

Iz dobro obveščenih virov na Facebooku se je izvedelo, kakšen je namen marsovske invazije: prišli so nam krast paradižnik! (Resno. To je povedal ugleden znanstvenik, ki je skoraj Nobelov nagrajenec za ekonomijo.)

Rešitev celotnega izpita

Najprej celotna rešitev, za predstavo o njegovem obsegu. Tule so preprosto razumljive, a ne nujno najkrajše različice. Posamične rešitve - in še kakšna alternativa - so potem razložene spodaj, skupaj z opisom nalog.

```
def paketov(teze, nosilnost):
    skupna = 0
    for i, teza in enumerate(teze):
        if skupna + teza > nosilnost:
            return i
        skupna += teza
    return len(teze)

def uravnotezena(tovor):
    return abs(sum(tovor[:,2]) - sum(tovor[1::2])) <= 10

def razporedi(paketi, kapacitete):
    ladje = []
    for paket in paketi:
        for ladja in ladje:
            if sum(ladja) + paket <= kapacitete:
                ladja.append(paket)
                break
        else:
            ladje.append([paket])
    return ladje

def popis(ime_datoteke):
    kolicine = defaultdict(int)
    for vrstica in open(ime_datoteke):
        kraj, zelenjava = vrstica.split(":")
        zel_kol = zelenjava.split()
        if len(zel_kol) == 2 and zel_kol[0] == "paradižnik":
            kolicine[kraj] += int(zel_kol[1])
    return kolicine

def skladiscniki(sef, hierarhija):
```

```

if not hierarhija[sef]:
    return 1
return sum(skladiscniki(spodaj, hierarhija)
           for spodaj in hierarhija[sef])

```

1. Število paketov

Napiši funkcijo `paketov(teze, nosilnost)`, ki prejme seznam tež paketov in nosilnost ladje in vrne število paketov, ki jih bodo natovorili. Ladje vedno natovarjajo tako, da zlagajo nanjo pakete po vrsti; če bi naslednji paket prekoračil dovoljeno nosilnost, se nalaganje konča, tudi če morda sledijo lažji paketi.

Rešitev

Najbolj (osnovno)šolska rešitev je takšna:

```

def paketov(teze, nosilnost):
    skupna = 0
    i = 0
    while i < len(teze) and skupna + teze[i] <= nosilnost:
        skupna += teze[i]
        i += 1
    return i

```

Z zanko `while` nalagamo pakete, dokler je vsota skupne teže doslej naloženih in naslednjega paketa manjša ali enaka dovoljeni.

Različic tega je še mnogo, kot recimo

```

def paketov(teze, nosilnost):
    skupna = 0
    for i, teza in enumerate(teze):
        if skupna + teza > nosilnost:
            return i
        skupna += teza
    return len(teze)

```

ali

```

def paketov(teze, nosilnost):
    for i in range(len(teze)):
        if sum(teze[:i]) > nosilnost:
            return i - 1
    return len(teze)

```

2. Uravnotežene ladje

Če je marsovska tovorna vesoljska ladja naložena tako, da je razlika med težo tovora na levi in desni strani večja od 10, se prevrne. Tudi v vesolju se zgodi.

Zato jih natovarjajo tako, da pakete izmenično odlagajo na levo in na desno. Vendar si tega niso najboljše zamislili: če imajo pakete [2, 10, 3, 8, 1], se ladja prevrne, ker je skupna teža paketov na levi enaka 6, na desni pa 18, kar je več kot 10 več.

Napiši funkcijo `uravnotezena(tovor)`, ki prejme seznam paketov in vrne `True`, če je ladja uravnotežena in `False`, če ni. Zaželeno je, da funkcijo napišeš v eni vrstici.

Rešitev

Absolutna vrednost razlike vsote elementov na sodih in na lihih mestih mora biti manjša ali enaka 10.

```
def uravnotezena(tovor):  
    return abs(sum(tovor[::2]) - sum(tovor[1::2])) <= 10
```

3. Razpored

V resnici bodo nalagali pakete malo drugače. Imamo seznam tež paketov. Pakete jemljejo po vrsti in vedno (ne tako kot v prvi nalogi) pregledajo seznam paketov do konca. Če je paket možno naložiti, gre na to ladjo, sicer ga pustijo v vrsti, da ga bo (morda) pobrala naslednja. Na voljo imajo neomejeno število ladij; vse imajo enako nosilnost. Ta je večja od teže najtežjega paketa.

Napiši funkcijo `razporedi(teze, nosilnost)`, ki prejme seznam tež paketov in nosilnost ladij. Vrniti mora seznam seznamov tež paketov po ladjah. Klic `razporedi([5, 3, 8, 1, 2, 3, 5, 4, 2, 4], 9)` vrne `[[5, 3, 1], [8], [2, 3, 4], [5, 2], [4]]`. Prva ladja odpelje pakete s težami 5, 3, (8 ne more), 1 ... in potem ne more nobenega več. Druga odpelje paket s težo 8, potem ne more naložiti nobenega več. Tretje odpelje pakete s težami 2, 3, in 4 (tistega s težo 5 mora izpustiti) ... In tako naprej.

Rešitev

Ta funkcija ne deluje pravilno!

```
def razporedi(paketi, kapaciteta):  
    ladje = []  
    while paketi:  
        ladja = []  
        for paket in paketi:  
            if sum(ladja) + paket <= kapaciteta:  
                ladja.append(paket)  
                paketi.remove(paket)  
        ladje.append(ladja)  
    return ladje
```

Tole zgoraj je bolj "idejni osnutek". "Ideja" je torej v tem, da pripravimo seznam, v katerega bomo zlagali ladje. Nato imamo zanko, ki teče, dokler obstaja še kakšen nenaložen paket. Znotraj te zanke pripravimo seznam tež paketov, ki gredo na to ladjo. Z zanko for se zapeljemo čez vse pakete in vsakega damo na ladjo, če vsota vseh tež paketov, ki so že na ladji in tega paketa ne presega dovoljene. Ko paket naložimo, ga odstranimo s seznama paketov.

Funkcija ne deluje iz dveh razlogov.

Prvi, manj pomembni, je ta, da funkcije ne smejo kar tako, nepooblaščno spreminjati seznamov, ki jih dobijo kot argument. To rešimo preprosto tako, da seznam na začetku funkcije skopiramo.

Drugi, pomembnejši, je, da znotraj zanke for ne moremo brisati elementov seznama, saj bo zanka preskakovala elemente, ki sledijo pobrisanim. Kot smo se učili, se temu izognemo, recimo, z zanko while.

```
def razporedi(paketi, kapaciteta):
    ladje = []
    s = paketi.copy()
    while s:
        i = 0
        ladja = []
        while i < len(s):
            if sum(ladja) + s[i] <= kapaciteta:
                ladja.append(s.pop(i))
            else:
                i += 1
        ladje.append(ladja)
    return ladje
```

Tule pa je čisto drugačna - in boljša rešitev -, ki sem se je domislil med popravljanjem izpitov FRIjevcev, kasneje pa dejansko naletel na študenta, ki je naredil isto in še malo preprosteje (jaz sem brez potrebe kompliciral z `enumerate`). Takole:

```
def razporedi(paketi, kapacitete):
    ladje = []
    for paket in paketi:
        for ladja in ladje:
            if sum(ladja) + paket <= kapacitete:
                ladja.append(paket)
                break
        else:
            ladje.append([paket])
    return ladje
```

Gremo čez pakete in vsak paket dodamo na prvo ladjo, ki ga lahko sprejme. Če ga ne more sprejeti nobena, dodamo novo ladjo.

4. Popis

Marsovski obveščevalci so zbrali popis zelenjave po slovenskih mestih in jih shranili v datoteko takšne oblike.

```
Ljubljana: paradižnik 18
Maribor: rumena koleraba 5
Ljubljana: rdeča pesa 13
Ljubljana: paradižnik 5
Škofja Loka: buče 5
Škofja Loka: paradižnik 1
```

Mesto se lahko pojavi tudi večkrat z isto zelenjavo. Napiši funkcijo `popis(ime)`, ki prejme ime datoteke in vrne slovar, katerega ključi so imena mest, vrednosti pa količina paradižnika, na primer `{"Ljubljana": 23, "Škofja Loka": 1}`. (V izogib težavam na MS Windows datoteko odpri z `open(ime, encoding="utf-8")`.)

Rešitev

Uporabili bomo `defaultdict(int)`, v katerega bomo prištevali količine.

Vrstico razdelimo najprej glede na `:`, da dobimo kraj, kot prvi element, in ime ter količino zelenjave kot drugi. Ostanek razdelimo glede na beli prostor (presledke). Če ima dva elementa in je prvi element paradižnik, potem je drugi element niz s količino paradižnika.

```
def popis(ime_datoteke):
    kolicine = defaultdict(int)
    for vrstica in open(ime_datoteke):
        kraj, zelenjava = vrstica.split(":")
        zel_kol = zelenjava.split()
        if len(zel_kol) == 2 and zel_kol[0] == "paradižnik":
            kolicine[kraj] += int(zel_kol[1])
    return kolicine
```

Malenkost bolj Pythonovska rešitev je

```
def popis(ime_datoteke):
    kolicine = defaultdict(int)
    for vrstica in open(ime_datoteke):
        kraj, zelenjava = vrstica.split(":")
        *zelenjava, kolicina = zelenjava.split()
        if zelenjava == ["paradižnik"]:
            kolicine[kraj] += int(kolicina)
    return kolicine
```

5. Skladiščniki

Ladjo vodi hierarhija skladiščnikov: general skladiščniki, skladiščniki prvega razreda, višji skladiščniki in tako naprej. Čisto na dnu so običajni skladiščniki (na sliki so to Tadeja, Cilka, Ludvik, Franc, Alenka, Petra, Aleksander, Barbara, Margareta in Erik).

Napiši funkcijo `skladiscniki` (`marsovec`, `hierarhija`), ki prejme ime nekega marsovca in slovar s hierarhijo, vrne pa število običajnih skladiščnikov, ki jim ta marsovec poveljuje (vključno s samim seboj). Adam, recimo, poveljuje, desetim. Juri poveljuje štirim. Petra poveljuje samo sebi.

Rešitev

Če marsovec nima podrejenih, je takšen eden, namreč on sam. Če jih ima, pa je število podrejenih skladiščnikov enako vsoti števila podrejenih skladiščnikov, ki jih imajo njegovi podrejeni.

```
def skladiscniki(sef, hierarhija):
    if not hierarhija[sef]:
        return 1

    return sum(skladiscniki(spodaj, hierarhija)
               for spodaj in hierarhija[sef])
```

Če se komu mudi, pa napiše

```
def skladiscniki(sef, hierarhija):
    return sum(skladiscniki(spodaj, hierarhija)
               for spodaj in hierarhija[sef]) or 1
```

Razmislite, zakaj.