

## 1. Statistika

Klicni center beleži podatke o številu klicev v numpyjevo tabelo z  $24 * 60$  elementi. Element z indeksom  $i$  pove število klicev v  $i$ -ti minuti dneva. Napišite naslednje funkcije:

- `po_urah(a)` prejme tabelo, kot jo opisujemo zgoraj in vrne tabelo s 24 elementi, ki vsebujejo število klicev v posamezni uri dneva;
- `naj_ura(a)` vrne uro z največ klici (npr. 5, če je največ klicev med peto in šesto uro);
- `brez_klicev(a)` naj vrne število minut, ko ni bilo klicev (torej število ničelnih elementov podane tabele  $a$ ).

Pri ocenjevanju bom upošteval tudi vašo spretnost uporabe knjižnice numpy.

### Rešitev

Pri prvi nalogi uporabimo `reshape`, s katerim spremenimo tabelo tako, da je vsaka vrstica ena ura. Potem to seštejemo po osi 1.

```
def po_urah(a):  
    return a.reshape(24, 60).sum(axis=1)
```

Za drugo nalogo se moramo spomniti `argmax`.

```
def naj_ura(a):  
    return po_urah(a).argmax()
```

Tretjo pa najlažje uženemo, če se spomnimo, da je `True` isto (ali vsaj enako) kot 1.

```
def brez_klicev(a):  
    return np.sum(a == 0)
```

Obstajajo tudi drugačne rešitve; nekatere morda nič slabše od teh.

## 2. Izpis

Napišite funkcijo `izpis(a)`, ki prejme tabelo s 24 elementi, ki predstavljajo, število klicev po urah in vrne niz, oblikovan natančno tako (do presledka enako!), kot kaže slika.

```
0 - 1      20 ##  
1 - 2      34 ###  
2 - 3      66 #####  
3 - 4      82 #####  
4 - 5     114 #####  
5 - 6     125 #####  
6 - 7     204 #####  
7 - 8     272 #####  
8 - 9     364 #####
```

```

9 - 10    453 #####
10 - 11   522 #####
(in tako naprej do)
23 - 24    36 ###

```

Prvi številki povesta uro dneva; začetek je poravnan desno, konec levo. Sledi število klicev v tej uri in "histogram", pri čemer vsak znak # predstavlja (dopolnjenih 10) klicev; pri 66, na primer, imamo 6 znakov #.

### Rešitev

Kdor zna, naredi tako

```

def izpis(ure):
    return "\n".join(f'{i:>2} - {i + 1:<2}    {x:3} {"#" * (x // 10)}' for i, x in enumerate(ure))

```

Kdor ni prijatelj generatorjev, pa tako

```

def izpis(ure):
    s = ""
    for i, x in enumerate(ure):
        s += f'{i:>2} - {i + 1:<2}    {x:3} {"#" * (x // 10)}\n'
    return s

```

Bistvo naloge je oblikovanje nizov, to pa je v obeh rešitvah enako.

## 3. Pravilnost

Napišite funkcijo `preveri(ime_datoteke)`, ki prejme ime datoteke, ki vsebuje besedilo, kakršnega vrne prejšnja funkcija. Funkcija vrne `True`, če je izpis pravičen in `False` če ni.

Predpostaviti smete, da datoteka vsebuje pravilno število (24) vrstic in da so oblikovane pravilno. Preveriti pa mora, da so pravilni začetki in konci (torej, da si ure sledijo v vrstnem redu 0 – 1, 1 – 2, 2 – 3 in tako naprej) ter da se število znakov # ujema s številom klicev (deljenim z 10).

### Rešitev

```

def preveri(ime_datoteke):
    for i, vrstica in enumerate(open(ime_datoteke)):
        od, minus, do, koliko, hashi = vrstica.split()
        if int(od) != i or int(do) != i + 1 or len(hashi) != int(koliko) // 10:
            return False
    return True

```

Odpremo datoteko, beremo po vrsticah, zraven pa uporabimo še `enumerate`, da vemo, v kateri vrstici smo.

Vsako vrstico razkosamo glede na presledke. Ker naloga zagotavlja, da je oblika vrstice pravilna, vemo, da bo "sestavnih delov" pet. Potem za vsakega preverimo, da je takšen, kot mora biti. Če ni vrnemo `False`.

Če se zanka izteče brez napake, pa vrnemo `True`.

## 4. Operaterji

V datoteki v formatu csv so shranjeni prejeti klici – za vsakega vemo, kateri operater ga je sprejel, koliko minut je trajal, kdaj se je začel in za kakšno vrsto klica je šlo (zadnja dva podatka sta nepomembna).

```
operater,dolzina,zacetek,tip
Ana,10,123,I
Berta,2,453,I
Cilka,5,134,0
Berta,10,500,T
Ana,3,135,I
Dani,5,245,T
Berta,3,573,I
Cilka,4,262,I
cbgadhbb,5,157,T
```

Napišite funkcijo `obremenitve(ime_datoteke)`, ki vrne slovar, katerega ključi so imena operaterjev, vrednosti pa število minut, ki jih je operater preživel na klicih. Za gornji primer vrne `{"Ana": 13, "Berta": 15, "Cilka": 9, "Dani": 5, "cbgadhbb": 5}`.

Poleg tega napišite funkcijo `naj_obremenjeni(ime_datoteke)`, ki vrne ime najbolj obremenjenega operaterja glede na skupno število minut; v gornjem primeru vrne `"Berta"`. Če je najbolj obremenjenih več, lahko vrne poljubnega med njimi.

### Rešitev

```
def obremenitve(ime_datoteke):
    operaterji = defaultdict(int)
    for klic in csv.DictReader(open(ime_datoteke)):
        operaterji[klic['operater']] += int(klic['dolzina'])
    return operaterji

def naj_obremenjeni(ime_datoteke):
    ob = obremenitve(ime_datoteke)
    return max(ob, key=ob.get)
```

Tule smo uporabili dva trika. Prvi je `defaultdict`. Če bi namesto njega uporabili običajen slovar, bi morali v zanki pred prištevanjem preveriti, da smo

na tega operaterja že kdaj naleteli, torej, da ključ s tem imenom operaterja že obstaja.

V `naj_obremenjeni` pa smo uporabili `max` z argumentom `key=ob.get`: ključne primerjamo glede na to, kaj zanje vrača slovarjeva metoda `get`. Brez tega bi morali pisati to, kar smo pač vedno pisali na predavanjih pri reševanju te klasične naloge (preden sem - če sem - pokazal ta trik).

## 5. Hierarhija

Mogoče je klicnih centrov več, postavljeni pa so v neko hierarhijo. Osnovna sta Koper in Maribor. Celje je v hierarhiji podrejeno Mariboru. Ptuj tudi. Ormož je podrejen Ptuj. Ljutomer tudi Ptuj. Laško Celju. Postojna Kopru. Vse doslej naštetu bi lahko shranili v takšen seznam: `[("Celje", "Maribor"), ("Ptuj", "Maribor"), ("Ormož", "Ptuj"), ("Ljutomer", "Ptuj"), ("Laško", "Celje"), ("Postojna", "Koper")]`. Krajev je seveda veliko več in hierarhija je lahko poljubno globoka.

Napiši funkcijo `centri(odvisnosti)`, ki prejme seznam, kot je gornji in vrne slovar obrnjenih parov: ključi so kraji, vrednosti pa množice krajev, ki so mu podrejeni. V gornjem primeru mora vrniti `{"Maribor": {"Celje", "Ptuj"}, "Ptuj": {"Ormož", "Ljutomer"}, "Celje": {"Laško"}, "Koper": {"Postojna"}}`. Kraji, ki jim ni podrejen noben kraj, naj se ne pojavijo kot ključi.

Pomoč: to ni nujno naloga iz rekurzije.

### Rešitev

To nikakor ni naloga iz rekurzije. Čeprav imam rekurzijo rad, se mi ne sanja, zakaj bi hotel to nalogo reševati rekurzivno.

```
def centri(odvisnosti):
    obratni = defaultdict(set)
    for od, do in odvisnosti:
        obratni[do].add(od)
    return obratni
```