Programski jezici

http://www.programskijezici.matf.bg.ac.rs/

Univerzitet u Beogradu
Matematički fakultet

Programske paradigme

Materijali za vežbe

Nastavnik: Milena Vujošević Janičić
Asistent: Branislava Živković

Beograd
2016.

Priprema materijala:
dr Milena Vujošević Janičić, docent na Matematičkom fakultetu u Beogradu
Marjana Šolajić, asistent na Matematičkom fakultetu u Beogradu
Branislava Živković, asistent na Matematičkom fakultetu u Beogradu

Sadržaj

1

Skript programiranje

Potrebno je imati instaliran Python 2.7 na računaru.
Literatura:

(a) https://www.python.org/

(b) http://www.tutorialspoint.com/python

(c) https://wiki.python.org/moin/

1.1 Uvod, kolekcije, matematičke funkcije

1.1.1 Uvodni primeri

Zadatak 1.1 Ispisivanje pozdravne poruke, komentari.

```
# Ovako se pisu komentari
#
# Pokretanje programa iz terminala:
#  $python  hello.py
#
print "Hello world! :)"
```

Zadatak 1.2 Promenljive, niske, formatiran ispis, učitavanje sa standardnog ulaza, aritmetičke i logičke operacije, naredbe grananja.

```

# Promenljive se dinamicki tipiziraju
a = 45
b = 67.45
istina = True
# Niske su konstantne tj. nisu promenljive.
# To znaci da se menjanjem nekog karaktera u niski
# pravi nova niska u memoriji.
niska = "I believe i can fly!"

# Ispis na standardni izlaz
print a
print b
print a, b, istina

# Formatiran ispis
print "\n------Formatiran ispis------\n"
print "Ceo broj: {0:d} \nBroj u pokretnom zarezu {1:f}\nBulovska vrednost: {2:b}\
    \nNiska: {3:s}\n".format(a,b,istina,niska)


# Ucitavanje niske sa standardnog ulaza
print "\n------Ucitavanje sa standardnog ulaza------\n"
string_broj = raw_input("Unesite ceo broj: ")
broj = int(string_broj) # vrsi se konverzija stringa u ceo broj, slicno: float, str

```

```
# Osnovne aritmeticke operacije:
# +, -, *, /, %, ** (stepenovanje)
print "\n------Osnovne aritmeticke operacije------\n"
print broj+4

# Osnovne logicke operacije:
# not, and, or
print "\n------Osnovne logicke operacije------\n"
print istina or False

# Blokovi se ne ogranicavaju viticastim zagradama kao sto je u C-u
# vec moraju biti uvuceni tabulatorom.

# Naredba grananja
print "\n------Naredba granjanja------\n"
if broj%2 == 0:
  print "Unet je paran broj \n"
elif broj%3 == 0:
  print "Unet je broj deljiv sa 3\n"
# Naredbi <<elif>> moze biti vise
else:
  print "Unet je broj koji nije ni paran ni deljiv sa 3\n"

# Naredba <<switch>> ne postoji

# Petlja
print "\n------Petlja <<while>>------\n"
i=1
while i<=10:
  print i
  i=i+1 # i++ ne postoji, moze ili ovako ili i+=1

# Naredba <<break>> iskace iz bloka, isto kao i u C-u
# Naredba <<pass>> je ista kao naredba <<continue>> u C-u

# Funkcije
#
# def ime_funkcije(argumenti):
#     telo funkcije
#

def f1(x,y):
  return x+y

print f1(2,2)

def f2(a):
  return [a,2*a,3*a]

print f2(11.1)
```

Zadatak 1.3 Moduli math i random.

```
# Matematicke funkcije

# Ukljucujemo modul <<math>>
import math

# U ovom moduli se nalaze brojne funkcije kao sto su:
#
# math.sqrt(broj)
# math.log(broj, osnova)
# math.sin(ugao_u_radijanima), math.cos(), ...
# math.exp(stepen)
# math.factorial(broj)
# i druge...
print "\n------Matematicke funkcije------\n"
print math.factorial(6)
print math.log(125, 5)

# Pseudo slucajni brojevi

# Ukljucujemo modul <<random>>
```

```
21 import random
22
23 # Funkcija random() vraca pseudo slucajan broj tipa float iz opsega [0.0, 1.0)
24 print "\n------Pseudo slucajni brojevi------\n"
25 print "Pseudo slucajan broj iz opsega [0.0,1.0)\n"
26 print random.random()
27
28 # Korisne funkcije:
29 #
30 # randint(a,b) - vraca pseudo slucajan ceo broj n iz opsega [a,b]
31 # choice(lista) - vraca pseudo slucajan element iz liste
32 #
```

Zadatak 1.4 Liste.

```
1  # LISTA
2  #
3  # Notacija: [element1, element2, ...]
4  #
5  # Liste mogu sadrzati razlicite tipove podataka
6  lista = [1,2,3.4,"Another brick in the wall",True,[5,False,4.4,'Layla']]
7
8  print "\n------Lista------\n"
9  print lista
10
11 # Prazna lista
12 prazna = []
13
14 # Pristupanje elementima liste
15 print "\n------Pristupanje elementima liste------\n"
16 # Indeksiranje elemenata liste
17 print lista[0]
18 print lista[3][1]
19 print lista[0:3]
20 # Mozemo indeksirati liste unazad, pozicija -1 odgovara poslednjem elementu
21 print lista[-1]
22 # Ukoliko pokusama da pristupimo elementu liste
23 # koji se nalazi na poziciji van opsega interpreter ce nam prijaviti gresku
24 # IndexError: list index out of range
25 # print lista[100]
26
27 print "\n------Provera da li se element nalazi u listi-----\n"
28 if 1 in lista:
29   print "1 se nalazi u listi\n"
30
31 print "\n------Korisne funkcije za rad sa listama------\n"
32 # Ubacivanje elementa na kraj
33 print "Ubacivanje elementa na kraj liste\n"
34 lista.append(3.14)
35 print lista
36
37 # Ubacivanje elementa na odredjenu poziciju u listi
38 print "\nUbacivanje elementa na odredjenu poziciju u listi\n"
39 # list.insert(pozicija, element)
40 lista.insert(2, "Jana")
41 print lista
42
43 #
44 # Korisne funkcije:
45 #
46 # list.remove(x) - izbacuje prvo pojavljivanje elementa x iz liste
47 # list.count(x) - vraca broj koliko puta se element x nalazi u listi
48 # list.index(x) - vraca indeks prvog pojavljivanja elementa x u listi
49 # len(lista) - vraca broj elemenata liste
50 # del lista[a:b] - brise elemente liste od pozicije a do b
51 #
52 # Nadovezvivanje dve liste
53 print "\n------Nadovezvivanje dve liste------\n"
54 lista = lista+["Plava", "Zuta", "Crna"]
55 print lista
56
57
58 # Prolazak kroz listu
```

```
print "\n------Prolazak kroz listu petljom <<for>>------\n"
for i in lista:
  print i

# Poredjenje listi
#
# Dve liste se porede tako sto se njihovi elementi porede redom leksikografski
print "\n------Poredjenje listi------\n"
print "[1,2,3] < [1,2,5]"
print [1,2,3] < [1,2,5]
print "\n['abc','abc','abc'] < ['abc', 'ab', 'abcd']"
print ['abc','abc','abc'] < ['abc', 'ab', 'abcd']
print "\n['a','b','c'] > ['a', 'b']"
print ['a','b','c'] > ['a', 'b']


# Koriscenje liste kao stek strukture podataka
stek = [9,8,7]
# Operacija push je implementirana funkcijom append
stek.append(6)
stek.append(5)
print "\n------Ispisujemo stek------\n"
print stek
# Operacija pop je implementirana funkcijom pop
print "\n------Ispisujemo element dobijem funkcijom pop------\n"
print stek.pop()
print "\n------Ispisujemo znanje nakon pozivanja funkcije pop------\n"
print stek
```

Zadatak 1.5 Skup, katalog, uređene n-torke.

```


# SKUP
#
# Pravljenje skupa od liste
print "\n------Pravljenje skupa od liste------\n"
lista1 = [4,56,34,2,5,6,4,4,6]
skup = set(lista1)

for i in skup:
  print i


# Funkcija <<range>>
#
# range(kraj)
# range(pocetak, kraj[, korak])
print "\n------Funkcija <<range>>------\n"
brojevi = range(10)
for i in brojevi:
  print i

# KATALOG
#
# Katalog je kolekcija uredjenih parova oblika (kljuc, vrednost)
#
# Notacija: {kljuc:vrednost, kljuc:vrednost, ...}
print "\n------Katalog------\n"
prazna_mapa = {} # prazna mapa

mapa1 = {'a' : 3, 'b' : 4, 'c' : 5}

print mapa1

mapa = {"kljuc1":67.7, 6:"Vrednost 2"}

# Pristupanje elementima u mapi
print "\n------Pristupanje elementima u katalogu------\n"
print mapa['kljuc1']

# Prolazak kroz mapu
print "\n------Prolazak kroz katalog------\n"
for kljuc in mapa:
```

```
    print "'{0:s}' => {1:s}\n".format(str(kljuc),str(mapa[kljuc]))
44
# Korisne funkcije
46 #
# map.keys() - vraca listu kljuceva iz kataloga
48 # map.values() - vraca listu vrednosti iz kataloga
# map.has_key(kljuc) - vraca True/False u zavisnosti od toga da li se element
50 # sa kljucem kljuc nalazi u katalogu

52 # Uredjene N-TORKE
print "\n------Torke------\n"
54 torka = ("Daffy","Duck",11)

56 # Pristupanje elementima u torki
print "\n------Pristupanje elementima u torki------\n"
58 print torka[1]

60 print "\n------Ispisivanje torke------\n"
print torka
62
# Poredjenje torki
64 #
# Dve torke se porede tako sto se njihovi elementi porede redom leksikografski
66 print "\n------Poredjenje torki------\n"
print "(1,2,'a') < (1,2,'b')"
68 print (1,2,'a') < (1,2,'b')
print "\n([1,2,3], 'Bugs', 4) < ([1,1,1], 'Bunny', 6)"
70 print ([1,2,3], 'Bugs', 4) < ([1,1,1], 'Bunny', 6)
# Ukoliko torke ne sadrze elemente istog tipa na istim pozicijama, i dalje ih mozemo
    porediti,
72 # ali poredjenje se vrsi na osnovu imena tipa elementa leksikografski
# npr.  element tipa List < element tipa String < element tipa Tuple i slicno
74 print "\n(1,2,['a','b']) < (1,2,'ab')"
print (1,2,['a','b']) < (1,2,'ab')
```

1.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 1.6 **Pogodi broj** Napisati program koji implementira igricu "Pogodi broj". Na početku igre računar zamišlja jedan slučajan broj u intervalu [0,100]. Nakon toga igrač unosi svoje ime i započinje igru. Igrač unosi jedan po jedan broj sve dok ne pogodi koji broj je računar zamislio. Svaki put kada igrač unese broj, u zavisnosti od toga da li je broj koji je unet veći ili manji od zamišljenog broja ispisuje se odgovarajuća poruka. Igra se završava u trenutku kada igrač pogodio zamišljen broj.

[Rešenje 1.6]

Zadatak 1.7 **Aproksimacija broja PI metodom Monte Karlo** Napisati program koji aproksimira broj PI koriscenjem metode Monte Karlo. Sa standardnog ulaza unosi se broj N. Nakon toga N puta se bira tačka na slučajan način tako da su obe koordinate tačke iz intervala [0,1]. Broj PI se računa po sledecoj formuli:

$$PI = 4 * A/B$$

- A je broj slučajno izabranih tačaka koje pripadaju krugu poluprečnika 0.5, sa centrom u tački $(0.5, 0.5)$
- B je broj slučajno izabranih tačaka koje pripadaju kvadratu čija temena su tačke $(0, 0), (0, 1), (1, 1), (1, 0)$.

[Rešenje 1.7]

Zadatak 1.8 **X-O** Napisati program koji implementira igricu X-O sa dva igrača.

[Rešenje 1.8]

1.1.3 Zadaci za vežbu

Zadatak 1.9 **Anjc** Napisati program koji implementira igricu Anjc sa jednim igračem. Igra se sa špilom od 52 karte. Na početku igrač unosi svoje ime nakon čega računar deli dve karte igraču i dve karte sebi. U svakoj sledećoj iteraciji računar deli po jednu kartu igraču i sebi. Cilj igre je sakupiti karte koje u zbiru imaju 21 poen. Karte sa brojevima nose onoliko bodova koliki je broj, dok žandar, dama, kralj nose 10 bodova. Karta As može da nosi 1 ili 10 bodova, u zavisnosti od toga kako igraču odgovara. Igrač koji sakupi 21 je pobedio. Ukoliko igrač premaši 21 bod, pobednik je njegov protivnik. https://en.wikipedia.org/wiki/Blackjack

Zadatak 1.10 **4 u liniji** Napisati program koji implementira igricu 4 u nizu sa dva igrača. Tabla za igru je dimenzije 8x8. Igrači na početku unose svoja imena, nakon čega računar nasumično dodeljuje crvenu i žutu boju igračima. Igrač sa crvenom bojom igra prvi i bira kolonu u koju ce da spusti svoju lopticu. Cilj igre je da se sakupe 4 loptice iste boje u liniji. Prvi igrač koji sakupi 4 loptice u liniji je pobedio. https://en.wikipedia.org/wiki/Connect_Four

1.2 Datoteke, niske, JSON format, datum

1.2.1 Uvodni primeri

Zadatak 1.11 Funkcije za rad sa niskama.

```
# Niske
#
# Mozemo ih pisati izmedju jednostrukih i dvostrukih navodnika

niska1 = 'Ovo je neka niska.'
niska2 = "People are strange when you're a stranger ."

print "\n------Niske------\n"
print niska1
print niska2

# Karakterima u niski mozemo pristupati koristeci notaciju [] kao kod listi
print "\n------Pristupanje karakterima u niski------\n"
print niska2[4]
print niska2[6:10]

# Duzinu niske racunamo koristeci funkciju len(niska)
print "\n------Duzina niske------\n"
print len(niska1)

# Funkcija count
# niska.count(podniska [, pocetak [, kraj]]) - vraca broj koliko se puta
# podniska nalazi u niski (u intervalu od pocetak do kraj)
print "\n------Funkcija <<count>>------\n"
print niska2.count("strange")

# Funkcija find
# niska.find(podniska [, pocetak [, kraj]]) - vraca poziciju prvog pojavljivanja
# podniska u niski (u intervalu od pocetak do kraj), -1 ukoliko se podniska ne nalazi
    u niski
print "\n------Funkcija <<find>>------\n"
print niska2.find("are")

# Funkcija join
# niska_separator.join([niska1,niska2,niska3,...]) - spaja listu niski separatorom
print "\n------Funkcija <<join>>------\n"
print ' '.join(["Olovka",'pise','srcem.'])


# Korisne funkcije za rad sa niskama:
#
# niska.isalnum()
#       isalpha()
#       isdigit()
#       islower()
```

```
#       isspace()
#       isupper()
# niska.split(separator) - razlaze nisku u listu koristeci separator
# niska.replace(stara, nova [, n]) - zamenjuje svako pojavljivanje niske stara
# niskom nova (ukoliko je zadat broj n, onda zamenjuje najvise n pojavljivanja)
```

Zadatak 1.12 Datoteke.

```
# Datoteke
#
# Datoteku otvaramo koristeci funkciju
#
# open(ime_datoteke, mod)
#
# mod: "r" -> read, "w" -> write, "a" -> append, "r+" -> read + append
#
# Datoteku zatvaramo koristeci funkciju
#
# datoteka.close()

f = open("dat1.txt","r")

# f.read(n) cita n karaktera iz datoteke
print "\n-----Funkcija <<read>>-----\n"
while True:
    c = f.read(2)
    if c == '':
        break
    print c

# f.readline() cita jednu liniju iz Datoteke
f.close()

g = open("dat2.txt","r")

# Liniju po liniju mozemo ucitavati koristeci petlju
# tako sto 'iteriramo' kroz Datoteku
print "-----Iteriranje kroz datoteku <<for>> petljom-----\n"
for linija in g:
    print linija

g.close()
# f.readlines() i list(f)
# vracaju listu linija datoteke
#
# f.write(niska) upisuje nisku u datoteku
print "-----Upisivanje u datoteku-----\n"
h = open("dat3.txt","r+")
h.write("water\n")

print h.readlines()

h.close()
```

Zadatak 1.13 Modul datetime.

```
# Datumi

# Ukljucujemo klasu datetime iz modula datetime

from datetime import datetime

# Nov objekat datuma:
#
# datetime.datetime(godina, mesec, dan [, sat [, minut [, sekund]]])
#
# Korisne funkcije:
#
# datetime.now() - vraca trenutno vreme odnosno datum
# datetime.strptime(datum_niska, format)
# datetime.year, datetime.month, datetime.day, datetime.hour, datetime.minute,
    datetime.second,
```

```
# datetime.strftime(format) - vraca string reprezentaciju objekta datuma na osnovu
    zadatog formata
# datetime.strptime(niska, format) - vraca objekat datetime konstruisan na osnovu
    niske u zadatom formatu
# datetime.time([sat [, minut [, sekund]]]) - vraca objekat koji predstavlja vreme
# datetime.date(dan, mesec, godina) - vraca objekat datuma
# format:
#   %A - dan u nedelji (Monday, Tuesday,...)
#   %w - dan u nedelji (0, 1, 2,..., 6)
#   $d - dan (01, 02, 03,...)
#   %B - mesec (January, February,...)
#   %m - mesec (01, 02, ...)
#   %Y - godina (1992, 1993,...)
#   %H - sat (00, 01, ..., 23)
#   %M - minut (00, 01, ..., 59)
#   %S - sekund (00, 01, ..., 59)
#



print "\n-----Datumi-----\n"
print datetime.now().strftime("Dan u nedelji: %a/%w, Dan: %d, Mesec: %b/%m, Godina: %
    y, Vreme: %H:%M:%S\n")
print datetime.now().time()
print datetime.now().date()
```

Zadatak 1.14 JSON format.

```
# JSON format
#
# Funkcije za rad sa JSON formatom se nalaze u modulu json
import json

# json.dumps(objekat) vraca string koji sadrzi JSON reprezentaciju objekta x

print "\n-----JSON reprezentacija objekta-----\n"
junak = {"Ime":"Dusko", "Prezime":"Dugousko", "Godine":11}
print json.dumps(junak)

# json.dump(x,f) upisuje string sa JSON reprezentacijom objekta x u datoteku f

f = open("dat4.json","w")
json.dump(junak,f)
f.close()

# json.load(f) ucitava iz datoteke string koji sadrzi JSON format objekta i vraca
    objekat
print "\n-----Ucitavanje objekta iz datoteke-----\n"
f = open("dat4.json","r")
x = json.load(f)
print x['Ime']
print x['Prezime']
print x['Godine']
f.close()
```

1.2.2 Zadaci za samostalni rad sa rešenjima

Zadatak 1.15 Napisati program koji sa standardnog ulaza učitava ime datoteke i broj n i računa broj pojavljivanja svakog n-grama u datoteci koji su sačinjeni od proizvoljnih karaktera i rezultat upisuje u datoteku `rezultat.json`.

Na primer:

Listing 1.1: *dat.txt*

```
Ovo je datoteka dat
```

Listing 1.2: *rezultat.json*

```
{
'a ': 1, 'ka': 1, 'ot': 1, 'ek': 1,
' d': 2, ' j': 1, 'da': 2, 'e ': 1,
'o ': 1, 'to': 1, 'at': 2, 'je': 1,
'Ov': 1, 'te': 1, 'vo': 1
}
```

[Rešenje 1.15]

Zadatak 1.16
U datoteci `korpa.json` se nalazi spisak kupljenog voća u json formatu:

```
[ { 'ime' : ime_voca, 'kolicina' : broj_kilograma } , ...]
```

U datotekama `maxi_cene.json`, `idea_cene.json`, `shopngo_cene.json` se nalaze cene voća u json formatu:

```
[ { 'ime' : ime_voca, 'cena' : cena_po_kilogramu } , ...]
```

Napisati program koji izračunava ukupan račun korpe u svakoj prodavnici i ispisuje cene na standardni izlaz.

[Rešenje 1.16]

1.2.3 Zadaci za vežbu

Zadatak 1.17 Napisati program koji iz datoteke `ispiti.json` učitava podatke o ispitima i njihovim datumima. Ispisati na standardni izlaz za svaki ispit njegovo ime i status "Prosao" ukoliko je ispit prosao, odnosno "Ostalo je jos n dana.", gde je n broj dana od trenutnog datuma do datuma ispita.

Listing 1.3: *ispiti.json*

```
[  {'ime': 'Relacione baze podataka',
    'datum': '21.09.2016.'},
    {'ime': 'Vestacka inteligencija',
    'datum': '17.06.2017.'},
    {'ime': 'Linearna algebra i analiticka geometrija',
    'datum': '08.02.2017.'} ]
```

Zadatak 1.18 Napisati program koji izdvaja sve jednolinijske i višelinijske komentare iz .c datoteke čije ime se unosi sa standardnog ulaza, listu jednih i drugih komentara upisuje u datoteku `komentari.json`. Jednolinijski komentari se navode nakon // a višelinijski između /∗ i ∗/.

Listing 1.4: *program.c*

```
#include <stdio.h>

// Primer jednolinijskog komentara

int main(){
/*
  Na ovaj nacin ispisujemo tekst
  na standardni izlaz koristeci jezik C.
*/
  printf("Hello world!");

  // Na ovaj nacin se ispisuje novi red
  printf("\n");
/*
  Ukoliko se funkcija uspesno zavrsila
  vracamo 0 kao njen rezultat.
```

```
*/
  return 0;
}
```

Listing 1.5: *komentari.json*

```
{
  'jednolinijski' : ['Primer jednolinijskog komentara',
                     'Na ovaj nacin se ispisuje novi red'],
  'viselinijski' : ['Na ovaj nacin ispisujemo tekst na standardni
                     izlaz koristeci jezik C.',
                     'Ukoliko se funkcija uspesno zavrsila
                     vracamo 0 kao njen rezultat.']
}
```

Zadatak 1.19 Napisati program upoređuje dve datoteke čija imena se unose sa standardnog ulaza. Rezultat upoređivanja je datoteka `razlike.json` koja sadrži broj linija iz prve datoteke koje se ne nalaze u drugoj datoteci i obratno. *Napomena* Obratiti pažnju na efikasnost.

Listing 1.6: *dat1.txt*

```
//netacno

same=1;

for(i=0;s1[i]!='\0' && s2[i]!='\0';i++) {
 if(s1[i]!=s2[i]) {
  same=0;
  break;
 }
}
return same;
```

Listing 1.7: *dat2.txt*

```
 //tacno

 for(i=0;s1[i]!='\0' && s2[i]!='\0';i++){

 if(s1[i]!=s2[i])
   return 0;
}
 return s1[i]==s2[i];
```

Listing 1.8: *razlike.json*

```
{
  'dat1.txt' : 7,
  'dat2.txt' : 4
}
```

1.3 Argumenti komandne linije, sortiranje, obilazak direktorijuma

1.3.1 Uvodni primeri

Zadatak 1.20 Modul sys i argumenti komandne linije

```
# modul sys ima definisan objekat argv koji predstavlja listu argumenata komandne
    linije (svi argumenti se cuvaju kao niske karaktera)

```

```
import sys

if len(sys.argv)==1:
  print "Niste naveli argumente komandne linije"
  # funkcija exit() iz modula sys prekida program
  # (ukoliko se ne prosledi argument, podrazumevano
  # se salje None objekat)
  exit()

# ispisujemo argumente komandne linije
# prvi argument, tj. sys.argv[0] je uvek ime skript fajla koji se pokrece
for item in sys.argv:
  print item

# korisnik moze da zada ime datoteke kao prvi argument komandne linije
# u tom slucaju datoteku otvaramo sa f = open(sys.argv[1], "r")
```

Zadatak 1.21 Modul os

```
# Prolazak kroz direktorijume

import os
# ispisuje oznaku za tekuci direktorijum
print os.curdir
# ispisuje oznaku za roditeljski direktorijum tekuceg direktorijuma
print os.pardir
# ispisuje separator koji koristi za pravljenje putanja
print os.sep


# funkcija za prosledjenu putanju direktorijuma vraca listu imena svih fajlova u tom
    direktorijumu, . je zamena za putanju tekuceg direktorijuma

print os.listdir(".")

#
# os.walk() - vraca listu torki (trenutni_direktorijum, poddirektorijumi, datoteke)
# os.path.join(putanja, ime) - pravi putanju tako sto nadovezuje na prosledjenu
    putanju zadato ime odvojeno /

print "\n-----Prolazak kroz zadati direktorijum-----\n"
for (trenutni_dir, poddirektorijumi, datoteke) in os.walk("."):
    print trenutni_dir
    for datoteka in datoteke:
        print os.path.join(trenutni_dir, datoteka)

# os.path.abspath(path) - vraca apsolutnu putanju za zadatu relativnu putanju nekog
    fajla
# os.path.isdir(path) - vraca True ako je path putanja direktorijuma, inace vraca
    False
# os.path.isfile(path) - vraca True ako je path putanja regularnog fajla, inace vraca
     False

print "\n-----Regularni fajlovi zadatog direktorijuma-----\n"
for ime in os.listdir("."):
   # ako je regularan fajl u pitanju ispisujemo njegovu apsolutnu putanju
  if os.path.isfile(os.path.join(".", ime)):
    print os.path.abspath(os.path.join(".", ime))
```

Zadatak 1.22 Sortiranje

```
# Sortiranje
#
# sorted(kolekcija [, poredi [, kljuc [, obrni]]]) - vraca sortiranu kolekciju
#
# kolekcija - kolekcija koju zelimo da sortiramo
# poredi - funkcija poredjenja
# kljuc - funkcija koja vraca kljuc po kome se poredi
# obrni - True/False (opadajuce/rastuce)
#
# za poziv sorted(kolekcija) koristi se funkcija cmp za poredjenje
# cmp(x, y) -> integer
```

```
# vraca negativnu vrednost za x<y, 0 za x==y, pozitivnu vrednost za x>y
# ako su x i y niske, cmp ih leksikografski poredi
#

import json
import math

l = ["A", "C", "D", "5", "1", "3"]
print l
print "sortirana lista: ", sorted(l)

# u sledecem primeru je neophodno da definisemo svoje funkcije za poredjenje i
    vracanje kljuca jer je kolekcija lista recnika i za to cmp nema definisano
    ponasanje
tacke = [{"teme":"A" , "koordinate": [10.0, 1.1]}, {"teme":"B" , "koordinate": [1.0,
    15.0]}, {"teme":"C" , "koordinate": [-1.0, 5.0]}]


# funkcija koja tacke x i y poredi po njihovoj udaljenosti od koordinatnog pocetka
def poredi(x,y):
    if (x[0]*x[0] + x[1]*x[1]) > (y[0]*y[0] + y[1]*y[1]):
        return 1
    else:
        return -1
# funkcija kljuc kao argument ima element kolekcije koja se poredi, u ovom slucaju je
     to jedan recnik
# povratna vrednost funkcije kljuc je u stvari tip argumenata funkcije poredi
def kljuc(x):
    return x["koordinate"]

sortirane_tacke = sorted(tacke, poredi, kljuc) # ili sorted(tacke, poredi, kljuc,
    True) ako zelimo opadajuce da se sortira
print "Tacke pre sortiranja:"
for item in tacke:
  print item["teme"],
print "\nTacke nakon sortiranja: "
for item in sortirane_tacke:
  print item["teme"],
print
```

1.3.2 Zadaci za samostalni rad sa rešenjima

Zadatak 1.23 Napisati program koji računa odnos kardinalnosti skupova duže i šire za zadati direktorijum. Datoteka pripada skupu duže ukoliko ima više redova od maksimalnog broja karaktera po redu, u suprotnom pripada skupu šire. Sa standardnog ulaza se unosi putanja do direktorijuma. Potrebno je obići sve datoteke u zadatom direktorijumu i njegovim poddirektorijumima (koristiti funkciju os.walk()) i ispisati odnos kardinalnosti skupova duže i šire.

[Rešenje 1.23]

Zadatak 1.24 Napisati program koji obilazi direktorijume rekurzivno i računa broj datoteka za sve postojeće ekstenzije u tim direktorijumima. Sa standardnog ulaza se unosi putanja do početnog direktorijuma, a rezultat se ispisuje u datoteku `rezultat.json`. Na primer:

Listing 1.9: *rezultat.txt*

```
{
'txt' : 14,
'py' : 12,
 'c' : 10
}

```

[Rešenje 1.24]

Zadatak 1.25 U datoteci `radnici.json` nalaze se podaci o radnom vremenu zaposlenih u preduzeću u sledecem formatu:

```
[  { 'ime' : 'Pera Peric',
   'odmor' : ['21.08.2016.', '31.08.2016.'],
   'radno_vreme' : ['08:30', '15:30'] }, ...]
```

Napisati program koji u zavisnosti od unete opcije poslodavcu ispisuje trenutno dostupne radnike odnosno radnike koji su na odmoru. Moguće opcije su 'd' - trenutno dostupni radnici i 'o' - radnici koji su na odmoru. Radnik je dostupan ukoliko nije na odmoru i trenutno vreme je u okviru njegovog radnog vremena.

[Rešenje 1.25]

Zadatak 1.26 Napisati program koji učitava ime datoteke sa standardnog ulaza i na standardni izlaz ispisuje putanje do svih direktorijuma u kojima se nalazi ta datoteka.

[Rešenje 1.26]

1.3.3 Zadaci za vežbu

Zadatak 1.27 Napisati program koji ispisuje na standardni izlaz putanje do lokacija svih Apache virtuelnih hostova na računaru. Smatrati da je neki direktorijum lokacija Apache virtuelnog hosta ukoliko u sebi sadrži `index.html` ili `index.php` datoteku.

Zadatak 1.28 Napisati program koji realizuje autocomplete funkcionalnost. Sa standardnog ulaza korisnik unosi delove reči sve dok ne unese karakter !. Nakon svakog unetog dela reči ispisuju se reči koje počinju tim karakterima. Spisak reči koje program može da predloži se nalazi u datoteci `reci.txt`.

1.4 Rešenja

Rešenje 1.6 Pogodi broj

```
# Pogodi broj

import random

print "-------- IGRA: Pogodi broj ---------\n"

zamisljen_broj = random.randint(0,100)

ime = raw_input("Unesite Vase ime: ")

print "Zdravo {0:s}. :) \nZamislio sam neki broj od 1 do 100. Da li mozes da pogodis
    koji je to broj? ".format(ime)

pogodio = 0;
while not pogodio:
     print "Unesi broj:"
     broj = int(raw_input())
     if broj == zamisljen_broj:
         pogodio = 1
     elif broj > zamisljen_broj:
         print "Broj koji sam zamislio je MANJI od {0:d}.".format(broj)
     else:
         print "Broj koji sam zamislio je VECI od {0:d}.".format(broj)

print "BRAVO!!! Pogodio si! Zamislio sam {0:d}. Bilo je lepo igrati se sa tobom. :)".
    format(zamisljen_broj)
```

Rešenje 1.7 Aproksimacija broja PI metodom Monte Karlo

```
# Aproksimacija broja PI metodom Monte Karlo


import random

print "Izracunavanje broja PI metodom Monte Karlo \n"
N = int(raw_input("Unesite broj iteracija: "))
# Broj tacaka koje se nalaze u krugu
A = 0
# Broj tacaka koje se nalaze u kvadratu
B = 0

i = N
while i >= 0:
    tacka = (random.random(), random.random())
    print "Tacka: "
    print tacka
    # Ukoliko se tacka nalazi u krugu, povecavamo broj tacaka u krugu
    if ((float(tacka[0])-0.5)**2 + (float(tacka[1])-0.5)**2) < (0.5**2):
        A = A + 1
    B = B + 1
    i = i - 1

print "Broj PI aproksimiran metodom Monte Karlo: "
print 4.0*A/B
```

Rešenje 1.8 X-O

```
# X-O
#
#  - | O | X
# --- --- ---
#  X | - | -
# --- --- ---
#  - | X | O

import random

def ispisi_tablu(tabla):
    print "\n    TABLA \n"
    print "    1   2   3   "
    print "   --- --- ---  "
    indeks = 1
    for i in tabla:
        print indeks,"|",i[0],"|",i[1],"|",i[2],"|"
        print "   --- --- --- "
        indeks = indeks + 1
    print "\n"

def pobedio(tabla):
    if (tabla[0][0] != "-" and tabla[0][2] != "-") and ((tabla[0][0] == tabla[1][1]
    == tabla[2][2]) or (tabla[0][2] == tabla[1][1] == tabla[2][0])):
        return True
    for i in range(3):
        if (tabla[0][i] != "-" and tabla[i][0] != "-") and ((tabla[0][i] == tabla[1][
    i] == tabla[2][i]) or (tabla[i][0] == tabla[i][1] == tabla[i][2])):
            return True
    return False

def ucitaj_koordinate(ime):
    while True:
        print "{0:s} unesite koordinate polja koje zelite da popunite u posebnim
    linijama:\n".format(ime)
        x = int(raw_input("Unesite vrstu: "))
        y = int(raw_input("Unesite kolonu: "))
        if 1<=x<=3 and 1<=y<=3:
            return x-1,y-1
        else:
            "Morate uneti brojeve 1,2 ili 3\n"

def korak(igrac):
```

```
    while True:
        x,y = ucitaj_koordinate(igrac[0])
        if tabla[x][y] == "-":
            tabla[x][y] = igrac[1]
            ispisi_tablu(tabla)
            break
        else:
            print tabla[x][y]
            print "Uneto polje je popunjeno!\n"

print "IGRA: X-O pocinje\n"

ime1 = raw_input("Unesite ime prvog igraca: ")
print "Zdravo {0:s}!\n".format(ime1)
ime2 = raw_input("Unesite ime drugog igraca: ")
print "Zdravo {0:s}!\n".format(ime2)

indikator = random.randint(1,2)
if indikator == 1:
    prvi_igrac = (ime1, "X")
    drugi_igrac = (ime2, "O")
else:
    prvi_igrac = (ime2, "X")
    drugi_igrac = (ime1, "O")

print "Igrac {0:s} igra prvi. \n".format(prvi_igrac)
print "X : {0:s}\n".format(prvi_igrac)
print "O : {0:s}\n".format(drugi_igrac)

tabla = [['-','-','-'],['-','-','-'],['-','-','-']]

print "Zapocnimo igru \n"

ispisi_tablu(tabla)

na_redu = 0
iteracija = 0
igraci = [prvi_igrac, drugi_igrac]
while iteracija < 9:
    korak(igraci[na_redu])
    if pobedio(tabla) == True:
        print "BRAVO!!!!!!! {0:s} je pobedio!\n".format(igraci[na_redu][0])
        break
    na_redu = (na_redu+1)%2
    iteracija = iteracija + 1

if iteracija == 9:
    print "NERESENO! Pokusajte ponovo.\n"
```

Rešenje 1.15

```
# dat.txt:
# Ovo je datoteka dat
#
# rezultat.json:
#
# {"a ": 1, "ka": 1, "ot": 1, "ek": 1, " d": 2, " j": 1, "da": 2, "e ": 1, "o ": 1, "
    to": 1, "at": 2, "je": 1, "Ov": 1, "te": 1, "vo": 1}

import json

ime_datoteke = raw_input("Unesite ime datoteke: ")
n = int(raw_input("Unesite broj n: "))

# Otvaramo datoteku i citamo njen sadrzaj
f = open(ime_datoteke, "r")
sadrzaj = f.read()
f.close()

recnik = {}
i = 0
# Prolazimo kroz sadrzaj i uzimamo jedan po jedan n-gram
```

```
while i < len(sadrzaj) - n:
    ngram = sadrzaj[i : i+n]
    # Ukoliko se n-gram vec nalazi u recniku,
    #  povecavamo mu broj pojavljivanja
    if ngram in recnik:
        recnik[ngram] = recnik[ngram]+1
    # Dodajemo n-gram u recnik i postavljamo mu broj na 1
    else:
        recnik[ngram] = 1
    i = i + 1

f = open("rezultat.json", "w")
json.dump(recnik,f)
f.close()
```

Rešenje 1.16

```
import json

# Ucitavamo podatke iz datoteka
f = open('korpa.json', "r")
korpa = json.load(f)
f.close()

f = open('maxi_cene.json', "r")
maxi_cene = json.load(f)
f.close()

f = open('idea_cene.json', "r")
idea_cene = json.load(f)
f.close()

f = open('shopngo_cene.json', "r")
shopngo_cene = json.load(f)
f.close()

maxi_racun = 0
idea_racun = 0
shopngo_racun = 0
# Za svako voce u korpi dodajemo njegovu cenu u svaki racun posebno
for voce in korpa:
    ime = voce['ime']
    maxi_racun = maxi_racun + korpa[ime]['kolicina']*maxi_cene[ime]['cena']
    idea_racun = idea_racun + korpa[ime]['kolicina']*idea_cene[ime]['cena']
    shopngo_racun = shopngo_racun + korpa[ime]['kolicina']*shopngo_cene[ime]['cena']

print "Maxi: " + str(maxi_racun) + " dinara"
print "Idea: " + str(idea_racun) + " dinara"
print "Shopngo: " + str(shopngo_racun) + " dinara"
```

Rešenje 1.23

```

import os

dat_u_duze = 0
dat_u_sire = 0

# Funkcija koja obilazi datoteku i vraca 1 ukoliko datoteka pripada skupu duze
# odnosno 0 ukoliko datoteka pripada skupu sire
def obilazak(ime_datoteke):
    br_linija = 0
    najduza_linija = 0
    f = open(ime_datoteke, "r")
    for linija in f:
        br_linija = br_linija + 1
        if len(linija) > najduza_linija:
            najduza_linija = len(linija)
    f.close()
    if br_linija > najduza_linija:
```

```
        return 1
    else:
        return 0

ime_direktorijuma = raw_input("Unesite putanju do direktorijuma: ")

for (tren_dir, pod_dir, datoteke ) in os.walk(ime_direktorijuma):
    for dat in datoteke:
        if obilazak(os.path.join(tren_dir, dat)) == 0:
            dat_u_sire += 1
        else:
            dat_u_duze += 1

print "Kardinalnost skupa duze: kardinalnost skupa sire"
print str(dat_u_duze)+":"+str(dat_u_sire)
```

Rešenje 1.24

```

import os
import json

ime_direktorijuma = raw_input("Unesite putanju do direktorijuma: ")

ekstenzije = {}

for (tren_dir, pod_dir, datoteke ) in os.walk(ime_direktorijuma):
    for dat in datoteke:
        pozicija = dat.find(".")
        # Ukoliko datoteka ima ekstenziju, pretpostavljamo da su datoteke imenovane
    tako da posle . ide ekstenzija u ispravnom obliku
        if pozicija >= 0:
            # Ukoliko ekstenzija postoji u mapi, povecavamo njen broj
            if dat[pozicija:] in ekstenzije:
                ekstenzije[dat[pozicija:]] += 1
            else:
            # Dodajemo novu ekstentiju u mapu i postavljamo njen broj na 1
                ekstenzije[dat[pozicija:]] = 1

f = open("rezultat.json","w")
json.dump(ekstenzije, f)
f.close()
```

Rešenje 1.25

```
import json, os
from datetime import datetime

f = open("radnici.json", "r")
radnici = json.load(f)
f.close()

opcija = raw_input("Unesite opciju koju zelite (d - dostupni radnici, o - radnici na
    odmoru): \n")

if opcija != "d" and opcija != "o":
    print "Uneta opcija nije podrzana."
    exit()

tren_dat = datetime.now()

# funkcija datetime.strptime(string, format) pravi objekat tipa datetime na osnovu
    zadatih podataka u stringu i odgovarajuceg formata, na primer ako je datum
    zapisan kao "21.08.2016" odgovarajuci format je "%d.%m.%Y." pa se funkcija poziva
     sa datetime.strptime("21.08.2016", "%d.%m.%Y.")

for radnik in radnici:
    kraj_odmora = datetime.strptime(radnik['odmor'][1],"%d.%m.%Y.").date()
    pocetak_odmora = datetime.strptime(radnik['odmor'][0],"%d.%m.%Y.").date()
    kraj_rad_vrem = datetime.strptime(radnik['radno_vreme'][1],"%H:%M").time()
```

```
    pocetak_rad_vrem = datetime.strptime(radnik['radno_vreme'][0],"%H:%M").time()
    if opcija == "o":
        # Ukoliko je radnik trenutno na odmoru ispisujemo ga
        if  pocetak_odmora < tren_dat.date() < kraj_odmora:
            print radnik["ime"]
    else:
        # Ukoliko je radnik trenutno dostupan i nije na odmoru, ispisujemo ga
        if not (pocetak_odmora < tren_dat.date() < kraj_odmora) and pocetak_rad_vrem
    < tren_dat.time() < kraj_rad_vrem:
            print radnik["ime"]
```

Rešenje 1.26

```
import os

ime_datoteke = raw_input("Unesite ime datoteke: ")

# pretrazujemo ceo fajl sistem, odnosno pretragu krecemo od root direktorijuma /
# imajte u vidu da ce vreme izvrsavanja ovog programa biti veliko posto se pretrazuje
    ceo fajl sistem, mozete ga prekinuti u svakom trenutku sa CTRL+C
for (tren_dir, pod_dir, datoteke ) in os.walk("/"):
  # objekat datoteke predstavlja listu imena datoteka iz direktorijuma
  # ta imena poredimo sa zadatim
  for dat in datoteke:
    # ako smo naisli na trazenu datoteku, pravimo odgovarajucu putanju
    if dat == ime_datoteke:
      print os.path.join(os.path.abspath(tren_dir), ime_datoteke)
```

2

Programiranje ograničenja - Python

Potrebno je imati instaliran Python 2.7 i biblioteku python-constraint. Na Ubuntu 14.04 operativnom sistemu, biblioteka python-constraint se moze instalirati pomocu Pip alata:

```
sudo apt-get -y install python-pip
sudo pip install python-constraint
```

Korisni linkovi i literatura:

http://labix.org/doc/constraint/
https://pypi.python.org/pypi/python-constraint
http://www.hakank.org/constraint_programming_blog/

2.1 Programiranje ograničenja

2.1.1 Uvodni primeri

Zadatak 2.1 Modul constraint, osnovne funkcije

```
# Programiranje ogranicenja

# Ukljucujemo modul za rad sa ogranicenjima
import constraint

# Definisemo problem
problem = constraint.Problem()
# Dodajemo promenljive
#
# problem.addVariable(ime_promenljive, domen_lista)
# problem.addVariables(lista_imena_promenljivih, domen_lista)
problem.addVariable('x',[1,2,3])
problem.addVariable('y',['a','b','c'])
# Ispisujemo resenja
# print problem.getSolutions()

problem.addVariable('z',[0.1,0.2,0.3])
# Dodajemo ogranicenja
#
# problem.addConstraint(ogranicenje [, redosled_promenljivih])
#
# ogranicenje moze biti:
#   constraint.AllDifferentConstraint() - razlicite vrednosti svih promenljivih
#   constraint.AllEqualConstraint() - iste vrednosti svih promenljivih
#   constraint.MaxSumConstraint(s [,tezine]) - suma vrednosti promenljivih (pomnozena
     sa tezinama) ne prelazi s
#   constraint.MinSumConstraint(s [,tezine]) - suma vrednosti promenljivih (pomnozena
     sa tezinama) nije manja od s
#   constraint.ExactSumConstraint(s [,tezine]) - suma vrednosti promenljivih (
    pomnozena sa tezinama) je s
#   constraint.InSetConstraint(skup) - vrednosti promenljivih se nalaze u skupu skup
```

```
#   constraint.NotInSetConstraint(skup) - vrednosti promenljivih se ne nalaze u skupu
     skup
#   constraint.SomeInSetConstraint(skup) - vrednosti nekih promenljivih se nalaze u
    skupu skup
#   constraint.SomeNotInSetConstraint(skup) - vrednosti nekih promenljivih se ne
    nalaze u skupu skup
#
# redosled_promenljivih predstavlja listu promenljivih
# i zadaje se zbog definisanja tacnog redosleda
# ogranicenja koja se primenjuju na promenljive
#
# Mozemo napraviti i svoju funkciju ogranicenja
def ogranicenje(x,z):
    if x / 10.0 == z:
        return True

# Prosledjujemo funkciju ogranicenja i redosled promenljivih koji treba da odgovara
    redosledu argumenata funkcije ogranicenja
problem.addConstraint(ogranicenje,['x','z'])
resenja = problem.getSolutions()
print "\n-----Resenja-----\n"
for resenje in resenja:
    print resenje
```

2.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 2.2 Napisati program koji pronalazi trocifren broj `ABC` tako da je količnik `ABC / (A + B + C)` minimalan i `A`, `B` i `C` su različiti brojevi.

[Rešenje 2.2]

Zadatak 2.3 Dati su novčići od 1, 2, 5, 10, 20 dinara. Napisati program koji pronalazi sve moguće kombinacije tako da zbir svih novčića bude 50.

[Rešenje 2.3]

Zadatak 2.4 Napisati program koji ređa brojeve u magičan kvadrat. Magičan kvadrat je kvadrat dimenzija 3x3 takav da je suma svih brojeva u svakom redu, svakoj koloni i svakoj dijagonali jednak 15 i svi brojevi različiti. Na primer:

```
4 9 2
3 5 7
8 1 6
```

[Rešenje 2.4]

Zadatak 2.5 Napisati program koji pronalazi sve vrednosti promenljivih X, Y i Z za koje važi da je $X >= Z$ i $X * 2 + Y * X + Z <= 34$ pri čemu promenljive pripadaju narednim domenima $X \in \{1, 2, ..., 90\}$, $Y \in \{2, 4, 6, ...60\}$ i $Z \in \{1, 4, 9, 16, ..., 100\}$

[Rešenje 2.5]

Zadatak 2.6 Napisati program koji dodeljuje različite vrednosti različitim karakterima tako da suma bude zadovoljena:

```
  TWO
 +TWO
 ---------
 FOUR
```

[Rešenje 2.6]

Zadatak 2.7 Napisati program koji pronalazi sve vrednosti promenljivih X, Y, Z i W za koje važi da je $X >= 2 * W$, $3 + Y <= Z$ i $X - 11 * W + Y + 11 * Z <= 100$ pri čemu promenljive pripadaju narednim domenima $X \in \{1, 2, ..., 10\}$, $Y \in \{1, 3, 5, ...51\}$, $Z \in \{10, 20, 30, ..., 100\}$ i $W \in \{1, 8, 27, ..., 1000\}$.

[Rešenje 2.7]

Zadatak 2.8 Napisati program koji raspoređuje brojeve 1-9 u dve linije koje se seku u jednom broju. Svaka linija sadrži 5 brojeva takvih da je njihova suma u obe linije 25 i brojevi su u rastućem redosledu.

```
1   3
 2 4
  5
 6 8
7   9
```

[Rešenje 2.8]

Zadatak 2.9 Pekara *Kiflica* proizvodi hleb i kifle. Za mešenje hleba potrebno je 10 minuta, dok je za kiflu potrebno 12 minuta. Vreme potrebno za pečenje ćemo zanemariti. Testo za hleb sadrži 300g brašna, a testo za kiflu sadrži 120g brašna. Zarada koja se ostvari prilikom prodaje jednog hleba je 7 dinara, a prilikom prodaje jedne kifle je 9 dinara. Ukoliko pekara ima 20 radnih sati za mešenje peciva i 20kg brašna, koliko komada hleba i kifli treba da se umesi kako bi se ostvarila maksimalna zarada (pod pretpostavkom da će pekara sve prodati)?

[Rešenje 2.9]

Zadatak 2.10 Napisati program pronalazi vrednosti A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S (svako slovo predstavlja različit broj) koje su poređane u heksagon na sledeći način:

```
  A,B,C
 D,E,F,G
H,I,J,K,L
 M,N,O,P
  Q,R,S
```

tako da zbir vrednosti duž svake horizontalne i dijagonalne linije bude 38 (A+B+C = D+E+F+G = ... = Q+R+S = 38, A+D+H = B+E+I+M = ... = L+P+S = 38, C+G+L = B+F+K+P = ... = H+M+Q = 38).

[Rešenje 2.10]

Zadatak 2.11 Kompanija Start ima 250 zaposlenih radnika. Rukovodstvo kompanije je odlučilo da svojim radnicima obezbedi dodatnu edukaciju. Da bi se radnik obučio programskom jeziku Elixir potrebno je platiti 100 evra po osobi za kurs, ali bi njegovo produktivno znanje ovog programskog jezika donelo 150 projekat/sati mesečno, što bi za kompaniju značilo dobit od 5 evra po projekat/satu. Da bi se radnik obučio programskom jeziku Dart potrebno je platiti 105 evra po osobi za kurs, ali bi njegovo produktivno znanje ovog programskog jezika donelo 170 projekat/sati mesečno, koji bi za kompaniju značili dobit od 6 evra po satu. Ukoliko Start ima na raspolaganju 26000 evra za obuku i maksimalan broj 51200 mogućih projekat/sati mesečno, odrediti na koji nacin kompanija treba da obuči svoje zaposlene kako bi ostvarila maksimalnu dobit.

[Rešenje 2.11]

Zadatak 2.12 Napisati program koji raspoređuje 8 topova na šahovsku tablu tako da se nikoja dva topa ne napadaju.

[Rešenje 2.12]

Zadatak 2.13 Napisati program koji raspoređuje 8 dama na šahovsku tablu tako da se nikoje dve dame ne napadaju.

[Rešenje 2.13]

Zadatak 2.14 Napisati program koji učitava tablu za Sudoku iz datoteke čije ime se zadaje sa standardnog ulaza i korišćenjem ograničenja rešava Sudoku zagonetku.

[Rešenje 2.14]

2.1.3 Zadaci za vežbu

Zadatak 2.15 Za svaku narednu zagonetku, napisati program koji dodeljuje različite vrednosti različitim karakterima tako da suma bude zadovoljena:

```
GREEN + ORANGE = COLORS
MANET + MATISSE + MIRO + MONET + RENOIR = ARTISTS
COMPLEX + LAPLACE = CALCULUS
THIS + IS + VERY = EASY
CROSS + ROADS = DANGER
FATHER + MOTHER = PARENT
WE + WANT + NO + NEW + ATOMIC = WEAPON
EARTH + AIR + FIRE + WATER = NATURE
SATURN + URANUS + NEPTUNE + PLUTO = PLANETS
SEE + YOU = SOON
NO + GUN + NO = HUNT
WHEN + IN + ROME + BE + A = ROMAN
DONT + STOP + THE = DANCE
HERE + THEY + GO = AGAIN
OSAKA + HAIKU + SUSHI = JAPAN
MACHU + PICCHU = INDIAN
SHE + KNOWS + HOW + IT = WORKS
COPY + PASTE + SAVE = TOOLS
```

Zadatak 2.16 Za svaku narednu zagonetku, napisati program koji dodeljuje različite vrednosti različitim karakterima tako da suma bude zadovoljena:

```
THREE + THREE + ONE = SEVEN
NINE + LESS + TWO = SEVEN
ONE + THREE + FOUR = EIGHT
THREE + THREE + TWO + TWO + ONE = ELEVEN
SIX + SIX + SIX = NINE + NINE
SEVEN + SEVEN + SIX = TWENTY
ONE + ONE + ONE + THREE + THREE + ELEVEN = TWENTY
EIGHT + EIGHT + TWO + ONE + ONE = TWENTY
ELEVEN + NINE + FIVE + FIVE = THIRTY
NINE + SEVEN + SEVEN + SEVEN = THIRTY
TEN + SEVEN + SEVEN + SEVEN + FOUR + FOUR + ONE = FORTY
TEN + TEN + NINE + EIGHT + THREE = FORTY
FOURTEEN + TEN + TEN + SEVEN = FORTYONE
NINETEEN + THIRTEEN + THREE + TWO + TWO + ONE + ONE + ONE = FORTYTWO
FORTY + TEN + TEN = SIXTY
SIXTEEN + TWENTY + TWENTY + TEN + TWO + TWO = SEVENTY
SIXTEEN + TWELVE + TWELVE + TWELVE + NINE + NINE = SEVENTY
TWENTY + TWENTY + THIRTY = SEVENTY
FIFTY + EIGHT + EIGHT + TEN + TWO + TWO = EIGHTY
FIVE + FIVE + TEN + TEN + TEN + TEN + THIRTY = EIGHTY
SIXTY + EIGHT + THREE + NINE + TEN = NINETY
ONE + NINE + TWENTY + THIRTY + THIRTY = NINETY
```

Zadatak 2.17 Za svaku narednu zagonetku, napisati program koji dodeljuje različite vrednosti različitim karakterima tako da jednakost bude zadovoljena:

```
MEN * AND = WOMEN
COGITO = ERGO * SUM
((JE + PENSE) - DONC) + JE = SUIS
FERMAT * S = LAST + THEOREM.
WINNIE / THE = POOH
TWO * TWO + EIGHT = TWELVE
```

Zadatak 2.18 Uraditi sve zadatke koji su pobrojani ovde:
http://www.primepuzzle.com/leeslatest/alphameticpuzzles.html

Zadatak 2.19 Napisati program koji učitava ceo broj n i ispisuje magičnu sekvencu S brojeva od 0 do $n-1$. $S = (x_0, x_1, ..., x_{n-1})$ je magična sekvenca ukoliko postoji x_i pojavljivanja broja i za $i = 0, 1, ..., n-1$.

Zadatak 2.20 Čistačica Mica sređuje i čisti kuće i stanove. Da bi sredila i počistila jedan stan potrebno joj je 1 sat, dok joj je za kuću potrebno 1.5 sati. Prilikom čišćenja, Mica potroši neku količinu deterdženta, 120ml po stanu, odnosno 100ml po kući. Mica zaradi 1000 dinara po svakom stanu, odnosno 1500 dinara po kući. Ukoliko Mica radi 40 sati nedeljno i ima 5l deterdženta na raspolaganju, koliko stanova i kuća je potrebno da očisti kako bi imala najveću zaradu?

Zadatak 2.21 Marija se bavi grnčarstvom i pravi šolje i tanjire. Da bi se napravila šolja, potrebno je 6 minuta, dok je za tanjir potrebno 3 minuta. Pri pravljenju šolje potroši se 75 gr, dok se za tanjir potroši 100 gr gline. Ukoliko ima 20 sati na raspolaganju za izradu svih proizvoda i 250 kg gline, a zarada koju ostvari iznosi 2 evra po svakoj šolji i 1.5 evra po tanjiru, koliko šolja i tanjira treba da napravi kako bi ostvarila maksimalnu zaradu?

Zadatak 2.22 Jovanin komšija preprodaje računare i računarsku opremu. Očekuje isporuku računara i štampača. Pri tom, računari su spakovani tako da njihova kutija zauzima 360 kubnih decimetara prostora, dok se štampači pakuju u kutijama koje zauzimaju 240 kubnih decimetara prostora. Komšija se trudi da mesečno proda najmanje 30 računara i da taj broj bude bar za 50% veći od broja prodatih štampača. Računari koštaju 200 evra po nabavnoj ceni, a prodaju se po ceni od 400 evra, dok štampači koštaju u nabavci 60 evra i prodaju se za 140 evra. Magacin kojim komšija raspolaže ima svega 30000 kubnih decimetara prostora i mesečno može da nabavi robu u iznosu od najviše 14000 evra. Koliko računara, a koliko štampača komšija treba da proda kako bi se maksimalno obogatio?

2.2 Rešenja

Rešenje 2.2

```
1
2  import constraint
3
4  problem = constraint.Problem()
5  # Definisemo promenljive i njihove vrednosti
6  problem.addVariable('A',range(1,10))
7  problem.addVariable('B',range(10))
8  problem.addVariable('C',range(10))
9  # Dodajemo ogranicenje da su vrednosti svih promenljivih razlicite
10 problem.addConstraint(constraint.AllDifferentConstraint())
11 resenja = problem.getSolutions()
12 # Znamo da minimalni kolicnik mora biti manji od 999
13 min_kolicnik = 999
14 min_resenje = {}
15 for resenje in resenja:
16     a = resenje['A']
17     b = resenje['B']
18     c = resenje['C']
19     kolicnik = (float)(a*100 + b*10 + c) / (a+b+c)
20     if kolicnik < min_kolicnik:
21         min_kolicnik = kolicnik
22         min_resenje = resenje
```

```
print min_resenje['A']*100 + min_resenje['B']*10 + min_resenje['C']
```

Rešenje 2.3

```
import constraint

problem = constraint.Problem()
# Definisemo promenljive za svaki novcic
# ako bi se zahtevalo da u kombinaciji bude od svake vrednosti bar po jedan novcic
    samo treba promeniti da domen za svaku promenljivu krece od 1
problem.addVariable("1 din",range(0,51))
problem.addVariable("2 din",range(0,26))
problem.addVariable("5 din",range(0,11))
problem.addVariable("10 din",range(0,6))
problem.addVariable("20 din",range(0,3))

# Problem koji je uocen pri ispisu resenja je sledeci, redosled u kom ce biti dodate
    promenljive problemu ne mora uvek da odgovara redosledu kojim smo mi definisali
    promenljive, u konkretnom primeru (videti oblik u kom ispisuje resenje),
    promenljive ce se dodati u sledecem redosledu:'1 din', '2 din', '10 din', '20 din
    ', '5 din' (nacin na koji se kljucevi organizuju u recniku nije striktno
    definisan, primetimo da niske nisu sortirane)
# posledica je da postavljanje ogranicenja
# problem.addConstraint(constraint.ExactSumConstraint(50,[1,2,5,10,20]))
# nece ispravno dodeliti tezine, na primer, tezinu 5 dodeli promenljivoj '10 din'
    umesto '5 din' kako bismo ocekivali

# I nacin da se resi ovaj problem je da redosled promenljivih koji odgovara redosledu
     tezina za ExactSumConstraint prosledimo kao dodatni argument za funkciju
    addConstraint

problem.addConstraint(constraint.ExactSumConstraint(50,[1,2,5,10,20]), ["1 din", "2
    din", "5 din","10 din", "20 din"])

# II nacin je da definisemo svoju funkciju koja predstavlja ogranicenje, samo ce sada
     solver nesto sporije da radi posto ugradjene funkcije imaju optimizovanu
    pretragu i brze dolaze do resenja
#
#def o(a, b, c, d, e):
# if a + 2*b + 5*c + 10*d + 20*e == 50:
#   return True
#
#problem.addConstraint(o, ["1 din", "2 din", "5 din","10 din", "20 din"])
#
resenja = problem.getSolutions()

for r in resenja:
  print r
  # Provera da je suma bas 50
  print r["1 din"] + r["2 din"]*2 + r["5 din"]*5 + r["10 din"]*10 + r["20 din"]*20
```

Rešenje 2.4

```
#  4 9 2
#  3 5 7
#  8 1 6
#

import constraint

def o(x,y,z):
    if x+y+z == 15:
        return True

problem = constraint.Problem()
# Promenljive:
# a b c
# d e f
# g h i
problem.addVariables("abcdefghi", range(1,10))
```

```
problem.addConstraint(constraint.AllDifferentConstraint())
# Dodajemo ogranicenja za svaku vrstu
problem.addConstraint(o,"abc")
problem.addConstraint(o,"def")
problem.addConstraint(o,"ghi")
# Dodajemo ogranicenja za svaku kolonu
problem.addConstraint(o,"adg")
problem.addConstraint(o,"beh")
problem.addConstraint(o,"cfi")
#Dodajemo ogranicenja za dijagonale
problem.addConstraint(o,"aei")
problem.addConstraint(o,"ceg")

resenja = problem.getSolutions()
for r in resenja:
    print " ------- "
    print "| {0:d} {1:d} {2:d} |".format(r['a'],r['b'],r['c'])
    print "| {0:d} {1:d} {2:d} |".format(r['d'],r['e'],r['f'])
    print "| {0:d} {1:d} {2:d} |".format(r['g'],r['h'],r['i'])
    print " ------- "
```

Rešenje 2.5

```
# X,Y,Z
#
# X >= Z
# X*2 + X*Y + Z <= 34
#
# X <- {1,2,3,...90}
# Y <- {2,4,6,...60}
# Z <- {1,4,9,16,...100}
#
#

import constraint

problem = constraint.Problem()

# Dodajemo promenljivu X i definisemo njen domen
problem.addVariable('X', range(1,91))

# Dodajemo promenljivu Y i definisemo njen domen
problem.addVariable('Y', range(2,61,2))

domenZ = [];
for i in range(1,11):
    domenZ.append(i*i)

# Dodajemo promenljivu Z i definisemo njen domen
problem.addVariable('Z', domenZ)

def o1(x,z):
    if x >= z:
        return True

def o2(x,y,z):
    if x*2 + x*y + z <= 34:
        return True;

# Dodajemo ogranicenja
problem.addConstraint(o1, 'XZ')
problem.addConstraint(o2, 'XYZ')

resenja = problem.getSolutions()

for r in resenja:
    print "-------------------------"
    print "X = {0:d} , Y = {1:d} , Z = {2:d}".format(r['X'],r['Y'],r['Z'])
print "-------------------------"
```

Rešenje 2.6

```
#    TWO
#  +TWO
#  ------
#  FOUR
#

import constraint

problem = constraint.Problem()
# Definisemo promenljive i njihove vrednosti
problem.addVariables("TF",range(1,10))
problem.addVariables("WOUR",range(10))

# Definisemo ogranicenje za cifre
def o(t, w, o, f, u, r):
    if 2*(t*100 + w*10 + o) == f*1000 + o*100 + u*10 + r:
        return True

# Dodajemo ogranicenja za cifre
problem.addConstraint(o,"TWOFUR")
# Dodajemo ogranicenje da su sve cifre razlicite
problem.addConstraint(constraint.AllDifferentConstraint())

resenja = problem.getSolutions()

for r in resenja:
    print "-----"
    print "  "+str(r['T'])+str(r['W'])+str(r['O'])
    print " +"+str(r['T'])+str(r['W'])+str(r['O'])
    print "="+str(r['F'])+str(r['O'])+str(r['U'])+str(r['R'])
```

Rešenje 2.7

```
# Dati sistem nejednacina nema resenje, tj. metog getSolutions() vraca praznu listu
# Ukoliko se za promenljivu W domen promeni na {1,...,100} sistem ce imati resenje
import constraint

problem = constraint.Problem()

# Dodajemo promenljivu X i definisemo njen domen
problem.addVariable('X', range(1,11))

# Dodajemo promenljivu Y i definisemo njen domen
problem.addVariable('Y', range(1,52,2))

domenZ = []
domenW = []

for i in range(1,11):
  domenZ.append(i*10)
  domenW.append(i**3)

# Dodajemo promenljivu Z i definisemo njen domen
problem.addVariable('Z', domenZ)

# Dodajemo promenljivu W i definisemo njen domen
problem.addVariable('W', domenW)

# Za ovako definisan domen za promenljivu W sistem ce imati resenje
# problem.addVariable('W', range(1,101))


def o1(x,w):
    if x >= 2*w:
        return True

def o2(y,z):
    if 3 + y <= z:
        return True
```

```
def o3(x,y,z,w):
    if x - 11*w + y + 11*z <= 100:
        return True;

# Dodajemo ogranicenja
problem.addConstraint(o1, 'XW')
problem.addConstraint(o2, 'YZ')
problem.addConstraint(o3, 'XYZW')

resenja = problem.getSolutions()
# Proveravamo da li postoji resenje za sistem nejednacina
if resenja==[]:
  print "Sistem nema resenje."
else:
  for r in resenja:
        print "-------------------------"
        print "X = {0:d} , Y = {1:d} , Z = {2:d}, W = {3:d}".format(r['X'],r['Y'],r['
    Z'], r['W'])
    print "-------------------------"
```

Rešenje 2.8

```
#       1   3
#        2 4
#         5
#        6 8
#       7   9
#

import constraint

# Definisemo ogranicenje za jednu dijagonalu
def o(a,b,c,d,e):
    if a<b<c<d<e and a+b+c+d+e==25:
        return True

problem = constraint.Problem()
# Definisemo promenljive za svaku poziciju
problem.addVariables('abcdeABDE',range(1,10))
# Dodajemo ogranicenja za obe dijagonale
problem.addConstraint(o,'abcde')
problem.addConstraint(o,'ABcDE')
# Dodajemo ogranicenje da su vrednosti svih promenljivih razlicite
problem.addConstraint(constraint.AllDifferentConstraint())

resenja = problem.getSolutions()
for r in resenja:
    print "-------------"
    print "{0:d}   {1:d}".format(r['a'],r['A'])
    print " {0:d} {1:d} ".format(r['b'],r['B'])
    print "  {0:d}  ".format(r['c'])
    print " {0:d} {1:d} ".format(r['D'],r['d'])
    print "{0:d}   {1:d}".format(r['E'],r['e'])
    print "-------------"
```

Rešenje 2.9

```
#
# Potrebno je napraviti H komada hleba i K komada kifli
#
# Zarada iznosi:
# - 7din/hleb, tj. zarada za H komada hleba bice 7*H
# - 9din/kifla tj. zarada za K komada kifli bice 9*K
#
# Ukupna zarada iznosi:
# 7*H + 9*K - funkcija koju treba maksimizovati
#
# Ogranicenja vremena:
# - vreme potrebno za mesenje jednog hleba je 10min,
#   tj. za mesenje H komada hleba potrebno je 10*H minuta
```

```
# - vreme potrebno za mesenje jedne kifle je 12min,
#   tj. za mesenje K komada kifli potrebno je 12*K minuta
#
# Ukupno vreme koje je na raspolaganju iznosi 20h, tako da je:
# 10*H + 12*K <= 1200
#
# Ogranicenje materijala:
# - za jedan hleb potrebno je 300g brasna, a za H komada hleba potrebno je H*300 grama
# - za jednu kifli potrebno je 120g brasna, a za K komada kifli potrebno je K*120 grama
#
# Ukupno, na raspolaganju je 20kg brasna, tako da je:
# 300*H + 120*K <= 20000
#
# Broj kifli i hleba je najmanje 0, tako da:
# H>=0
# K>=0
#
# S obzirom na to da imamo 20kg brasna na raspolaganju, mozemo napraviti:
# - najvise 20000/120 kifli
# - najvise 20000/300 hleba
#
# H <= 20000/120 ~ 167
# K <= 20000/300 ~ 67
#
# S obzirom na to da imamo 20h na raspolaganju, mozemo napraviti:
# - najvise 1200/12 kifli
# - najvise 1200/10 hleba
#
# H <= 1200/10 = 120
# K <= 1200/12 = 100
#
# najoptimalnije je za gornju granicu domena postaviti minimum od dobijenih vrednosti, tj. sve ukupno H <= 120, K <= 67

import constraint

problem = constraint.Problem()

# Dodajemo promenljivu H i definisemo njen domen
problem.addVariable('H', range(0,121))

# Dodajemo promenljivu K i definisemo njen domen
problem.addVariable('K', range(0,68))

def ogranicenje_vremena(h,k):
    if 10*h + 12*k <= 1200:
        return True

def ogranicenje_materijala(h,k):
    if 300*h + 120*k <= 20000:
        return True;

# Dodajemo ogranicenja vremena i matrijala
problem.addConstraint(ogranicenje_vremena, 'HK')
problem.addConstraint(ogranicenje_materijala, 'HK')

resenja = problem.getSolutions()

# Pronalazimo maksimalnu vrednost funkcije cilja
max_H = 0
max_K = 0

for r in resenja:
    if 7*r['H'] + 9*r['K'] > 7*max_H + 9*max_K:
        max_H = r['H']
        max_K = r['K']

print "----------------------------------------------------------------"
print "Maksimalna zarada je {0:d}, komada hleba je {1:d}, a komada kifli {2:d}".
    format(7*max_H + 9*max_K, max_H, max_K)
print "----------------------------------------------------------------"
```

Rešenje 2.10

```
import constraint

def o1(x,y,z):
    if x+y+z == 38:
        return True
def o2(x,y,z,w):
    if x+y+z+w == 38:
        return True
def o3(x,y,z,w,h):
    if x+y+z+w+h == 38:
        return True

problem = constraint.Problem()
problem.addVariables("ABCDEFGHIJKLMNOPQRST", range(1,38))
problem.addConstraint(constraint.AllDifferentConstraint())
# Dodajemo ogranicenja za svaku horizontalnu liniju
#  A,B,C
# D,E,F,G
#H,I,J,K,L
# M,N,O,P
#  Q,R,S
problem.addConstraint(o1,"ABC")
problem.addConstraint(o2,"DEFG")
problem.addConstraint(o3,"HIJKL")
problem.addConstraint(o2,"MNOP")
problem.addConstraint(o1,"QRS")

# Dodajemo ogranicenja za svaku od glavnih dijagonala
#  A,B,C
# D,E,F,G
#H,I,J,K,L
# M,N,O,P
#  Q,R,S

problem.addConstraint(o1,"HMQ")
problem.addConstraint(o2,"DINR")
problem.addConstraint(o3,"AEJOS")
problem.addConstraint(o2,"BFKP")
problem.addConstraint(o1,"CGL")

# Dodajemo ogranicenja za svaku od sporednih dijagonala
#  A,B,C
# D,E,F,G
#H,I,J,K,L
# M,N,O,P
#  Q,R,S

problem.addConstraint(o1,"ADH")
problem.addConstraint(o2,"BEIM")
problem.addConstraint(o3,"CFJNQ")
problem.addConstraint(o2,"GKOR")
problem.addConstraint(o1,"LPS")

resenja = problem.getSolutions()
for r in resenja:
    print " -----------------"
    print "  {0:d},{1:d},{2:d}".format(r['A'],r['B'],r['C'])
    print " {0:d},{1:d},{2:d},{3:d}".format(r['D'],r['E'],r['F'],r['G'])
    print "{0:d},{1:d},{2:d},{3:d},{4:d}".format(r['H'],r['I'],r['J'],r['K'],r['L'])
    print " {0:d},{1:d},{2:d},{3:d}".format(r['M'],r['N'],r['O'],r['P'])
    print "  {0:d},{1:d},{2:d}".format(r['Q'],r['R'],r['S'])
    print " -----------------"
```

Rešenje 2.11

```
import constraint

problem = constraint.Problem()
# Kompanija ima 250 zaposlenih radnika
```

```
# za sve njih organizuje dodatnu obuku
# ako je E promenjiva za Elixir, a D za Dart
# mora da vazi E<=250, D<=250 i E + D = 250
#
# Dodajemo promenljivu E i definisemo njen domen
problem.addVariable('E', range(0,251))

# Dodajemo promenljivu D i definisemo njen domen
problem.addVariable('D', range(0,251))

def ukupno_radnika(e,d):
  if e+d == 250:
    return True

def ogranicenje_projekat_sati(e,d):
    if 150*e + 170*d <= 51200:
        return True

def ogranicenje_sredstava(e,d):
    if 100*e + 105*d <= 26000:
        return True;

# Dodajemo ogranicenja za broj projekat/sati i ukupna sredstva
# na raspolaganju kao i za broj radnika u firmi
problem.addConstraint(ogranicenje_projekat_sati, 'ED')
problem.addConstraint(ogranicenje_sredstava, 'ED')
problem.addConstraint(ukupno_radnika, 'ED')

resenja = problem.getSolutions()

# Pronalazimo maksimalnu vrednost funkcije cilja
max_E = 0
max_D = 0
# Od ostvarene dobiti preko broja projekat/sati oduzimamo gubitak za placanje kurseva
     radnicima
for r in resenja:
    if 150*5*r['E'] + 170*6*r['D'] - (100*r['E'] + 105*r['D']) > 150*5*max_E + 170*6*
    max_D - (100*max_E + 105*max_D) :
        max_E = r['E']
        max_D = r['D']


print "Maksimalna zarada je {0:d}, broj radnika koje treba poslati na kurs Elixir je
    {1:d}, a broj radnika koje treba poslati na kurs Dart je {2:d}.".format(170*6*
    max_E + 150*5*max_D - (100*max_E + 150*max_D) , max_E, max_D)
```

Rešenje 2.12

```
# Jedan od mogucih rasporeda
# sva ostala resenja su permutacije
# takve da je u svakoj vrsti i koloni samo jedan top
#
# 8 T - - - - - - -
# 7 - T - - - - - -
# 6 - - T - - - - -
# 5 - - - T - - - -
# 4 - - - - T - - -
# 3 - - - - - T - -
# 2 - - - - - - T -
# 1 - - - - - - - T
#   1 2 3 4 5 6 7 8
#

import constraint

problem = constraint.Problem()
# Dodajemo promenljive za svaku kolonu i njihove vrednosti 1-8
problem.addVariables("12345678", range(1,9))
# Dodajemo ogranicenje da se topovi ne napadaju medjusobno po svakoj vrsti
problem.addConstraint(constraint.AllDifferentConstraint())
resenja = problem.getSolutions()
```

```
# Broj svih mogucih permutacija je 8! = 40320
# Za prikaz svih najbolje pozvati program sa preusmerenjem izlaznih podataka
# python 2_12.py > izlaz.txt
print "Broj resenja je: {0:d}.".format(len(resenja))

for r in resenja:
    print "---------------"
    for i in "12345678":
        for j in range(1,9):
            if r[i] == j:
                print "T",
            else:
                print "-",
        print ""
    print "---------------"
```

Rešenje 2.13

```
# 8 D - - - - - - -
# 7 - - - - D - - -
# 6 - D - - - - - -
# 5 - - - - - D - -
# 4 - - D - - - - -
# 3 - - - - - - D -
# 2 - - - D - - - -
# 1 - - - - - - - D
#   1 2 3 4 5 6 7 8
#

import constraint
import math

problem = constraint.Problem()
# Dodajemo promenljive za svaku kolonu i njihove vrednosti 1-8
problem.addVariables("12345678", range(1,9))
# Dodajemo ogranicenje da se dame ne napadaju medjusobno po svakoj vrsti
problem.addConstraint(constraint.AllDifferentConstraint())

for k1 in range(1,9):
    for k2 in range(1,9):
        if k1 < k2:
            # Definisemo funkciju ogranicenja za dijagonale
            def o(vrsta1, vrsta2, kolona1=k1, kolona2=k2):
                if math.fabs(vrsta1 - vrsta2) != math.fabs(kolona1 - kolona2):
                    return True
            problem.addConstraint(o, [str(k1),str(k2)])

resenja = problem.getSolutions()
# Za prikaz svih najbolje pozvati program sa preusmerenjem izlaznih podataka
# python 2_13.py > izlaz.txt
print "Broj resenja je: {0:d}.".format(len(resenja))
for r in resenja:
    print "---------------"
    for i in "12345678":
        for j in range(1,9):
            if r[i] == j:
                print "D",
            else:
                print "-",
        print ""
    print "---------------"
```

Rešenje 2.14

```
# 9 - - - - - - - - -
# 8 - - - - - - - - -
# 7 - - - - - - - - -
# 6 - - - - - - - - -
# 5 - - - - - - - - -
# 4 - - - - - - - - -
```

```
# 3 - - - - - - - - -
# 2 - - - - - - - - -
# 1 - - - - - - - - -
#   1 2 3 4 5 6 7 8 9
#

import constraint
import json

problem = constraint.Problem()

# Dodajemo promenljive za svaki red i njihove vrednosti 1-9
for i in range(1, 10):
    problem.addVariables(range(i * 10 + 1, i * 10 + 10), range(1, 10))

# Dodajemo ogranicenja da se u svakoj vrsti nalaze razlicite vrednosti
for i in range(1, 10):
    problem.addConstraint(constraint.AllDifferentConstraint(), range(i * 10 + 1, i *
    10 + 10))

# Dodajemo ogranicenja da se u svakoj koloni nalaze razlicite vrednosti
for i in range(1, 10):
    problem.addConstraint(constraint.AllDifferentConstraint(), range(10 + i, 100 + i,
     10))

# Dodajemo ogranicenja da svaki podkvadrat od 3x3 promenljive ima razlicite vrednosti
for i in [1,4,7]:
    for j in [1,4,7]:
        pozicije = [10*i+j,10*i+j+1,10*i+j+2,10*(i+1)+j,10*(i+1)+j+1,10*(i+1)+j
    +2,10*(i+2)+j,10*(i+2)+j+1,10*(i+2)+j+2]
        problem.addConstraint(constraint.AllDifferentConstraint(),pozicije)


ime_datoteke = raw_input("Unesite ime datoteke sa tablom za sudoku: ")
f = open(ime_datoteke, "r")
tabla = json.load(f)
f.close()

# Dodajemo ogranicenja za svaki broj koji je zadat na tabli
for i in range(9):
    for j in range(9):
        if tabla[i][j] != 0:
            def o(vrednost_promenljive, vrednost_na_tabli = tabla[i][j]):
                if vrednost_promenljive == vrednost_na_tabli:
                    return True

            problem.addConstraint(o, [((i+1)*10 + (j+1))])

resenja = problem.getSolutions()

for r in resenja:
    print "========================="
    for i in range(1,10):
        print "|",
        for j in range(1,10):
            if j%3 == 0:
                print str(r[i*10+j])+" |",
            else:
                print str(r[i*10+j]),
        print ""
        if i%3 == 0 and i!=9:
            print "-------------------------"
    print "========================="
```

3

Funkcionalno programiranje

Potrebno je imati instaliran GHC kompajler na računaru.
Literatura:

(a) https://www.haskell.org/

(b) https://wiki.haskell.org/Haskell

3.1 Uvod

3.1.1 Uvodni primeri

Zadatak 3.1 Korišćenje kompilatora i interpretera.

```
-- Ovako pisemo jednolinijske komentare

{-
    Ovako pisemo
    viselinijske
    komentare
-}

{-
  Interpreter pokrecemo iz terminala komandom:

  ghci

  i otvorice nam se interaktivni interpreter:

  GHCi, version 8.0.1: http://www.haskell.org/ghc/  :? for help
  Prelude>

  Sa interpreterom mozemo direktno komunicirati, npr
  Prelude> print "Zdravo! :)"
  "Zdravo! :)"

  Interpretatoru dodajemo mogucnost izvrsavanja funkcija iz izvorne
  ime_programa.hs komandom:

  :load ime_programa.hs

  Nakon toga mozemo pokrenuti sve funkcije koje su u toj datoteci
  definisane, npr.

  Prelude> main
  "Zdravo! :)"

  Iz interpretera se izlazi komandom :quit
  Prelude> :quit

  Haskell programe mozemo kompajlirati komandom:

  ghc ime_programa.hs
```

```
40
   koja ce napraviti izvrsni program koji pokrecemo sa
42
   ./ime_programa
44
-}
46
main = print "Zdravo! :)"
```

Zadatak 3.2 Tipovi podataka. Tipski razredi. Funkcije.

```
1  {-
     Osnovni tipovi:
3    Bool
     Char
5    String
     Int
7    Integer
     Float
9    Double

11   Tipski razredi (definisu funkcije koje neki tip mora da implementira):
     Eq - tipovi sa jednakoscu (==,/=)
13   Ord - tipovi sa uredjenjem (<,<=,>,>=,...)
     Num - numericki tipovi (+,-,*,...)
15   Integral - celobrojni tipovi (div, mod,...)
     Fractional - razlomacki tipovi (/, recip,...)
17
     Funkcije preslikavaju vrednosti jednog tipa (ili tipskog razreda) u vrednost drugog
        tipa (ili tipskog razreda):
19
     duplo :: Int -> Int
21
     Informacije o bilo kom objektu mozete dobiti naredbom:
23   :info ime_objekta
   -}
25
   duplo :: Int -> Int
27 duplo x = x+x

29 {-
     Aritmeticke operacije nad tipovima Int, Integer, Float:
31   +, -, *, /,
     ^^ - stepen (tip Int)
33   ** - stepen (tip Float)

35   Funkcije:
     mod, div, log, sqrt, sin, cos, tan
37
   -}
39
   ostatak3 x = mod x 3
41  -- ekvivalentno sa x `mod` 3

43 -- ukoliko koristimo funkcije koje su definisane za realne tipove nad argumentima
       koji su celobrojnog tipa potrebno je prevesti vrednost u nadklasu Num koriscenjem
        funkcije fromIntegral, nece se izvrsiti eksplicitno kastovanje
   -- primer: funkcija racuna vrednost korena samo za celobrojni argument
45 korenCeli :: Int -> Double
   korenCeli n = sqrt (fromIntegral n)
47
   -- Blokovi se u Haskelu odvajaju tabulatorom.
49 -- Uslovni izraz (mora imati else granu)

51 jedan n = if n == 1
53           then True
             else False

55 -- Ogradjene jednacine (guarded equations) - mozemo ih koristiti umesto uslovnih
       izraza, bitan je redosled navodjenja

57 sumaPrvih n
       | n == 0 = 0
```

```
59      | otherwise = n + sumaPrvih(n-1)
```

Zadatak 3.3 Liste. N-torke.

```
{-
  Lista je niz vrednosti istog tipa:

  [element1,element2,...]

  Operacije:

  x : lista - dodaje element x na pocetak liste
  lista1 ++ lista1 - nadovezuje dve liste
  lista !! pozicija - vraca element liste koji se nalazi na poziciji pozicija
  [a..b] - konstruise listu sa elementima od a do b (u zavisnosti od tipa)
-}

lista = [1..5]
obrnutaLista = [5,4..1]

{-
  Korisne funkcije:
  head lista - vraca prvi element liste
  tail lista - vraca rep liste
  length lista - vraca broj elemenata liste
  take n lista - vraca listu prvih n elemenata
  drop n lista - vraca listu bez prvih n elemenata
  null lista - vraca True ukoliko je lista prazna, False inace
-}

-- Niske su liste karaktera

niska1 = ['A','l','a','d','d','i','n']
niska2 = "Aladdin"

{-
  N-torke su nizovi vrednosti razlicitog tipa:
  ('a',1,[1.2, 2.3],"torka")

  Funkcije definisane za parove (n=2):
  fst () - vraca prvi element
  snd () - vraca drugi element
-}
```

3.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 3.4 Napisati sledeće funkcije:

- `proizvodPrvih n` - računa proizvod prvih n pozitivnih brojeva (rekurzivno, bez korišćenja formule)
- `prost n` - vraća True ako je broj prost, False inače
- `nzd a b` - računa najmanji zajednički delilac brojeva a i b (koristiti Euklidov algoritam)
- `tipJednacine a b c` - vraća tip kvadratne jednačine $a * x^2 + b * x + c = 0$ (đegenerisana", "jedno resenje", đva resenja","bez resenja")
- `izDekadne x osn` - prebacuje broj x iz dekadne u osnovu osn (pretpostaviti da je osn>1 i osn<10)
- `uDekadne x osn` - prebacuje broj x iz osnove osn u dekadnu osnovu (pretpostaviti da je osn>1 i osn<10)
- `ceoDeo x` - računa ceo deo korena broja x (bez korišćenja ugrađenih funkcija za koren i/ili stepen)

[Rešenje 3.4]

Zadatak 3.5 Napisati sledeće funkcije koristeći liste:

- `harm n` - vraća listu prvih n elemenata harmonijskog reda
- `delioci n` - vraća listu svih delilaca pozitivnog broja n
- `nadovezi lista1 lista2 n` - nadovezuje na lista1 n puta lista2

[Rešenje 3.5]

3.1.3 Zadaci za vežbu

Zadatak 3.6 Napisati sledeće funkcije:

- `sumaKvadrata n` - računa sumu kvadrata prvih n brojeva (rekurzivno, bez korišćenja formule)
- `brojDelilaca n` - vraća broj delilaca broja n
- `fib n` - računa n-ti element Fibonačijevog reda
- `osnova x osn1 osn2` - prebacuje broj x iz osnove osn1 u osnovu osn2 (pretpostaviti da su osn1 i osn2 brojevi veći od 1 i manji od 10)

Zadatak 3.7 Napisati sledeće funkcije koristeći liste:

- `parni n` - vraća listu prvih n parnih brojeva
- `fibLista n` - vraća listu prvih n elemenata Fibonačijevog niza
- `jednocifreniDelioci n` - vraća listu svih jednocifrenih delilaca broja n

3.2 Liste

3.2.1 Uvodni primeri

Zadatak 3.8

```
{-
  Uparivanje sablona (pattern matching)

  not :: Bool -> Bool
  not False = True
  not True = False

  Ako pozovemo not True uparice se prvi sablon,
  a ako pozovemo not False uparice se drugi sablon

  Dzoker (wildcard)

  prvi nacin (ako bismo zamenili redosled sablona, javlja pattern match overlapped)
  (&&) :: Bool -> Bool -> Bool
  True && True = True
  _ && _ = False   --- moglo bi i: n && m = False, bitno je samo da su razlicito
    imenovani

  drugi nacin, efikasniji:
  (&&) :: Bool -> Bool -> Bool
  True && x = x
  False && _ = False

  _ predstavlja dzoker, tj. sablon koji odgovara bilo kojoj vrednosti (obratiti
    paznju na redosled definisanja sablona!)

  Sabloni lista:

  length [_] = 1 - [_] predstavlja listu sa jednim elementom
```

```
  length x = 1 + length (tail x) - x predstavlja listu

-}

length1 [] = 0
length1 x = 1 + length1 (tail x)

-- (x:xs) - predstavlja sablon liste, x je glava, xs je rep

length2 [] = 0
length2 (_:xs) = 1 + length2 xs

-- preslikava iz liste elemenata u jedan element
glava :: [a] -> a
glava (x:_) = x

-- preslikava iz liste u listu
rep :: [a] -> [a]
rep (_:xs) = xs

-- restrikcija na liste elemenata koji pripadaju numerickim tipovima
glavaNum :: (Num a) => [a] -> a
glavaNum (x:_) = x

-- Generatori liste (list comprenhension)

-- even proverava da li je argument paran, a odd da li je neparan
-- elementi listi pripadaju skupu od 1 do 50
parni = [x | x<-[1..50], even x]
neparni = [x | x<-[1..50], odd x]

deljiviPet = [x | x<-[1..50], mod x 5 == 0] -- elementi liste pripadaju skupu od 1 do
     50 i deljivi su sa 5

kvadrati = [x^2 | x<-[1..10]]

-- Mozemo imati vise generatora

torke1 = [(x,y) | x<-[1..5], y<-[6..9]]

-- obratiti paznju kako utice zamena mesta generatora
torke2 = [(x,y) | y<-[6..9], x<-[1..5]]

-- sa negativnim korakom
torke3 = [(x,y) | y<-[9,8..6], x<-[1..5]]

-- Generatori mogu zavisiti jedni od drugih

-- kasniji generatori mogu zavisiti samo od promenljivih koje su uvedene ranijim
    generatorima citajuci sa leva na desno
torke4 = [(x,y) | x<-[1..10], y<-[x..10]]

{-
  Sekcija where

  where ime = vrednost
  - imena koja definisemo u where sekciji su vidljiva samo u okviru funkcije
  u kojoj su definisana
-}

bmiIzracunaj tezina visina
    | bmi <= 18.5 = "Mrsavko"
    | bmi <= 25.0 = "Sportista"
    | bmi > 25.0    = "Bucko"
    where bmi = tezina / visina ^ 2
```

3.2.2 Zadaci za samostalni rad sa rešenjima

Zadatak 3.9 Napisati sledeće funkcije:

- `bezbedanRep lista` - ukoliko je lista prazna vraća praznu listu, inače vraća rep liste, kori-

steći: a) uslovne izraze b) ograđene jednačine c) uparivanje šablona

- `savrseni n` - vraća listu savršenih brojeva manjih od n (broj je savršen ako je jednak sumi svojih faktora (tj. delilaca), ne uključujući taj broj)
- `zbirPar n` - vraća listu parova (a,b) takvih da su a i b prirodni brojevi i da je zbir a i b jednak n
- `polovina lista` - deli listu na dve polovine
- `poslednji lista` - vraca poslednji element liste
- `spoji lista` - spaja listu listi u jednu listu
- `sufiksi lista` - vraca listu svih sufiksa liste
- `obrni lista` - obrce listu
- `izbaci k lista` - izbacuje k-ti element iz liste
- `duplira lista` - duplira svaki element iz liste
- `ubaci k x lista` - ubacuje u listu na poziciju k element x
- `izbaciSvaki n lista` - izbacuje svaki n-ti element iz liste (broji se od 1)

[Rešenje 3.9]

3.2.3 Zadaci za vežbu

Zadatak 3.10 Napisati sledeće funkcije:

- `prosti n` - vraća listu prvih n prostih brojeva
- `pitagorineTrojke n` - vraća listu Pitagorinih trojki (a,b,c) takvih da su a,b,c<=n
- `proizvodPar n` - vraća listu parova (a,b) takvih da su a i b prirodni brojevi i da je proizvod a i b manji od n
- `podeli n lista` - deli listu na dve liste, prva lista sadrži prvih n elemenata, a druga lista sadrži ostatak
- `pretposlednji lista` - vraća pretposlednji element liste
- `izbrisiPrvi x lista` - briše prvo pojavljivanje broja x u listi
- `palindrom lista` - ispituje da li je lista palindrom
- `pozNeg lista` - vraća par (a, b) takav da a predstavlja broj True elemenata u listi, a b predstavlja broj False elemenata u listi
- `izmedju a b lista` - vraca listu elemenata koji su veći od a, a manji od b

3.3 Funkcije

3.3.1 Uvodni primeri

Zadatak 3.11

```
{-
  Karijeve funkcije (uzimaju jedan po jedan element)

  add :: (Int, Int) -> Int
  Karijeva verzija:
  add1 :: Int -> Int -> Int
-}

add1 :: Int -> Int -> Int
{-
  -> je desno asocijativna
  add1 :: Int -> (Int -> Int)
-}
add1 x y = x + y

{-
  Primena funkcije je levo asocijativna

  add1 x y === ((add1 x) y)

  Polimorfne funkcije - sadrze jednu ili vise tipskih promenljivih

  length1 :: [a] -> Int (a je tipska promenljiva, moze da bude bilo kog tipa)

  Preopterecene funkcije (overloaded) - sadrze jednu ili vise tipskih promenljivih
    koje propadaju nekom tipskom razredu

  sum :: Num a => [a] -> a (a mora biti numerickog tipa)
-}

sum1 :: Num a => [a] -> a
sum1 [x] = x
sum1 (x:xs) = x + sum1 xs

{-
  Lambda (anonimne) funkcije

  \ x = x + x
-}

add2 = \x -> (\y -> x + y )

{-
  Sekcije

  Operatore mozemo zapisati kao Karijeve funkcije:
  (+) x y <-> x+y
  (+2) x <-> x+2

  Generalno ako je op operator
  (op), (x op), (op y) nazivamo sekcijama
-}

duplo x = (*2) x


{-
 Funkcija zip pravi listu torki od dve liste
 tako sto spaja elemente prve liste sa elementima druge liste
-}

spojeno1 = zip ['a', 'b', 'c'] [1,2,3]
spojeno2 = zip ['a', 'b', 'c'] [1,2,3,4,5,6]
spojeno3 = zip ['a', 'b', 'c'] [1,2]
```

Zadatak 3.12

```
{-
  Funkcije viseg reda
  -funkcije koje uzimaju funkciju kao argument ili vracaju funkciju kao rezultat

  Funkcija map
```

```
  - kao prvi argument uzima funkciju koju primenjuje na sve elemente liste
  koju uzima kao drugi argument

  map :: (a -> b) -> [a] -> [b]

-}

povecaj lista = map (+1) lista

{-
  Funkcija filter
  - kao prvi argument uzima funkciju uslova (funkcija koja vraca logicku vrednost)
  i izdvaja iz liste koju uzima kao drugi argument
  sve elemente koji zadovoljavaju zadat uslov

  filter :: (a -> Bool) -> [a] -> [a]

-}

pozitivni lista = filter (>0) lista

{-
  Jos neke funkcije viseg reda za rad sa listama:

  any uslov lista - vraca True ako postoji element u listi koji zadovoljava uslov,
    False inace
  all uslov lista - vraca True ako svi elementi u listi zadovoljavaju uslov, False
    inace
  takeWhile uslov lista - izdvaja jedan po jedan element liste sve dok ne stigne do
    elementa koji ne zadovoljava uslov
  dropWhile uslov lista - izbacuje jedan po jedan element liste sve dok ne stigne do
    elementa koji ne zadovoljava uslov
  sum lista - sabira elemente liste
  product lista - mnozi elemente liste
  reverse lista - obrce listu
  unzip - spaja listu parova u dve liste
  concat lista - spaja liste iz liste listi u jednu listu
  elem e lista - vraca True ako e postoji u listi, False inace
  replicate n x - kopira broj x n puta

-}

prviNePozitivni lista = takeWhile (<=0) lista

{-
  Funkcija foldr
  - enkapsulira sledeci sablon rekurzije

  f [] = v - f preslikava praznu listu u vrednost v
  f (x:xs) = x op (f xs) - nepraznu listu preslikava u funkciju op
  primenjenu na glavu liste (x) i f od repa liste (f xs)

  Npr:
-}

sum1 [] = 0 -- v = 0
sum1 (x:xs) = x + sum1 xs -- op = +

sum2 lista = foldr (+) 0 lista

{-
  Kompozicija funkcija

  (f . g) x  <-> f (g x)
-}

negativneSume liste = map (negate . sum) liste
-- negativneSume [[1..5], [6..10], [11..15]]

{-
  Cesto nam je potrebno da listu zadamo na sledeci nacin:

  lista = [f(x) | x<-D, p(x)]
```

```
75
   to mozemo jednostavno uraditi koristeci funkcije map i filter
77
   map f(filter p xs)
79 -}

81 povecajPozitivne [] = []
   povecajPozitivne lista = map (+1) (filter (>0) lista)
```

3.3.2 Zadaci za samostalni rad sa rešenjima

Zadatak 3.13 Napisati sledeće funkcije:

- `pozicije x lista` - vraća listu pozicija broja x u listi
- `qsort lista` - sortira listu algoritmom qsort
- `brisiPonavljanja lista` - briše sva uzastopna ponavljanja elemenata u listi
- `podlistePonavljanja lista` - pakuje sva uzastopna ponavljanja nekog elementa u podlistu tako da rezultat bude lista listi
- `imaDuplikata lista` - ispituje da li u listi postoje duplikati
- `uzastopniParovi lista` - vraća listu uzastopnih parova
- `broj lista` - vraća broj određen ciframa koje se nalaze u listi u obrnutom poretku
- `broj1 lista` - vraća broj određen ciframa koje se nalaze u listi

[Rešenje 3.13]

Zadatak 3.14 Napisati sledeće funkcije (koristiti funkcije `map`, `foldr` ili `filter`):

- `sumaKvNeg lista` - računa sumu kvadrata negativnih elemenata iz liste (koristiti funkciju
- `malaSlova lista` - računa broj malih slova unutar liste
- `sume lista` - vraća listu suma elemenata svake podliste
- `spoji lista` - spaja sve podliste u jednu listu
- `listaUPar lista` - pretvara listu parova u par dve liste, tako da prva lista sadrži sve prve elemente parova a druga sve druge (implementacija funkcije unzip)
- `parOdListi lista1 lista2` - pravi listu parova od dve liste, tako da prvi element svakog para bude iz prve liste, a drugi element svakog para bude iz druge liste (implementacija funkcije zip)
- `ucesljaj lista1 lista2` - pravi listu naizmeničnim učešljavanjem elemenata listi

[Rešenje 3.14]

3.3.3 Zadaci za vežbu

Zadatak 3.15 Napisati sledeće funkcije:

- `paroviPonavljanja lista` - pravi listu parova (x,k) takvih da je x element iz liste, a k broj koliko se puta x uzastopno pojavljuje u listi
- `sortirana lista` - ispituje da li je lista sortirana
- `permutacija lista1 lista2` - ispituje da li je lista1 permutacija liste2
- `izdvoji a b lista` - izdvaja elemente liste od pozicije a do pozicije b

- `izbrisi x lista` - brise sva pojavljivanja broja x u listi
- `ponovi n lista` - svaki element iz liste ponavlja n puta

Zadatak 3.16 Napisati sledeće funkcije (koristiti funkcije `map`, `foldr` ili `filter`):

- `brojPoz lista` - računa broj pozitivnih elemenata liste
- `eliminisiProste lista` - eliminiše sve proste brojeve iz listi
- `saberiAbs lista` - sabira apsolutne vrednosti elemenata iz liste
- `nadoveziProizvod lista` - svakoj podlisti nadovezuje proizvod njenih elemenata
- `prosek lista` - računa prosek liste
- `brojPojavljivanja x lista` - vraća broj pojavljivanja broja x u listi

3.4 Rešenja

Rešenje 3.4

```
-- Pretpostavljamo da funkciju koristimo samo za n>=1
proizvodPrvih n
      | n == 1 = 1
      | otherwise = n * proizvodPrvih (n-1)

-- Ispitujemo da li je broj prost tako sto pozovemo pomocnu funkciju
-- koja ispituje da li postoji neki broj pocev od 2 do n koji deli broj n
prost n = prost1 n 2

-- Proveravamo da li postoji broj koji deli n (pocev od broja 2 - poziv iz prethodne
    funkcije)
-- ukoliko je k == n to znaci da smo proverili za svaki broj od 2 do n-1
-- i ustanovili da nijedan od njih ne deli n tako da vracamo True kao indikator da
    broj n jeste prost
-- ukoliko je n deljivo sa k, vracamo False (n nije prost)
-- ukoliko n nije deljivo sa k, pozovemo funkciju rekurzivno za sledeci broj (k+1)
prost1 n k
  | n == k = True
  | n `mod` k == 0 = False
  | otherwise = prost1 n (k+1)

nzd a b
  | b == 0 = a
  | otherwise = nzd b (a `mod` b)

tipJednacine a b c
  | a == 0 = "Degenerisana"
  | (b*b - 4*a*c) == 0 = "Jedno resenje"
  | (b*b - 4*a*c) > 0 = "Dva resenja"
  | otherwise = "Nema resenja"

uDekadnu x osn = if x==0 then 0 else uDekadnu (x `div` 10) osn * osn + (mod x 10)

izDekadne x osn = if x==0 then 0 else izDekadne (x `div` osn) osn * 10 + (mod x osn)

-- Trazimo ceo deo korena broja x
-- trazimo prvi broj (pocev od 1) ciji je kvadrat veci od kvadrata naseg broja x
-- i kao rezultat vracamo broj koji je za jedan manji
ceoDeo x = ceoDeo1 x 1

ceoDeo1 x i
  | (i*i) > x = (i-1)
  | otherwise = ceoDeo1 x (i+1)
```

Rešenje 3.5

```
-- Pravimo listu prvih n elemenata harmonijskog reda (n>0)
-- ukoliko je n = 1, vracamo listu koja sadrzi prvi element harmonijskog reda
-- inace pozovemo funkciju rekurzivno za n-1
-- i na rezultat nadovezemo reciprocnu vrednost broja n
harm n
  | n == 1 = [1.0]
  | otherwise = harm (n-1) ++ [recip n]

-- Pravimo listu delioca broja n
-- ukoliko je n = 1, vracamo listu koja sadrzi samo 1
-- inace pozivamo pomocnu funkciju sa jos jednim parametrom
-- koji nam govori do kog potencijalnog delioca smo stigli
delioci n
    | n == 1 = [1]
    | otherwise = delioci1 n 2

-- Pravimo listu delioca broja n pocev od broja k
-- ukoliko smo stigli do broja n (k==n) vracamo praznu listu
-- inace proveravamo da li je k delioc broja n
-- ako jeste, stavljamo ga u listu i pozivamo funkciju rekurzivno
-- za k+1 (sledeci potencijalni delilac)
-- inace samo pozivamo funkciju rekurzivno za k+1
delioci1 n k
    | k == n = []
    | n `mod` k == 0 = [k] ++ (delioci1 n (k+1))
    | otherwise = delioci1 n (k+1)

-- Nadovezujemo listu2 na listu1 n puta
-- ukoliko je n == 1 na listu1 nadovezemo listu2 operatorom ++
-- inace pozovemo funkciju rekurzivno za n-1 i na rezultat nadovezemo listu2
nadovezi lista1 lista2 n
  | n == 1 = lista1 ++ lista2
  | otherwise = (nadovezi lista1 lista2 (n-1)) ++ lista2
```

Rešenje 3.9

```
bezbedanRep1 lista = if lista == [] then [] else tail lista

bezbedanRep2 lista
  | lista == [] = []
  | otherwise = tail lista

bezbedanRep3 [] = []
bezbedanRep3 (x:xs) = xs
-- bezbedanRep3 lista = tail lista

savrseni n = [x | x<-[1..n], sum(faktori x) == x]
faktori x = [i | i<-[1..x-1], x `mod` i ==0]

zbirPar n = [(a,b)| a<-[1..n], b<-[1..n], a+b==n]
-- drugi, efikasniji nacin
zbirPar2 n = [(a,b)| a<-[1..n], b<-[n-a], b/=0]

polovina [] = ([],[])
polovina lista = (take n lista, drop n lista)
              where n = length lista `div` 2

poslednji lista = lista !! poz
                 where poz = length lista - 1  -- moze i: length(tail lista) -1

spoji [] = [] -- nije neophodno, prolazi drugi sablon i za praznu listu
spoji lista = [x | podlista <- lista, x <- podlista]

sufiksi [] = [[]]
-- lista je sama svoj sufiks, a ostali sufiksi su sufiksi njenog repa
sufiksi (x:xs) = (x:xs) : sufiksi xs

obrni [] = []
obrni (x:xs) = obrni xs ++ [x]

```

```
izbaci _ [] = []
izbaci 0 (x:xs) = xs
izbaci k (x:xs) = x : (izbaci (k-1) xs)

duplira [] = []
duplira (x:xs) = [x,x] ++ duplira xs

ubaci 0 n lista = n : lista
ubaci k n [] = [n]
ubaci k n (x:xs) = x : (ubaci (k-1) n xs)

izbaciSvaki n [] = []
izbaciSvaki n lista = take (n-1) lista ++ izbaciSvaki n (drop n lista)
```

Rešenje 3.13

```
-- Racunamo sve pozicije broja x u listi
-- tako sto spajamo listu sa listom brojeva od 0 do n
-- i od liste parova (broj, pozicija)
-- izdvajamo one pozicije kod kojih je broj = x
pozicije :: Eq a => a -> [a] -> [Int]
pozicije x [] = []
pozicije x lista = [i | (x1,i) <- zip lista [0..n], x == x1]
                 where n = length lista - 1

qsort :: Ord a => [a] -> [a]
qsort [] = []
qsort (x:xs) = qsort manji ++ [x] ++ qsort veci
            where manji = [a | a<-xs, a <= x]
                  veci = [b | b<-xs, b > x]

-- Pravimo listu bez uzastopnih ponavljanja
-- tako na glavu nadovezemo rezultat rekurzivnog poziva funkcije nad korigovanim repom
-- sa pocetka repa izbacujemo sve elemente koji su jednaki glavi
-- [1,1,1,1,1,2]
-- 1 : brojPonavljanja [2]
-- [1,2]
brisiPonavljanja :: Eq a => [a] -> [a]
brisiPonavljanja [] = []
brisiPonavljanja (x:xs) = x : brisiPonavljanja(dropWhile (==x) xs)

-- Pravimo listu koja sadrzi sva uzastopna pojavljivanja elemenata u posebnim listama
-- [1,1,1,2,2,3] -> [[1,1,1], [2,2], [3]]
-- tako sto pravimo listu uzastopnih pojavljivanja glave
-- i pozivamo funkciju rekurzivno pocev od elementa koji nije jednak glavi
-- [[1,1,1] , podlistePonavljanja [2,2,3]]
-- [[1,1,1], [2,2], podlistePonavljanja [3]]
-- [[1,1,1], [2,2], [3], podlistePonavljanja []]
podlistePonavljanja :: Eq a => [a] -> [[a]]
podlistePonavljanja [] = []
podlistePonavljanja (x:xs) = (x : (takeWhile (==x) xs)) : podlistePonavljanja(
    dropWhile (==x) xs)

-- Ispitujemo da li lista sadrzi duplikate
-- ukoliko je lista prazna vracamo False
-- inace, ukoliko se glava nalazi jos negde u repu vracamo True
-- odnosno pozivamo funkciju rekurzivno za rep
imaDuplikata :: Eq a => [a] -> Bool
imaDuplikata [] = False
imaDuplikata (x:xs) = elem x xs || imaDuplikata xs

-- Pravimo listu uzastopnih parova tako sto spajamo svaki element liste
-- sa elementima iz repa
-- [1,2,3,4] [2,3,4] -> [(1,2), (2,3), (3,4)]
uzastopniParovi :: [a] -> [(a,a)]
uzastopniParovi [] = []
uzastopniParovi lista = zip lista (tail lista)

-- [1,2,3] -> 321
broj :: Num a => [a] -> a
broj [] = 0
```

```
broj (x:xs) = (broj xs)*10 + x

-- [1,2,3] -> 123
broj1 [] = 0
broj1 (x:xs) = x*10^(length xs) + broj xs

-- drugi nacin koriscenjem funkcije broj
-- broj1 lista = broj (reverse lista)
```

Rešenje 3.14

```
-- Racunamo sumu kvadrata negativnih elemenata liste
-- tako sto izdvajamo negativne elemente funkcijom filter u listu negativni
-- i primenjujemo operaciju kvadriranja nad svim elementima liste negativni
-- koristeci funkciju map i rezultat nazivamo kvadrati
-- nakon toga sabiramo elemente liste kvadrati u akumulator cija je pocetna vrednost
    = 0
-- podsetnik: celobrojni stepen -> ^, realni stepen -> ^^
sumaKvNeg1 :: (Ord a, Fractional a) => [a] -> a
sumaKvNeg1 [] = 0
sumaKvNeg1 lista =  foldr (+) 0 kvadrati
    where kvadrati = map (^2) negativni
          negativni = filter (<0) lista

-- Drugi nacin - pravimo kompoziciju funkcija filter, map i fold
sumaKvNeg2 :: [Double] -> Double
sumaKvNeg2 [] = 0
sumaKvNeg2 lista = suma lista
suma = (foldr (+) 0 ) . (map (^2)) . (filter (<0))

-- Racunamo broj malih slova u listi tako sto primenjujemo kompoziciju funkcija
-- filter - filtriramo karaktere tako da nam ostanu samo mala slova
-- i nakon toga racunamo duzinu rezultujuce liste - funkcija length
malaSlova :: [Char] -> Int
malaSlova [] = 0
malaSlova lista = (length . (filter (\x -> if x>='a' && x<='z' then True else False)
    )) lista

-- Pravimo listu suma svih elemenata podlisti
-- tako sto primenjujemo funkciju sum nad svakom podlistom koristeci funkciju map
sume :: Num a => [[a]] -> [a]
sume [] = []
sume lista = map (sum) lista

-- Spajamo podliste u jednu listu
-- tako sto nadovezujemo svaku podlistu na akumulator cija je pocetna vrednost
    jednaka []
-- koristeci funkciju foldr
-- [[1,1], [2,2,2], [3,3,3]] - []
-- [[2,2,2], [3,3,3]] - [1,1]
-- [[3,3,3]] - [1,1,2,2,2]
-- [] - [1,1,2,2,2,3,3,3]
spoji :: [[a]] -> [a]
spoji [] = []
spoji lista = foldr (++) [] lista

-- Pravimo par listi od liste parova kao sto radi funkcija unzip
-- [(1,2), (1,2), (1,2)] -> ([1,1,1], [2,2,2])
-- tako sto svaki par iz liste dodajemo u akumulator cija je pocetna vrednost par
    praznih listi
-- [(1,2), (1,2), (1,2)]  - ([],[])
-- [(1,2), (1,2)]  - ([1],[2])
-- [(1,2)]  - ([1,1],[2,2])
-- []  - ([1,1,1],[2,2,2])
listaUPar :: [(a,b)] -> ([a],[b])
listaUPar [] = ([],[])
listaUPar lista = foldr (\ (a,b) (c,d) -> (a:c, b:d)) ([],[]) lista

{- drugi nacin bez foldr:
listaUPar [] = ([],[])
listaUPar ((a,b):xs) = (a:l1, b:l2)
    where
```

```
      l1 = fst rep
59    l2 = snd rep
      rep = listaUPar xs
61 -}
-- implementacija funkcije zip
63 -- [1,2,3] [4,5,6] -> [(1,4), (2,5), (3,6)]
-- da bi radila po duzini krace liste neophodno je dodati sablone za takve situacije
65 parOdListi [] _ = []
parOdListi _ [] = []
67 -- od glava listi pravimo par koji dodajemo na pocetak rezultujuce liste za rep
parOdListi (x:xs) (y:ys) = ( (x, y) : (parOdListi xs ys))
69
-- [1,1,1] [2,2,2] -> [1,2,1,2,1,2]
71 -- [1,1] [2,2,2,2] -> [1,2,1,2,2,2]
-- treba obuhvatiti slucajeve listi razlicitih duzina
73 -- sledeca linija koda ne radi jer nije dozvoljeno koriscenje dzoker promenljive u
    izrazu kojim definisete vrednost funkcije
-- ucesljaj [] _ = _
75 ucesljaj [] lista = lista
ucesljaj lista [] = lista
77 ucesljaj (x:xs) (y:ys) = [x]++[y]++(ucesljaj xs ys)
```