

MOL je poslal zemljevid ovir na kolesarski poti. Zemljevid je shranjen kot seznam nizov, ki predstavljajo "vrstice": # predstavlja oviro, . pa prosto pot.

Ker smo prejšnji teden napisali program, ki sprejme opis ovir kot seznam trojk ( $x_0$ ,  $x_1$ ,  $y$ ), bi radi pretvorili MOlov zemljevid v takšno obliko.

## Ogrevalna naloga

Napiši program, ki se začne z opisom ovire v eni vrstici, na primer

```
ovire = ".##.####...##"
```

Program mora izpisati seznam **parov**, ki predstavljajo začetke in konce ovir. Za gornjo postavitev je to [(2, 3), (6, 9), (13, 14)]. **Ne spreglej**, da koordinate niso opisane po "Pythonovski": prvi stolpec ima indeks 1 in obe meji sta vključeni ((6, 9) pokriva tudi vrstico 9, torej vsebuje 4 #).

Program obvezno preskusi na nizih ".##.####...##" in ##.##...# (ki se pretvori v [(1, 2), (5, 5), (9, 9)]). **Pazi: v podatkih morajo biti napisani nizi v točno takšni obliki. Tu ne dodajaj ničesar na začetel ali konec; če želiš, pa lahko to kasneje naredi program sam.**

## Rešitev

Da bomo preskušali vse sitnosti naenkrat, vzemimo tale niz:

```
ovire = "##...#.###.##"
#      12345678901234
```

Vsebuje oviro na začetku in na koncu, oviro dolžine 1 in oviri, ki sta razmaknjeni za 1. Spodaj smo napisali številke, da lažje razberemo pričakovani rezultat: [(1, 2), (6, 6), (7, 9), (13, 14)].

Nalogo je možno rešiti na kup načinov. Vsem je skupno to, da se je potrebno dobro zorganizirati. Da naloga ne bi bila pretežka, sem na predavanju, namenjenem začetnikom (starejši mački pa naj se kar malo potrudijo) predlagal, da na začetek in konec niza prilepijo piko, potem pa opazujejo vse #, ki so takoj za ali tik pred piko. Zanimajo nas njihovi indeksi; tisti za piko predstavljajo začetke, tisti po pa konce ovir.

```
ovire = "##...#.###.####.##"
```

```
ovire = "." + ovire + "."
bloki = []
for i in range(len(ovire)):
    if ovire[i] == "#":
        if ovire[i - 1] == ".":
            zacetek = i
        if ovire[i + 1] == ".":
```

```

        bloki.append((zacetek, i))
print(bloki)

[(1, 2), (6, 6), (8, 10), (12, 15), (17, 18)]

```

Če vidimo #, pred katerim je ., si zapomnim, da je to začetek. Če vidimo #, za katerim je ., v seznam blokov dodamo blok od začetka do trenutnega indeksa.

Za posamične # se zgodi oboje naenkrat, zato tu nismo uporabili kekega `else` ali `elif`.

Stolpce v tej nalogi štejemo od 1 ne od 0. Naši indeksi bodo pravilni, saj smo na začetek dodali ..

Zdaj pa izboljšave.

Na predavanjih običajno stokam, da `range(len(ovire))` ni lepo pisati - sploh, kadar potrebuješ tako indeks kot element. Raje pišimo:

```

ovire = "##...#.###.##"

ovire = "." + ovire + "."
bloki = []
for i, znak in enumerate(ovire):
    if znak == "#":
        if ovire[i - 1] == ".":
            zacetek = i
        if ovire[i + 1] == ".":
            bloki.append((zacetek, i))
print(bloki)

[(1, 2), (6, 6), (8, 10), (13, 14)]

```

Se lahko zgodi, da bi bil indeks `i + 1` prevelik? To bi bilo možno le, če bi bil zadnji znak # in bi se torej spraševali po znaku za njim. To pa ne more biti, saj smo na konec dodali piko. Prav tako `i - 1` nikoli ne bo -1, saj je ničti znak vedno . in ne #.

Še lepše je, če uporabimo čistokrvni `zip`. V tem primeru lahko pike dodamo kar v klicu `zip`-a.

```

ovire = "##...#.###.##"

bloki = []
for i, (prej, znak, potem) in enumerate(zip(".", ovire, ovire[1:] + "."), start=1):
    if znak == "#":
        if prej == ".":
            zacetek = i
        if potem == ".":

```

```

        bloki.append((zacetek, i))
print(bloki)

[(1, 2), (6, 6), (8, 10), (13, 14)]

```

Tule se je sicer zdaj zgodilo kar nekaj reči, o katerih se na predavanjih še nismo pogovarjali.

Najprej, kaj zipamo. Tole

Okrog `zip`-a pride še `enumerate`, da bomo imeli indekse; vse skupaj je potrebno razpakirati kot `for i, (prej, znak, potem)`, saj imamo pare (indeks, element), pri čemer je element trojka.

Končno, ker morajo indeksi teči od 1, smo v `enumerate` dodali `start=1`.

Zdaj pa malo drugačna različica: zabeležimo vse spremembe.

```
ovire = "##...#.###.##"
```

```

ovire = "." + ovire + "."
spremembe = []
for i in range(len(ovire)):
    if ovire[i] != ovire[i - 1]:
        spremembe.append(i)

```

```
spremembe
```

```
[1, 3, 6, 7, 8, 11, 13, 15]
```

Te indekse je potrebno popariti:

```
list(zip(spremembe[::2], spremembe[1::2]))
```

```
[(1, 3), (6, 7), (8, 11), (13, 15)]
```

Žal so drugi elementi teh terk - konci blokov - preveliki. Ker terk ni mogoče spreminjati, nam (dokler se ne naučimo, kako to storiti drugače) ne preostane drugega kot prepisati jih v nov seznam. No, saj ni tako hudo.

```
ovire = "##...#.###.##"
```

```

ovire = "." + ovire + "."
spremembe = []
for i in range(len(ovire)):
    if ovire[i] != ovire[i - 1]:
        spremembe.append(i)

```

```
bloki = []
```

```
for zac, kon in zip(spremembe[::2], spremembe[1::2]):
```

```

        bloki.append((zac, kon - 1))

bloki

[(1, 2), (6, 6), (8, 10), (13, 14)]

Pa brez dodajanja pik na začetek in konec? Gre, samo malo več sitnosti je.
Recimo tako:

ovire = "##...#.###.##"

prej = "."
bloki = []
for i, znak in enumerate(ovire, start=1):
    if znak != prej:
        if znak == "#":
            zacetek = i
        else:
            bloki.append((zacetek, i))
    prej = znak

if ovire[-1] == "#":
    bloki.append((zacetek, len(ovire)))

```

```

bloki

[(1, 3), (6, 7), (8, 11), (13, 14)]

```

V spremenljivko `prej` beležimo, kakšen je bil prejšnji znak. V začetku se pretvarjamo, da je ".". To poskrbi, da zaznamo oviro na začetku niza. Za oviro na koncu pa poskrbimo z `if` po zanki.

Če ne vemo, da ima `enumerate` lahko tudi argument `start`, namesto `i` vedno pišemo `i + 1`.

## Obvezna naloga

Napiši program, ki prejme celotni zemljevid in izpiše seznam ovir, kot smo ga vajeni iz prejšnje naloge. Če zemljevid definiramo z

```

zemljevid = [
    ".....",
    "..##..",
    ".##.#.",
    "...###",
    "###.##",
]

```

mora izpisati `[(3, 4, 2), (2, 3, 3), (5, 5, 3), (4, 6, 4), (1, 3, 5), (5, 6, 5)]`.

Pazi, tudi vrstice so oštevilčene od 1.

### Rešitev

Vzamemo katerokoli od prejšnjih rešitev in jo zapremo v zanko

```
for y, ovire in enumerate(zemljevid, start=1):
```

ter k paru koordinat  $x$  dodamo še številko vrstice,  $y$ . Če ne vemo, da lahko funkciji `enumerate` podamo argument `start=2`, pa namesto  $y$  dodajamo  $y + 1$ .

```
zemljevid = [  
    ".....",  
    "..##..",  
    ".##.#",  
    "...###",  
    "###.##",  
]
```

```
bloki = []
```

```
for y, ovire in enumerate(zemljevid):  
    ovire = "." + ovire + "."  
    for i, znak in enumerate(ovire):  
        if znak == "#":  
            if ovire[i - 1] == ".":  
                zacetek = i  
            if ovire[i + 1] == ".":  
                bloki.append((zacetek, i, y + 1))
```

```
print(bloki)
```

```
[(3, 4, 2), (2, 3, 3), (5, 5, 3), (4, 6, 4), (1, 3, 5), (5, 6, 5)]
```

### Dodatna naloga

Naredi obratno: napiši program, ki na podlagi seznama ovir izpiše zemljevid.

Podatki naj bodo zapisani v spremenljivki `ovire`, na primer

```
ovire = [(3, 4, 2), (2, 3, 3), (5, 5, 3), (4, 6, 4), (1, 3, 5), (5, 6, 5)]
```

### Rešitev

Najprej ugotovimo širino in višino ter sestavimo prazen zemljevid:

```
sirina = visina = 0  
for _, x1, y in ovire:  
    if x1 > sirina:  
        sirina = x1  
    if y > visina:
```

```

visina = y

# Kot bomo nekoč videli, je tole malo nevarno,
# ampak pri nizih deluje
zemljevid = ["." * sirina] * visina

zemljevid

['.....', '.....', '.....', '.....', '.....']

Potem napolnimo zemljevid: gremo čez ovire in menjamo ustrezne vrstice
zemljevida s takšnimi, v katere dodamo ovire.

for x0, x1, y in ovire:
    vrstica = zemljevid[y - 1]
    zemljevid[y - 1] = vrstica[:x0 - 1] + "#" * (x1 - x0 + 1) + vrstica[x1:]
    #
    #               del pred oviro           ovira           po oviri

```

V resnici je kar preprosto, le o indeksih moramo razmišljati. Od  $y$  je potrebno odšteti 1, ker ima prva vrstica v ovire številko 1, Python pa šteje od 0. Iz istega razloga je potrebno odšteti 1 od  $x0$ . Dolžina ovire je  $x1 - x0 + 1$ ; ena prištejemo, ker ovira obsega tudi  $x1$ . Del po oviri pa se začne po  $x1$ ; tu ne prištejemo ničesar.

Preverimo rezultat:

```

for vrstica in zemljevid:
    print(vrstica)

.....
..##..
.##.#.
...###
###.##

```

Celotna rešitev dodatne naloge je torej:

```

sirina = visina = 0
for _, x1, y in ovire:
    if x1 > sirina:
        sirina = x1
    if y > visina:
        visina = y
zemljevid = ["." * sirina] * visina

for x0, x1, y in ovire:
    vrstica = zemljevid[y - 1]
    zemljevid[y - 1] = vrstica[:x0 - 1] + "#" * (x1 - x0 + 1) + vrstica[x1:]

```

Zanimivo, da je del s iskanjem širine in višine celo daljši od dejanske naloge. Ko bomo znali več, bomo pisali

```
zemljevid = [ "." * max(x1 for _, x1, _ in ovire)] * max(y for _, _, y in ovire)
```

In če bi bile ovire podane bolj Pythonovsko - tako da bi šteli od 0 in ne vključili gornje meje, bi bil drugi del nekoliko preprostejši:

```
for x0, x1, y in ovire:
    vrstica = zemljevid[y]
    zemljevid[y] = vrstica[:x0] + "#" * (x1 - x0) + vrstica[x1:]
```

### Izziv: rešitev z numpy

Nalogo je čisto preprosto rešiti z numpy-jem. Ampak samo tistim, ki obvladajo numpy.

```
import numpy as np
```

```
zemljevid = [
    ".....",
    "..##..",
    ".##.#.",
    "...###",
    "###.##",
]
```

```
zemljevid = np.array([[0] + [int(c == "#")
                             for c in vrstica] + [0] for vrstica in zemljevid])
```

zemljevid je po tem tabela v numpy-ju, ki vsebuje enke na mestih, kjer so ovire; levo in desno sta dodana še stolpca ničel.

```
zemljevid
```

```
array([[0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 0, 0, 0],
       [0, 0, 1, 1, 0, 1, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 0],
       [0, 1, 1, 1, 0, 1, 1, 0]])
```

Ovire se začnejo tam, kjer enica sledi ničli in končajo tam, kjer enici sledi ničla.

S tremi vrsticami numpy-ja - **brez kakršnekoli zanke v Pythonu** - je možno pridelati tabelo, ki je v bistvu rešitev naloge:

```
array([[3, 4, 2],
       [2, 3, 3],
       [5, 5, 3],
       [4, 6, 4],
       [1, 3, 5],
```

```
[5, 6, 5]])
```

Znaš?

Nasveti:

- glavni trik je odštevanje po stolpcih. Tabela, ki ji odbijemo prvi stolpec je potrebno odšteti od tabele, ki ji odbijemo zadnjega. Ali nekaj podobnega. Potem pa preveriti, kje v rezultatu so enice.
- Poglej funkcijo `np.nonzero`. Z njo hitro prideš do koordinat začetkov in koncev ovir. Rezultat razpakiraš v `y` in `x0` oziroma `y` in `x1`. (Če boš prav naredil, boš dvakrat dobil isti `y`).
- Če so vse številke v kaki tabeli prevelike ali premajhne za 1, ji odštej ali prištej 1.
- Če imaš tri enodimenzionalne tabele, ki jih želiš zložiti v eno, lahko uporabiš `np.column_stack`.

Če rešiš nalogo na ta način (in si prepričan, da deluje pravilno), jo lahko oddaš kot rešitev obveznega dela domače naloge.

## Rešitev

Tole je zemljevid brez prvega stolpca:

```
zemljevid[:, 1:]  
array([[0, 0, 0, 0, 0, 0, 0],  
       [0, 0, 1, 1, 0, 0, 0],  
       [0, 1, 1, 0, 1, 0, 0],  
       [0, 0, 0, 1, 1, 1, 0],  
       [1, 1, 1, 0, 1, 1, 0]])
```

In tole brez zadnjega:

```
zemljevid[:, :-1]  
array([[0, 0, 0, 0, 0, 0, 0],  
       [0, 0, 0, 1, 1, 0, 0],  
       [0, 0, 1, 1, 0, 1, 0],  
       [0, 0, 0, 0, 1, 1, 1],  
       [0, 1, 1, 1, 0, 1, 1]])
```

Če ju odštejemo, dobimo tole:

```
zemljevid[:, 1:] - zemljevid[:, :-1]  
array([[ 0,  0,  0,  0,  0,  0,  0],  
       [ 0,  0,  1,  0, -1,  0,  0],  
       [ 0,  1,  0, -1,  1, -1,  0],  
       [ 0,  0,  0,  1,  0,  0, -1],  
       [ 1,  0,  0, -1,  1,  0, -1]])
```



Enice so natančno tam, kjer se začenjajo ovire. Koordinate začetkov dobimo z

```
np.nonzero(zemljevid[:, 1:] - zemljevid[:, :-1] == 1)
(array([1, 2, 2, 3, 4, 4]), array([2, 1, 4, 3, 0, 4]))
```

Koordinate koncev dobimo na enak način, le da odštevamo v drugo smer. Ali, preprosteje, gledamo, kje so -1.

```
np.nonzero(zemljevid[:, 1:] - zemljevid[:, :-1] == -1)
(array([1, 2, 2, 3, 4, 4]), array([4, 3, 5, 6, 3, 6]))
```

Rešitev celotne obvezne naloge - če dopuščamo, da je rezultat tabela trojk namesto seznama terk - je potem:

```
y, x0 = np.nonzero(zemljevid[:, 1:] - zemljevid[:, :-1] == 1)
_, x1 = np.nonzero(zemljevid[:, :-1] - zemljevid[:, 1:] == 1)
ovire = np.column_stack((x0 + 1, x1, y + 1))
```

ovire

```
array([[3, 4, 2],
       [2, 3, 3],
       [5, 5, 3],
       [4, 6, 4],
       [1, 3, 5],
       [5, 6, 5]])
```

Tako bi to nalogo rešil profesionalni programer v Pythonu - sploh, če bi bilo ovir veliko in bi mu hitrost numpy-ja kaj pomenila.