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

def cena_voca(prodavnica, ime_voca):
  for voce in prodavnica:
    if voce['ime'] == ime_voca:
      return voce['cena']

# 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
i = 0
# Za svako voce u korpi dodajemo njegovu cenu u svaki racun posebno
while i < len(korpa):
    ime_voca = korpa[i]['ime']
    maxi_racun = maxi_racun + korpa[i]['kolicina']*cena_voca(maxi_cene, ime_voca)
    idea_racun = idea_racun + korpa[i]['kolicina']*cena_voca(idea_cene, ime_voca)
    shopngo_racun = shopngo_racun + korpa[i]['kolicina']*cena_voca(shopngo_cene,
    ime_voca)
    i += 1

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)
```