

Če izpustimo zgodbo in povemo bolj suho: prazne vrstice razdelijo te podatke v skupine vrstic. V prvem delu nas za vsako skupino zanima, koliko različnih črk vsebuje. V drugem delu nas zanima, koliko črk se pojavi v vsaki vrstici.

V teh operacijah takoj prepoznamo unijo in presek vrstic v posamezni skupini, pri čemer si vsako vrstico predstavljamo kot eno množico.

Naloga, konkretno, sprašuje po vsoti velikosti teh unij in presekov. (Za detajle in primer preberi opis naloge).

Dve temi iz Pythona

Vezane in nevezane metode

Pa se naučimo nekaj (morda) novega v Pythonu.

Metode so lahko vezane ali nevezane. Vzemimo recimo `append`. Funkcije `append` ni; `append` je metoda seznama.

```
s = [1, 2, 3]
s.append(4)
```

`s.append` je metoda, seznam `s`. Vezana je na ta konkretni `s`.

```
s.append
```

```
<function list.append(object, /)>
```

Sicer pa je `append`, seveda, metoda razreda `list`. Torej obstaja tudi

```
list.append
```

```
<method 'append' of 'list' objects>
```

Razlika je v tem, da `list.append` ni vezana na noben objekt.

```
list.append(5)
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-4-4726f696a85b> in <module>  
----> 1 list.append(5)
```

```
TypeError: descriptor 'append' requires a 'list' object but received a 'int'
```

Objekt je prvi element, ki ga prejme metoda, le da je pri vezanih metodah že fiksiran. Če ga ni, ga lahko vedno dodamo.

```
list.append(s, 5)
```

```
s
```

```
[1, 2, 3, 4, 5]
```

Tega sicer nikoli ne počnemo, ker ni treba in je nenavadno in, poleg tega, zahteva, da vemo, da je `s` res `list`. Tega torej nisem pokazal zato, da bi uporabljali, temveč predvsem, da veste, da so metode dosegljive tudi direktno prek razredov.

Unije in preseki več množic

Vzemimo dve množici.

```
a = {1, 2, 3}
```

```
b = {4, 5, 6}
```

Njuno unijo praviloma izračunamo z

```
a | b
```

```
{1, 2, 3, 4, 5, 6}
```

Obstaja tudi metoda `union`, ki pa je ne uporabljamo, ker izgleda ... asimetrično.

```
a.union(b)
```

```
{1, 2, 3, 4, 5, 6}
```

To je videti, kot da bi z `b.union(a)` morda dobili kaj drugega. Kot da je enkrat "šef", osnova, `a`, enkrat pa `b`. Pač pa ima obstoj te metode smisel, če - skladno z onim zgoraj - pomislimo, da jo je mogoče poklicati takole:

```
set.union(a, b)
```

```
{1, 2, 3, 4, 5, 6}
```

Lepota `set.union` je, da sprejme tudi več množic.

```
set.union(a, b, {6, 7}, {8, 5, 3})
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

Kaj pa, če imamo seznam množic?

```
m = [{1, 2, 3}, {3, 1, 6}, {3, 5, 1}, {8, 6, 3}]
```

Tu se spomnimo, kako poklicati funkcijo tako, da kot argumente uporabimo elemente določenega seznama (terke, ... generatorja): z zvezdico (znano tudi kot *spread operator*).

```
set.union(*m)
```

```
{1, 2, 3, 5, 6, 8}
```

Zdaj vemo vse in se lahko lotimo reševanja.

Rešitev

Spet lepo počasi, korak za korakom, skonstruirajmo branje podatkov.

Datoteko preberemo v niz in ga razbijemo glede na `\n\n`.

```
open("example.txt").read().split("\n\n")
```

```
['abc', 'a\nb\nc', 'ab\nac', 'a\na\na\na', 'b']
```

Vzemimo vrstico `ab\nac` - ostale so dolgečasne.

```
vrstica = 'ab\nac'
```

Če enkrat jo je potrebno razcepiti in iz vsakega kosa narediti množico.

```
[set(x) for x in vrstica.split()]
```

```
[{'a', 'b'}, {'a', 'c'}]
```

In to storimo za vsako vrstico datoteke.

```
groups = [[set(x) for x in vrstica.split()] for vrstica in open("input.txt").read().split("\n\n")]
```

Preostane nam le še izračunati presek in unijo za vsako skupino ter sešteti njihove velikosti.

```
print(sum(len(set.union(*g)) for g in groups))
```

```
print(sum(len(set.intersection(*g)) for g in groups))
```

```
6382
```

```
3197
```