

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 pravilen 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 json 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. Primer datoteke je na sliki (prelom vrstic je lahko tudi drugačen!).

```
[{"operator": "Ana", "dolzina": 10, "zacetek": 123, "tip": "I"},
 {"operator": "Berta", "dolzina": 2, "zacetek": 453, "tip": "I"},
 {"operator": "Cilka", "dolzina": 5, "zacetek": 134, "tip": "O"},
 {"operator": "Berta", "dolzina": 10, "zacetek": 500, "tip": "T"},
 {"operator": "Ana", "dolzina": 3, "zacetek": 135, "tip": "I"},
 {"operator": "Dani", "dolzina": 5, "zacetek": 245, "tip": "T"},
 {"operator": "Berta", "dolzina": 3, "zacetek": 573, "tip": "I"},
 {"operator": "Cilka", "dolzina": 4, "zacetek": 262, "tip": "I"},
 {"operator": "eaaudgef", "dolzina": 5, "zacetek": 157, "tip": "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, "eaaudgef": 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.

Pomagajte si s funkcijo `json.load(datoteka)`. Funkcija prejme odprto datoteko, ne niza z imenom!

Rešitev

```
def obremenitve(ime_datoteke):
    operaterji = defaultdict(int)
    for klic in json.load(open(ime_datoteke)):
        operaterji[klic['operator']] += 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. Brez klicev

Napišite funkcijo `obdobje_brez(a)`, ki prejme takšen argument kot funkcije iz prejšnje naloge, in vrne začetek in konec najdaljšega obdobja brez prehodov. Če so vsi elementi na indeksih od, na primer, 150 do (vključno) 180 enaki 0 in je to tudi najdaljše zaporedje ničel, mora funkcija vrniti (150, 180). Da bo reševanje lažje, so v testih tudi trije primeri s tabelami, ki nimajo 24×60 temveč le 13 števil.

Rešitev

Izpit je bil kar lahek (meni se je zdel, pa tudi asistent, ki ga je pregledal, ga je ocenil tako). Samo ta naloga je bila bolj zafrknjena, zato sem jo dal na konec.

Ena možna rešitev je

```
def obdobje_brez(a):
    naj_zac = 1
    naj_kon = 0
    zac = None
    for i, x in enumerate(a):
        if x == 0:
            if zac is None:
                zac = i
            if i - zac > naj_kon - naj_zac:
                naj_zac, naj_kon = zac, i
        else:
            zac = None
    return naj_zac, naj_kon
```

Gremo prek seznama. `zac` hrani indeks začetka trenutnega zaporedja ničel. Če trenutno nismo v zaporedju ničel, je `zac` enak `None`.

Če je trenutni element enak 0, pogledamo, ali je trenutno zaporedje daljše od najdaljšega doslej. Če, potem si zapomnimo njegove meje:

```
if i - zac > naj_kon - naj_zac:
    naj_zac, naj_kon = zac, i
```

Če trenutni element ni enak 0, pa le postavimo `zac` na `None`.

Ostane le še malo administracije. V začetku se delamo, da se je najdaljše zaporedje začelo z 1 in končalo z 0. Tako je dolgo -1 in bo že prva ničla, na katero bomo naleteli, predstavljala daljše zaporedje.

Drugi košček administracije je

```
if zac is None:
    zac = i
```

Če naletimo na 0 in doslej še nismo bili v zaporedju ničel, si zapomnimo trenutni indeks kot začetek trenutnega zaporedja.

Nalogo je možno rešiti še na veliko načinov. Gornji morda ni najbolj učinkovit, ker prepogosto nastavlja `naj_zac` in `naj_kon`. Načelno bi bilo boljše, če bi vsakič, ko se zaporedje konča, preverili, ali je bilo to zaporedje daljše od najdaljšega doslej. To bi povzročilo nekaj sitnosti, če je najdaljše zaporedje ravno na koncu ...

Tule je zabavnejša rešitev.

```
def obdobje_brez(a):
    s = ""
    for x in a:
        if x == 0:
            s += "x"
        else:
            s += " "
    naj = max(s.split())
    zac = s.index(naj)
    return zac, zac + len(naj) - 1
```

zaporedje števil pretvorimo v niz, sestavljen iz x-ov in presledkov: ničle spreminimo v `x` in ne-ničle v presledke. Nato na tem nizu pokličemo `split`. Dobimo "besede"; vsaka je sestavljena iz toliko x-ov, kolikor je bilo zaporednih ničel. "Maksimalna" beseda je beseda, ki je zadnja po abecedi; vse besede so sestavljene iz enakih črk (x-ov); v tem primeru je kasneje po abecedi tista, ki je daljša. `naj` bo torej najdaljša beseda. Pogledamo, kje v nizu se nahaja, pa imamo začetni indeks zaporedja; končni indeks dobimo tako, da k začetnemu prištejemo dolžino besede.

Rešitev lahko skrajšamo v

```
def obdobje_brez(a):
    s = "".join(" x"[x == 0] for x in a)
    naj = max(s.split())
    zac = s.index(naj)
    return zac, zac + len(naj) - 1
```

Ta rešitev je sicer precej kavbojska.

Najbrž najlepše pa je poiskati vse začetke in konce intervalov ničel. Potem zipnemo oba seznama (ali generatorja, če hočemo biti še bolj *fancy*) skupaj, in vrnemo par z največjo razliko.

```
def obdobje_brez(a):
    zacetki = (i for i in range(len(a))
               if a[i] == 0 and (i == 0 or a[i - 1] != 0))
    konci = (i for i in range(len(a))
             if a[i] == 0 and (i == len(a) - 1 or a[i + 1] != 0))
    return max(zip(zacetki, konci), key=lambda x: x[1] - x[0])
```

To gre potem čisto lepo celo v enovrstični izraz (čeprav je večvrstični natančno enako učinkovit, hkrati pa preglednejši).

```
def obdobje_brez(a):
    zacetki = (i for i in range(len(a))
               if a[i] == 0 and (i == 0 or a[i - 1] != 0))
    konci = (i for i in range(len(a))
             if a[i] == 0 and (i == len(a) - 1 or a[i + 1] != 0))
    return max(zip(zacetki, konci), key=lambda x: x[1] - x[0])
```

Če povem po pravici: ta rešitev mi ob sestavljanju izpita ni prišla na misel. Idejo sem dobil od študenta,