**Izpit iz predmeta Programiranje 1, 26. januar 2011 ob 10.00.**

Rešitve **vseh** nalog shranite **v eno samo datoteko s končnico .py** in jo oddajte prek Učilnice tako kot domače naloge. Vse funkcije naj imajo enaka imena in argumente, kot jih predpisuje naloga. Rezultate naj vrnejo, ne izpišejo.

Pozorno preberi naloge in ne rešuj le na podlagi podanih primerov!

Da vaša rešitev ne bi imela trivialnih napak, jo preverite s testi v ločeni datoteki na Učilnici. Za rešitev naloge lahko dobite določeno število točk, tudi če ne prestane vseh testov. Funkcija, ki prestane vse teste, še ni nujno pravilna.

Pri reševanju nalog je dovoljena vsa literatura na poljubnih medijih, ves material, ki je objavljen na Učilnici, vključno z objavljenimi programi; njihova uporaba in predelava se ne šteje za prepisovanje.

Izpit morate pisati na fakultetnih računalnikih, ne lastnih prenosnikih.

Študenti s predolgimi vratovi in podobnimi hibami bodo morali zapustili izpit, katerega opravljanje se bo štelo kot neuspešno. Hujše kršitve bomo prijavili disciplinski komisiji za študente.

Čas pisanja: 90 minut.

Nalogi A in B se ne točkujeta, temveč sta obvezni za uspešno opravljen izpit. Kdor ju ne reši pravilno, je s tem že padel.

Nalogi morate rešiti tako, da predelate SVOJO rešitev domače naloge. Samo tisti, ki naloge niso oddali, oziroma jim je bila ocenjena negativno, naj vzamejo objavljeno rešitev.

Vse ostale naloge so vredne enako število točk.

## A. Napadalne kraljice (obvezna naloga!)

Vzemimo, da imamo posebno šahovnico, ki ima deset stolpcev namesto osem (poleg prvih osem imamo torej še stolpca `i` in `j`). Ustrezno popravi funkcijo `prost_stolpec`!

## B. Funkcije za delo z vektorji (obvezna naloga!)

Funkcija `mul` je množila vektor s številom. Spremeni jo tako, da bo množila vektor z vektorjem: produkt vektorjem `[1, 2, 5]` in `[7, 3, 12]` naj bo `[7, 6, 60]`, produkt vektorjev `[1, 2, 5]` in `[7, 3]` pa naj bo `[7, 6, 0]`. Namig: skopiraj, preimenuj in predelaj funkcijo `add`, staro funkcijo `mul` pa kar pobriši.

## 1. Večkratniki 7 raus!

Napiši funkcijo `v7raus(s)`, ki kot argument dobi seznam števil, kot rezultat pa vrne seznam, iz katerega so izbrisani vsi večkratniki 7. Funkcija naj uporablja izpeljane sezname, tako da bo dolga le eno vrstico (poleg prve vrstice, glave funkcije). Če ne znaš tako, napiši na daljši način, a boš dobil za to le polovico točk.

**Primer**

```
>>> v7raus([1, 2, 7, 14, 33, 140, 5])
[1, 2, 33, 5]
>>> v7raus([0, 7, 14])
[]
>>> v7raus([])
[]
```

## 2. Sekajoči se krogi

Napiši funkcijo `sekajo(krogi)`, ki dobi seznam krogov, podanih s terkami `(x, y, r)`, kjer sta `x` in `y` koordinati središča kroga, `r` pa polmer. Funkcija naj vrne `True`, če se vsaj dva kroga sekata in `False`, če se ne.

**Namig**: da bo vse prav, boš moral najbrž uporabiti `enumerate(krogi)` ali `range(len(krogi))`.

**Primer**

```
>>> sekajo([(0, 0, 1), (2, 2, 1), (4, 4, 1), (6, 6, 1)])
False
>>> sekajo([(0, 0, 1), (2, 2, 1), (4, 4, 1), (4, 6, 2)])
True
>>> sekajo([(1, 5, 2), (1, 5, 5])
True
```

## 3. Blagajna

Napiši razred `Blagajna`. Konstruktor (`__init__`) naj prejme slovar, ki pove, kakšni bankovci so v blagajni, npr. `{100: 5, 50: 6, 10: 7, 5: 3}` bi pomenilo, da je v blagajni pet bankovcev za 100 evrov, šest bankovcev za 50, sedem za 10 in trije za 5 evrov.

`Blagajna` naj ima dve metodi. Metoda `vsota()` naj pove, koliko denarja je v blagajni.

Metoda `izplacaj(bankovci)` dobi kot argument slovar bankovcev (v enaki obliki kot zgoraj), ki jih je potrebno izplačati. Metoda mora ustrezno popraviti stanje blagajne. Rezultata ne vrača.

Če se podanih bankovcev ne da izplačati, ker jih v blagajni ni, naj blagajna sproži izjemo (exception) vrste `ValueError`. V tem primeru naj blagajna *ne izplača ničesar*, torej ne spremeni stanja! (**Namig**: najprej preveri, če bo šlo in šele nato izplačaj!)

**Primer**

```
>>> b = Blagajna({100: 5, 50: 6, 10: 7, 5: 3})
>>> b.vsota()
885
>>> b.izplacaj({100: 2, 10: 3})
>>> b.vsota()
655
>>> b.izplacaj({100: 7})
ValueError: nimam toliko bankovcev po 100
>>> b.vsota()
655
```

## 4. Povedi

Napiši funkcijo `v_stavke(s)`, ki kot argument dobi seznam nizov, ki predstavlja neko besedilo, razbito po vrsticah (recimo tako, kot bi ga dobili iz datoteke, če bi jo prebrali z `readlines()`, le brez znakov `\n`.) Kot rezultat naj vrne taisto besedilo v seznamu, katerega elementi predstavljajo stavke. Predpostavi, da se stavki vedno končajo s piko, klicajem ali vprašajem ter da vse pike, klicaji in vprašaji pomenijo konec stavka. Če tega ne znaš, pozabi na klicaje in vprašaje ter išči le pike, vendar boš dobil(a) za to le polovico točk.

**Primer**

```
>>> t = ["Napiši funkcijo povedi, ki kot argument ", "sprejme nekaj. Kot rezultat ",
         "vrne nekaj drugega, ", "namreč seznam. ", "Kaj naj ta vsebuje? Eh, ",
         "kaj neki, nize! Same ", "nize. ", "Da, tako bodi."]
>>> v_stavke(t)
["Napiši funkcijo povedi, ki kot argument sprejme nekaj.", "Kot rezultat vrne nekaj
drugega, namreč seznam.", "Kaj naj ta vsebuje?", "Eh, kaj neki, nize!", "Same nize.",
"Da, tako bodi."]
```

## 5. Seznam po modulu

Napiši razred `listmod`, ki se vede podobno kot seznam (konstruktor sestavi prazen seznam, poleg tega pa pozna razred vsaj še metodo `append` in indeksiranje, kot kažejo spodnji primeri). Od "pravega" seznama naj se razlikuje po tem, da obravnava indekse po modulu. Če ima seznam `s`, recimo, pet elementov in napišemo `s[17]`, naj to pomeni element `2`, saj je `17 % 5` enako `2`. Torej: če želimo element z indeksom `i`, naj razred v resnici vrne oz. spreminja element z indeksom `i % len(s)`.

Delujejo naj tudi negativni indeksi: `s[-7]` naj bo isto kot `s[-2]`.

**Namig:** ne vznemirjaj se zaradi negativnih indeksov: če boš programiral pravilno, bodo delovali sami od sebe.

**Primer**

```
>>> b = listmod()
>>> for e in "Benjamin":
...     b.append(e)
...
>>> b[0]
'B'
>>> b[7]
'n'
>>> b[8]
'B'
>>> b[17]
'e'
>>> b[-2]
'i'
>>> b[-10]
'i'
>>> b[-18] = 42
>>> b[-10]
42
```