Izpit iz predmeta Programiranje 1, 23. januar 2011 ob 10.00.

Vse rešitve shranite **v eno samo datoteko s končnico .py** in jo oddajte prek Učilnice. Vse funkcije naj imajo enaka imena, kot jih predpisuje naloga. **Pozorno preberite** naloge in ne rešujte le na podlagi primerov!

Da 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.

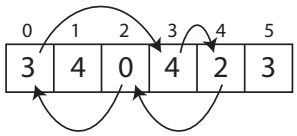
Pri reševanju nalog je dovoljena vsa literatura na poljubnih medijih, ves material, ki je objavljen na Učilnici, vključno z objavljenimi programi; njihova uporaba in predelava se ne šteje za prepisovanje.

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.

Čas pisanja: 90 minut.

1. Skoki



Napišite funkcijo `skoki(s)`, ki prepotuje podani seznam števil `s`. Recimo, da dobi kot argument seznam `[3, 4, 0, 4, 2, 3]`. Funkcija vedno začne na ničtem polju. Ta v našem primeru vsebuje `3`, zato skoči na tretje polje. Na tretjem polju je `4`, torej skoči na četrto polje. Vsebina četrtega polja je `2`, zato skoči na drugo polje. To vsebuje `0`: spet smo na začetku, zato končamo.

Funkcija naj kot rezultat vrne število skokov do vrnitve na polje 0; v gornjem primeru je to `4`.

Poleg tega naj funkcija pazi še na dve možnosti:

- Če se slučajno zgodi, da kako polje, *na katerega pride*, vsebuje preveliko število, naj se funkcija ustavi in vrne `-1`. Če so v seznamu prevelike številke, vendar je pot ne pripelje nanje, ni nič narobe.
- Če se funkcija zacikla – najpreprostejši primer je seznam `[1, 1]`, kjer s prvega polja stalno skače na prvo polje – naj vrne `-2`.

Namig: če se vrnemo na začetno polje, za to potrebujemo največ toliko skokov, kolikor je dolg seznam.

Predpostaviti smete, da seznam ni prazen in vsebuje samo cela pozitivna števila.

Primer

```
>>> skoki([3, 4, 0, 4, 2, 3])
4
>>> skoki([1, 1]) # stalno ponavlja polje 1
-2
>>> skoki([1, 2, 3, 4, 5, 3]) # ponavlja 4, 5, 3, 4, 5, 3, 4, 5, 3 ...
-2
>>> skoki([1, 2, 3, 8]) # pride do 8 in "skoci ven"
-1
>>> skoki([0])
1
```

2. Transakcije

V začetku je imela Ana štiri zlatnike, Berta 8 in CIlka 10, torej `[('Ana', 4), ('Berta', 8), ('Cilka', 10)]`. Nato je dala Cilka Ani 3 zlatnike; potem je dala Cilka Ani še 2; na koncu je dala Ana Berti 2, kar zapišemo `[('Cilka', 'Ana', 3), ('Cilka', 'Ana', 2), ('Ana', 'Berta', 2)]`. Kdo ima na koncu največ?

Napišite funkcijo `transakcije(zacetek, dajatve)`, ki dobi gornja seznama in vrne ime najbogatejše osebe po koncu transakcij. Če je na koncu več enako bogatih najbogatejših oseb, naj vrne poljubno izmed njih.

Namig: delo si boste poenostavili, če bo funkcija takoj pretvorila sezmam v primernejšo podatkovno strukturo.

Primer

```
>>> transakcije([('Ana', 4), ('Berta', 8), ('Cilka', 10)], [('Cilka', 'Ana', 3),
('Cilka', 'Ana', 2), ('Ana', 'Berta', 2)])
Berta
```

3. Deljenje nizov

Nize v Pythonu lahko pomnožimo s številom: `"bla"*3` vrne `"blablabla"`. Napišite funkcijo `deli_niz(s, k)`, ki niz `s` "deli" s številom `k`. Torej: `deli_niz(s, k)` mora vrniti tak niz, da bi, če ga pomnožimo s `k`, spet dobili `s`.

Če niz ni sestavljen iz `k` ponovitev nekega niza, naj funkcija vrne `None`.

Primer

```
>>> deli_niz('toktoktoktok', 4)
tok
>>> deli_niz('tktktktk', 2)
tktk
>>> deli_niz('XXX', 3)
X
>>> print(deli_niz('toktoktoktok', 3))
None
>>> print(deli_niz('tiktoktak', 3))
None
```

4. Sekajoči se pravokotniki

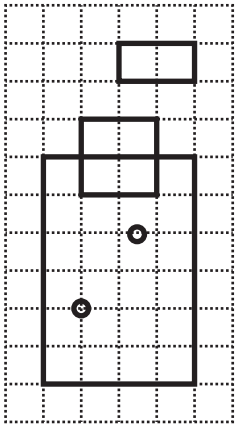
Napišite funkcijo `v_prav(pravokotnik, tocka)`, ki za podano točko pove, ali se nahaja v podanem pravokotniku. Pravokotnik je opisan s terko `(x0, y0, x1, y1)`, ki pove koordinati nasprotnih oglišč, pri čemer je vedno `x0` < `x1` in `y0` < `y1`. Točka je opisana s terko `(x, y)`. Točka je v pravokotniku tudi, če je na njegovi stranici.

Poleg tega napišite funkcijo `se_sekata(prav1, prav2)`, ki pove, ali se dana pravokotnika sekata (ali vsaj dotikata). Namig: sekata se, če je vsaj eno od štirih oglišč pravokotnika `prav2` znotraj `prav1` ali obratno.

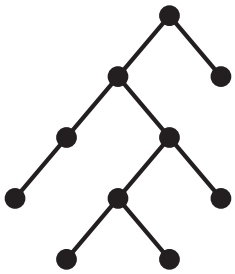
Končno, napišite funkcijo `sekajoci_par(s)`, ki dobi seznam pravokotnikov in vrne par pravokotnikov `(prav1, prav2)`, ki se sekata. Če je takšnih parov več, naj vrne kateregakoli. Če ni nobenega, naj vrne `None`.

Primer (kvadrati in točki iz primerov so označeni na sliki)

```
>>> v_prav((1, 1, 5, 7), (2, 3)) # spodnji pravokotnik in točka
True
>>> v_prav((2, 6, 4, 8), (3.5, 5)) # srednji pravokotnik in točka
False
>>> se_sekata((1, 1, 5, 7), (2, 4, 6, 8))
True
>>> se_sekata((1, 1, 5, 7), (3, 9, 5, 10))
False
>>> sekajoci_par([(1, 1, 5, 7), (3, 9, 5, 10), (2, 4, 6, 8)])
((2, 4, 6, 8), (1, 1, 5, 7))
>>> print(sekajoci_par([(1, 1, 5, 7), (3, 9, 5, 10)]))
None
```



5. Globina drevesa



Spomnimo se potomcev Karla Velikega, ki smo jih zložili v drevo, katerega koren je bil Karel, pod njim pa so bili njegovi potomci. V tej nalogi bomo imeli podobno drevo, s to razliko, da bo imel vsak element največ dva potomca (imenovali ju bomo *levi* in *desni* potomec).

Vozlišča drevesa predstavlja razred `Vozlisce` (uporabite definicijo razreda, ki je priložena testnim primerom). Razred ima dve metodi: `levi` vrne levega potomca in `desni` vrne desnega potomca. Če levega (oz. desnega) potomca ni, metoda vrne `None`.

Iz razreda `Vozlisce` izpeljite nov razred, `VozliscePlus`, ki ima poleg podedovanih metod še metodo `globina`, ki pove, kako globoko je drevo pod njim. Če ima, recimo, neko vozlišče (vsaj enega) pravnuka, je globina pod njim enaka 3 (sin-vnuk-pravnuk). Če ima, recimo, sina (ali dva), ta pa nima(ta) potomcev, je globina enaka 1. Če vozlišče sploh nima potomcev, je njegova globina enaka 0.

Koren drevesa na desni sliki ima globino 4, saj je najnižji potomec štiri nivoje pod njim.