Izpit iz predmeta Programiranje 1, 26. januar 2011 ob 12.30.

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.

Ostale naloge so vredne enako število točk.

A. Napadalne kraljice (obvezna naloga!)

Napiši funkcijo `proste_vrstice`, ki počne isto kot `prosti_stolpci`, le z vrsticami.

B. Iskanje datotek (obvezna naloga!)

Spremeni funkcijo `preisci`, tako da preskoči vse datoteke in direktorije, pri katerih je prvi znak imena pika.

1. Sodi vs. lihi

Napiši funkcijo `sodi_vs_lihi(s)`, ki kot argument dobi seznam števil. Funkcija naj ugotovi, ali je v seznamu več sodih ali lihih števil in vrne seznam tistih, ki jih je več. Če je obojih enako, naj vrne vsa soda števila.

Primer

```
>>> sodi_vs_lihi([1, 2, 7, 14, 33, 140, 5])
[1, 7, 33, 5]
>>> sodi_vs_lihi([1, 2, 7, 4])
[2, 4]
>>> sodi_vs_lihi([])
[]
```

2. Hamilkon

Šahovski konj se premika v obliki črke L, vedno za dve polji v eni smeri in eno v drugi, npr. a1 -> c2 ali e5 -> d3. Na ta način lahko prehodi celo šahovnico tako, da začne na poljubnem polju, preskače vsa polja (na vsako polje skoči le enkrat!) in konča tam, kje je začel. Napiši funkcijo `hamilkon(s)`, ki prejme seznam polj (npr. `["g3", "h1", "f2", "d1"]`) in vrne `True`, če podani seznam vsebuje pravilni obhod in `False`, če ne.

Predpostaviti smeš, da seznam vsebuje sama pravilno opisana polja (nize v obliki e5, a8...), tako da moraš preveriti le še, da seznam vsebuje vsa polja, da vsebuje vsako le enkrat, razen prvega, ki mora biti enako zadnjemu, in da so vse poteze pravilne, torej, da je konj vedno skočil tako, kot mora.

Primeri so v testni skripti.

3. Naslednji avtobus

Napiši funkcijo `naslednji_avtobus(prihodi)`, ki dobi seznam prihodov avtobusov in vrne "številko" (npr. `"1"` ali `"6b"`) avtobusa, ki bo prišel prvi. Če je za več avtobusov napovedan isti čas do prihoda, naj vrne tistega z nižjo številko (a pazite: avtobus 2 ima nižjo številko od avtobusa 11, prav tako ima avtobus 6b nižjo številko od 11). Prihodi avtobusov so podani kot slovar, katerega ključi so "številke" avtobusov ("1", "6b"...), vrednosti pa seznami napovedanih časov do prihoda. Časi niso nujno urejeni po velikosti.

Spodnji primeri so razvrščeni po naraščajoči težavnosti. Če ne znaš rešiti vseh, jih poskusi pokriti čim več.

Primer

```
>>> naslednji_avtobus({"1": [5, 7], "3": [3, 11], "6b": [7]})
"3"
>>> naslednji_avtobus({"1": [5, 7], "2": [11, 3, 18], "11": [3, 7]})
"2"
>>> naslednji_avtobus({"1": [5, 7], "2": [11, 3, 18], "6b": [3, 7]})
"2"
```

4. Cenzura

Napiši funkcijo `cenzura(besedilo, prepovedane)`, ki prejme niz `besedilo` in vrne nov niz, ki je enak staremu, le da so pobrisane vse besede, ki so podane v drugem argumentu, množici `prepovedane`. Predpostaviti smeš, da v besedilu ni nobenih ločil, le besede in presledki; funkcija sme v rezultatu pustiti odvečne presledke. V množici prepovedanih besed so vse besede napisane z malimi črkami, vendar so prepovedane tudi, če so napisane z velikimi.

Funkcija naj uporablja izpeljane sezname, tako da bo dolga le eno vrstico (poleg prve vrstice, glave funkcije in morebitnih uvozov modulov). Če ne znaš tako, napiši na daljši način, a boš dobil za to le polovico točk.

Primer

```
>>> prepovedane = {"zadnjica", "tepec", "pujs", "kreten"}
>>> cenzura("Pepe je ena navadna zadnjica in pujs in še kaj hujšega", prepovedane)
'Pepe je ena navadna in in še kaj hujšega'
>>> cenzura("Pepe je ena velika Zadnjica in PUJS in še kaj hujšega", prepovedane)
'Pepe je ena navadna in in še kaj hujšega'
>>> cenzura("Pepe je okreten", prepovedane)
'Pepe je okreten'
```

5. Dvosmerni slovar

Sestavi razred `two_way_dict`, ki se obnaša kot dvosmerni slovar: če indeksiramo s ključem, dobimo vrednost, če indeksiramo z vrednostjo, dobimo ključ. Če se iskana vrednost v slovarju pojavi kot ključ in kot vrednost, naj se upošteva kot ključ. Predpostaviti smeš, da se bo vsaka vrednost pojavila le enkrat.

Razred ima lahko konstruktor, ki ne sprejme argumentov (lahko pa jih sprejema, če želiš) in mora podpirati vsaj prirejanje in branje elementov ter metode `keys`, `values` in `items`.

Primer (metodi `keys` in `items` lahko vračata elemente v drugačnem vrstnem redu ali pa celo sploh ne vrneta seznama, temveč kaj drugega primernega)

```
>>> b = two_way_dict()
>>> b[3] = 1
>>> b[1] = 5
>>> b["Berta"] = "Cilka"
>>> b[1]
5
>>> b["Cilka"]
'Berta'
>>> b.keys()
[3, 1, 'Cilka']
>>> b.items()
[(3, 1), (1, 5), ('Berta', 'Cilka')]
```