

## Izpit 13. februar 2024

### 1. Skupna cena

Napišite funkcijo `cena(zasedenost, cene)`, ki prejme numpyjevo tabelo `zasedenost` tipa `bool`, ki ima šest stolpcev in toliko vrstic, kolikor vrst ima letalo ter vsebuje `True` na mestih, ki ustrezajo zasedenim sedežem in `False` na mestih, ki ustrezajo prostim. Poleg tega dobi Pythonov seznam `cene`, ki vsebuje cene vozovnic za sedeže v prvih treh stolpcih, na primer `[200, 175, 193]`, če sedeži ob oknu (stolpec 0) stanejo 200, sedeži na sredini (stolpec 1) 175 in sedeži ob hodniku (stolpec 2) 193 evrov. Cene sedežev v drugih treh stolpcih so zrcalno enake tem (na primer 193, 175 in 200).

Funkcija mora vrniti vsoto cen vseh prodanih vozovnic. Funkcija mora biti napisana v numpyju. Točkovanje bo upoštevalo eleganco rešitve.

#### Rešitev

Tule je najprej rešitev. Njena razlaga je spodaj.

```
def cena(zasedenost, cene):  
    return np.sum(zasedenost * (cene + cene[::-1]))
```

Zdaj pa razložimo počasi, po korakih.

Najprej smo razširili cene na vseh šest stolpcev. Če imamo *Pythonov seznam* `cene`, na primer

```
cene = [200, 175, 193]
```

lahko k njemu pripnemo še prezrcaljen seznam,

```
cene + cene[::-1]
```

```
[200, 175, 193, 193, 175, 200]
```

pa dobimo cene za vse stolpce. (Nekateri študenti so prezrli, da so cene na desni strani letala zrcalne tem na levi. To rešitev sem štel za "skoraj pravilno", saj bi njihovi programi z malo spremembo delovali pravilno.)

Zdaj pa imamo dve poti. Očitnejša (a, kot se bo izkazalo, malenkost daljša, vendar vam pri ocenjevanju tega nisem štel v slabo), je, da najprej preštejemo `zasedenost` vseh stolpcev.

```
import numpy as np
```

```
zasedenost = np.array([  
    [True, False, False, False, False, False],  
    [False, True, True, False, False, False],  
    [False, False, False, True, True, True],  
    [False, False, False, True, True, True],  
    [False, True, False, True, True, True],
```

```

[False, False, False, True, True, False],
[False, False, False, False, True, False]
])

```

Zasedenost po stolpcih dobimo tako, da tabelo seštejemo po osi 0,

```

np.sum(zasedenost, axis=0)
array([1, 2, 1, 4, 5, 3])

```

To tabelo pomnožimo s cenami.

```

np.sum(zasedenost, axis=0) * (cene + cene[::-1])
array([200, 350, 193, 772, 875, 600])

```

pa dobimo vsote cen po stolpcih. To, končno, seštejemo v skupno ceno.

```

np.sum(np.sum(zasedenost, axis=0) * (cene + cene[::-1]))
2990

```

Druga rešitev je, da celotno tabelo pomnožimo s cenami.

```

zasedenost * (cene + cene[::-1])
array([[200,  0,  0,  0,  0,  0],
       [ 0, 175, 193,  0,  0,  0],
       [ 0,  0,  0, 193, 175, 200],
       [ 0,  0,  0, 193, 175, 200],
       [ 0, 175,  0, 193, 175, 200],
       [ 0,  0,  0, 193, 175,  0],
       [ 0,  0,  0,  0, 175,  0]])

```

In to seštejemo.

```

np.sum(zasedenost * (cene + cene[::-1]))
2990

```

Če bi nas bolelo za učinkovitost, bi bila prva rešitev boljša, saj ima manj množenj. Vendar letala niso tako velika, da bi se to res poznalo, zato je morda boljša druga, krajša. V resnici sta obe dovolj hitri, za potrebe tega predmeta pa sploh.

## 2. Preberi sporočilo

Potovalne agencije pošiljajo potnikom sporočila v slogu

Spoštovani,

potrjujemo nakup vozovnice. Detajli so zapisani spodaj.

Posebne želje: nima posebnih želja.

Ime potnika: Janez Tone Novak

Starost: 34

Cena: 184E

Dodelili smo vam sedež 12A, kot ste želeli.

Vsebina je lahko zelo različna, vedno pa velja naslednje:

- če sporočilo vsebuje ime potnika, je le-to zapisano v vrstici, ki vsebuje dvopičje, desno od dvopičja pa sta dve ali več besed, ki se začnejo z veliko začetnico in ničesar drugega. Predpostaviti smete tudi, da imena in priimki ne vsebujejo šumnikov.
- Če sporočilo vsebuje oznako sedeža, je ta sestavljeno iz ene ali dveh števk, ki jima sledi črka A, B, C, D, E ali F. (Funkcija naj torej deluje le za manjša letala.)

Napišite funkcijo `preberi_sporocilo(ime_datoteke)`, ki kot argument dobi ime datoteke s takšnim sporočilom ter vrne par z imenom potnika in oznako sedeža. Če sporočilo ne vsebuje katerega od teh dveh podatkov, namesto tega podatka vrne `None`. Za primer na sliki funkcija vrne ("`Janez Tone Novak`", "`12A`").

## Rešitev

V nalogi moramo pokazati, da znamo prebrati datoteko in uporabljati regularne izraze. Gre tudi brez, vendar je precej bolj zoprno; bomo videli.

Regularni izraz za besedo, ki se začne z veliko črko in je dolgo vsaj dve črki `[A-Z]\w+` -- `[A-Z]` je velika črka (angleške abecede) in `\w+` pomeni eno ali več črk. Zapleteni del je, da ima potnik dve ali več imen, ločenih s presledki. To dobimo z `(([A-Z]\w+)([A-Z]\w+)+)`. V prvi skupini je prvo ime, v drugi skupini so nadaljnja. Vse to zapremo še v ene oklepaje, da dobimo "skupino", ki vsebuje vsa potnikova imena.

Pred vse skupaj postavimo `.+:` - dvopičje, ki mu sledi presledek. Celoten regularni izraz je torej `(([A-Z]\w+)([A-Z]\w+)+)`. Kar zoprno. Na srečo bo prav ta del možno kolikor toliko spodobno narediti brez regularnih izrazov.

Oznako sedeža poiščemo z `\d\d?[A-F]` - števka, nato morda še ena števka in potem črka med A in F. Manjša težava je, da bo tole našlo tudi `184E` (reklo bo, da gre za sedež 84E, saj najde dve števki, ki ji sledi črka). Da bo naloga rešena popolnoma pravilno (in preživela vse teste), moramo zato povedati še, da pred prvo števko ne sme biti drugi števka, `\D\d\d?[A-F]`. Del, ki vsebuje številko sedeža, zapremo v skupino, da jo bomo lažje izvlekli iz aajdenega podniza, `\D(\d\d?[A-F])`.

Funkcija je potem preprosta: v začetku postavimo `potnik` in `sedez` na `None`, potem v vsaki vrstici poskusimo najti enega od gornjih izrazov. Če uspemo, postavimo ustrežno spremenljivko, kot je treba.

```

import re

def preberi_sporocilo(ime_datoteke):
    potnik = None
    sedez = None
    for vrstica in open(ime_datoteke, encoding="utf-8"):
        mo = re.match(r".+: (([A-Z]\w+)( [A-Z]\w+)+)", vrstica)
        if mo:
            potnik = mo.group(1)
            mo = re.search(r"\D(\d\d?[A-F])", vrstica)
            if mo:
                sedez = mo.group(1)
    return potnik, sedez

```

Pa če nočemo uporabiti regularnih izrazov? (Morda zato, ker jih ne znamo?) V tem primeru postanejo stvari veliko nadležnejše.

V funkciji bomo uporabljali metode `s.isupper()`, `s.isdigit()` in `s.isalpha()`, ki povedo, ali niz vsebuje same velike črke, same števke ali same črke. Očitno bi šlo tudi brez njih, vendar obstajajo meje, do katerih sem si pripravljen v teh zapiskih zaplesti življenje. :)

```

def preberi_sporocilo(ime_datoteke):
    potnik = None
    sedez = None
    for vrstica in open(ime_datoteke, encoding="utf-8"):

        # Imena potnika
        if ":" in vrstica:
            ime = vrstica.split(":")[-1]
            for beseda in ime.split():
                if not (beseda[0].isupper() and beseda.isalpha()):
                    break
            else:
                potnik = ime.strip()

        # Oznaka sedeža
        for i, c in enumerate(vrstica):
            if c.isdigit() \
                and (i == 0 or not vrstica[i - 1].isdigit()):
                if i + 1 < len(vrstica) \
                    and vrstica[i + 1] in "ABCDEF":
                    sedez = vrstica[i:i + 2]
                if i + 2 < len(vrstica) \
                    and vrstica[i + 1].isdigit() \
                    and vrstica[i + 2] in "ABCDEF":
                    sedez = vrstica[i:i + 3]

```

```
return potnik, sedez
```

Da dobimo ime potnika, preverimo, ali vsebuje vrstica dvopičje in v tem primeru predpostavimo, da bi lahko bilo ime tisto, kar sledi (zadnjemu) dvopičju. (Mimogrede: zgoraj smo napisali `ime = vrstica.split(":")[-1]`; bolj Pythonovski način bi bil `*`, `ime = vrstica.split(":", 1)[-1]`, vendar se s takšnim leporečjem pri tem predmetu ne obremenjujemo.)

Vse, kar je sledilo dvopičju, razbijemo na besede in preverimo, ali je prva črka velika in je niz sestavljen iz samih črk. Če naletimo na kakšno besedo, ki ne ustreza temu pogoju, prekinemo zanko. Če se zanka izteče brez prekinitve, smo našli ime.

Na nekem predavanju smo omenili generatorje. Tu bi nam prišli nekoliko prav: namesto

```
for beseda in ime.split():
    if not (beseda[0].isupper() and beseda.isalpha()):
        break
else:
    potnik = ime.strip()
```

bi lahko napisali tudi

```
if all(beseda[0].isupper() and beseda.isalpha() for beseda in ime.split()):
    potnik = ime.strip()
```

Drugi del, iskanje oznake sedeža, je še bolj zoprn. Gremo čez vrstico. Zanimajo nas tista mesta, ki vsebujejo številko in pred njimi ni števke (ali pa smo na začetku; v tem primeru ne gledamo prejšnjega znaka, čeprav bi to tu slučajno delovalo -- pogledali bi pač znak z indeksom `-1`, torej zadnji znak, ta pa je `\n`, kar ni števka).

Če neko mesto ustreza temu pogoju, gre lahko za oznako sedeža. Da bi bilo tako, mu mora slediti črka, ali pa števka in nato črka. Pač ... veliko `if`-ov.

Pri pregledovanju izdelkov se je - po pričakovnjih - pokazalo, da so mnogi pravilno napisali regularni izraz za številke sedežev, izraz za imena potnikov pa je bil seveda trši oreh. V tem primeru bi lahko dobili imena potnikov brez regularnega izraza (prvi, preprostejši del spodnje različice programa), številko sedeža pa z regularnim izrazom.

Z regularnimi izrazi pa je življenje seveda preprostejše.

### 3. Razpored

Neka letalska družba pa sedežev ne dodeljuje vnaprej, temveč lahko potniki zgolj izrazijo željo. Potnike nato vkrcavajo po seznamu. Vsak potnik se usede na želeni sedež, če je ta že zaseden, pa se pomika nazaj po vrstah, dokler ne naleti na vrsto, v kateri je sedež v želenem stolpcu prazen, ter ga zasede. Če so, na

primer, že zasedeni sedeži 12C, 13C in 14C, se bo potnik, ki bi želel sedeti na 12C, usedel na 15C. Letalska družba uporablja dolga letala, ki imajo 130 vrst, obenem pa ji ne gre prav dobro (čudno, čudno), zato smete predpostaviti, da se bo po tem postopku našel sedež za vsakogar.

Napišite funkcijo `razpored(seznam)`, ki dobi seznam s pari (`ime_potnika`, `zeleni_sedež`) in vrne nov seznam, ki je enak temu, le da so sedeži zamenjani z sedeži, ki jih bodo potniki dejansko dobili.

Pomoč: če je `c` črka A, B, C, D, E ali F, je `ord(c) - 65` enak 0, 1, 2, 3, 4 oz. 5. Mogoče vam pride prav.

## Rešitev

Tole je naloga iz množic in iz zanke `while`. Lahko pa tudi ni iz množic. Vsekakor pa je iz zanke `while`. :)

Nekam bo potrebno shranjevati zasedena mesta. Lahko, torej, pripravimo množico `zasedeni` z zasedenimi mesti, poleg tega pa seznam `dejanski`, v katerega bomo postavljali potnike in ga na koncu vrnili.

```
def razpored(zelje):
    zasedeni = set()
    dejanski = []
    for potnik, sedež in zelje:
        vrsta, stolpec = int(sedež[:-1]), sedež[-1]
        while (vrsta, stolpec) in zasedeni:
            vrsta += 1
        zasedeni.add((vrsta, stolpec))
        dejanski.append((potnik, f"{vrsta}{stolpec}"))
    return dejanski
```

Ker smo toliko delali z `numpy`-jem -- pa tudi zaradi tega, kako so bili shranjeni podatki v prvi nalogi -- je gotovo prišlo komu na misel namesto množice uporabiti `numpy`-jevo tabelo. (Priznanje: tudi sam sem nalogo najprej rešil tako. Vendar je to predvsem posledica tega, kaj sem razmišljal, ko sem jo sestavljal.)

```
def razpored(zelje):
    zaseden = np.zeros((130, 6), dtype=bool)
    razpored = []
    for potnik, sedež in zelje:
        vrsta, stolpec = int(sedež[:-1]), ord(sedež[-1]) - 65
        while zaseden[vrsta, stolpec]:
            vrsta += 1
        zaseden[vrsta, stolpec] = True
        razpored.append((potnik, f"{vrsta}{sedež[-1]}"))
    return razpored
```

Takšna rešitev je malenkost bolj zapletena, ker potrebujemo še indeks stolpca. Tega dobimo z `ord`, kot je svetoval opis naloge.

Morda bi koga zmotilo, dvojno shranjevanje podatkov. To, kaj je zasedeno, je razvidno že iz `razpored`, torej je `zasedeni` v bistvu nepotreben. To je res, vendar je nerodno, da je `razpored` seznam, v katerem je zoprno preverjati, ali je določen sedež zaseden. Če že hočemo, pa lahko uporabimo slovar, ki za vsak sedež pove, kdo sedi na njem.

```
def razpored(zelje):
    zasedeni = {}
    for potnik, sedez in zelje:
        vrsta, stolpec = int(sedež[:-1]), sedez[-1]
        while (vrsta, stolpec) in zasedeni:
            vrsta += 1
        zasedeni[vrsta, stolpec] = potnik

    dejanski = []
    for (vrsta, stolpec), potnik in zasedeni.items():
        dejanski.append((potnik, f"{vrsta}{stolpec}"))
    return dejanski
```

Prvi del se je malenkost poenostavil. V `return` pa moramo iz slovarja sestaviti pare potnikov in sedežev.

Drugi del lahko poenostavimo, če znamo sestavljati izpeljane sezname:

```
return [(potnik, f"{vrsta}{stolpec}")
        for (vrsta, stolpec), potnik in zasedeni.items()]
```

## 4. Ravnotežje

Napišite funkcijo `ravnotezje(ime_datoteke)`, ki prejme ime datoteke s podatki, ločenimi z vejico. V prvi vrstici so zapisana imena stolpcev. Eden od njih se imenuje sedež; sedeži so zapisani v običajni obliki, npr. 12F ali 1C ali 128E. Funkcija mora vrniti razliko med številom potnikov, ki sedijo na desni (stolpci D, E, F) in levi (stolpci A, B, C) strani letala. Če, recimo, na desni sedi 7 potnikov več, vrne 7; če je 7 potnikov več na levi, vrne -7.!

### Rešitev

Ta, ki je normalno in pridno delal domače naloge, se ob tej nalogi lahko zahvali za poceni točke. Potrebno je le uporabiti `DictReader` in brati vrednosti v ustreznem stolpcu - to, v čemer so nas vadile naloge od vremena do medvedov. Glede na zadnji znak je bilo potrebno spreminjati ravnotežje letala. Lahko ločeno štejemo levo- in desnosedeeče ter na koncu vrnemo razliko ...

```
import csv
```

```
def ravnotezje(ime_datoteke):
    levo = desno = 0
    for vrstica in csv.DictReader(open(ime_datoteke, encoding="utf-8"), delimiter=","):
        if vrstica["sedez"][-1] <= "C":
            levo += 1
        else:
            desno += 1
    return desno - levo
```

... lahko pa razliko računamo že kar sproti.

```
def ravnotezje(ime_datoteke):
    ravno = 0
    for vrstica in csv.DictReader(open(ime_datoteke, encoding="utf-8"), delimiter=","):
        if vrstica["sedez"][-1] <= "C":
            ravno -= 1
        else:
            ravno += 1
    return ravno
```

*Kakor vam drago, je rekel Vili.*

## 5. Leti

Napišite funkcijo `vozni_redi(potniki, ime_datoteke)`. Argument `potniki` je seznam z imeni potnikov, številkami letov in pari (ura, minuta) za odhod in prihod. Primer podatkov je, recimo `[("Ana Argon", "LH2832", (12, 10), (13, 20)), ("Berta Bor", "U0391", (15, 5), (20, 30)), ("Cilka Cankar", "LH192", (7, 0), (12, 30))]`. Funkcija mora v datoteko s podanim imenom zapisati tabelico z imeni, leti in časi, v naslednji obliki.

Ana Argon	LH2832	12:10-13:20
Berta Bor	U0391	15:05-20:30
Cilka Cankar	LH192	7:00-12:30

Točno obliko - število presledkov - razberite iz testov.

### Rešitev

Tudi tale je bila kar poceni.

```
def vozni_redi(potniki, ime_datoteke):
    f = open(ime_datoteke, "w", encoding="utf-8")
    for potnik, let, (od_h, od_m), (pri_h, pri_m) in potniki:
        f.write(f"{potnik:20} {let:>6} {od_h:2}:{od_m:02}-{pri_h:2}:{pri_m:02}\n")
```

Ta naloga se je izkazala za - do neke mere - najlažjo. Preglavice so vam delale le vodilne ničle - večina je pisala, na primer `{od_m:2}` namesto `{od_m:02}`. Nekateri so se znašli tako, da so pred izpisom preverili, ali je število manjše od 10 in v tem primeru dodalo manjkajočo ničlo. No, tudi tako se da.



## Celotna rešitev

Ker smo tule napisali veliko različnih rešitev, združimo vse skupaj še v "pričakovano" rešitev izpita. Tudi za vtis o dolžini celotnega izpita.

```
import re
import csv

import numpy as np

def cena(zasedenost, cene):
    return np.sum(zasedenost * (cene + cene[::-1]))

def preberi_sporocilo(ime_datoteke):
    potnik = None
    sedez = None
    for vrstica in open(ime_datoteke, encoding="utf-8"):
        mo = re.match(r".+: (([A-Z]\w+)( [A-Z]\w+)+)", vrstica)
        if mo:
            potnik = mo.group(1)
        mo = re.search(r"\D(\d\d?[A-F])", vrstica)
        if mo:
            sedez = mo.group(1)
    return potnik, sedez

def razpored(zelje):
    zasedeni = set()
    dejanski = []
    for potnik, sedez in zelje:
        vrsta, stolpec = int(sedež[:-1]), sedez[-1]
        while (vrsta, stolpec) in zasedeni:
            vrsta += 1
        zasedeni.add((vrsta, stolpec))
        dejanski.append((potnik, f"{vrsta}-{stolpec}"))
    return dejanski

def ravnotezje(ime_datoteke):
    ravno = 0
    for vrstica in csv.DictReader(open(ime_datoteke, encoding="utf-8"), delimiter=","):
        if vrstica["sedež"][-1] <= "C":
            ravno -= 1
        else:
```

```

        ravno += 1
    return ravno

def vozni_redi(potniki, ime_datoteke):
    f = open(ime_datoteke, "w", encoding="utf-8")
    for potnik, let, (od_h, od_m), (pri_h, pri_m) in potniki:
        f.write(f"{potnik:20} {let:>6} {od_h:2}:{od_m:02}-{pri_h:2}:{pri_m:02}\n")

```