

## Sestavljanje

V tej nalogi bomo sestavljali sestavljanke. Takšne.

```

      /;      ;\
      --  \ \____//
      /{ _ \ /  ' \____
      \____ (o) (o }
              :--'
      -----/
, -, ' ^ccccccccc      cccccc      \_      ^__\
;:(  cccccccccc      ccc      \____(o'o)
:: )  cccc      cccccc      , 'cc(  ^===='
:: : ccccc:      cccc      ^ccc:
:: \  ccccc:      cccccccc)  (  'ccc'
;; /\      /\ ,      cccccccccc\  :cccc)
:/ )      { _-----:  :~^,~^;
;; '^; :  )      :  / ^; ;
;;; : : ;      : ; ; :
'^'^ / : :      : : : :
      ) _ \__ ;      " ;"      : _ ;  \ _ \      ^ , ' , '
      : _ \ \      * ^ , ' *      \ \ : \      * 8 ^ ; ' * *
      ^ ^      \ : /      ^ ^ ^ ^      \ v / :  \ /

```

(Inspiracija prihaja iz Advent of Code, naloga Jurassic Jigsaw, vendar je narejena na novo.)

Kako so videti posamezni koščki, si lahko pogledate v priloženih datotekah. V programu jih bomo zapisovali s terkami nizov. Košček

```

Qm55
7 _H
2o\R
H3cc

```

torej zapišemo kot

```
kos = ("Qm55", "7 _H", "2o\R", "H3cc")
```

Iz teh koščkov se da sestaviti sestavljanke, s tem da je potrebno koščke vrteti in celo zrcaliti. Ko jo sestavimo, bo potrebno še odbiti robove, ki niso del slike - od gornjega koščka bo ostal samo srednji kvadrat 2x2:

```

^
o\

```

A do tam pridemo šele pri oceni 10; šli pa bomo lepo počasi.

1. **Pretvarjanje v terko:** če imamo seznam `s` in potrebujemo terko `s`, uporabimo `tuple(s)`. Toliko, da me ne boste spraševali vsak posebej. :)

2. **Vzratne poševnice:** V spodnjih izpisih so vzratne poševnice zaradi preglednosti napisane z enim samim znakom. V testih sta po dve, ker, vemo, napišeš dve in dobiš eno.
3. **Oblika sestavljanek:** vsi koščki so kvadratni. Same sestavljanke pa so pravokotne, ne pa nujno kvadratne. Gornja krava je bolj dolga kot visoka.
4. Naloga ni tako težka, zato ker jo bomo reševali po korakih in vse lepo sproti testirali. Držite se navodil in ko programirate določeno funkcijo, razmišljajte o tem, zakaj sem vam dal sprogramirati tiste prej. Ne uporabljajte pa jih vedno in na silo.

## Testi

Testi in datoteke: sestavljanek.zip.

## Za oceno 6

- Napiši funkcijo `zrcaljen_navp(kos)`, ki vrne navpično prezrcaljen košček. Za košček iz uvoda v nalogo vrne

```
("H3cc",
 "2o\R",
 "7 _H",
 "Qm55")
```

- Napiši funkcijo `zrcaljen_vod(kos)`, ki prejme košček in vrne vodoravno prezrcaljen košček. Za gornji košček vrne

```
("55mQ",
 "H_ 7",
 "R\o2",
 "cc3H")
```

- Napiši funkcijo `obrnjen(kos)`, ki vrne košček, obrnjen za 90 stopinj v smeri urinega kazalca. Za košček iz uvoda v nalogo vrne

```
('H27Q',
 '3o m',
 'c\_5',
 'cRH5')
```

V gornji vrstici se pojavi, kar je bil prej levi rob. Gornja vrstica je postala desni rob...

- Napiši funkcijo `obrnjen_n(kos, n)`, ki vrne košček, ki je n-krat obrnjen za 90 stopinj v smeri urinega kazalca. Košček in število obratov sta lahko tudi precej velika.
- Napiši funkcijo `stranice(kos)`, ki vrne seznam s stranicami kosa v naslednjem vrstnem redu: gornja, desna, spodnja (od leve proti desni!), leva

(od zgoraj navzdol) in nato še zrcaljena gornja, desna, spodnja in leva. Pozorno pogledaj tale primer: za košček iz primera,

```
Qm55
7 _H
2o\R
H3cc
```

mora vrniti ['Qm55', '5HRc', 'H3cc', 'Q72H', '55mQ', 'cRH5', 'cc3H', 'H27Q'].

- Napiši funkcijo `obradi(kos)`, ki vrne množico vseh koščkov, ki jih je mogoče dobiti z obračanjem in zrcaljenjem podanega koščka. Za gornji košček vrne

```
{('55mQ', 'H_ 7', 'R\o2', 'cc3H'),
 ('5HRc', '5_\c', 'm o3', 'Q72H'),
 ('H27Q', '3o m', 'c\_5', 'cRH5'),
 ('H3cc', '2o\R', '7 _H', 'Qm55'),
 ('Q72H', 'm o3', '5_\c', '5HRc'),
 ('Qm55', '7 _H', '2o\R', 'H3cc'),
 ('cRH5', 'c\_5', '3o m', 'H27Q'),
 ('cc3H', 'R\o2', 'H_ 7', '55mQ')}
```

## Rešitev

Prva naloga je bila naloga iz dela s seznamami. Reševali pa jo bomo z izpeljanimi seznamami; ker smo že dovolj stari.

Za lažje opazovanje dogajanja si napišimo funkcijo, ki izpiše dva kosa: `izpisi(kos1, kos2)`.

```
def izpisi(kos1, kos2):
    for v1, v2 in zip(kos1, kos2):
        print(v1, " ", v2)
```

`zrcali_navp` Tu je potrebno le obrniti terko vrstic.

```
def zrcaljen_navp(kos):
    return kos[::-1]
```

```
izpisi(kos, zrcaljen_navp(kos))
```

```
Qm55      H3cc
7 _H      2o\R
2o\R      7 _H
H3cc      Qm55
```

`zrcaljen_vod` Tu pa je potrebno sestaviti seznam obrnjenih vrstic. Pravzaprav ne seznama, temveč terko.

```
def zrcaljen_vod(kos):
    return tuple(x[::-1] for x in kos)
```

```
izpisi(kos, zrcaljen_vod(kos))
```

```
Qm55      55mQ
7 _H      H_ 7
2o\R      R\o2
H3cc      cc3H
```

`x[::-1]` `for x in kos` je generator in ta generator podamo kot argument funkciji (tipu, konstruktorju, kakor mu že hočete reči) `tuple`, da iz generiranih elementov sestavi terko.

**obrnjen** Ta je pa malo bolj sitna. Ne posebej imenitna rešitev je takšna.

```
def obrnjen(kos):
    obrnjeni = ()
    for stolpec in range(len(kos)):
        nova_vrstica = ""
        for vrstica in kos[::-1]:
            nova_vrstica += vrstica[stolpec]
        obrnjeni = obrnjeni + (nova_vrstica, )
    return obrnjeni
```

V seznam `obrnjeni` nabiramo nove vrstice. V začetku bo to prazna terka, nato bomo vanjo z `obrnjeni = obrnjeni + (nova_vrstica, )` dodajali nove vrstice. Reč je kar sitna: terke so nespremenljive, torej nimajo metode `append`. Z `obrnjeni = ...` v bistvu priredimo imenu `obrnjeni` novo terko, ki je vsota tega, kar je bilo v tej terki doslej in terke, ki vsebujejo novo vrstico (`nova_vrstica,` ). Če bi pisali le `obrnjeni + nova_vrstica`, to ne bi delovalo, ker Python ne more sešteti terke `obrnjeni` in niza `nova_vrstica`. Če bi pisali `obrnjeni + (nova_vrstica)` (brez vejice) pa ne bi šlo, ker je (`nova_vrstica`) isto kot `nove_vrstica`, le da traja nanosekundo dlje, ker se Python vmes začudi, zakaj smo dali to ime v oklepaj.

Drug način je, da je `obrnjeni` seznam, `obrnjeni = []`. Potem vrstice dodajamo kar s `obrnjeni.append(nova_vrstica)`, zadnjo vrstico pa spremenimo v `return tuple(obrnjeni)`.

```
def obrnjen(kos):
    obrnjeni = []
    for stolpec in range(len(kos)):
        nova_vrstica = ""
        for vrstica in kos[::-1]:
            nova_vrstica += vrstica[stolpec]
        obrnjeni.append(nova_vrstica)
    return tuple(obrnjeni)
```

Da, tako je morda še lažje.

Novo vrstico pa sestavljamo tako, da pripravimo prazen niz. Nato gremo čez vse bodoče stolpce (za kar bi morali pravzaprav pisati `range(len(kos[0]))`), vendar so koščki kvadratni, zato je `len(kos)` vedno enako `len(kos[0])`. Pri vsakem stolpcu gremo čez vse vrstice obstoječega kosa, vendar od spodaj navzgor. Pobiramo črke na ustreznem stolpcu in jih zlagamo v niz.

```
izpisi(kos, obrnjen(kos))
```

```
Qm55      H27Q
7 _H      3o m
2o\R      c\_5
H3cc      cRH5
```

Neneobičajna komplikacija, ki se je domislilo študenti, je takšna.

```
def obrnjen(kos):
    obrnjeni = []
    nova_vrstica = "" # Zakaj že tu?!

    for stolpec in range(len(kos)):
        nova_vrstica = ""
        for vrstica in kos[::-1]:
            nova_vrstica += vrstica[stolpec]
        obrnjeni.append(nova_vrstica)
        nova_vrstica = "" # To moramo dodati za kazen...
    return tuple(obrnjeni)
```

Tu na začetku funkcije pripravimo prazno novo vrstico in jo pobrišemo po tem, ko jo uporabimo. Čeprav to deluje, je logika nekako ... izkrivljena. C-jevska. Pripravimo prostor, ga polnimo in praznimo. V gornji, preprostejši rešitvi, pa znotraj zanke `for stolpec ...` pripravimo vrstico in jo napolnimo.

Zdaj pa skrajšajmo to funkcijo. Črke po stolpcih lahko pobremo z `vrstica[stolpec]` `for vrstica in kos[::-1]`. To damo metodi `"".join`, da jih združi, torej

```
def obrnjen(kos):
    obrnjeni = []
    for stolpec in range(len(kos)):
        nova_vrstica = "".join(vrstica[stolpec] for vrstica in kos[::-1])
        obrnjeni.append(nova_vrstica)
    return tuple(obrnjeni)
```

```
izpisi(kos, obrnjen(kos))
```

```
Qm55      H27Q
7 _H      3o m
2o\R      c\_5
H3cc      cRH5
```

Odtod pa, očitno

```
def obrnjen(kos):  
    return tuple(  
        "".join(vrstica[stolpec] for vrstica in kos[::-1])  
        for stolpec in range(len(kos))  
    )
```

```
izpisi(kos, obrnjen(kos))
```

```
Qm55      H27Q  
7 _H      3o m  
2o\R      c\_5  
H3cc      cRH5
```

Radovedni pa bodo raziskali, kako deluje tole:

```
def obrnjen(kos):  
    return zrcaljen_vod(map("".join, zip(*kos)))
```

```
izpisi(kos, obrnjen(kos))
```

```
Qm55      H27Q  
7 _H      3o m  
2o\R      c\_5  
H3cc      cRH5
```

**obrnjen\_n** Eden od testov je poskušal 1000000000000001-krat obrniti kos velikosti 100x100, kar namiguje na to, da si bo potrebno izmisliti bližnjico. V vprašanih študentov sem videl kup zanimivih ugibanj, resnica pa je preprosta: če list papirja štirikrat obrnemo za 90 stopinj, bo obrnjen tako, kot je bil v začetku. In če ga obrnemo  $n$ -krat, lahko preskočimo  $4 * \text{bogveliko vrtenj}$ . Zanima nas le ostanek po deljenju s štiri. Ne liha in soda in podobno. Če ga obrnemo 1000000000000001-krat je to isto, kot če smo ga 1-krat. Če ga 1000000000000003-krat, je isto, kot če ga 3-krat. In če ga  $n$ -krat, je isto, kot če ga  $n \% 4$ -krat.

To sem potem videl sprogramirano na veliko zanimivih načinov -- pretežno z nekimi zankami `while` v temu duhu:

```
while n != 0:  
    kos = obrnjen(kos)  
    n = n % 4 - 1
```

Ena in edina prava rešitev je

```
def obrnjen_n(kos, n):  
    for i in range(n % 4):  
        kos = obrnjen(kos)  
    return kos
```

Vseh druge rešitve te naloge je že davno, res davno, nekdo povsem pravilno pokomentiral z *"kar je več, je od hudega"*.

**stranice** Še ena vaja iz seznamov. In izpeljanih seznamov.

- gornja: kos[0];
- desna: vrstica[-1] for vrstica in kos; združiti jim moramo v nov niz, torej `"".join(vrstica[-1] for vrstica in kos)`, seveda pa obstajajo tudi daljši načini;
- spodnja: kos[-1];
- leva: vrstica[0] for vrstica in kos. Tako dobimo prve štiri. Damo jih v seznam; ostale štiri dobimo kot obrnjene elemente tega seznama.

```
def stranice(kos):
    t = [kos[0],
         "".join(vrstica[-1] for vrstica in kos),
         kos[-1],
         "".join(vrstica[0] for vrstica in kos),
         ]
    return t + [x[::-1] for x in t]

izpisi(kos, [""] * 4)

Qm55
7 _H
2o\R
H3cc

stranice(kos)

['Qm55', '5HRc', 'H3cc', 'Q72H', '55mQ', 'cRH5', 'cc3H', 'H27Q']
```

**obradi** Še zadnja funkcija za oceno 6. Košček je potrebno štirikrat obrniti, zrcaliti in še štirikrat obrniti, ter vse pridno zlagati v množico.

```
def obrati(kos):
    vsi_obradi = set()
    for i in range(2):
        for j in range(4):
            vsi_obradi.add(kos)
            kos = obrnjen(kos)
            kos = zrcaljen_navp(kos)
    return vsi_obradi
```

V množici, ki jo dobimo, so res vsi obrati, v nekem naključnem vrstnem redu. Ker gre pač za množico.

(S tem, kako deluje spodnji izpis, naj se zabavajo tisti, ki jih to zanima.)

```

for vrstice in zip(*obrati(kos)):
    print("      ".join(vrstice))

Qm55    H3cc    H27Q    cc3H    cRH5    5HRc    55mQ    Q72H
7 _H    2o\R    3o m    R\o2    c\_5    5\_c    H_ 7    m o3
2o\R    7 _H    c\_5    H_ 7    3o m    m o3    R\o2    5\_c
H3cc    Qm55    cRH5    55mQ    H27Q    Q72H    cc3H    5HRc

```

### Vse funkcije za oceno 6

Da dolžina gornjih opisov ne bi pustila vtisa, da je bili potrebno za oceno 6 sprogramirati tisoč vrstic kode:

*# Funkcije za oceno 6*

```

def zrcaljen_vod(kos):
    return tuple(x[::-1] for x in kos)

def zrcaljen_navp(kos):
    return kos[::-1]

def obrnjen(kos):
    return tuple("".join(vrstica[i] for vrstica in kos[::-1])
                  for i in range(len(kos)))

def obrnjen_n(kos, n):
    for i in range(n):
        kos = obrnjen(kos)
    return kos

def stranice(kos):
    t = [kos[0],
          "".join(vrstica[-1] for vrstica in kos),
          kos[-1],
          "".join(vrstica[0] for vrstica in kos),
          ]
    return t + [x[::-1] for x in t]

def obrati(kos):
    vsi_obrati = set()
    for i in range(2):
        for j in range(4):

```



```

        vsi_obrazi.add(kos)
        kos = obrnjen(kos)
        kos = zrcaljen_navp(kos)
    return vsi_obrazi

```

## Za oceno 7

- Napiši funkcijo `preberi_datoteko(ime_datoteke)`, ki prebere koščke iz podane datoteke. Vrniti mora seznam koščkov, torej seznam terk nizov, na primer

```

[('bmq8', 'b__8', '0__s', 'fyuX'),
 ('Qm55', '7_H', '2o\R', 'H3cc'),
 ('cj97', 'c 7', 'x \m', 'KZ9K'),
 ... in tako naprej

```

- Napiši funkcijo `zbirka_stranic(kosi)`, ki prejme seznam koščkov (tak, kot ga vrača gornja funkcija). Vrniti mora slovar, katerega ključi so vse možne stranice (vključno s prezrcaljenimi), pripadajoče vrednosti pa so množice koščkov, ki vsebujejo takšno stranico.

```

{'bmq8': {('bmq8', 'b__8', '0__s', 'fyuX'),
          ('8J99', 'q_ k', 'm_ 2', 'bUee')},
 '88sX': {('XH00', 's)_P', '8 H', '8v2H'),
          ('bmq8', 'b__8', '0__s', 'fyuX')},
 'fyuX': {('bmq8', 'b__8', '0__s', 'fyuX')},
 'bb0f': {('o855', 'A__Y', 'X__j', 'f0bb'),
          ('bmq8', 'b__8', '0__s', 'fyuX')},
 ... in tako naprej

```

Stranica "bmq8" je zgornja stranica koščka v prvi vrstici in zrcaljena leva stranica koščka v drugi vrstici (od leve proti desni preberemo prve črke nizov v drugi terki).

V množicah naj se koščki pojavljajo v svojih originalnih legah - takšnih, ko so v podanem seznamu - in ne obrnjeni ali zrcaljeni.

Vsak košček se pojavi v osmih različnih množicah - toliko, kolikor različnih stranih ima (skupaj z zrcaljenimi).

(Tale opis je dolg, sama funkcija pa ima pet vrstic, brez kakšnih fint z izpeljanim čemerkoli.)

Opazil(a) boš nekaj prijetnega: vsaka množica vsebuje le en ali dva koščka. Kaj to pomeni, razmisli sam(a). Še preden greš naprej.

## Rešitev

`preberi_datoteko` Povsem običajna rešitev je takšna.

```
def preberi_datoteko(ime_dat):
    kosi = []

    trenutni = []
    for vrstica in open(ime_dat):
        vrstica = vrstica.strip()
        if not vrstica:
            kosi.append(tuple(trenutni))
            trenutni = []
        else:
            trenutni.append(vrstica)

    kosi.append(tuple(trenutni))
    return kosi
```

`kosi` bo seznam kosov, ki ga bomo vrnili, v `trenutni` pa bomo med branjem nabirali vrstice trenutnega kosa.

Gremo po vrsticah datoteke. Od-strip-amo `\n` na koncu vrstice. Če je vrstica s tem prazna, je trenutni kos končan in ga (pretvorjenega v terko) dodamo v seznam kosov ter pripravimo nov `trenutni`. Če vrstica še ni prazna, pa dodamo vrstico v `trenutni` kos.

Ko je zanka končana, s `kosi.append(tuple(trenutni))` dodamo še zadnji kos.

Datoteka je oblikovana tako, da pred prvim in za zadnjim kosom ni praznih vrstic. Ako bi ne bilo tako, bi morali opaziti še na primere, kot je `trenutni` prazen.

Zanimivo je, da smo tu naredili natančno tisto, o čemer sem pri `obrnjen` tožil, naj ne počnemo. Tam je bilo potrebno polniti in prazniti vrstico, če smo jo pripravili pred zanko. Tu se temu ne da izogniti, saj imamo le eno zanko.

```
preberi_datoteko("muc-puzzle.txt")
```

```
[('bm q8', 'b__8', '0__s', 'fyuX'),
 ('Qm55', '7 _H', '2o\\R', 'H3cc'),
 ('cj97', 'c 7', 'x \\m', 'KZ9K'),
 ('KZ9K', 'K /K', 'a )0', '0Y04'),
 ('QRbK', 'Q= K', 'h_m0', '41K4'),
 ('8J99', 'q_ k', 'm_ 2', 'bUee'),
 ('ww2F', 'J _n', 'M _y', 'YYTq'),
 ('XH00', 's)_P', '8 _H', '8v2H'),
 ('o855', 'A__Y', 'X__j', 'f0bb'),
 ('qqzn', '6/_c', '2~_1', '558o'),
 ('Qm55', 'Ro/k', 'b \\C', 'Km77'),
 ('5v00', 'Y_ V', 'j_ f', 'bUee'),
 ('YYTq', 'T _6', 'P _2', '00v5'),
 ('9J6c', '9 /c', 'J/ _3', '8v2H'),
```

```
('FF84', 'n M', 'y (J', 'qqzn'),
('HHPO', '2=_E', '7ømw', 'QQh4'])
```

Funkcijo se da napisati tudi precej preprosteje. Tu so bili v prednosti tisti, ki so morda brali moje rešitve Advent of Code. Tam smo letos pogosto srečali podatke, ločene s prazno vrstico. Prvič v nalogi Password Processing. Trik je `open(ime_dat).read().split("\n\n")`: preberemo celo vsebino datoteke in jo razdelimo glede na `\n\n`, dva znaka za novo vrstico. Poglejmo, kaj dobimo.

```
bloki = open("muc-puzzle.txt").read().split("\n\n")
```

Vzemimo prvi blok in si ga oglejmo.

```
prvi = bloki[0]
```

```
prvi
```

```
'bmq8\nb__8\n0__s\nfyuX'
```

To reč je potrebno le še razdeliti glede na `\n` ali kar s `splitlines()` in spremeniti v terko.

```
tuple(prvi.splitlines())
```

```
('bmq8', 'b__8', '0__s', 'fyuX')
```

Rešitev naloge je potem

```
def preberi_datoteko(ime_dat):
    return [tuple(kos.splitlines())
            for kos in open(ime_dat).read().split("\n\n")]
```

```
kosi = preberi_datoteko("muc-puzzle.txt")
```

```
kosi
```

```
[('bmq8', 'b__8', '0__s', 'fyuX'),
 ('Qm55', '7_H', '2o\\R', 'H3cc'),
 ('cj97', 'c_7', 'x \\m', 'KZ9K'),
 ('KZ9K', 'K /K', 'a )0', '0Y04'),
 ('QRbK', 'Q= K', 'h_m0', '41K4'),
 ('8J99', 'q_ k', 'm_ 2', 'bUee'),
 ('ww2F', 'J_n', 'M_y', 'YYTq'),
 ('XH00', 's)_P', '8_H', '8v2H'),
 ('o855', 'A__Y', 'X__j', 'f0bb'),
 ('qqzn', '6/_c', '2~_1', '558o'),
 ('Qm55', 'Ro/k', 'b \\C', 'Km77'),
 ('5v00', 'Y_V', 'j_f', 'bUee'),
 ('YYTq', 'T_6', 'P_2', '00v5'),
 ('9J6c', '9 /c', 'J/ 3', '8v2H'),
 ('FF84', 'n M', 'y (J', 'qqzn'),
 ('HHPO', '2=_E', '7ømw', 'QQh4')]
```

**zbirka\_stranic** To je preprosto: za vsak kos pokličemo **stranice**, da izvemo vse njegove stranice. To bodo ključi slovarja. Vrednosti bodo množice; v te množice (torej: tiste, množice, ki kot vrednosti pripadajo tem stranicam) dodamo ta kos. Seveda bomo uporabili **defaultdict**.

```
from collections import defaultdict
```

```
def zbirka_stranic(kosi):
    vse_stranice = defaultdict(set)
    for kos in kosi:
        for stranica in stranice(kos):
            vse_stranice[stranica].add(kos)
    return vse_stranice
```

```
po_stranicah = zbirka_stranic(kosi)
```

```
po_stranicah["bmq8"]
```

```
{('8J99', 'q_k', 'm_2', 'bUee'), ('bmq8', 'b__8', '0__s', 'fyuX')}
```

Nekateri študenti so to funkcijo napisali tako, da je bila bistveno počasnejša. Kako, ne vem. Zna biti, da so najprej nabrali množico vseh možnih stranic, nato pa za vsako stranico gledali, kateri kosi jo vsebujejo. To bi potencialno lahko vodilo v rešitev v eni vrstici. Oziroma v rešitev, ki bi lepo demonstrirala, zakaj so rešitve v eni vrstici lahko tudi slaba ideja. Takšni programi so bili namreč zelo počasni.

## Vse funkcije za oceno 7

Spet pokažimo, koliko je bilo v resnici tega programiranja. Toliko. :)

```
def preberi_datoteko(ime_dat):
    return [tuple(kos.splitlines())
            for kos in open(ime_dat).read().split("\n\n")]

def zbirka_stranic(kosi):
    vse_stranice = defaultdict(set)
    for kos in kosi:
        for stranica in stranice(kos):
            vse_stranice[stranica].add(kos)
    return vse_stranice
```

## Za oceno 8

Po razmisleku o gornjem napiši naslednje funkcije.

- Funkcija **kotni(kosi)** prejme vse kose sestavljanke (kot jih vrne **preberi\_datoteko**) in vrne množico kosov, ki so v vogalih. Kosi naj ne bodo obrnjeni ali zavrteni, temveč takšni, kot so v seznamu kosi.

Če nimaš pojma, kako to napisati, ponovno preveri zadnji odstavek zadnje funkcije naloge za oceno 7.

- Funkcija `robni(kosi)` vrne množico kosov, ki so na robovih sestavljanke (vendar ne v kotih). Če nimaš pojma, kako to napisati, potem ne vem, kako ti je uspelo napisati prejšnjo funkcijo. :)
- Funkcija `prvi_kot(kosi)` vrne tisti, kot, pri katerem je prva vrstica prva po abecedi. Uporabljaljaj Pythonovo urejenost (števke pred črkami, velike črke pred malimi; z drugimi besedami, samo uporabljaljaj Pythonove funkcije in vse bo dobro).

(Funkcij, ki jih pišeš tu, morda ne potrebuješ, so pa uporabne, če želiš kaj izvedeti o sestavljanke. In za dodatno nalogo onstran one za oceno 10.)

## Rešitev

Na koncu naloge za oceno 7 sem namignil, da je zanimivo opazovati množice v slovarju, ki ga vrne `zbirka_stranic`. Nekatere imajo en sam kos, nekatere dva. Stranice z enim samim kosom so na robu sestavljanke, tiste z dvema, so notranje.

Robni in vogalni kosi so tisti kosi, ki imajo zunanje stranice, torej, ki nastopajo v množicah z enim samim elementom.

Vogalni kosi imajo dve takšni, unikatni stranici. Vendar vsako stranico gledamo dvakrat - enkrat tako, kot se pojavi in drugič zrcaljeno. Torej se vogalni kosi štirikrat pojavijo v množici z enim samim elementom.

Robni kosi imajo eno unikatno stranico, torej se dvakrat pojavijo v množici z enim samim elementom.

Funkciji `robni` in `vogalni` si bosta očitno zelo podobni, torej napišimo funkcijo `_po_enojnih(kosi, n)`, ki vrne množico vseh kosov, ki se `n`-krat pojavijo v množici z enim elementom. Za `n = 2` bomo dobili stranice, za `n = 4` pa vogale.

```
def _po_enojnih(kosi, n):
    enojnih = defaultdict(int)
    for m_kosov in zbirka_stranic(kosi).values():
        if len(m_kosov) == 1:
            enojnih[m_kosov.pop()] += 1
    return {kos for kos, stevec in enojnih.items() if stevec == n}
```

Naredimo torej `zbirka_stranic` in gremo čez množice, `values()`. V `enojnih` štejemo, kolikokrat posamični kos nastopa v množici z enim samim elementom: če ima množica (`m_kosov`) le en element, ga s `pop` pobremo iz množice in povečamo ustrezen števec. Na koncu v množico nabereimo tiste ključe `enojnih`, pri katerih je števec enak 1.

Ljubitelji jedrnatega izražanja vedo, da lahko za štetje uporabimo `Counter` in napišejo

```
from collections import Counter
```

```
def _po_enojnih(kosi, n):
    enojnih = Counter(m_kosov.pop()
                      for m_kosov in zbirka_stranic(kosi).values()
                      if len(m_kosov) == 1)
    return {kos for kos, enojnih in enojnih.items() if enojnih == n}
```

Ali, na tiste dni, ko se jim res tlači brez potrebe,

```
def _po_enojnih(kosi, n):
    return {kos
            for kos, stevec in Counter(m_kosov.pop()
                                       for m_kosov in zbirka_stranic(kosi).values()
                                       if len(m_kosov) == 1).items()

            if stevec == n}
```

### Rešitve za oceno 8

Če `_po_enojnih` sprogramiramo ... razumno, je celotna rešitev naloge za oceno 8 takšna.

```
def _po_enojnih(kosi, n):
    enojnih = Counter(m_kosov.pop()
                      for m_kosov in zbirka_stranic(kosi).values()
                      if len(m_kosov) == 1)
    return {kos for kos, stevec in enojnih.items() if stevec == n}
```

```
def kotni(kosi):
    return _po_enojnih(kosi, 4)
```

```
def robni(kosi):
    return _po_enojnih(kosi, 2)
```

```
def prvi_kot(kosi):
    return min(kotni(kosi))
```

### Za oceno 9

- Funkcija `preobrne(kos, levo, gor, po_stranicah)` prejme kos sestavljanke, podatek o tem, kaj mora biti levo in zgoraj, ter slovar `po_stranicah`, ki je takšen, kot ga vrača funkcija `zbirka_stranic`. Vrniti mora kos, ki ga dobi z vrtenjem in/ali zrcaljenjem podanega kosa tako, da le-ta ustreza pogojema levo in gor.

Argumenta `levo` in `gor` povesata, kakšna morata biti leva in zgornja stranica vrnjene kosa. Če je argument enak 1, to pomeni, da mora biti leva oz

gor-nja stranica robna stranica. Če je argument niz, pa pove, kakšna mora biti stranica.

Klic

```
preobrni(("Qm55",
          "7 _H",
          "2o\R",
          "H3cc"), "cc3H", 1, po_stranicah)
```

vrne

```
'c\_5',
'3o m',
'H27Q')
```

To je namreč tista oblika tega koščka, pri katerem je leva stranica enaka "cc3H" (argument `leva`), zgornja, "cRH5" pa se v `po_stranicah` pojavi le enkrat (argument `gor`).

Funkcija sme predpostaviti, da je košček možno obrniti v zahtevano lego na natančno en način. Če razmisliš, to pomeni tudi, da je vsaj eden od argumentov vedno niz.

- Funkcija `vzemi_kos(leva, gor, po_stranicah, uporabljeni)` prejme pogoja `leva` in `gor`, ki imata enak pomen kot pri prejšnji funkciji (in je vsaj eden od njiju niz); slovar `po_stranicah`, ki je prav tako enak onemu iz prejšnje funkcije; ter množico `uporabljeni`, ki vsebuje koščke, ki so že uporabljeni.

**Tako kot slovar `po_stranicah`, naj/bo tudi `uporabljeni` vsebuje koščke v njihovih originalnih legah.**

Funkcija `vzemi_kos` mora:

- vrniti košček, ki ustreza pogojema in še ni bil uporabljen; **košček mora biti obrnjen/zrcaljen tako, da sta `leva` in `gor` stranici, ki ju predpisujeta argumenta.**
- poleg tega mora ta košček dodati v množico `uporabljeni`. (Footnote: to je grdo, ker funkcija vrača rezultat in hkrati nekaj spreminja. Vendar: če v Pythonu napišete `import this`, se med drugim izpiše tudi "*practicality beats purity*")

Ker je vaš profesor kljub temu, da vam daje takšne domače naloge, neke globoko v srcu vseeno za silo prijazen človek, so stvari so organizirane tako, da bo v vsakem trenutku obstajal natančno en košček, ki bo ustrezal pogojem. (Tako je bilo tudi v nalogi v Advent of Code.)

## Rešitev

Kos je potrebno preprosto vrteti, dokler ni tak, kot mora biti.

Za vsak obrat (`for kos in obrati(kos)`) pokličemo `stranice(kos)` in si zabeležimo kakšna je gornja stranica tega kosa (`ta_gor`) in leva stranica (`ta_levo`). Dobili bi jih lahko z `ta_gor = stranice(kos)[0]`, `ta_levo = stranice(kos)[3]`, lahko pa preprosto razpakiramo terko; odvečno desno in spodnjo stranico razpakiramo v `_1` in `_2`, ostale štiri pa v `_` (pred katerega damo zvezdico, da bo pogoltnil vse odvečne stranice).

Da je kos pravilno obrnjen, mora biti zahtevani `gor` enak `ta_gor`; če je `gor` slučajno število, pa mora biti enak številu kosov, pri katerih se pojavi takšna stranica `ta_gor`, torej `len(po_stranicah[ta_gor])`. Torej mora biti `gor` enak bodisi `ta_gor` bodisi `len(po_stranicah[ta_gor])`. Z levo pa je enako.

```
def preobrni(kos, levo, gor, po_stranicah):
    for kos in obrati(kos):
        ta_gor, _1, _2, ta_levo, *_ = stranice(kos)
        if gor in (ta_gor, len(po_stranicah[ta_gor])) \
            and levo in (ta_levo, len(po_stranicah[ta_levo])):
            return kos
```

Izkaže pa se, da je obrat kosa določen že s stranicami, ki so podane. Tega, ali gre za robno, niti ni potrebno preverjati. Še več, če sta podani dve stranici, je dovolj, da se ujema ena in ujemala se bo tudi druga. Tako so pač sestavljeni podatki; če bi bili bolj zapleteni, bi bilo programiranje težje. Spodnja funkcija torej ne sledi povsem navodilom, vendar preživi teste in bi jih preživela tudi za vsake druge podatke, ki ne bi neprimerno zapletli naloge.

```
def preobrni(kos, levo, gor, po_stranicah):
    for kos in obrati(kos):
        ta_gor, _1, _2, ta_levo, *_ = stranice(kos)
        if gor == ta_gor or levo == ta_levo:
            return kos
```

Zanimivo je, da argumenta `po_stranicah` torej niti ne potrebujemo. (Ob sestavljanju naloge se teh stvari nisem zavedal, čeprav sem preživel skoraj dva dni ob vrtenju teh koščkov. :)

**vzemi\_kos** Rešitev naloge je načelno takšna.

```
def vzemi_kos(levo, gor, po_stranicah, uporabljeni):
    stranica = levo if gor == 1 else gor
    kos = (po_stranicah[stranica] - uporabljeni).pop()
    uporabljeni.add(kos)
    return preobrni(kos, levo, gor, po_stranicah)
```

Najprej si izberemo stranico, po kateri bomo iskali košček: to bo `levo` stranica, če je `gor` enak 1, sicer pa zgornja. Lahko pa bi naredili tudi obratno.

Iz slovarja `po_stranicah` dobimo množico kosov s to stranico. Odstranimo vse, ki so bili že uporabljeni. Naloga zagotavlja, da bo vedno obstajal natančno



en košček s takšno stranico, torej vemo, da ima množica natančno en element. Vzamemo ga s `pop`. Dodamo ga med uporabljene in vrnemo ustrezen obrat tega kosa.

Zdaj pa poškilimo malo naprej. V nalogi za oceno 10 bo potrebno sestaviti sestavljanko, pri čemer bomo kot argument dobili vse kose in gornji levi kos. Najprej bomo sestavili prvo vrstico (potem pa vsako naslednjo), tako da bomo začeli s prvim kosom vrstice in klicali `vzemi_kos` z desno stranico zadnjega kosa, da bi dobil kos z ustrezno levo stranico. Očitno bo veselja konec, ko pridemo do konca vrstice. Testi za oceno 9 sicer te funkcije nikoli ne pokličejo, ko ustreznega kosa ni (to nam zagotavlja besedilo naloge). Ob koncu vrstice pa bomo sami poklicali to funkcijo tako, da ne bo mogla vrniti ustreznega kosa. Torej je smiselno, da predvidimo še ta scenarij. V primeru, da je množica prazna, vrnimo `None`.

```
def vzemi_kos(levo, gor, po_stranicah, uporabljeni):
    stranica = levo if gor == 1 else gor
    kandidati = po_stranicah[stranica] - uporabljeni
    if len(kandidati) != 1:
        return None
    kos = kandidati.pop()
    uporabljeni.add(kos)
    return preobrni(kos, levo, gor, po_stranicah)
```

## Za oceno 10

- Napiši funkcijo `sestavi_koscke(kosi, kos)`, ki dobi kose, kot jih vrne `preberi_datoteko` in za pomoč še kos, ki ga je potrebno postaviti v gornji levi kot. Ta košček je tudi pravilno obrnjen (glede na zeleni končni izgled sestavljanke.)

Ta funkcija mora v bistvu sestaviti sestavljanko.

Vrniti mora seznam seznamov koščkov (vsak košček pa je terka nizov, torej funkcija vrača seznam seznamov terk nizov). Vsak seznam znotraj tega seznama predstavlja vrstico sestavljanke. Slika muca je sestavljena iz dveh vrstic in osmih stolpcev, torej mora funkcija vrniti seznam, ki vsebuje dva seznama s po osmimi seznamami terk, takole:

```
[
    [('wJMY', 'w Y', '2 T', 'Fnyq'), ('YTP0', 'Y 0', 'T v', 'q625'),
     ('0Vfe', '0 e', 'v_U', '5Yjb'), ('e2k9', 'e 9', 'U__J', 'bmq8'),
     ('9J6c', '9 /c', 'J/ 3', '8v2H'), ('cRH5', 'c\ 5', '3o m', 'H27Q'),
     ('5kC7', '5/\7', 'mo m', 'QRbK'), ('79jc', '7 c', 'm\ x', 'K9ZK')],
    [('Fnyq', 'F q', '8 (z', '4MJn'), ('q625', 'q/~5', 'z__8', 'nc1o'),
     ('5Yjb', '5__b', '8__0', 'oAXf'), ('bmq8', 'b__8', '0__s', 'fyuX'),
     ('8v2H', '8 H', 's)_P', 'XH00'), ('H27Q', 'H=øQ', 'P_mh', '0Ew4'),
     ('QRbK', 'Q= K', 'h_m0', '41K4'), ('K9ZK', 'K/ K', '0) a', '40Y0')]
```

]

Prvi košček v prvem seznamu je podani kotni košček (v podani legi). Ostali se z njim ujemajo (torej so po potrebi obrnjeni drugače kot v podatkih).

Če pozorno pogledamo, vidimo, da je desna stranica prvega koščka YYTq, in prav taka je leva stranica drugega. Prvi košček prve vrstice se konča z FNyq in z istim robom se začne prvi košček druge vrstice. In tako povsod drugje.

Funkcijo je najlažje napisati tako, da postavimo podani prvi košček in nato dokončamo prvo vrstico, tako da izbiramo ustrezne koščke za nadaljevanje. (Kako dolga je vrstica, vidimo sproti.) Pod prvo vrstico sestavimo drugo vrstico. Pod drugo dodamo tretjo. In tako naprej.

Če kaj pomaga (ni pa nujno potrebno): ko zaključimo prvo vrstico, vemo tudi, koliko vrstic bo.

- Napiši funkcijo `slika(kosi)`, ki prejme kose, kot jih vrne prejšnja funkcija, in vrne niz s sliko: od vsakega koščka mora odstraniti rob, kar ostane, pa lepo zložiti. Med vrsticami niza so `\n`.

`print(slika(kosi))`, kjer so `kosi` gornji seznam, bi izpisal

```
      /\_/\
    ----/  o o \
  /~-----=φ= /
(-----)__m_m)
```

- Napiši funkcijo `sestavljanka(kosi, kot)`, ki prejme enake argumente kot `sestavi_koscke` in vrne enak rezultat kot `slika`.

## Rešitev

Za oceno 10 pa je potrebno začeti programirati.

Najprej si pripravimo zbirko stranic in množico uporabljenih kosov. V začetku je uporabljen vogalni kos, ki ga dobimo kot argument. Tu nas zamika napisati `uporabljeni = {kos}`, vendar to ne bo delovalo, ker ima množica `uporabljeni` kose v originalni legi, vogalni kos, ki ga dobi funkcija kot argument, pa je lahko obrnjen. Torej potrebujemo `uporabljeni = obrati(kos)` - za uporabljene razglasimo kar vse obrate tega kosa. Če hočemo biti "pravilnejši", pa napišemo `uporabljeni = obrati(kos) & set(kosi)`, tako da množica vsebuje le tisti obrat, ki se dejansko pojavi v originalnih podatkih.

V `slika` bomo sestavljali sliko. Vanjo najprej dodamo (prazen) seznam `prva_vrstica`, ki ga potem v prvi zanki dopolnimo s `kosi`, ki jih vrača `vzemi_kos`. (Poglejte, kako smo jo obrnili!)

Nato na podoben način dodamo preostale stranice.

```

def sestavi_koscke(kosi, kos):
    po_stranicah = zbirka_stranic(kosi)

    uporabljeni = obrati(kos) & set(kosi)
    prva_vrstica = []
    slika = [prva_vrstica]

    # Dopolni prvo vrstico
    while kos:
        prva_vrstica.append(kos)
        levo = stranice(prva_vrstica[-1])[1]
        kos = vzemi_kos(levo, 1, po_stranicah, uporabljeni)

    # Dodaj ostale vrstice
    N = len(prva_vrstica)
    for i in range(len(kosi) // N - 1):
        gor = stranice(slika[-1][0])[2]
        vrstica = [vzemi_kos(1, gor, po_stranicah, uporabljeni)]
        for i in range(1, N):
            gor = stranice(slika[-1][i])[2]
            levo = stranice(vrstica[-1])[1]
            vrstica.append(vzemi_kos(levo, gor, po_stranicah, uporabljeni))
        slika.append(vrstica)

    return slika

```

Obstaja tudi precej krajša rešitev.

Za vsako stranico obstajata le dva koščka, ki ji ustrezata (ali eden, če gre za rob). To med drugim pomeni, da nam pri sestavljanju nadaljnjih vrstic ni potrebno opazovati prejšnje.

```

def sestavi_koscke(kosi, kos):
    po_stranicah = zbirka_stranic(kosi)
    uporabljeni = obrati(kos) & set(kosi)
    slika = [[kos]]
    while len(uporabljeni) < len(kosi):
        kos = vzemi_kos(stranice(slika[-1][-1])[1], 1, po_stranicah, uporabljeni)
        if kos:
            slika[-1].append(kos)
        else:
            slika.append([vzemi_kos(1, slika[-1][0][-1], po_stranicah, uporabljeni)])
    return slika

```

Ta funkcija v sliko najprej vstavi kot. Nato izvaja zanko, dokler ne porabi vseh kosov. V vsakem koraku vzame kos, ki sledi (doslej) zadnjemu kosu v (doslej) zadnji vrstici. Ta kos je `slika[-1][-1]`, izmed njegovih stranic nas zanima desna (`stranice(kos[-1][-1])[1]`). Če tak kos obstaja, ga doda v to vrstico.

Če ga ni, pa v sliko doda novo vrstico, ki se začne s koščkom, katerega zgornja stranica se ujema s spodnjo stranico prvega koščka v zadnji vrstici.

**Ta funkcija deluje samo**, če v `preobrni_kos` ne preverjamo, ali je stranica, za katero podamo argument 1, res robna. Uporabiti moramo torej ono različico s pogojem `if gor == ta_gor or levo == ta_levo`.

Preostali funkciji sta v primerjavi s to trivialni.

```
def slika(kosi):
    bitmap = []
    for vrsta_kosov in kosi:
        for vrsta in zip(*(tile[1:-1] for tile in vrsta_kosov)):
            bitmap.append("".join(row[1:-1] for row in vrsta))
    return "\n".join(bitmap)
```

```
def sestavljanje(kosi, kos):
    return slika(sestavi_koscke(kosi, kos))
```

```
zadnji = preberi_datoteko("zadnji-puzzle.txt")
kot = next(kos for kos in zadnji if kos[0] == "TTwADdOW")
print(sestavljanje(zadnji, zrcaljen_vod(obrnjen(kot))))
```

```
Lepe praznike in še          :~::~: .~~~\          :~::~:
veliko programerskih        |:::| / \ _ |:::|
uspehov v 2021! :) _      l~~~~! ~x .~~_)_ l~~~~!
      .~~ ~. \ / ~x".~~ ~. \ /
      / \ || _ ( / \ ||
|| T o o Y || || T o o Y||
==:l l < ! (3 ==:l l < ! (3
\\ \ ._/ / || \\ \ ._/ / ||
\\ ,r"-,___-'r.//|| \\ ,r"-,___-'r.//||
}^ \.( ) _.'//. || }^ \.( ) _.'//. ||
/ }~Xi--- // || / }~Xi--- // ||\
Y Y I\ \ " || Y Y I\ \ " || Y
| | |o\ \ || | | |o\ \ || |
| l_l Y T || | l_l Y T || |
l "o l_j |! l "o l_j || !
\ || \ ||/
.~~^ . o .^||. .~~^ . o ||--.
" ~ ^ "
```

## Za dodatno zabavo

- Spremeni funkcijo `sestavi_koscke` tako, da bo lahko argument `kos`, ki pove gornji levi kot, tudi `None`. To ustreza situaciji, ko preberemo datoteko in pravzaprav ne vemo, kateri košček bo levo zgoraj. Funkcija naj v tem

primeru vzame enega od kotnih kosov (lahko kar `prvi_kot`) in ga ustrezno zasuka.

Dobljena sestavljanke bo sicer morda obrnjena in prezrcaljena. Vseeno pa je pomembno, da prvi kot pravilno zasukaš: če predpostavimo, da gre za gornji levi kot, mora biti košček obrnjen tako, da sta zgornja in leva stranica robni.

Tega v resnici sploh ni težko sprogramirati, samo testov se mi ne piše in naloga je že dovolj dolga. :)

- Napiši program, ki prejme datoteko z ASCII artom in sestavi sestavljanke. Se pravi to, kar sem moral jaz početi s temi kravami in mačkami. :)

## Rešitev

Tole razreže sestavljanke, doda unikatne robove, prevrti in premeša koščke.

```
from random import choice, shuffle, seed
import string
from functools import reduce
from operator import add

seed(0)

#fn = "muc"
#n = 2

#fn = "krava"
#n = 5

fn = "zadnji"
n = 6

vrstice = [x.strip("\n") for x in open(fn + ".txt")]

w, h = max(map(len, vrstice)), len(vrstice)

hh = (h + n - 1) // n
ww = (w + n - 1) // n
print(hh, ww)
pad_hor = " " * (n * ww - h)
pad_ver = [" " * n * ww] * (ww * n - h)

vrstice = [x + pad_hor for x in vrstice] + pad_ver

allowed = string.ascii_letters + string.digits
print(allowed)
```

```

def rl():
    return choice(allowed)

sides = sorted({"".join(rl() for _ in range(n))
                for _ in range(10 * (ww + 1) * (hh + 1))})
shuffle(sides)
sides = iter(sides)

d = [[rl() for _ in range(ww + 1)] for _ in range(hh + 1)]
vers = [[d[row][col] + next(sides) + d[row + 1][col]
          for col in range(ww + 1)]
         for row in range(hh)]
hors = [[d[row][col] + next(sides) + d[row][col + 1]
          for col in range(ww)]
         for row in range(hh + 1)]

def tile(x, y):
    xs = slice(x * n, x * n + n)
    left, right = vers[y][x:x + 2]
    print(y * n + n - 1, len(vrstice))
    return [hors[y][x],
            *[left[yi] + vrstice[y * n + yi][xs] + right[yi] for yi in range(n)],
            hors[y + 1][x]]

def obrati(kos):
    vsi_obrati = []
    for i in range(2):
        for j in range(4):
            vsi_obrati.append(kos)
            kos = ["".join(v[i] for v in kos)[::-1] for i in range(len(kos))]
            kos = kos[::-1]
    return vsi_obrati

tiles = [[tile(x, y) for x in range(ww)] for y in range(hh)]
print("\n".join(repr(x) + "," for x in tiles))

flat_tiles = reduce(add, tiles)
shuffle(flat_tiles)
open(fn + "-puzzle.txt", "wt").write(
    "\n\n".join("\n".join(choice(obrati(tile))) for tile in flat_tiles))

```