Izpit iz predmeta Programiranje 1, 9. junij 2015 ob 12.00. Čas pisanja: 90 minut.

Vse rešitve shranite **v eno samo datoteko s končnico .py** in jo oddajte prek Učilnice. Vse funkcije naj imajo takšna imena, kot jih predpisuje naloga. 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 testov**. Funkcija, ki prestane vse teste, **še ni nujno pravilna**.

Pri reševanju nalog je dovoljena vsa literatura na poljubnih medijih.

1. Prelivanje

Napisali bomo funkcijo, ki je delček znane naloge, v kateri imamo različno velike vrče in "cisterno", naloga pa je nameriti določeno količino vode (mleka, piva...).

Napišite funkcijo `pretoci(velikosti, vsebina, iz, v)`, ki prejme seznam `velikosti` vrčev, koliko tekočine trenutno vsebujejo (`vsebina`) ter `iz` katerega vrča pretakamo `v` katerega. Če je `iz` enak `None`, pretakamo v cisterno. Če je `v` enak `None`, točimo v cisterno. Vedno pretakamo "do konca" – dokler ni ciljni vrč poln ali pa vrč, iz katerega točimo, prazen. Če točimo v cisterno, vedno izpraznimo vrč; če nalivamo iz cisterne, napolnimo vrč do roba.

Primer

```
>>> velikosti = [5, 7, 4]
>>> vsebine = [3, 4, 1]
>>>
>>> # iz prvega vrča smo napolnili ničtega; v prvem ostane 2
>>> pretoci(velikosti, vsebine, 1, 0)
>>> vsebine
[5, 2, 1]
>>>
>>> # celotno vsebino drugega vrča smo pretočili v prvega
>>> pretoci(velikosti, vsebine, 2, 1)
>>> vsebine
[5, 3, 0]
>>>
>>> # drugi vrč napolnimo iz cisterne
>>> pretoci(velikosti, vsebine, None, 2)
>>> vsebine
[5, 3, 4]
```

2. Čolni

Recimo, da na prvi čoln naložimo tri tovore, težke 4, 2 in 4 enote, na drugega en tovor, težak 10 enot in na tretjega tri tovore težke eno enoto. Tako obremenitev bi lahko opisali s seznamom [[4, 2, 4], [10], [1, 1, 1].

Napišite funkcijo `ni_preobremenjenih(tovori, nosilnost)`, ki prejme seznam v takšni obliki in nosilnost čolnov (ta je za vse enaka). Funkcija naj vrne `True`, če ni nobeden od čolnov preobremenjen.

Primer

```
>>> ni_preobremenjenih([[4, 2, 4], [10], [1, 1, 1]], 11)
>>> True
>>> ni_preobremenjenih([[4, 5, 4], [10], [1, 1, 1]], 11)
>>> False
```

V drugem primeru je preobremenjen prvi čoln, saj nameravamo nanj naložiti za 4 + 5 + 4 = 13 eno tovora, kar je več kot dovoljena nosilnost 11.

Če znaš, se potrudi napisati to funkcijo s pomočjo generatorjev oz. izpeljanih seznamov.

3. Pari

Napišite *rekurzivno* funkcijo `vecji(moski, zenske)`, ki prejme seznam višin moških in višin žensk. Prvi moški bo plesal s prvo žensko, drugi z drugo in tako naprej. Funkcija naj vrne `True`, če so vsi moški večji od žensk, s katerimi bodo plesali (prvi moški mora biti večji od prve ženske in tako naprej).

4. Podobnosti

Podobnost med dvema stavkoma lahko izmerimo tako, da število besed, ki se pojavljajo v obeh stavkih delimo s številom besed, ki se pojavijo v vsaj enem. Podobnost med stavkoma "Ana in Berta hočeta jesti kumarice" ter "Berta je videla Benjamina jesti kumarice" je 3 / 9, saj se besede Berta, jesti in kumarice pojavljata v obeh stavkih, vseh (različnih!) besed pa je 9.

Napišitev funkcijo `podobnosti(stavki)`, ki vrne slovar, katerega ključi so terke s pari stavkov, vrednosti pa podobnosti med stavki. Predpostaviti smete, da so v stavku le besede, ločene s presledki, in nobenih spremenljivk.

Primer

```
>>> besedila = ["A B C", "A B D", "D E F", "F G"]
>>> podobnosti(besedila)
{("A B C", "A B D"): 2 / 4, ("A B D", "A B C"): 2 / 4, # A in B izmed A, B, C, D
 ("A B C", "D E F"): 0,     ("D E F", "A B C"): 0,     # ni skupnih
 ("A B C", "F G"): 0,       ("F G", "A B C"): 0,       # ni skupnih
 ("A B D", "D E F"): 1 / 5, ("D E F", "A B D"): 1 / 5, # D izmed A, B, D, E, F
 ("A B D", "F G"): 0,       ("F G", "A B D"): 0,       # ni skupnih
 ("D E F", "F G"): 1 / 4,   ("F G", "D E F"): 1 / 4,   # F izmed D, E, F, G
}
```

Tule smo poenostavili primer tako, da so "besede" dolge eno črko – A, B, C ...

"Komentarji", ki smo jih dodali v primer, so le razlaga primera in se v resnici seveda ne izpišejo.

5. Dolžina poti

Podan je razred Robot, ki je videti takole (ni ga potrebno pretipkavati, saj je v datoteki s testi):

```
class Robot:
    def __init__(self):
        self.x = self.y = 0
        self.pot = [(0, 0)]

    def premik(self, kot, koliko):
        self.x += koliko * cos(kot)
        self.y += koliko * sin(kot)
        self.pot.append((self.x, self.y))
```

Gre za robota, ki ga lahko premikamo naokrog. Ve, kje je in na katerih točkah se je ustavljal doslej.

Ne da bi spreminjali obstoječi metodi dodajte metodo `dolzina_poti`, ki bo vrnila dolžino poti, ki jo je doslej prehodil robot.

Primer

```
>>> r = Robot()
>>> r.dolzina_poti()
0
>>> r.premik(0.2, 40)
>>> r.dolzina_poti()
40
>>> r.premik(0.6, 30)
>>> r.dolzina_poti()
70
```

Namig: ne poglabljajte se preveč v trigonometrijo. ;) Zares pomembna je le ena vrstica podanega razreda.