Izpit iz predmeta Programiranje 1, 11. junij 2013 ob 13.00. **Čas pisanja: 90 minut.**

Rešitve shranite **v eno samo datoteko s končnico .py** in jo oddajte prek Učilnice. Imena funkcij naj bodo takšna, kot jih predpisuje naloga. Rešitev preverite s testnimi primeri na Učilinici. Za rešitev naloge lahko dobite določeno število točk, tudi če ne prestane vseh testov. Funkcija, ki prestane vse teste, še ni nujno pravilna.

Normalno sestavljene rešitve prvih dveh nalog imajo po 10 vrstic, tretja ima eno, četrta in peta pa 15. Čeprav ne zahtevamo takšne učinkovitosti, pa bomo posebej okorno napisane rešitve kaznovali z nekaj odbitimi točkami.

Dovoljena je vsa literatura na poljubnih medijih in ves material, ki je objavljen na Učilnici. Študenti s predolgimi vratovi in podobnimi hibami bodo morali (najmanj!) zapustiti izpit, katerega opravljanje se bo štelo kot neuspešno.

1. Blagajna

Zaporedje dogodkov na blagajni v trgovini lahko predstavimo z zaporedjem plusov in minusov: plus naj pomeni, da se je v vrsto postavila nova stranka in minus, da je stranka plačala in šla. Tako bi zaporedje `"++-++---"` pomenilo, da je najprej prišla ena stranka, nato še ena, nato je bila ena postrežena, nato sta dve prišli in potem so bile tri postrežene.

Napišite funkcijo `blagajna(s)`, ki kot argument prejme niz, kakršen je gornji, kot rezultat pa vrne najdaljšo dolžino vrste. Predpostaviti smete, da niz vsebuje le pluse in minuse.

Primeri

```
>>> blagajna("++-++---")
3
>>> blagajna("+++++-----")
5
>>> blagajna("+-+-+-+-+-+-")
1
```

2. Kockarji

N kockarjev eden za drugim meče kocko in mete beleži v seznam. Če bi prvi kockar vrgel 2, drugi 3, tretji 5, prvi 4, drugi 3, tretji 6, bi to zapisali kot [2, 3, 5, 4, 3, 6].

Napišite funkcijo kockarji(s, n), ki prejme seznam in število kockarjev, kot rezultat pa vrne zaporedno številko kockarja, ki je vrgel največ šestic. (Prvi kockar naj ima zaporedno številko 1, ne 0.) Predpostaviti smete, da v igri sodeluje vsaj en igralec in, če potrebujete, tudi, da je dolžina seznama večkratnik n.

Namig: najprej sestavi seznam dolžine n, v njem naj bodo same ničle. Nato se sprehodi prek seznama s in vsakič, ko naletiš na šestico, prištej 1 ustreznemu tekmovalcu (prav ti utegne priti operator %). Ko je to opravljeno, poiščeš zaporedno številko kockarja z največ šesticami.

Primeri

```
>>> kockarji([1, 2, 6,   1, 2, 6,   1, 6, 6,   1, 2, 1], 3)
3
>>> kockarji([1, 6,   1, 6,   2, 2], 2)
2
```

3. Enaka seznama

Dva seznama sta enaka, če sta oba prazna ali pa sta oba neprazna in imata enaka prva elementa in sta enaka tudi ostanka seznama (vse razen prvega elementa).

Sprogramirajte *rekurzivno funkcijo* `enaka(s, t)`, ki vrne `True`, če sta seznama `s` in `t` enaka in `False`, če nista. Funkcija naj deluje tako, kot je opisano v prejšnjem odstavku. (Duhovitosti, kot je `return s == t`, niso dovoljene; to tudi ni rekurzivna rešitev.)

4. Lego

Vsebino škatle Legovih kock lahko opišemo s slovarjem: ključi predstavljajo tip kocke, vrednosti pa so število kock takšne oblike. Zapis `{"A": 2, "B": 1, "C": 3}` pomeni, da škatla vsebuje dve kocki vrste A, eno kocko vrste B in tri kocke vrste C. (V resnici Lego kock ne označuje s črkami, a to za nalogo ni pomembno.)

Na podoben način lahko opišemo kocke, ki jih potrebujemo za izdelavo neke reči. `{"A": 4, "C": 3}` pomeni, da potrebujemo štiri kocke A in tri kocke C.

Opazimo lahko, da s kockami iz gornje škatle ne moremo narediti spodnje reči, ker imamo premalo kock vrste A (imamo dve, potrebovali bi štiri). Prav tako ne moremo narediti `{"A": 2, "D": 3}`, ker nimamo kock vrste D. Lahko pa bi naredili, recimo `{"A": 1, "C": 2}`.

Napišite dve funkciji. `vsi_deli(skatla, potrebno)` naj vrne `True`, če vsebina škatle, opisana s slovarjem `skatla`, zadošča, da naredimo reč, za katere potrebujemo vse, kar piše v slovarju potrebno.

Podobna funkcija `kaj_manjka(skatla, potrebno)` naj vrne slovar delov, ki manjkajo, da bi lahko iz škatle, katere vsebina je opisana v `skatla`, naredili reč, katere sestavni deli so opisani v potrebno.

Funkciji se lahko kličeta med sabo, če želite.

Pozor: funkciji ne smeta spremeniti nobenega od slovarjev, ki jih dobita kot argument!

Primeri

```
>>> vsi_deli ({"A": 2, "B": 1, "C": 3}, {"A": 3, "C": 2, "D": 2})
False

>>> kaj_manjka({"A": 2, "B": 1, "C": 3}, {"A": 3, "C": 2, "D": 2})
{"A": 1, "D": 2}

>>> vsi_deli({"A": 2, "B": 1, "C": 3}, {"A": 2, "C": 2})
True

>>> kaj_manjka({"A": 2, "B": 1, "C": 3}, {"A": 2, "C": 2})
{}
```

5. Lego - objektni

Sestavite razred `Skatla`. Njegov konstruktor dobi vsebino škatle, opisane s slovarjem na takšen način kot v prejšnji nalogi.

Metoda `vsi_deli(self, potrebno)` pove, ali škatla vsebuje vse, kar je potrebno (kot v prejšnji nalogi). Tudi metoda `kaj_manjka(self, potrebno)`, dela enako kot funkcija iz prejšnje naloge.

Pri tej nalogi smete seveda uporabiti (oz. predelati), kar ste napisali za četrto nalogo.

Če vam ni uspelo rešiti četrte naloge, lahko vseeno rešujete peto; če bodo metode zastavljene dovolj pravilno, da bom vedel, da bi znali dopolniti četrto v peto (če bi rešili četrto), boste za to vseeno dobili večino točk.