def stevilo\_otrok(oseba):

return len(otroci[oseba])

def stevilo\_vnukov(oseba):

v = 0

for otrok in otroci[oseba]:

v += len(otroci[otrok])

return v

def velikost\_rodbine(oseba):

v = 0

for otrok in otroci[oseba]:

v += velikost\_rodbine(otrok)

return v + 1

def obstaja\_ime(oseba, ime):

if oseba == ime:

return True

for otrok in otroci[oseba]:

if obstaja\_ime(otrok, ime):

return True

return False

def najvec\_otrok(oseba):

najvec = len(otroci[oseba])

for otrok in otroci[oseba]:

koliko = najvec\_otrok(otrok)

if koliko > najvec:

najvec = koliko

return najvec

def najdaljse\_ime(oseba):

najdaljse = oseba

for otrok in otroci[oseba]:

naj\_pod = najdaljse\_ime(otrok)

if len(naj\_pod) > len(najdaljse):

najdaljse = naj\_pod

return najdaljse

def globina(oseba):

najvecja = -1

for otrok in otroci[oseba]:

g = globina(otrok)

if g > najvecja:

najvecja = g

return najvecja + 1

def stevilo\_potomcev(oseba):

potomcev = len(otroci[oseba])

for otrok in otroci[oseba]:

potomcev += stevilo\_potomcev(otrok)

return potomcev

clani = []

def rodbina(oseba):

clani.append(oseba)

for otrok in otroci[oseba]:

rodbina(otrok)

return clani

def globina\_imena(oseba, ime):

if oseba == ime:

return 0

for otrok in otroci[oseba]:

n = globina\_imena(otrok, ime)

if n != None:

return n + 1

return None

def palindrom(s):

return s == "" or s[0] == s[-1] and palindrom(s[1:-1])

def vsota(s):

if not s:

return 0

return s[0] + vsota(s[1:])

def soda(s):

return not s or s[0] % 2 == 0 and soda(s[1:])

def vsota\_sodih(s):

if not s:

return 0

if s[0] % 2 == 0:

return s[0] + vsota\_sodih(s[1:])

else:

return vsota\_sodih(s[1:])

def prvi\_sodi(s):

if s[0] % 2 == 0:

return s[0]

else:

return prvi\_sodi(s[1:])

def sodolih(s):

return s == [] or s[0] % 2 == 0 and lihosod(s[1:])

def lihosod(s):

return s == [] or s[0] % 2 == 1 and sodolih(s[1:])

def faktoriela(n):  
 if n == 1 or n == 0:  
 return 1  
 return faktoriela(n-1) \* n  
  
def kolikokrat\_ime(rodovnik, oseba, ime ):  
 st = 0  
 if oseba.split(" ")[0] == ime:  
 st += 1  
 for osebek in rodovnik[oseba]:  
 st += kolikokrat\_ime(rodovnik, osebek, ime)  
  
 return st  
def zensk\_v\_rodbini(rodovnik, oseba):  
 st = 0  
 if oseba.split(" ")[0][-1] == "a":  
 st += 1  
 for osebek in rodovnik[oseba]:  
 st += zensk\_v\_rodbini(rodovnik, osebek)  
  
 return st  
  
def najvec\_otrok(rodovnik, oseba):  
 naj = len(rodovnik[oseba])  
 for oseba in rodovnik[oseba]:  
 if naj < najvec\_otrok(rodovnik, oseba):  
 naj = najvec\_otrok(rodovnik, oseba)  
  
  
 return naj  
  
  
def potomstvo(rodovnik, oseba):  
 set2 = set()  
 for oseba2 in rodovnik[oseba]:  
 set2 |= potomstvo(rodovnik, oseba2) | {oseba2}  
  
 return set2  
rodovnik = {'Ulrik I.': ['Viljem'], 'Margareta': [], 'Herman I.':  
 ['Herman II.', 'Hans'], 'Elizabeta II.': [], 'Viljem': ['Ana Poljska'],  
 'Elizabeta I.': [], 'Ana Poljska': [], 'Herman III.': ['Margareta'],  
 'Ana Ortenburška': [], 'Barbara': [], 'Herman IV.': [], 'Katarina': [],  
 'Friderik III.': [], 'Herman II.': ['Ludvik', 'Friderik II.', 'Herman III.',  
 'Elizabeta I.', 'Barbara'], 'Ulrik II.': ['Herman IV.', 'Jurij',  
 'Elizabeta II.'], 'Hans': [], 'Ludvik': [], 'Friderik I.': ['Ulrik I.',  
 'Katarina', 'Herman I.', 'Ana Ortenburška'], 'Friderik II.': ['Friderik III.',  
 'Ulrik II.'], 'Jurij': []}  
  
def preberi\_izdelke(ime):  
 izdelki = collections.defaultdict(set)  
  
 for vrstica in open(ime):  
 id,stars,ime = vrstica.strip().split(";")  
 izdelki[stars].add((id, ime))  
  
 print(dict(izdelki))  
 return dict(izdelki)

def pot\_do(od\_ime, do\_ime):

if od\_ime == do\_ime:

return [od\_ime]

for otrok in otroci[od\_ime]:

d = pot\_do(otrok, do\_ime)

if d is not None:

return [od\_ime] + d

4. Rekurzivni štumfi, zokni, kalcete, fuzetlne in kucjte  
Napiši rekurzivno funkcijo brez\_para(nogavica, nogavice), ki prejme številko nogavice in podoben seznam  
kot prva naloga. Vrniti mora True, če je nogavica brez para, in False, če ni.  
Klic brez\_para(39, [41, 39, 39, 41, 41, 39, 39]) vrne False in klic brez\_para(41, [41, 39, 39, 41, 41,  
39, 39]) vrne True, saj imamo eno 41 brez para.

def brez\_para(stevilka, nogavice):

if not nogavice:

return False

if nogavice[0] == stevilka:

return not brez\_para(stevilka, nogavice[1:])

else:

return brez\_para(stevilka, nogavice[1:])

2. Vreme  
Napiši funkcijo izpis\_vrstice(kraj, vreme, temperatura, veter, tlak) , ki prejme vremenske podatke: prva  
dva argumenta sta niza. Ostali argumenti so številke ali pa prazen niz, če podatek ni znan. Funkcija vrne niz z  
opisom vremena (glej teste). Kraj je izpisan na 35 mest, vreme na 20, temperatura in veter na 5 ter tlak na 8.  
Napiši funkcijo izpisi\_vreme(datoteka) , ki prejme ime datoteke z vremenskimi podatki in vrne niz s celotnim  
“vremenskim poročilom” (glej teste). Kako je videti datoteka, si oglej v vreme.txt, ki se uporablja tudi v testih.

def izpisi\_vreme(fname):

s = []

for line in open(fname):

data = (line.strip().split("\t") + [""] \* 5)[:5]

kraj, vreme = data[:2]

temperatura, veter, tlak = [int(x) if x else "" for x in data[2:]]

s.append(izpis\_vrstice(kraj, vreme, temperatura, veter, tlak))

return "\n".join(s)

podatki = (podatki + [""] \* 5)[:5]

Imamo naselje vrtičkarjev, ki je pravokotne oblike in je razdeljeno na  
nxm kvadratnih polj s stranico 1. Med dvema poljema je ograja, če imata  
različna lastnika. Primer je narisan na desni. Kdo je lastnik katerega  
vrtička, je zapisano v seznamu nizov; za primer na desni bi bil seznam  
takšen: ["AAABC", "ABCDC", "ACCDA"] .  
Napišite funkcijo ograje(s) , ki dobi seznam v tej obliki in vrne skupno  
dolžino vsej ograj – tako teh med vrtički kot vseh okrog njih.  
Namig: morda vam bo lažje reševati tako, da najprej izračunate, koliko ograje bi potrebovali, če bi bili  
vsi lastniki različni. Nato od tega preštejte, koliko je polj, med katerimi je ograja odveč. Nekaj  
podobnega smo namreč že počeli.

def ograje(s): ista = 0 for v in s: for i, j in zip(v, v[1:]): if i == j: ista += 1 for v1, v2 in zip(s, s[1:]): for i, j in zip(v1, v2): if i == j: ista += 1 return 2 \* len(s) \* len(s[0]) + len(s) + len(s[0]) – ista

1. Ravnotežje  
Proti gugalnici prihaja kolona otrok z znanimi težami. Prvih nekaj bomo poslali na levo stran gugalnice,  
ostale na desno. Koliko otrok z začetka je potrebno poslati na levo, da bo čimbolj uravnotežena?  
Z drugimi besedami, napiši funkcijo ravnotezje(s) , ki bo za podani seznam s vrnila tak indeks k , da  
bo vsota velikosti prvih k elementov čim bližje vsoti velikosti preostalih k . Lahko je manjša ali večja!

def ograje(s): ista = 0 for v in s: for i, j in zip(v, v[1:]): if i == j: ista += 1 for v1, v2 in zip(s, s[1:]): for i, j in zip(v1, v2): if i == j: ista += 1 return 2 \* len(s) \* len(s[0]) + len(s) + len(s[0]) - ista

3. Čebele z leve in desne  
V nekem vrtu so rože posejane ob potki, zato je količina nektarja v njih opisana z običajnim  
(enodimenzionalnim) seznamom. Obiranja se lotita dve čebeli: prva gre z leve proti desni, druga z  
desne proti levi. Čebela potrebuje eno sekundo, da se premakne iz cveta na cvet in eno sekundo, da  
obere eno enoto nektarja. Napišite funkcijo srecanje(vrt) , ki pove na katerem cvetu se bosta  
čebeli srečali. Za vrt [1, 4, 1, 2, 8, 3] mora vrniti 4.  
Funkcija mora upoštevati, da se lahko čebeli srečata tudi med letom in ne samo na cvetu. Za seznam  
[0, 0, 0, 0] , naj vaš program vrne 1.5 .  
Lahko se zgodi, da ob trenutku srečanja ostane kakšen cvet neobran. Čebeli se na vrtu [6, 0]  
srečata na prvem cvetu, na katerem je v tem trenutku še 5 enot nektarja.  
Namig: najprej razmislite, koliko časa bosta potrebovali obe čebeli skupaj, nato za eno od čebel  
izračunajte, kje bo takrat

def srecanje(vrt): cas = (len(vrt) + sum(vrt)) / 2 cvet = -1 while cas > 0: cvet += 1 cas -= 1 + vrt[cvet] if cas == 0: cvet += 0.5 return cvet

def crv(drevo): naj = 0 for veja, naprej in drevo.items(): if veja == "zelena": naj = max(naj, 1 + crv(naprej)) return naj

4. Največ sester  
Imejmo rodovnike, ki so podani tako, kot so bili na predavanjih iz rekurzije. Napišite funkcijo  
najvec\_sester(ime, rodovnik) , ki pove, koliko sester ima oseba z največ sestrami. Funkcija dobi  
kot argument slovar z rodovnikom in ime osebe, med katere potomci iščemo (sestre podane osebe  
nas ne zanimajo). Delo si poenostavimo tako kot na predavanjih: ime je žensko, če se konča s črko "a".  
Pazi: v družini z otroki Ana, Berta, Jože sta dve hčeri in nekdo (Jože) ima dve sestri. V družini z  
otrokoma Ano in Berto sta prav tako dve hčeri, vendar ima vsako od njiju le eno sestro.  
Nasvet: mogoče vam bo lažje, če najprej napišete funkcijo sester\_pod(ime, rodovnik) , ki pove,  
koliko sester ima tisti otrok osebe ime , ki ima največ sester.

def sester\_pod(ime, rodovnik): otroci = rodovnik[ime] sester = len([otrok for otrok in otroci if otrok.split()[0][-1] == "a"]) if sester and sester == len(otroci): sester -= 1 return sester def najvec\_sester(ime, rodovnik): return max([sester\_pod(ime, rodovnik)] + [najvec\_sester(otrok, rodovnik) for otrok in rodovnik[ime]])