resitev

January 28, 2024

0.1 Day 04: Passport Processing

(Povezava na nalogo)

Naloga je vobče dolgočasna. Izkoristili jo bomo za to, da se bomo naučili lepo prebrati podatke z eno vrstico kode.

V datoteki so shranjeni podatki iz potnih listov.

```
ecl:gry pid:860033327 eyr:2020 hcl:#fffffd
byr:1937 iyr:2017 cid:147 hgt:183cm

iyr:2013 ecl:amb cid:350 eyr:2023 pid:028048884
hcl:#cfa07d byr:1929

hcl:#ae17e1 iyr:2013
eyr:2024
ecl:brn pid:760753108 byr:1931
hgt:179cm

hcl:#cfa07d eyr:2025 pid:166559648
iyr:2011 ecl:brn hgt:59in
```

Potni listi so ločeni s prazno vrstico. Tako se prvi dve nanašata na isti potni list. Tiste tričrkovne zadeve predstavljajo različne podatke. Podatek `cid` je v obeh delih naloge opcijski, zato bomo podatke že prebrali tako, da ga bomo preskočili.

0.1.1 Branje podatkov

Edini zabavni del te naloge je branje podatkov. Stvar bo poučne predvsem, če ga bomo izvedli v eni vrstici. :) Resno, to bomo storili zato, da bomo videli, kako po korakih sestaviti takšen izraz in kako ga zapisati (skoraj) čitljivo.

Najprej preberemo celotno datoteko in jo razbijmo na kose, med katerimi je prazna vrstica. Torej na kose, ki jih ločuje niz `\n\n`.

[1]:
```
open("example.txt").read().split("\n\n")
```

Dobili smo štiri elemente, kar je že dober znak. Zdaj se skoncentrirajmo le na prvo.

[2]:
```
vrstica = open("example.txt").read().split("\n\n")[0]

vrstica
```

Vidim, da so podatki ločeni s presledki in z `\n`. To je dobro: metoda `split` loči glede na beli prostor in tretira `\n` enako kot presledke.

[3]:
```
vrstica.split()
```

[3]:
```
['ecl:gry',
 'pid:860033327',
 'eyr:2020',
 'hcl:#fffffd',
 'byr:1937',
 'iyr:2017',
 'cid:147',
 'hgt:183cm']
```

Vsakega od teh nizov razbijemo glede na `":"`.

[4]:
```
[x.split(":") for x in vrstica.split()]
```

[4]:
```
[['ecl', 'gry'],
 ['pid', '860033327'],
 ['eyr', '2020'],
 ['hcl', '#fffffd'],
 ['byr', '1937'],
 ['iyr', '2017'],
 ['cid', '147'],
 ['hgt', '183cm']]
```

Izgleda lepo, živce pa nam žre `cid`. Kako se ga znebiti? Vse te pare spustimo na novo skozi generator in odstranimo tiste, katerih prvi element je `cid`.

[5]:
```
[(k, v)
 for k, v in (x.split(":") for x in vrstica.split())
 if k != "cid"]
```

[5]:
```
[('ecl', 'gry'),
 ('pid', '860033327'),
 ('eyr', '2020'),
 ('hcl', '#fffffd'),
 ('byr', '1937'),
 ('iyr', '2017'),
 ('hgt', '183cm')]
```

Če tole malo težko razumete, samo poglejte, kje v tem, zadnjem izrazu se je znašel predzadnji, pa bo najbrž jasno.

Dobimo torej pare. Če te pare podtaknemo tipu `dict`, dobimo točno takšen slovar, kakršnega si želimo.

[6]:
```
dict((k, v)
     for k, v in (x.split(":") for x in vrstica.split())
     if k != "cid")
```

[6]:
```
{'ecl': 'gry',
 'pid': '860033327',
 'eyr': '2020',
 'hcl': '#fffffd',
 'byr': '1937',
 'iyr': '2017',
 'hgt': '183cm'}
```

Slovar, ki smo ga sestavili tule, moramo sestaviti za vsako vrstico datoteke, ne le prvo. Tale `dict` torej "vložimo" v izpeljan seznam, ki bo šel prek datoteke.

[7]:
```
passports = [dict((k, v)
                  for k, v in (x.split(":") for x in vrstica.split())
                  if k != "cid")
             for vrstica in open("example.txt").read().split("\n\n")]
```

[8]:
```
passports
```

[8]:
```
[{'ecl': 'gry',
  'pid': '860033327',
  'eyr': '2020',
  'hcl': '#fffffd',
  'byr': '1937',
  'iyr': '2017',
  'hgt': '183cm'},
 {'iyr': '2013',
  'ecl': 'amb',
  'eyr': '2023',
  'pid': '028048884',
  'hcl': '#cfa07d',
  'byr': '1929'},
 {'hcl': '#ae17e1',
  'iyr': '2013',
  'eyr': '2024',
  'ecl': 'brn',
  'pid': '760753108',
  'byr': '1931',
  'hgt': '179cm'},
 {'hcl': '#cfa07d',
  'eyr': '2025',
  'pid': '166559648',
```

```
 'iyr': '2011',
 'ecl': 'brn',
 'hgt': '59in'}]
```

Tega, kar smo zgoraj napisali v eni vrstici, ni čisto preprosto prebrati, je pa možno. Zato: pišite tako. Ne tako:

```
passports = [dict((k, v) for k, v in (x.split(":") for x in vrstica.split()) if k
!= "cid") for vrstica in open("example.txt").read().split("\n\n")]
```

0.2 Prvi del: potni listi s sedmimi podatki

V prvem delu naloga zahteva, da preštejemo potne liste, ki imajo vseh sedem podatkov.

Že v nalogi za drugi dan smo se naučili, kako s `sum` prešteti elemente, ki ustrezajo določenemu pogoju.

[9]:
```
passports = [dict((k, v)
                  for k, v in (x.split(":") for x in vrstica.split())
                  if k != "cid")
             for vrstica in open("input.txt").read().split("\n\n")]

print(sum(len(passport) == 7 for passport in passports))
```

```
254
```

0.3 Drugi del: pravilnost podatkov

Ta je malo dolgočasen. Različna polja imajo različne kriterije za pravilnost in preveriti je potrebno, ali jih potni list zadošča.

[10]:
```
def valid(p):
        return 1920 <= int(p["byr"]) <= 2002 \
            and 2010 <= int(p["iyr"]) <= 2020 \
            and 2020 <= int(p["eyr"]) <= 2030 \
            and (150 <= int(p["hgt"][:-2]) <= 193 if p["hgt"].endswith("cm")␣
 ↪else 59 <= int(p["hgt"][:-2]) <= 76) \
            and len(p["hcl"]) == 7 and p["hcl"][0] == "#" and all(c in␣
 ↪"0123456789abcdef" for c in p["hcl"][1:]) \
            and p["ecl"] in {"amb", "blu", "brn", "gry", "grn", "hzl", "oth"} \
            and len(p["pid"]) == 9 and all(map(str.isdigit, p["pid"]))

print(sum(valid(passport) for passport in passports if len(passport) == 7))
```

```
184
```

Spet bi lahko vse skupan tlačili v eno vrstico, vendar ne bi bilo ne pregledno ne poučno. Boljše je napisati ločeno funkcijo.

Nekateri so ob reševanju naloge tudi pregledovali - znotraj funkcije, kakršna je gornja `valid`, ali potni list v resnici vsebuje neko polje. Tu smo se tega dela otresli tako, da kličemo `valid` le za

potne liste, ki imajo vseh sedem polj.