Programski jezici

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

Univerzitet u Beogradu
Matematički fakultet

Programske paradigme

Materijali za vežbe

Nastavnik: Milena Vujošević Janičić
Asistenti: Milan Čugurović, Ivan Ristović

Beograd
2020.

Priprema materijala:
dr Milena Vujošević Janičić, docent na Matematičkom fakultetu u Beogradu
Marjana Šolajić, asistent na Matematičkom fakultetu u Beogradu
Branislava Živković
Nemanja Mićović, asistent na Matematičkom fakultetu u Beogradu
Milica Selaković, asistent na Matematičkom fakultetu u Beogradu
Milan Čugurović, asistent na Matematičkom fakultetu u Beogradu
Ivan Ristović, asistent na Matematičkom fakultetu u Beogradu

Sadržaj

1

Skript programiranje

Potrebno je imati instaliran Python 3.7 na računaru.
Literatura:

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

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

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

1.1 Žašto Python?

Programski jezik Python predstavlja trenutno najpopularniji programski jezik. Koristi se svuda, od prosvete preko privrede i industrije do medicine i mnogih drugih oblasti. Najbolji pokazatelj toga su trenutne rang liste popularnosti programskih jezika, na kojima Python dominira. Organizacija IEEE rangirala je Python kao #1 jezik za 2018. godinu, pre čega je bio rangiran kao #1 u 2017. godini, a #3 u 2016. godini. GitHut, vizualizacija GitHub zastupljenosti jezika, stavlja Python na izuzetno visoku #3 poziciju.

Python kultura propagira open-source ideje, zajednicu koja je povezana kako na lokalnom tako i na globalnom nivou, koja održava svoj jezik, i deli svoje znanje sa drugima. Filozofija jezika je toliko jaka da je čak ugrađena i u sam jezik. Ovo se može videti tako što se interpreteru zada komanda "import this". Tom prilikom na ekranu se prikaže kompletan manifest jezika kao i osnovne ideje i vrednosti istog.

Osnovne prednosti jezika jesu jasnost, jednostavnost, intuitivnost, konciznost i ekspresivnost, kao i izuzetno jaka i aktivna zajednica. Dodatno sjajne biblioteke koje implementiraju mnogobrojne funkcionalnosti. Cena svega ovoga jeste efikasnost, često su Python programi dosta sporiji od osnovnih konkurenata. Međutim, na ovome se aktivno radi, pišu se biblioteke u Pythonu koje su jako efikasne (primer *numpy*) i koje umnogome popravljaju efikasnost rada u Pythonu.

Detaljnije poređenje može se videti na linku: https://benchmarksgame-team.pages.debian.net/benchmarksgame/

1.2 Uvod, funkcije, kolekcije, moduli

1.2.1 Uvodni primeri

Zadatak 1.1 Napisati program koji na standardni izlaz ispisuje poruku *Hello world!*.

```
# Ovako se pisu komentari
#
# Pokretanje programa iz terminala:
#  $python  hello.py

"""
  Tradicionalni program koji na standardni izlaz ispisuje Zdravo svete.
  Ilustruje prednosti programskog jezika Python.
"""
print("Hello world.")
```

Zadatak 1.2 Napisati program koji za uneta dva cela broja i nisku ispisuje najpre unete vrednosti, a zatim i zbir brojeva, njihovu razliku, proizvod i količnik.

```
# Skup (Set)
imena = set()
for ime in ['Milan', 'Ivan', 'Milan', 'Ana', 'Ana', 'Milan']:
    imena.add(ime)
print(imena)
print(set(['Milan', 'Ivan', 'Milan', 'Ana', 'Ana', 'Milan']))


# Stek (LIFO struktura)
# Funkcije:
# * empty() - da li je stek prazan? - Time Complexity : O(1)
# * size() - čveliina steka? - Time Complexity : O(1)
# * top() - ćVraa referencu na element sa vrha steka - Time Complexity : O(1)
# * push(g) - Dodaje element g na vrh steka - Time Complexity : O(1)
# * pop() - Skida element sa vrha steka - Time Complexity : O(1)
#
# Implementacije: collections.deque, queue.LifoQueue etc. ili jednostavno koristiti
    listu!
stack = []
stack.append('a')
stack.append('b')
stack.append('c')
print('Inicijlani stek:', end=':')
print(stack)
print('pop():')
print(stack.pop())
print(stack.pop())
print(stack.pop())
print('Stanje steka:', end=':')
print(stack)

# Mapa (Dictionary)
# Notacija: {kljuc:vrednost, kljuc:vrednost, ...} - JSON format
prosek = {'Milan':9.45, 'Ana':9.87, 'Nikola':6.76}
print(prosek)
print(type(prosek))
print(prosek.keys())
print(prosek.values())
print('Milan' in prosek)

# Torke (Tuple)
# Torka predstavlja strukturu podataka koja je uredjena i imutabilna
torka = ("jabuka", "banana", "visnja", "pomorandza", "kivi", "lubenica", "mango")
print(torka[2:5])
try:
    torka[0] = 11
except TypeError:
    print("Torke su imutabilne.")
```

Zadatak 1.3 Napisati program koji sa standardnog ulaza ucitava jedan string i jedan ceo broj, a zatim ispisuje (dinamicki određene) tipove ucitanih promenljivih. Nakon toga, na standardni izlaz ispisati poruku `Ispisivanje vise stringova` koriscenjem separatora =, <-> i novog reda, redom.

```
# Hip (red sa prioritetom)
# https://docs.python.org/3.7/library/heapq.html
# Funkcije:
# * heapify - Konvertuje listu u hip u mestu.
# * heappush - Dodaje element u hip.
# * heappop - Vraca koren hipa.
# * heapreplace - Zamenjuje koren hipa novom vrednoscu prosledjenoj funckiji.
import heapq

H = [21,1,45,78,3,5]
heapq.heapify(H)
print(H)
heapq.heappush(H,8)
print(H)

```

```
# Heapsort
def heapsort(iterable):
    h = []
    for value in iterable:
        heapq.heappush(h, value)
    return [heapq.heappop(h) for i in range(len(h))]
print(heapsort([1, 3, 5, 7, 9, 2, 4, 6, 8, 0]))
```

Zadatak 1.4 Napisati program koji sa standardnog ulaza ucitava ceo broj n, a zatim na standardni izlaz ispisuje poruku o ostatku koji taj broj daje kada se podeli sa pet. Nakon toga, koristeci for i while petlju ispisati prvih n prirodnih brojeva.

```
# Grananje
print("if/elif: ")
x = int(input("Unesi jedan broj: "))
if x%5==0:
    print("Unet je broj deljiv sa 5.")
elif x%5==1 or x%5==2:
    print("Uneti broj daje ostatak 1 ili 2 pri deljenju sa 5.")
elif x%5==3:
    print("Uneti broj daje ostatak 3 pri deljenju sa 5.")
else:
    print("Preostaje da uneti broj daje ostatak 4 pri deljenju sa 5.")

# While petlja
print("while: ")
i = 0
while i < 10:
    print(i, end=' ')
    i += 1  # i++ nije podrzano

# For petlja
print("for: ")
for i in range(10):
    print(i, end=' ')
```

Zadatak 1.5 Napisati program koji sa standardnog ulaza ucitava ceo broj, koji je otporan na pogresan ulaz. Nakon toga, ilustrovati imutabilnost stringova.

```
# Try catch
try:
    n = int(input("Unesite ceo broj: "))
except ValueError:
    print("Greska. Parsiranje celog broja.")
    exit(1)

# Imutabilnost stringova
x = "Hello world"
print("x: ", x, id(x))
y = "Hello world"
print("y: ", y, id(y))
print(x==y)
#x[0] = '7'  # Type Error
#print("x: ", x, id(x))
```

Zadatak 1.6 Ako je prvi dan u mesecu ponedeljak napisati funkciju `radni_dan(dan)` koja kao argument dobija dan u mesecu i vraća tačno ako je dan radni dan. Napisati program koji testira ovu funkciju, korisnik sa standardnog ulaza u petlji unosi deset dana i dobija o poruku o tome da li su uneti dani radni ili ne.

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

def radni_dan(broj):
    # Osnovne logicke operacije:
        # not, and, or

```

```
        if broj % 7 == 1 or broj % 7 == 2 or broj % 7 == 3:
            return True
        elif broj % 7 == 4 or broj % 7 == 5:
            return True
        # Naredbi <<elif>> moze biti vise
        else:
            return False

# Blokovi se ne ogranicavaju viticastim zagradama kao sto je u C-u
# vec moraju biti indentovani. Naredbe na istom nivou indentacije se
# smatraju istim naredbama istog bloka

i = 0
while i <= 10:
    dan = input("Unesite dan: ")
    try:
      dan = int(dan)
    except ValueError:
      print("Greska. Parsiranje rednog broja dana.")
      exit(1)
    if radni_dan(dan):
        print("Uneti dan je radni dan.")
    else:
        print("Uneti dan je neradan.")
    i = i + 1 # i++ ne postoji, moze ili ovako ili i += 1

# Naredba <<break>> iskace iz bloka, isto kao i u C-u
```

Zadatak 1.7 Napisati program koji ilustruje osnovne sistemske funkcije i module programskog jezika Python tako što koristi funkcije faktorijel, logaritam (math), generise pseudoslucajne brojeve (modul random), ispisuje argumente komandne linije na izlaz (modul sys), poziva sistemski poziv listanja tekuceg direktorijuma (modul os), ispisuje broj sekundi od 1.1.1970. godine , finalno unosi string sa ulaza za koji proverava (korišćenjem regularnih izraza) da li predstavlja zapis celog broja.

```
"""
    Python moduli.
"""

# U <<math>> modulu se nalaze brojne naprednije matematicke funkcije kao sto su:
# math.sqrt(broj)
# math.log(broj, osnova)
# math.sin(ugao_u_radijanima), math.cos(), ...
# math.exp(stepen)
# math.factorial(broj)
import math
print(math.factorial(23))
print(math.log(225, 10))

# Pseudo slucajni brojevi nalaze se u modulu <<random>>
import random
print(random.random())       # [0.0, 1.0)
print(random.randint(0,1))   # {0,1,...,9,10}

# Preko <<sys>> modula se moze pristupiti objektima koje interpreter koristi,
# kao sto su npr argumenti komandne linije
import sys
print(sys.argv)

# U <<os>> modulu se nalaze, izmedju ostalog, sistemski pozivi
import os
os.system('ls -lah')

# Alias za modul
import time as t
print(t.time())

# Selektovani import
from re import search as re_src
print(re_src(r'[0-9]+', input('Unesite string za match: ')))
```

Zadatak 1.8 Napisati program koji na standardni izlaz ispisuje vrednost 6!, $\log_5 125$ i pseudo

slučajan broj iz opsega $[0, 1)$

```
# 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>>
import random

# Funkcija random() vraca pseudo slucajan broj tipa float iz opsega [0.0, 1.0)
print("\n------Pseudo slucajni brojevi------\n")
print("Pseudo slucajan broj iz opsega [0.0,1.0)\n")
print(random.random())

# Korisne funkcije:
#
# randint(a,b) - vraca pseudo slucajan ceo broj n iz opsega [a,b]
# choice(lista) - vraca pseudo slucajan element iz liste
```

Zadatak 1.9 Napisati program koji definiše praznu (bez tela) funkciju tri argumenta, kvadratnu funkciju, kao i funkciju translacije, koja prosleđeni argument umanjuje za jedan. Ilustrovati kompoziciju kvadratne funkcije i translacije. Funkciju kvadriranja definisati kao anonimnu funkciju.

```
"""
    Python funkcije.
"""

# Sintaksa:
# def name(args):
#   body

# Definicija funkcije (prazna funkcija)
def empty(x, y, z):
    pass
print(empty(1,2,3))

# Definicija anonimne funkcije (funkcija sabiranja)
sum = lambda x, y: x + y
print(sum(1,2))
print(sum)

def square(x):
    """ Vraca kvadrat svog argumenta """
    return x**2

print(square.__repr__())
print(help(square))
x = 9
print(square(x))
x = 9.3
print(square(x))

square = lambda x : x**2
print(square)
print(square(7))
```

Zadatak 1.10 Ilustrovati prenos argumenata funkciji: definisati dve funkcije koje rade sa listom, jednu koja menja, a drugu koja pravi kopiju prosleđenog joj argumenta.

```
# Prenos argumenata funkciji
# Postoje dva tipa objekata u Python-u: mutabilni i imutabilni
# - Za imutabilne objekte, menjanje unutar funkcije kreira novu instancu
#   i originali se ne menjaju. Imutabilni objekti su niske, brojevi i torke.
# - Za mutabilne objekte, svaka izmena unutar funkcije menja original.
#   Medjutim, ponovna dodela instance nece promeniti instancu izvan funkcije.
#   Mutabilni objekti su (izmedju ostalih): liste, mape i klasne instance.

def translate(x):
    x = x-1
    return x

a = 5
print('a:', a)
translate(a)
print('a:', a)

l = [1, 2, 3]
def translate(l):
    l = l[:-1]
print('l: ', l)
translate(l)
print(l)

l = [1, 2, 3]
def translate(l):
    l[0] = "M"
print('l: ', l)
translate(l)
print(l)

x = 17
print("x=",x," id=",id(x))
def ref_demo(x):
    print("x=",x," id=",id(x))
    x=42
    print("x=",x," id=",id(x))
ref_demo(x)

x = 17
print("x=",x," id=",id(x))
def ref_demo1(x):
    print("x=",x," id=",id(x))
    x -= 1
    print("x=",x," id=",id(x))
ref_demo1(x)
```

Zadatak 1.11 Definisati funkciju printf, koja na standardni izlaz stampa svoje argumente (ne praviti nikakve pretpostavke o broju argumenata). Modifikovati definisanu funkciju tako da kao argumente prima proiyvoljan imenovanih argumenata.

Definisati funkciju koja racuna aritmeticku sredinu svojih argumenata (pretpostaviti da je nad argumentima definisane operacije deljenja i sabiranja, kao i da će broj samih argumenata biti veći od jedan.

```
# Funkcije sa nepoznatim brojem argumenata
def printf(*x):
    print(x)
printf()
printf(14)
printf([1,2,3])
printf(1,2,3,translate, printf)

def aritm_sredina(head, *tail):
    sum = head
    for i in tail:
        sum += i
    return sum / (1.0 + len(l))
print(aritm_sredina(1,2,3,4,5))
```

```
# Proizvoljan broj imenovanih argumenata
def printf(**x):
    print(x)
printf()
printf(de="German",en="English",fr="French")

# Lokalne funkcije
def average(a, b, c):
    def total(a, b, c):
        return a+b+c
    return total(a,b,c)/3
print(average(1, 2, 3))

# REFERENCE:
# https://www.python-course.eu/passing_arguments.php
# https://www.zentut.com/python-tutorial/advanced-python-function/
```

Zadatak 1.12 Napisati program koji sa standardnog ulaza učitava listu celih brojeva, a zatim na standardni izlaz ispisuje njenu dužinu, sumu njenih elemenata, njen maksimalni element kao i broj pojavljivanja broja jedan u unetoj listi.

```
# Lista (List)
l = [1, 2, 3, 4, 5]
print(l, type(l))
l += [-3, -3, -3]
print(l)
print('len:', len(l))
print('sum:', sum(l))
l.append(-6)
print(l)
l.pop()
print(l)
l = [1, 2, "Matf", 12.3, [1, 2, 3], [[1,2], [3,4]], 1]
print(l)
print(l[2])
print(l[-2])
print(l[2:])
print(l[:-2])
print(l.count(1))
l1 = [2, 3, 34, 341, 2, -23, 213]

l = list(input())
print('len:', len(l))
print('sum:', sum(l))
print('max: ', max(l1))
print(l.count(1))

# Poredjenje razlicitih tipova
lista = [[1,2,3], [1,2,5], ['abc','abc','abc'], ['abc', 'ab', 'abcd'], ['a','b','c']
    > ['a', 'b']]
try:
    print(max(lista))
except TypeError as err:
    print("Greska. Python3 ne poredi razlicite tipove: ", err)
```

Zadatak 1.13 Napisati program kojim se ilustruje upotreba struktura podataka: steka, skupa, mape i torke.

```
# Skup (Set)
imena = set()
for ime in ['Milan', 'Ivan', 'Milan', 'Ana', 'Ana', 'Milan']:
    imena.add(ime)
print(imena)
print(set(['Milan', 'Ivan', 'Milan', 'Ana', 'Ana', 'Milan']))


# Stek (LIFO struktura)
# Funkcije:
# * empty() - da li je stek prazan? - Time Complexity : O(1)
# * size() - čveliina steka? - Time Complexity : O(1)
```

```
# * top() - ćVraa referencu na element sa vrha steka - Time Complexity : O(1)
# * push(g) - Dodaje element g na vrh steka - Time Complexity : O(1)
# * pop() - Skida element sa vrha steka - Time Complexity : O(1)
#
# Implementacije: collections.deque, queue.LifoQueue etc. ili jednostavno koristiti
    listu!
stack = []
stack.append('a')
stack.append('b')
stack.append('c')
print('Inicijlani stek:', end=':')
print(stack)
print('pop():')
print(stack.pop())
print(stack.pop())
print(stack.pop())
print('Stanje steka:', end=':')
print(stack)

# Mapa (Dictionary)
# Notacija: {kljuc:vrednost, kljuc:vrednost, ...} - JSON format
prosek = {'Milan':9.45, 'Ana':9.87, 'Nikola':6.76}
print(prosek)
print(type(prosek))
print(prosek.keys())
print(prosek.values())
print('Milan' in prosek)

# Torke (Tuple)
# Torka predstavlja strukturu podataka koja je uredjena i imutabilna
torka = ("jabuka", "banana", "visnja", "pomorandza", "kivi", "lubenica", "mango")
print(torka[2:5])
try:
    torka[0] = 11
except TypeError:
    print("Torke su imutabilne.")
```

Zadatak 1.14 Korišćenjem hipa implementirati algoritam sortiranja hip-sort.

```
# Hip (red sa prioritetom)
# https://docs.python.org/3.7/library/heapq.html
# Funkcije:
# * heapify - Konvertuje listu u hip u mestu.
# * heappush - Dodaje element u hip.
# * heappop - Vraca koren hipa.
# * heapreplace - Zamenjuje koren hipa novom vrednoscu prosledjenoj funckiji.
import heapq

H = [21,1,45,78,3,5]
heapq.heapify(H)
print(H)
heapq.heappush(H,8)
print(H)

# Heapsort
def heapsort(iterable):
    h = []
    for value in iterable:
        heapq.heappush(h, value)
    return [heapq.heappop(h) for i in range(len(h))]
print(heapsort([1, 3, 5, 7, 9, 2, 4, 6, 8, 0]))
```

Zadatak 1.15 Napisati program koji imitira rad bafera. Maksimalni broj elemenata u baferu je 5. Korisnik sa standardnog ulaza unosi podatke do unosa reči *quit*. Program ih smešta u bafer, posto se bafer napuni unosi se ispisuju na standarni izlaz i bafer se prazni.

```
# LISTA
#
# Notacija: [element1, element2, ...]
#
# Liste mogu sadrzati razlicite tipove podataka
```

```
# lista = [1,2,3.4,"Another brick in the wall",True,[5,False,4.4,'Layla']]

# Prazna lista
buffer = []
i = 0
while True:
    unos = input()
    if unos == 'quit':
        break
    # Ubacivanje elementa na kraj
    buffer.append(unos)
    i += 1
    if i == 5:
        # Prolazak kroz listu
        for unos in buffer:
            print(unos)
        # Praznimo bafer
        buffer = []
        i = 0
```

Zadatak 1.16 Korisnik sa standardnog ulaza unosi ceo broj n, a potom ciklično pomeren rastuće sortiran niz (pr. 56781234) koji ima n elemenata. Napisati program koji na standarni izlaz ispisuje sortiran niz bez ponavljanja elementa.

```
# Korisne funkcije za liste:
#
# list.remove(x) - izbacuje prvo pojavljivanje elementa x iz liste
# list.count(x) - vraca broj koliko puta se element x nalazi u listi
# list.index(x) - vraca indeks prvog pojavljivanja elementa x u listi
# len(lista) - vraca broj elemenata liste
# del lista[a:b] - brise elemente liste od pozicije a do b

try:
    n = int(input("Unesite broj elemenata: "))
except ValueError:
    print("Greska. Parsiranje broja elemenata.")
    exit(1)

elementi = []
# Funkcija <<range>>
#
# range(kraj)
# range(pocetak, kraj[, korak])

for i in range(n):
    element = int(input())
    # Provera da li se element nalazi u listi
    if not element in elementi:
        # Ubacivanje elementa u listi
        elementi.append(element)

k = 0
for i in range(n-1):
    # Pristupanje elementima liste
    if elementi[i] > elementi[i+1]:
        k = i + 1
        break

prvi_deo = elementi[0:k]
drugi_deo = elementi[k:]

# Nadovezvivanje dve liste
sortirani = drugi_deo + prvi_deo
print("Sortirani elementi: ")
for element in sortirani:
    print(element)
```

Zadatak 1.17 Napisati funkciju `max_list(lista)` koja vraća najveći element u listi listi. Napisati program koji testira ovu funkciju.

```
def max_list(lista):
```

```
    # Mozemo indeksirati liste unazad, pozicija -1 odgovara poslednjem elementu
    maximum = lista[-1]
    for element in lista:
        if maximum < element:
            maximum = element
    return maximum

lista = [1, 4, -6, 7, 9, 0, 1]
print(max_list(lista))
```

Zadatak 1.18 Napisati program za rad sa stekom.

- Definisati stek koji sadrži elemente $9, 8, 7$
- Dodati na stek elemente 6 i 5
- Skinuti sa steka element i ispisati ga na standardni izlaz

```
# 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 1.19 Napisati program koji za uneti prirodan broj n ispisuje vrednosti funkcije x^2 u celobrojnim tačkama u intervalu $[0, n]$. Zadatak rešiti korišćenjem mape.

```
# KATALOG
#
# Katalog je kolekcija uredjenih parova oblika (kljuc, vrednost)
#
# Notacija: {kljuc:vrednost, kljuc:vrednost, ...}

mapa = {} # prazna mapa

try:
    n = int(input("Unesite n: "))
except ValueError as err:
    print("Greska. Parsiranje broja elemenata: ", err)
    exit(1)
for x in range(n):
    mapa[x] = x**2

# Prolazak kroz mapu
print("\n------Prolazak kroz katalog------\n")
for kljuc in mapa:
    print("f({0:s}) =  {1:s}\n".format(str(kljuc),str(mapa[kljuc])))

# Korisne funkcije
#
# map.keys() - vraca listu kljuceva iz kataloga
# map.values() - vraca listu vrednosti iz kataloga
# map.has_key(kljuc) - vraca True/False u zavisnosti od toga da li se element
# sa kljucem kljuc nalazi u katalogu
```

Zadatak 1.20 Sa standardnog ulaza se unose reči do reči *quit*. Napisati program koji ispisuje unete reči eliminišući duplikate.

```
lista = []

while True:
    unos = input()
```

```
    if unos == 'quit':
        break
    lista.append(unos)

# Pravljenje skupa od liste
skup = set(lista)

for i in skup:
    print(i)
```

Zadatak 1.21 Napisati funkciju `min_torka(lista)` koja vraća najmanji element u torci torki. Napisati program koji ovu funkciju testira.

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

def min_torka(torke):
    # Pristupanje elementima u torki
    minimum = torke[0]
    for torka in torke:
        # Ukoliko torke ne sadrze elemente istog tipa na istim pozicijama, i dalje ih
     mozemo porediti,
        # ali poredjenje se vrsi na osnovu imena tipa elementa leksikografski
        # npr.  element tipa List < element tipa String < element tipa Tuple i slicno
        if minimum > torka:
            minimum = torka
    return minimum

torke1 = ((1, 2, 'a'), (1, 2, 'b'), (1, 3, 'z'), (2, 2, '5'))
print(min_torka(torke1))

torke2 = ((1,2,'a'),(1,2,'b'),([1,2,3], 'Bugs', 4), ([1,1,1], 'Bunny', 6),(1,2,['a','
    b']),(1,2,'ab'))
try:
    print(min_torka(torke2))
except TypeError as err:
    print("Greska. Python3 ne poredi razlicite tipove: ", err)
    exit(1)
```

1.2.2 Zadaci za samostalni rad sa rešenjima

Zadatak 1.22 **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.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
POKRETANJE: pogodi_broj
 -------- IGRA: Pogodi broj ---------
 Unesite Vase ime:
 Milica
 Zdravo Milica. :)
 Zamislio sam neki broj od 1 do 100. Da li mozes da pogodis koji je to broj?
 Unesi broj
 50
 Broj koji sam zamislio je MANJI od 50
 Unesi broj
 25
 Broj koji sam zamislio je VECI od 25
 Unesi broj
 30
 Broj koji sam zamislio je MANJI od 30
 Unesi broj
 27
 "BRAVO!!! Pogodio si! Zamislio sam 27. Bilo je lepo igrati se sa tobom. :)
```

[Rešenje 1.22]

Zadatak 1.23 **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 $(0, 0), (0, 1), (1, 1), (1, 0)$.

Primer 1

```
INTERAKCIJA SA PROGRAMOM:
POKRETANJE: aproksimacija_PI
 Izracunavanje broja PI metodom Monte Karlo
 Unesite broj iteracija:
 5
 Tacka:
 (0.14186247318019474, 0.15644650897353152)
 Tacka:
 (0.8910898038304426, 0.2200563958299553)
 Tacka:
 (0.641604107090444, 0.03712366524007682)
 Tacka:
 (0.4893727376942526, 0.17230005349431587)
 Tacka:
 (0.6199558112390107, 0.32122922953511124)
 Tacka:
 (0.5821041171248978, 0.025052299437462566)
 Broj PI aproksimiran metodom Monte Karlo: 4.0
```

[Rešenje 1.23]

Zadatak 1.24 **X-O** Napisati program koji implementira igricu X-O sa dva igrača.

Primer 1

```
POKRETANJE: XO
INTERAKCIJA SA PROGRAMOM:
 IGRA: X-O pocinje
 Unesite ime prvog igraca:
 Ana
 Zdravo Ana
 Unesite ime drugog igraca:
 Petar
 Zdravo Petar!
 Igrac ('Ana', 'X') igra prvi.
 X : ('Ana', 'X')
 O : ('Petar', 'O')
 Zapocnimo igru
 TABLA
  1 2 3
  --- --- ---
 1 | - | - | - |
  --- --- ---
 2 | - | - | - |
  --- --- ---
 3 | - | - | - |
  --- --- ---
 Ana unesite koordinate polja koje
 zelite da popunite u posebnim linijama:
 Unesite vrstu:
 1
 Unesite kolonu:
 1
 TABLA
  1 2 3
  --- --- ---
 1 | X | - | - |
  --- --- ---
 2 | - | - | - |
  --- --- ---
 3 | - | - | - |
  --- --- ---
 Petar unesite koordinate polja koje
 zelite da popunite u posebnim linijama:
 Unesite vrstu:
 1
 Unesite kolonu:
 2
 TABLA
  1 2 3
  --- --- ---
 1 | X | O | - |
  --- --- ---
 2 | - | - | - |
  --- --- ---
 3 | - | - | - |
  --- --- ---
 Ana unesite koordinate polja koje
 zelite da popunite u posebnim linijama:
 Unesite vrstu:
 2
 Unesite kolonu:
 2
 TABLA
  1 2 3
  --- --- ---
 1 | X | O | - |
  --- --- ---
 2 | - | X | - |
  --- --- ---
 3 | - | - | - |
  --- --- ---
 Petar unesite koordinate polja koje
 zelite da popunite u posebnim linijama:
 Unesite vrstu:
 2
 Unesite kolonu:
 3
 TABLA
  1 2 3
  --- --- ---
 1 | X | O | - |
  --- --- ---
 2 | - | X | O |
  --- --- ---
 3 | - | - | - |
  --- --- ---
 Ana unesite koordinate polja koje
 zelite da popunite u posebnim linijama:
 Unesite vrstu:
 3
 Unesite kolonu:
 3
 TABLA
  1 2 3
  --- --- ---
 1 | X | O | - |
  --- --- ---
 2 | - | X | O |
  --- --- ---
 3 | - | - | X |
  --- --- ---
 BRAVO!!!!!!! Igrac Ana je pobedio!
```

[Rešenje 1.24]

1.2.3 Zadaci za vežbu

Zadatak 1.25 **Ajnc** Napisati program koji implementira igricu Ajnc 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. Detaljan opis igre može se naći na narednoj adresi: https://en.wikipedia.org/wiki/Blackjack

Primer 1

```
Pokretanje: ajnc
Interakcija sa programom:
 -------- IGRA: Ajnc ---------
 Unesite Vase ime:
 Pavle
 Zdravo Pavle. :)
 Igra pocinje
 Vase karte su:
 1 Herc 5 karo
 Hit ili stand?[H/S]
 H
 Vase karte su:
 1 Herc 5 karo 5 tref
 Cestitam!!! Pobedio si!
 Bilo je lepo igrati se sa tobom. :)
```

Primer 2

```
Pokretanje: ajnc
Interakcija sa programom:
 -------- IGRA: Ajnc ---------
 Unesite Vase ime:
 Pavle
 Zdravo Pavle. :)
 Igra pocinje
 Vase karte su:
 Q Tref 7 karo
 Hit ili stand?[H/S]
 H
 Vase karte su:
 Q Tref 7 karo K Herc
 Zao mi je, izgubio si!:(
 Bilo je lepo igrati se sa tobom. :)
```

Zadatak 1.26 **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. Detaljan opis igre može se naći na narednoj adresi: https://en.wikipedia.org/wiki/Connect_Four

Primer 1

```
Pokretanje: cetri_u_nizu
Interakcija sa programom:
 IGRA: Cetiri u nizu pocinje
 Unesite ime prvog igraca:
 Ana
 Zdravo Ana
 Unesite ime drugog igraca:
 Petar
 Zdravo Petar!
 Igrac ('Ana', 'C') igra prvi.
 C : ('Ana', 'C')
 Z : ('Petar', 'Z')
 Zapocnimo igru
 TABLA
  1 2 3 4 5 6 7 8
  --- --- --- --- --- --- --- ---
 1 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 2 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 3 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 4 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 5 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 6 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 7 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 8 | - | - | - | - | - | - | - | - |
 Ana unesite koordinate polja
 koje zelite da popunite
 u posebnim linijama:
 Unesite vrstu:
 1
 Unesite kolonu:
 1
 TABLA
  1 2 3 4 5 6 7 8
  --- --- --- --- --- --- --- ---
 1 | c | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 2 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 3 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 4 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 5 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 6 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 7 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 8 | - | - | - | - | - | - | - | - |
 Petar unesite koordinate polja
 koje zelite da popunite
 u posebnim linijama:
 Unesite vrstu:
 2
 Unesite kolonu:
 1
 TABLA
  1 2 3 4 5 6 7 8
  --- --- --- --- --- --- --- ---
 1 | c | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 2 | z | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 3 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 4 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 5 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 6 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 7 | - | - | - | - | - | - | - | - |
  --- --- --- --- --- --- --- ---
 8 | - | - | - | - | - | - | - | - |
```

```
Ana unesite koordinate polja
koje zelite da popunite
u posebnim linijama:
Unesite vrstu:
1
Unesite kolonu:
2
TABLA
 1 2 3 4 5 6 7 8
 --- --- --- --- --- --- --- ---
1 | c | c | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
2 | z | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
3 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
4 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
5 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
6 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
7 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
8 | - | - | - | - | - | - | - | - |
Petar unesite koordinate polja
koje zelite da popunite
u posebnim linijama:
Unesite vrstu:
2
Unesite kolonu:
2
TABLA
 1 2 3 4 5 6 7 8
 --- --- --- --- --- --- --- ---
1 | c | c | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
2 | z | z | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
3 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
4 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
5 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
6 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
7 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
8 | - | - | - | - | - | - | - | - |
Ana unesite koordinate polja
koje zelite da popunite
u posebnim linijama:
Unesite vrstu:
1
Unesite kolonu:
3
TABLA
 1 2 3 4 5 6 7 8
 --- --- --- --- --- --- --- ---
1 | c | c | c | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
2 | z | z | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
3 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
4 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
5 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
6 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
7 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
8 | - | - | - | - | - | - | - | - |
```

```
Petar unesite koordinate polja
koje zelite da popunite
u posebnim linijama:
Unesite vrstu:
2
Unesite kolonu:
3
TABLA
 1 2 3 4 5 6 7 8
 --- --- --- --- --- --- --- ---
1 | c | c | c | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
2 | z | z | z | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
3 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
4 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
5 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
6 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
7 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
8 | - | - | - | - | - | - | - | - |
Ana unesite koordinate polja
koje zelite da popunite
u posebnim linijama:
Unesite vrstu:
1
Unesite kolonu:
4
TABLA
 1 2 3 4 5 6 7 8
 --- --- --- --- --- --- --- ---
1 | c | c | c | c | - | - | - | - |
 --- --- --- --- --- --- --- ---
2 | z | z | z | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
3 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
4 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
5 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
6 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
7 | - | - | - | - | - | - | - | - |
 --- --- --- --- --- --- --- ---
8 | - | - | - | - | - | - | - | - |
BRAVO!!!!!!! Igrac Ana je pobedio!
```

1.3 OOP koncepti

Zadatak 1.27 Korišćenjem liste implementirati klasu stek.

```
"""
    Python klase. Uvod. Konstruktor. Instanciranje. self promenljiva.
"""
class Stek:

    # konstruktor
    def __init__(self):
        self.stek = []

    def push(self, elem):
        self.stek.append(elem)

    def pop(self):
        try:
            return self.stek.pop()
        except IndexError:
            print("Stek prazan, nemoguce izvesti pop.")
            return None

    def peek(self):
        try:
            return self.stek[-1]
        except IndexError:
            print("Stek prazan, nemoguce izvesti peek.")
            return None

s = Stek()
s.push(1)
s.push(2)
s.push(3)
print(s.peek())
print(s.pop())
print(s.pop())
print(s.pop())
print(s.pop())
print(s.peek())
```

Zadatak 1.28 Implementirati klasu Robot, koja će biti svesna broja trenutno živih instanci ove klase. Klasa kao atribut ima naziv robota. Od metoda definisati metod oglašavanja robota, kao i statički metod koji na standardni izlaz ispisuje broj trenutno aktivnih instanci ove klase. Ilustrovati upotrebu klase.

```
"""
    Python klase. Destruktor, promenljive.
"""
class Robot:
    """Predstavlja robota sa imenom."""

    # Mozemo pristupiti dokumentaciji preko __doc__() metoda

    populacija = 0  # klasna promenljiva

    def __init__(self, name):
        """Konstruktor."""
        self.name = name
        Robot.populacija += 1
        print(f"{self.name} je kreiran!")

    def __del__(self):
        """Destruktor."""
        print(f"{self.name} se unistava!")
        Robot.populacija -= 1

    def oglasi_se(self):
        """Oglasavanje za trenutnog robota."""
        print(f"Zdravo, moje ime je {self.name}.")
```

```
    @classmethod
    def koliko(cls):
        """Ispisuje broj trenutno aktivnih robota."""
        print(f"Imamo {cls.populacija} aktivnih robota.")

    # postoji i @staticmethod, takve funkcije nemaju parametre
    # koriste se kada nije potrebno da metod pristupa atributima klase


droid1 = Robot("R2-D2")
droid1.oglasi_se()
Robot.koliko()

droid2 = Robot("C-3PO")
droid2.oglasi_se()
Robot.koliko()

print("\nRoboti rade posao ovde...\n")

print("Roboti su zavrsili posao. Obrisimo ih.")
del droid1
del droid2

Robot.koliko()

# REFERENCE:
# https://python.swaroopch.com/oop.html
```

Zadatak 1.29 Implementirati klasu Oblik, koja kao atribute ima boju kao i informaciju o popunjenosti unutrašnjosti oblika bojom. Definisati metode za dohvatanje i postavljanje ovih atributa. Definisati klase Krug i Pravougaonik koje nasleđuju klasu oblik, od dodatnih atributa imaju informaciju o poluprečniku odnosno stranicama redom, i implementiraju funkcije računanja obima i površine. Ilustrovati upotrebu ovih klasa.

```
"""
    Python klase. Nasledjivanje.
"""

import math

class Oblik:
    def __init__(self, boja='crna', popuna=False):
        # protected promenljive pocinju sa _
        # private promenljive pocinju sa __
        # Ovo je samo konvencija o imenovanju, ne postoje zaista privatne promenljive
     u Pythonu.
        # Radi se o kvazi privatnim, Python promenljivim koje čpoinju sa __ menja
    naziv interno,
        # i na taj čnain švri skrivanje imena.
        self.__boja = boja
        self.__popuna = popuna
    def get_boja(self):
        return self.__boja
    def get_popuna(self):
        return self.__popuna
    def set_popuna(self, popuna):
        self.__popuna = popuna


class Pravougaonik(Oblik):
    def __init__(self, a, b):
        super().__init__(boja='crvena')
        self.__a = a
        self.__b = b
    def povrsina(self):
        return self.__a * self.__b
    def obim(self):
        return 2 * (self.__a + self.__b)


class Circle(Oblik):
    def __init__(self, r):
```

```
        super().__init__(boja='plava')
        self.__r = r
    def povrsina(self):
        return math.pi * self.__r ** 2
    def obim(self):
        return 2 * math.pi * self.__r


r1 = Pravougaonik(10.5, 2.5)
print("Povrsina pravougaonika r1:", r1.povrsina())
print("Obim pravougaonika r1:", r1.obim())
print("Boja pravougaonika r1:", r1.get_boja())
print("Da li je r1 popunjen? ", r1.get_popuna())
r1.set_popuna(True)
print("Da li je r1 popunjen? ", r1.get_popuna())

c1 = Circle(12)
print("\nPovrsina kruga c1:", format(c1.povrsina(), "0.2f"))
print("Obim kruga c1:", format(c1.obim(), "0.2f"))
print("Boja kruga c1:", c1.get_boja())
print("Da li je c1 popunjen? ", c1.get_popuna())
c1.set_popuna(True)
print("Da li je c1 popunjen? ", c1.get_popuna())

"""
Visestruko nasledjivanje:
class A:
    def __init__(self):
        super(A, self).__init__()
        print('A')
class B:
    def __init__(self):
        super(B, self).__init__()
        print('B')
class C:
    def __init__(self):
        super(C, self).__init__()
        print('C')
class X(A, B, C):
    def __init__(self):
        super(X, self).__init__()
x = X()
"""

"""
Virtualne funkcije:
class A:
    def metod(self):
        print("metod() iz klase A")
class B(A):
    def metod(self):
        print("metod() iz klase B")
b_obj = B()
a_obj = A()
b_obj.metod()
a_obj.metod()
"""

"""
Predefinisanje operatora:
class Tacka:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
    def __str__(self):
        return "({0},{1})".format(self.x,self.y)
    def __add__(self,other):
        x = self.x + other.x
        y = self.y + other.y
        return Tacka(x,y)
p1 = Tacka(2,3)
p2 = Tacka(-1,2)
print(p1 + p2)
```

```
"""

# REFERENCE:
# https://overiq.com/python-101/inheritance-and-polymorphism-in-python/
```

Zadatak 1.30 Implementirati klasu MyIterator koja predstavlja iterator od 0 do n, pri čemu je n atribut klase.

```
"""
    Iteratori.
"""
for i in [1,2,3,4]:
    print(i)
for i in ('a', 'b', 'c', 'd'):
    print(i)

# Da bi Python objekat bio iterabilan, potrebno je da implementira ćsledee metode:
# * __iter__ metod koji se poziva prilikom inicijalizacije iteratora
#            ćvraa objekat koji implementira __next__ metod.
# * __next__ metoda ćvraa ćsledeu vrednost iteratora.
#            Kada se iterator objekat nalazi u okviru ‘for ’in petlje, petlja
    implicitno
#            poziva __next__() metod na iterator objektu.
#            Metod žpodie izuzetak StopIteration kao signal kraja iteriranja.

class MojIterator:
    def __init__(self, limit):
        self.limit = limit

    def __iter__(self):
        self.x = 0
        return self

    def __next__(self):
        # Treba nam trenutna vrednost x
        x = self.x
        if x > self.limit:
            raise StopIteration
        else:
            self.x += 1
            return x

for i in MojIterator(10):
    print(i)

# Ovo se prevodi u (priblizno, naravno):
# foo = MojIterator(10)
# foo = foo.__iter__()
# try:
#     while (True):
#         i = foo.__next__()
#         print(i)
# except StopIteration:
#     pass
```

Zadatak 1.31 Implementirati generator Fibonačijevih brojeva.

```
def Fibonacci():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

f = Fibonacci()
for i in range(10):
    print(f, f.__next__())
```

1.4 Datoteke, serijalizacija, sortiranje

Serijalizacija predstavlja proces prevođenja struktura podataka ili objekata u format koji može da se čuva ili prenosi, takav da je moguće rekonstruisati polaznu strukturu/objekat. Da bismo demonstrirali serijalizaciju, prvo ćemo pokazati kako se radi sa datotekama u Python-u, a zatim ćemo serijalizovati objekte u datoteke u JSON formatu.

1.4.1 Datoteke

Zadatak 1.32 Korisnik na standardni ulaz unosi dve niske. Napisati program koji prvo pojavljivanje druge niske u prvoj zamenjuje karakterom $. U slučaju da nema pojavljivanja druge niske u prvoj i da je druga niska kraća ispisuje nadovezane niske sa separatorom -. Ako je druga niska duža od prve program treba da ispiše drugu nisku i njenu dužinu.

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

niska1 = input("Unesite nisku: ")
niska2 = input("Unesite nisku: ")

# Duzinu niske racunamo koristeci funkciju len(niska)
if len(niska1) >= len(niska2):
    # Funkcija count
    # niska.count(podniska [, pocetak [, kraj]]) - vraca broj koliko se puta
    # podniska nalazi u niski (u intervalu od pocetak do kraj)
    n = niska1.count(niska2)
    if n > 0:
        # 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
        i = niska1.find(niska2)
        # Karakterima u niski mozemo pristupati koristeci notaciju [] kao kod listi
        niska1 = niska1[0 : i] + '$' + niska1[i+len(niska2) : ]
        print(niska1)
    else:
        # Funkcija join
        # niska_separator.join([niska1,niska2,niska3,...]) - spaja listu niski
    separatorom
        print('-'.join([niska1,niska2]))
else:
    print("Duznina niske {0:s} je {1:d}".format(niska2, len(niska2)))

# 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 1.33 Napisati program koji ispisuje sadrvzaj datoteke *datoteka.txt* na standardni izlaz karakter po karakter.

```
# Datoteku otvaramo koristeci funkciju open koja vraca
# referencu na otvoreni tok podataka.
#
# rezimi:
# - "r"  -> read
# - "w"  -> write
# - "a"  -> append
# - "r+" -> read + append
#
```

```
# Datoteku smo duzni da zatvorimo sa 'datoteka.close()',
#
# datoteka = open("datoteka.txt","r")
# kod koji obradjuje datoteku
# ...
# datoteka.close()

# Python nudi 'with' koji omogucava da
# se datoteka automatski zatvori, cak i u situaciji
# kada se ispali izuzetak. Ovo je preporuceni nacin
# za citanje tokova podataka u Python-u.
with open("datoteka.txt", "r") as datoteka:
    # Citamo datoteku liniju po liniju, a potom
    # u liniji citamo karakter po karakter.
    for linija in datoteka:
        for karakter in linija:
            print(karakter)
    # datoteka.close() nam nije neophodno,
    # Python ce sam zatvoriti datoteku kada
    # se zavrsi 'with' blok
```

Zadatak 1.34 Napisati program koji ispisuje sadrvzaj datoteke *datoteka.txt* na standardni izlaz liniju po liniju.

```
# Ispitivanje da li je otvaranje datoteke uspelo:
try:
    with open("datoteka.txt","r") as g:
        # Liniju po liniju mozemo ucitavati koristeci petlju
        # tako sto 'iteriramo' kroz Datoteku
        print("-----Iteriranje kroz datoteku <<for>> petljom-----\n")
        # Metod f.readline() cita jednu liniju iz Datoteke
        for linija in g:
            print(linija)
except IOError:
    # Ukoliko ne uspe otvaranje datoteke, Python ispaljuje
    # izuzetak 'IOError'.
    print("Nije uspelo otvaranje datoteke.")
```

Zadatak 1.35 Napisati program koji dodaje u datoteku *datoteka.txt* nisku *water* a potom ispisuje njen sadržaj na standardni izlaz.

```
# f.readlines() i list(f)
# vracaju listu linija datoteke
#
# f.write(niska) upisuje nisku u datoteku
print("-----Upisivanje u datoteku-----\n")
# razresiti konfuziju izmedju a+ (upisuje na kraj) i r+ (upisuje na pocetak, tj.
    prepisace postojeci sadrzaj)
with open("datoteka.txt","a+") as h:
    h.write("water\n")
    # h.flush() - da budemo sigurni da je prebaceno iz bafera
    # nakon write file pointer se pomerio pa da bismo procitali ceo sadrzaj vracamo
    se na pocetak datoteke
    h.seek(0,0)
    print(h.readlines())
```

1.4.2 Serijalizacija, JSON

JSON (https://www.json.org/) reprezentacija objekata predstavlja jednostavni i pregledan način za serijalizaciju objekata. Koristi se svuda, pre svega u web programiranju kada je potrebno proslediti objekte preko mreže ali i lokalno kada je na primer potrebno sačuvati neki objekat u bazi podataka. JSON je nezamenjiv kod takozvanih *RESTful servisa* (servisa zasnovanih na REST-u, videti https://en.wikipedia.org/wiki/Representational_state_transfer). Korisnici obično pošalju zahtev preko HTTP protokola, a servis odgovara JSON reprezentacijom objekta koji ima informacije koje je korisnik tražio.

Zadatak 1.36 Korisnik na standarni ulaz unosi podatke o imenu, prezimenu i godinama. Program potom kreira JSON objekat *junak*, koji ima podatke *Ime*, *Prezime* i *Godine*, i ispisuje ga na standardni izlaz, a potom i u datoteku *datoteka.txt*.

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

ime = input("Unesite ime: ")
prezime = input("Unesite prezime: ")
godine = int(input("Unesite godine: "))

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

print("\n-----JSON reprezentacija objekta-----\n")
junak = {"Ime": ime, "Prezime":prezime, "Godine":godine}
print(json.dumps(junak))

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

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

Zadatak 1.37 Napisati program koji iz datoteke *datoteka.txt* učitava JSON objekat, a potom na standardni izlaz ispisuje podatke o *imenu*, *prezimenu* i *godininama*.

```
import json
# json.load(f) ucitava iz datoteke string koji sadrzi
# JSON format objekta i vraca referencu na mapu koja
# je konstruisana na osnovu .json datoteke.
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()
```

Zadatak 1.38 Napisati program koji od RESTful servisa `https://quotes.rest` uzima citat dana pomoću GET zahteva na `https://quotes.rest/qod` i ispisuje citat na standardni izlaz. Napomena: Kako bi kod radio za Python2.7 potrebno je instalirati modul *requests* (`pip install requests`, *pip* je Python modul menadžer, ukoliko nije instaliran može se instalirati pomoću `apt-get install python-pip` ili bilo kod drugog package-managera).

```
import requests

url = 'https://quotes.rest/qod'
response = requests.get(url)

if response.ok:
    json = response.json()
    print(json)
    citat = json['contents']['quotes'][0]['quote']
    print("---------------")
    print(citat)
else:
    print("Neuspesno dovlacenje citata.")
```

1.4.3 os modul

Zadatak 1.39 Napisati program koji na standarni izlaz ispisuje argumente 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)
```

Zadatak 1.40 Napisati program koji na standrani izlaz ispisuje oznaku za tekući i roditeljski direktorijum, kao i separator koji se koristi za pravljenje putanje.

```
import os

print(os.curdir)      # oznaka tekuceg direktorijuma
print(os.pardir)      # oznaka za roditeljski direktorijum tekuceg direktorijuma
print(os.sep)         # ispisuje separator koji koristi za pravljenje putanja
```

Zadatak 1.41 Napisati program koji imitira rad komande *ls*. Program na standardni izlaz ispisuje sadržaj tekućeg direktorijuma.

```
import os

# funkcija za prosledjenu putanju direktorijuma vraca listu imena
# svih fajlova u tom direktorijumu, . je zamena za putanju tekuceg direktorijuma
print(os.listdir(os.curdir))

# 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(os.curdir):
    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
```

Zadatak 1.42 Napisati program koji na standarni izlaz ispisuje sve apsolutne putanje regularnih fajlova koji se nalaze u tekućem direktorijumu.

```
import os

print("\n-----Regularni fajlovi zadatog direktorijuma-----\n")
for ime in os.listdir(os.curdir):
    # Funkcija 'join' vrsi konkatenaciju putanja koristeci sistemski separator
    if os.path.isfile(os.path.join(os.curdir, ime)):
        print(os.path.abspath(os.path.join(os.curdir, ime)))
```

1.4.4 Sortiranje

Zadatak 1.43 U datoteci *tacke.json* se nalaze podaci o tačkama u sledećem formatu.

Listing 1.1: *tacke.json*

```
[ {"teme":"A" , "koordinate": [10.0, 1.1]},
```

```
2 {"teme":"B" , "koordinate": [1.0, 15.0]},
3 {"teme":"C" , "koordinate": [-1.0, 5.0]} ]
```

Napisati program koji učitava podatke o tačkama iz datoteke *tacke.json* i sortira i po udaljenosti od koordinatnog početka. Na standarni izlaz ispisati podatke pre i posle sortiranja.

```
# Sortiranje
#
# sorted(kolekcija [, kljuc [, obrni]]) - vraca sortiranu kolekciju
#
# kolekcija - kolekcija koju zelimo da sortiramo
# kljuc - funkcija koja vraca kljuc po kome se poredi
# obrni - True/False (opadajuce/rastuce)
#
# Ukoliko zelimo da sortiramo po odredjenoj funkciji, mozemo iskoristiti
# funkciju iz `functools` modula `cmp_to_key()` koja ce prosledjenu
# funkciju "transformisati" u argument kljuc po kojem se poredi.
# Funkcija koja se prosledjuje funkciji cmp_to_key() ima potpis:
# foo(x, y) -> integer
# i vraca negativnu vrednost ako je x < y, 0 za x == y i pozitivnu vrednost za x > y
#
# primer:
#
# def foo(x, y):
#     return x + y
# sorted(kolekcija, key=cmp_to_key(foo))
#
# napomena: foo se prosledjuje bez zagrada jer se prosledjuje objekat koji
# predstavlja funkcija a ne povratna vrednost funkcije (zagrade znace da se
# funkcija evaluira za date argumente)

import json
import math

# l = ["A", "C", "D", "5", "1", "3"]
# print(l)
# print("sortirana lista: ", sorted(l))  # definisan operator < nad niskama

# U sledecem primeru je neophodno da definisemo svoje funkcije za vracanje kljuca po
    kom se poredi jer je kolekcija lista recnika i nad njima nije definisan operator
    '<'
with open("tacke.json","r") as f:
    tacke = json.load(f)

# Za definisanje kljuca poredjenja tacaka iskoristicemo anonimnu (lambda) funkciju (
    na mala vrata uvodimo ono sto cemo raditi detaljnije u programskom jeziku Haskell
    )
sortirane_tacke = sorted(tacke, key=lambda x: math.sqrt(x['koordinate'][0]**2 + x['
    koordinate'][1]**2))  # Optimizacija: bez .sqrt; reverse=True za obratan redosled
print("Tacke pre sortiranja:")
for item in tacke:
        print(item["teme"],)
print("\nTacke nakon sortiranja: ")
for item in sortirane_tacke:
        print(item["teme"],)
print()

# import functools
# TODO Probati isto koristeci cmp_to_key()
```

1.4.5 Zadaci za samostalni rad sa rešenjima

Zadatak 1.44 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`.

Primer 1

```
POKRETANJE: python n-anagram.py
INTERAKCIJA SA PROGRAMOM:
 Unesite ime datoteke:
 datoteka.txt
 Unesite n
 2
```

Sadržaj ulaznih datoteka koje se koriste u prethodnom primeru upotrebe programa:

Listing 1.2: *datoteka.txt*

```
Ovo je datoteka dat
```

Listing 1.3: *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 1.44]

Zadatak 1.45

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.

Primer 1

```
POKRETANJE: python korpa.py
INTERAKCIJA SA PROGRAMOM:
 Maxi: 631.67 dinara
 Idea: 575.67 dinara
 Shopngo: 674.67 dinara
```

[Rešenje 1.45]

Sadržaj ulaznih datoteka koje se koriste u prethodnom primeru upotrebe programa:

Listing 1.4: *korpa.json*

```
[ {"ime" : "jabuke" ,"kolicina": 3.3},
{"ime": "kruske" , "kolicina": 2.1},
{"ime": "đgroze" , "kolicina":  2.6},
```

Listing 1.5: *maksi_cene.json*

```
[ {"ime" :"jabuke", "cena" : 59.9},
{"ime" :"kruske", "cena" : 120},
{"ime" :"đgroze", "cena" : 70},
{"ime" :"narandze", "cena" : 49.9},
{"ime" :"breskve", "cena" : 89.9} ]
```

Listing 1.6: *idea_cene.json*

```
[ {"ime" :"jabuke", "cena" : 39.9},
{"ime" :"kruske", "cena" : 100},
{"ime" :"đgroze", "cena" : 90},
{"ime" :"breskve", "cena" : 59.9} ]
```

Listing 1.7: *shopngo_cene.json*

```
[ {"ime" :"jabuke", "cena" : 69.9},
{"ime" :"kruske", "cena" : 100},
{"ime" :"đgroze", "cena" : 90},
{"ime" :"maline", "cena" : 290},
```

[Rešenje 1.45]

Zadatak 1.46 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.

Primer 1

```
POKRETANJE: python duze_sire.py
INTERAKCIJA SA PROGRAMOM:
 Unesite putanju do direktorijuma:
 ..
 Kardinalnost skupa duze : Kardinalnost skupa sire
 10 : 15
```

[Rešenje 1.46]

Zadatak 1.47 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`.

Primer 1

```
POKRETANJE: python ekstenzije.py
INTERAKCIJA SA PROGRAMOM:
 Unesite putanju do direktorijuma:
 .
```

Sadržaj datoteke koja se generise u prethodnom primeru upotrebe programa:

Listing 1.8: *rezultat.txt*

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

[Rešenje 1.47]

Zadatak 1.48 U datoteci `radnici.json` nalaze se podaci o radnom vremenu zaposlenih u preduzeću u sledecem formatu:

```
[  { 'ime' : ime\_radnika, 'odmor' : [pocetak, kraj], 'radno_vreme'
: [pocetak, kraj] }, ...]
```

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.

Primer 1

```
POKRETANJE: python odmor.py
INTERAKCIJA SA PROGRAMOM:
 "Unesite opciju koju zelite
 d - dostupni radnici
 o - radnici na odmoru :
 m
 Uneta opcija nije podrzana
```

Primer 2

```
POKRETANJE: python odmor.py
INTERAKCIJA SA PROGRAMOM:
 "Unesite opciju koju zelite
 d - dostupni radnici
 o - radnici na odmoru :
 d
 Pera Peric
```

[Rešenje 1.48]

Sadržaj ulaznih datoteka koje se koriste u prethodnom primeru upotrebe programa:

Listing 1.9: *radnici.json*

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

Zadatak 1.49 Napisati program koji učitava ime datoteke sa standardnog ulaza i na standardni izlaz ispisuje putanje do svih direktorijuma u kojima se nalazi ta datoteka.

Primer 1

```
POKRETANJE: python pojavljivanja.py
INTERAKCIJA SA PROGRAMOM:
 Unesite ime datoteke:
 1.py
 /home/student/vezbe/cas1/1.py
 /home/student/vezbe/cas7/1.py
 /home/student/vezbe/cas9/1.py
```

[Rešenje 1.49]

1.4.6 Zadaci za vežbu

Zadatak 1.50 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.

Primer 1

```
POKRETANJE: python ispiti.py
INTERAKCIJA SA PROGRAMOM:
 Relacione baze podataka Prosao
 Vestacka inteligencija Prosao
 Linearna algebra i analiticka geometrija Prosao
```

Sadržaj ulaznih datoteka koje se koriste u prethodnom primeru upotrebe programa:

Listing 1.10: *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 1.51 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 ∗/.

Primer 1

```
POKRETANJE: python komentari.py
INTERAKCIJA SA PROGRAMOM:
 Unesite ime datoteke:
 program.c
```

Sadrzaj ulaznih datoteka koje se koriste u prethodnom primeru kao i datoteke koja se generiše:

Listing 1.11: *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 1.12: *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 1.52 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.

Primer 1

```
POKRETANJE: python razlika.py
INTERAKCIJA SA PROGRAMOM:
 Unesite ime datoteke:
 dat1.txt
 Unesite ime datoteke:
 dat2.txt
```

Sadrzaj ulaznih datoteka koje se koriste u prethodnom primeru kao i datoteke koja se generiše::

Listing 1.13: *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 1.14: *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 1.15: *razlike.json*

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

Zadatak 1.53 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.

Primer 1

```
POKRETANJE: python apache.py
INTERAKCIJA SA PROGRAMOM:
 /home/student/PVEB/prviPrimer
 /home/student/licna_strana
 /home/student/PVEB/ispit/jun
```

Zadatak 1.54 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`.

Primer 1

```
POKRETANJE: python autocomplete.py
INTERAKCIJA SA PROGRAMOM:
 ma
 mac macka mama maceta mađionicar
 mac
 mac macka maceta
 !
```

Sadržaj ulaznih datoteka koje se koriste u prethodnom primeru upotrebe programa:

Listing 1.16: *reci.txt*

```
mac pesma skola macka mama maceta igra đmaionicar
```

1.5 Rešenja

Rešenje 1.22 Pogodi broj

```
# Pogodi broj

import random

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

zamisljen_broj = random.randint(0,100)

ime = 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 = False
while not pogodio:
    print("Unesi broj:")
    broj = int(input())
    if broj == zamisljen_broj:
        pogodio = True
    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 1.23 Aproksimacija broja PI metodom Monte Karlo

```
# Aproksimacija broja PI metodom Monte Karlo

import random, math

def dist(A, B):
    """Funkcija izracunava euklidsko rastojanje izmedju tacaka A i B"""
    return math.sqrt((A[0]-B[0])**2 + (A[1]-B[1])**2)

print("Izracunavanje broja PI metodom Monte Karlo \n")
try:
    N = int(input("Unesite broj iteracija: "))
except ValueError as err:
    print("Greska. Parsiranje broja iteracija: ", err)
    exit(1)
A = 0   # Broj tacaka u krugu
B = 0   # Broj tacaka u kvadratu

i = N
while i >= 0:
    tacka = (random.random(), random.random())
    # Ukoliko se tacka nalazi u krugu, povecavamo broj tacaka u krugu
    if dist(tacka, (0.5, 0.5)) <= 0.5:
        A = A + 1
    B = B + 1
    i = i - 1

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

Rešenje 1.24 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(input("Unesite vrstu: "))
        y = int(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 = input("Unesite ime prvog igraca: ")
print("Zdravo {0:s}!\n".format(ime1))
ime2 = 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, "0")
else:
    prvi_igrac = (ime2, "X")
    drugi_igrac = (ime1, "0")

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

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!!!!!!! Igrac {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 1.44

```
# 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 = input("Unesite ime datoteke: ")
n = int(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 1.45

```
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 1.46

```
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
    with open(ime_datoteke, "r") as f:
        for linija in f:
            br_linija = br_linija + 1
            if len(linija) > najduza_linija:
                najduza_linija = len(linija)
    if br_linija > najduza_linija:
        return 1
    else:
        return 0

ime_direktorijuma = 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 1.47

```
import os
import json

ime_direktorijuma = 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

with open("rezultat.json","w") as f:
    json.dump(ekstenzije, f)
```

Rešenje 1.48

```
import json, os, sys
from datetime import datetime

try:
    with open("radnici.json", "r") as f:
        radnici = json.load(f)
except IOError:
    print("Otvaranje datoteke nije uspelo!")
    sys.exit()

opcija = 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 1.49

```
import os

ime_datoteke = 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 3.7 i biblioteku python-constraint. Na Ubuntu operativnom sistemu, biblioteka python-constraint se moze instalirati pomocu Pip alata:

```
sudo apt-get -y install python3-pip
sudo pip3 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 Napisati program koji na standardni izlaz ispisuje sve kombinacije oblika xyz, gde je $x \in \{a, b, c\}$, $y \in \{1, 2, 3\}$ i $z \in \{0.1, 0.2, 0.3\}$ tako da važi da je $10 \cdot z = y$.

```
# 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',['a','b','c'])
problem.addVariable('y',[1,2,3])
# 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(y,z):
    if y / 10.0 == z:
        return True

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

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.
Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

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

[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.
Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
print """
1 din: {0:d}
2 din: {1:d}
5 din: {2:d}
10 din: {3:d}
20 din: {4:d}
""".format(r["1 din"],r["2 din"],r["5 din"], r["10 din"], r["20 din"])
```

[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
```

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
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 2.4]

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]

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

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

[Rešenje 2.5]

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

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

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
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 2.6]

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 $W \in \{1, 8, 27, ..., 1000\}$.

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
print "--------------------------"
print "X = {0:d} , Y = {1:d} , Z = {2:d}, W = {3:d}".format(r['X'],r['Y'],r['Z'], r['W'
    ])
```

[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
```

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
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 2.8]

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)?

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
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 2.9]

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).

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
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 2.10]

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.

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
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 2.11]

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

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

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

[Rešenje 2.12]

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

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

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

[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.

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
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 "=========================="
```

Primer 1

```
Pokretanje: python sudoku.py
Interakcija sa programom:
 Unesite ime datoteke sa tablom za sudoku:
 sudoku.json
 ========================
 | 6 3 8 | 7 5 2 | 1 9 4 |
 | 9 2 7 | 4 1 8 | 5 6 3 |
 | 5 4 1 | 6 3 9 | 2 7 8 |
 ------------------------
 | 7 1 4 | 5 9 6 | 3 8 2 |
 | 2 8 9 | 1 7 3 | 4 5 6 |
 | 3 5 6 | 8 2 4 | 7 1 9 |
 ------------------------
 | 8 9 5 | 3 4 1 | 6 2 7 |
 | 4 7 2 | 9 6 5 | 8 3 1 |
 | 1 6 3 | 2 8 7 | 9 4 5 |
 ========================
```

[Rešenje 2.14]

Sadržaj datoteke koja se koriste u primeru 2.14:

Listing 2.1: *sudoku.json*

```
[[6, 3, 8, 7, 0, 0, 0, 0, 0],
[0, 2, 7, 0, 0, 0, 5, 0, 0],
[5, 4, 0, 6, 0, 9, 2, 0, 0],
[0, 1, 0, 5, 9, 0, 3, 0, 0],
[0, 0, 9, 0, 7, 0, 4, 0, 0],
[0, 0, 6, 0, 2, 4, 0, 1, 0],
[0, 0, 5, 3, 0, 1, 0, 2, 7],
[0, 0, 2, 0, 0, 0, 8, 3, 0],
[0, 0, 0, 0, 0, 7, 9, 4, 5]]
```

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
```

Sve rezultate ispisati na standardni izlaz koristeći sledeći format komande ispisa. Primer za prvu zagonetku.

Komanda ispisa rešenja:

```
print "   " + str(r['G']) + str(r['R']) + str(r['E']) + str(r['E']) + str(r['N'])
```

```
print " +" + str(r['O']) + str(r['R']) + str(r['A']) + str(r['N']) + str(r['G']) + str(r
    ['E'])
print "=" + str(r['C']) + str(r['O']) + str(r['L']) + str(r['O']) + str(r['R']) + str(r[
    'S'])
```

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
```

Sve rezultate ispisati na standardni izlaz koristeći sledeći format komande ispisa. Primer za prvu zagonetku.

Komanda ispisa rešenja:

```
print "   " + str(r['T']) + str(r['R']) + str(r['E']) + str(r['E'])
print "   " + str(r['T']) + str(r['R']) + str(r['E']) + str(r['E'])
print " +" + str(r['O']) + str(r['N']) + str(r['E'])
print "=" + str(r['S']) + str(r['E']) + str(r['V']) + str(r['E']) + str(r['N'])
```

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
```

Sve rezultate ispisati na standardni izlaz koristeći sledeći format komande ispisa. Primer za prvu zagonetku.

Komanda ispisa rešenja:

```
print "   " + str(r['M']) + str(r['A']) + str(r['N'])
print " *" + str(r['A']) + str(r['N']) + str(r['D'])
print "=" + str(r['W']) + str(r['O']) + str(r['M']) + str(r['E']) + str(r['N'])
```

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 $i = 0, 1, ..., n-1$.

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
print "( ",
for i in range(n-1):
  print str(r["x"+str(i)]) + ", ",
print str(r["xn-1"]) + ")"
```

Primer 1

```
Pokretanje: python sudoku.py
Interakcija sa programom:
 Unesite dužinu magične sekvence
 4
 (1,2,1,0)
 (2,0,2,0)
```

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?

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
print "Potrebno je da ocisti {0:d} kuca i {1:d} stanova da bi zarada bila najveca".
    format(r["kuca"], r["stan"])
```

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?

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
print "Potrebno je da napravi {0:d} solja i {1:d} tanjira da bi zarada bila najveca".
    format(r["solja"], r["tanjir"])
```

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?

Sve rezultate ispisati na standardni izlaz koristeći datu komandu ispisa.

Komanda ispisa rešenja:

```
print "Potrebno je da proda {0:d} stampaca i {1:d} racunara da bi se maksimalno obogatio
    ".format(r["stampac"], r["racunar"])
```

2.2 Rešenja

Rešenje 2.2

```
import constraint

problem = constraint.Problem()
# Definisemo promenljive i njihove vrednosti
problem.addVariable('A',range(1,10))
problem.addVariable('B',range(10))
problem.addVariable('C',range(10))
# Dodajemo ogranicenje da su vrednosti svih promenljivih razlicite
problem.addConstraint(constraint.AllDifferentConstraint())
resenja = problem.getSolutions()
# Znamo da minimalni kolicnik mora biti manji od 999
min_kolicnik = 999
min_resenje = {}
for resenje in resenja:
    a = resenje['A']
    b = resenje['B']
    c = resenje['C']
    kolicnik = (a*100 + b*10 + c) / (a+b+c)
    if kolicnik < min_kolicnik:
        min_kolicnik = kolicnik
        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("---")
  print("""1 din: {0:d}
2 din: {1:d}
5 din: {2:d}
10 din: {3:d}
20 din: {4:d}""".format(r["1 din"],r["2 din"],r["5 din"], r["10 din"], r["20 din"]))
  # Provera da je suma bas 50
  print("Ukupno:", r["1 din"] + r["2 din"]*2 + r["5 din"]*5 + r["10 din"]*10 + r["20
    din"]*20)
  print("---")
```

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
```

```
12
problem = constraint.Problem()
14
# Dodajemo promenljivu X i definisemo njen domen
16 problem.addVariable('X', range(1,91))

18 # Dodajemo promenljivu Y i definisemo njen domen
problem.addVariable('Y', range(2,61,2))
20
domenZ = [];
22 for i in range(1,11):
    domenZ.append(i*i)
24
# Dodajemo promenljivu Z i definisemo njen domen
26 problem.addVariable('Z', domenZ)

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

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

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

40 resenja = problem.getSolutions()

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

Rešenje 2.6

```
#    TWO
2 #  +TWO
#   ------
4 #  FOUR
#
6
import constraint
8
problem = constraint.Problem()
10 # Definisemo promenljive i njihove vrednosti
problem.addVariables("TF",range(1,10))
12 problem.addVariables("WOUR",range(10))

14 # Definisemo ogranicenje za cifre
def o(t, w, o, f, u, r):
16    if 2*(t*100 + w*10 + o) == f*1000 + o*100 + u*10 + r:
        return True
18
# Dodajemo ogranicenja za cifre
20 problem.addConstraint(o,"TWOFUR")
# Dodajemo ogranicenje da su sve cifre razlicite
22 problem.addConstraint(constraint.AllDifferentConstraint())

24 resenja = problem.getSolutions()

26 for r in resenja:
    print("-----")
28  print("  "+str(r['T'])+str(r['W'])+str(r['O']))
    print(" +"+str(r['T'])+str(r['W'])+str(r['O']))
30  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", end='')
            else:
                print("-", end='')
        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", end='')
            else:
                print("-", end='')
        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 = 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("|", end='')
        for j in range(1,10):
            if j%3 == 0:
                print(str(r[i*10+j])+" | ", end='')
            else:
                print(str(r[i*10+j]), end='')
        print("")
        if i%3 == 0 and i!=9:
            print("------------------")
    print("==================")
```

3

Funkcionalni koncepti u programskom jeziku Python

Literatura:

(a) https://docs.python.org/3.7/howto/functional.html

3.1 Funkcionalno programiranje

3.1.1 Uvodni primeri

Zadatak 3.1 Napisati program koji sa standardnog ulaza učitava dva broja i na standardni izlaz ispisuje njihov zbir.

```
def suma(a, b):
  return a + b

suma2 = lambda a, b : a + b

a = int(input("Unesi a: "))
b = int(input("Unesi b: "))
print("suma: {}".format(suma(a, b)))
print("suma1: {}".format(suma2(a, b)))
```

Zadatak 3.2 Napisati funkciju `parovi(a,b,c, d)` koja generiše listu parova celih brojeva (x, y), za koje `x` pripada segmentu $[a, b]$, a `y` pripada segmentu $[c, d]$. Testirati rad funkcije pozivom u programu.

```
a = int(input("Unesi a: "))
b = int(input("Unesi b: "))
c = int(input("Unesi c: "))
d = int(input("Unesi d: "))

lista = []
for i in range(a, b):
  for j in range(c, d):
    lista.append((i, j))
print(lista)

lista1 = [(i, j) for i in range(a, b) for j in range(c, d)]
print(lista1)
```

Zadatak 3.3 Marko, Petar i Pavle su polagali ispit iz predmeta programske paradigme. Napisati program koji sa standardnog ulaza učitava ocene koji su dobili, a potom ispisuje listu parova (student, ocena) na standardni izlaz.

```
imena = ["Marko", "Petar", "Pavle"]
ocene = [int(input("unesi ocenu koju je dobio "+ime+": ")) for ime in imena]

```

```
print(zip(imena, ocene))

for par in zip(imena, ocene):
  print(par)
```

Zadatak 3.4 Napisati program koji sa standarnog ulaza učitava nisku, a na standardni izlaz ispisuje nisku u kojoj su sva mala slova pretvorena u velika.

```
# pseudo kod za map.
# def map(func, seq):
#   # vraca `Map` objekat gde
#   # funkcija func primenjena na svaki element
#   return Map(
#   func(x)
#   for x in seq
#   )
niska = input("Unesi zeljenu nisku: ")
print(map(lambda c: c.upper(), niska))              # map object
print(list(map(lambda c: c.upper(), niska)))         # lista karaktera
print(''.join(list(map(lambda c: c.upper(), niska))))  # niska
```

Zadatak 3.5 Napisati program koji sa standarnog ulaza učitava nisku, a na standardni izlaz ispisuje sve karaktere koji nisu slova.

```
# 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
#
# def filter(evaluate , seq):
#   return Map(
#     x for x in seq
#     if evaluate(x) is True
#   )
niska = input("Unesite zeljenu nisku: ")
print(list(filter(lambda c: not c.isalpha(), niska)))  # lista karaktera koji nisu
    slova
```

Zadatak 3.6 Napisati program koji na standardni izlaz ispisuje sumu, koristeći funkciju `reduce` prvih n prirodnih brojeva gde se n unosi sa standardnog ulaza.

```
# funkcija koja odgovara funkciji fold u Haskelu je funkcija reduce
# ona se nalazi u modulu functools
from functools import reduce

try:
  n = int(input("Unesi n: "))
except ValueError:
  print("Error: cast error")

brojevi = range(0, n+1)
print("Suma(prvih {}) = {}".format(n, reduce(lambda x, y: x+y, brojevi)))
```

3.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 3.7 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.

Primer 1

```
POKRETANJE: python korpa.py
INTERAKCIJA SA PROGRAMOM:
 Maxi: 631.67 dinara
 Idea: 575.67 dinara
 Shopngo: 674.67 dinara
```

Sadržaj datoteka koje se koriste u primeru 3.7:

Listing 3.1: *korpa.json*

```
[ {"ime" : "jabuke" ,"kolicina": 3.3},
{"ime": "kruske" , "kolicina": 2.1},
{"ime": "grozdje" , "kolicina":  2.6},
```

Listing 3.2: *maksi_cene.json*

```
[ {"ime" :"jabuke", "cena" : 59.9},
{"ime" :"kruske", "cena" : 120},
{"ime" :"grozdje", "cena" : 70},
{"ime" :"narandze", "cena" : 49.9},
{"ime" :"breskve", "cena" : 89.9} ]
```

Listing 3.3: *idea_cene.json*

```
[ {"ime" :"jabuke", "cena" : 39.9},
{"ime" :"kruske", "cena" : 100},
{"ime" :"grozdje", "cena" : 90},
{"ime" :"breskve", "cena" : 59.9} ]
```

Listing 3.4: *shopngo_cene.json*

```
[ {"ime" :"jabuke", "cena" : 69.9},
{"ime" :"kruske", "cena" : 100},
{"ime" :"grozdje", "cena" : 90},
{"ime" :"maline", "cena" : 290},
```

[Rešenje 3.7]

Zadatak 3.8 U datoteci *tacke.json* se nalaze podaci o tačkama u sledećem formatu.

Listing 3.5: *tacke.json*

```
[ {"teme":teme , "koordinate": [x, y]}, ...]
```

Napisati program koji učitava podatke o tačkama iz datoteke *tacke.json* i sortira i po udaljenosti od koordinatnog početka opadajuće. Program na standardni izlaz treba da ispiše tačku iz zadatog kvadranta, koja je najudaljenija od koordinatnog početka. Kvadrant se zadaje kao argument komandne linije.

Primer 1

```
POKRETANJE: python tacke.py 1
INTERAKCIJA SA PROGRAMOM:
 B
```

Sadržaj datoteke koja se koristi u primeru 3.8:

Listing 3.6: *tacke.json*

```
[ {"teme":"A" , "koordinate": [10.0, 1.1]},
{"teme":"B" , "koordinate": [1.0, 15.0]},
{"teme":"C" , "koordinate": [-1.0, 5.0]} ]
```

[Rešenje 3.8]

3.1.3 Zadaci za vežbu

Zadatak 3.9 U datoteci *knjige.json* se nalaze podaci o knjigama u knjižari u sledećem formatu.

Listing 3.7: *knjige.json*

```
[ {"oznaka":broj , "naziv_autor": naziv knjige i autor, "kolicina":
   kolicina, "cena": cena po komadu}, ...]
```

Napisati program koji ispisuje listu parova (oznaka, ukupna vrednost). Ukupna vrednost se sračunava na osnovu cene i broja primeraka, a povećava se za 10 dinara ukoliko ukupna cena knjiga sa istom oznakom ne prelazi minimalnu cenu od 100 dinara.

Primer 1

```
POKRETANJE: python knjige.py
INTERAKCIJA SA PROGRAMOM:
 [('34587', 163.8), ('98762', 284.0), ('77226', 108.85000000000001), ('88112', 84.97)]
```

Sadržaj datoteke koja se koristi u primeru 3.9:

Listing 3.8: *knjige.json*

```
[ {"oznaka": 34587, "naziv_autor": "Learning Python, Mark Lutz", "
   kolicina": 4, "cena": 40.95},
{"oznaka": 98762, "naziv_autor": "Programming Python, Mark Lutz", "
   kolicina": 5, "cena": 56.80},
{"oznaka": 77226, "naziv_autor": "Head First Python, Paul Barry", "
   kolicina": 3, "cena": 32.95},
{"oznaka": 88112, "naziv_autor": "Einfuhrung in Python3, Bernd Klein",
   "kolicina": 3, "cena": 24.99} ]
```

Zadatak 3.10 Datoteka `fudbaleri.json` sadrži podatke o fudbalerima (ime, nacionalnost i broj golova) u sledećem formatu:

```
[ { "Ime" : "Alexis Sanchez",  "Nacionalnost" :"Cile", "Golovi" : 17} ,
   ...]
```

(a) Definisati funkciju `uporedi` koristeći lambda izraz, koja poredi dva fudbalera po broju postignutih golova. Funkcija vraća -1, 0 ili 1 ukoliko je prvi fubaler postigao manji, jednak i veći broj golova u odnosu na drugog fudbalera

(b) Napisati program koji iz izabrane datoteke izdvaja fudbalere određene nacionalnosti i sortira ih rastuće po broju golova. Željena nacionalnost (npr. 'Engleska') i ime datoteke u kojoj se nalaze podaci o fudbalerima se zadaju kao argumenti komandne linije, a rezultat rada programa se upisuje u datoteku `izabrana_nacionalnost.json` (npr. Engleska_nacionalnost.json). U slučaju greške prilikom pokretanja programa, ispisati tekst `Greska` na standardni izlaz.

Primer 1

```
POKRETANJE: python 1.py Engleska arsenal.json
ENGLESKA_NACIONALNOST.JSON
 [{"Nacionalnost": "Engleska",
  "Ime": "Alex Oxlade-Chamberlain",
  "Golovi": 2},
  {"Nacionalnost": "Engleska",
  "Ime": "Theo Walcott", "Golovi": 8}]
```

Sadržaj datoteke koja se koristi u primeru 3.10:

Listing 3.9: *fudbaleri.json*

```

[{"Nacionalnost": "Cile", "Ime": "Alexis Sanchez", "Golovi" : 17 },
```

```
{"Nacionalnost": "Francuska", "Ime": "Olivier Giroud", "Golovi" : 8},
{"Nacionalnost": "Engleska", "Ime": "Theo Walcott", "Golovi" : 8},}
{"Nacionalnost": "Engleska", "Ime": "Alex Oxlade-Chamberlain", "Golovi"
    : 2},
{"Nacionalnost": "Francuska", "Ime": "Laurent Koscielny", "Golovi" : 2
   } ]
```

Zadatak 3.11 Datoteka `utakmice.json` sadrži podatke o utakmicama (timovi koji se takmiče i vreme početka utakmice) za svaki sport posebno u sledećem formatu:

```
[ { "Timovi": "Liverpool-Arsenal", "Vreme": "18:00"} , ...]
```

(a) Napisati funkciju `u_vremenskom_intervalu(utakmice, pocetak, kraj)` koja vraća listu utakmica čiji se početak nalazi u vremenskom opsegu početak kraj. Funkciju implementirati bez korišćenja petlji.

(b) Napisati program koji sa standardnog ulaza učitava podatke o početku i kraju vremenskog intervala u formatu *%H:%M* i iz datoteka učitava podatke o utakmicama za sve sportove, i na standardni izlaz ispisuje listu utakmica čije se vreme početka nalazi u opsegu unešenog vremenskog intervala.

Primer 1

```
POKRETANJE: python 1.py
ULAZ:
  Unesite početak intervala: 17:00
  Unesite kraj intervala: 18:00
IZLAZ:
   {"Timovi": "Liverpool-Arsenal", "Vreme": "18:00"}
   {"Timovi": "Milan-Cheivo", "Vreme": "17:00"}
   {"Timovi": "Koln-Bayern", "Vreme": "17:30"}
```

Sadržaj datoteke koja se koristi u primeru 3.11:

Listing 3.10: *utakmice.json*

```
[{"Timovi": "Liverpool-Arsenal", "Vreme": "18:00"},
 {"Timovi": "Milan-Cheivo", "Vreme": "17:00"},
 {"Timovi": "Eibar-Real Madrid", "Vreme": "18:30"},
 {"Timovi": "Roma-Napoli", "Vreme": "16:15"},
 {"Timovi": "Koln-Bayern", "Vreme": "17:30"} ]
```

Zadatak 3.12 Napisati program koji na standardni izlaz ispisuje apsolutne putanje do svih datoteka koje sadrže reč koja se unosi sa standardnog ulaza ili u slučaju da ni jedna datoteka ne sadrži zadatu reč ispisati poruku `Ni jedna datoteka ne sadrzi zadatu rec`. Progam napisati korišćenjem lambda izraza i bez korišćenja petlji

Primer 1

```
POKRETANJE: python 1.py
ULAZ:
  Unesite zadatu rec: include
IZLAZ:
  /home/student/programiranje2/ispit/1.c
  /home/student/programiranje2/ispit/2.c
  /home/student/programiranje2/ispit/3.c
```

Primer 1

```
POKRETANJE: python 1.py
ULAZ:
  Unesite zadatu rec: Lenovo
IZLAZ:
  Ni jedna datoteka ne sadrzi zadatu rec
```

3.2 Rešenja

Rešenje 3.7

```
import json
from functools import reduce

def cena_voca(prodavnica, ime_voca): # [{"ime" :"jabuke", "cena" : 69.9},...], "
    jabuke"
  for voce in prodavnica:
    if voce['ime']==ime_voca:
      return voce['cena']

# Ucitavam podatke
try:
  with open('korpa.json', 'r') as f:  # [ {"ime" : "jabuke" ,"kolicina": 3.3},...]
    korpa = json.load(f)
  with open('maxi_cene.json', 'r') as f:
    maxi_cene = json.load(f)
  with open('idea_cene.json', 'r') as f:
    idea_cene = json.load(f)
  with open('shopngo_cene.json', 'r') as f:
    shopngo_cene = json.load(f)
except IOError:
  print("Error: open json file")
  exit(1)

maxi_racun = reduce(lambda a, b: a+b, [voce['kolicina']*cena_voca(maxi_cene, voce['
    ime']) for voce in korpa])
idea_racun = reduce(lambda a, b: a+b, [voce['kolicina']*cena_voca(idea_cene, voce['
    ime']) for voce in korpa])
shopngo_racun = reduce(lambda a, b: a+b, [voce['kolicina']*cena_voca(shopngo_cene,
    voce['ime']) for voce in korpa])

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

Rešenje 3.8

```
import json
import math
import sys
from functools import partial

# funkcija koja proverava da li tacka pripada kvadrantu
def pripada(kvadrant, tacka):  # 1, {'teme': 'A', 'koordinate': [10.0, 1.1]}
    koordinate = tacka["koordinate"]
    if kvadrant == 1:
        return koordinate[0] >=0 and koordinate[1] >=0
    elif kvadrant == 2:
        return koordinate[0] <=0 and koordinate[1] >=0
    elif kvadrant == 3:
        return koordinate[0] <=0 and koordinate[1] <=0
    elif kvadrant == 4:
        return koordinate[0] >=0 and koordinate[1] <=0
    else:
        print("Error: Kvadrant nije ispravno unet")
        exit()

if len(sys.argv) != 2:
    print("Niste naveli dovoljno argumenta komandne linije: kvadrant")
    exit(1)

try:
  with open("tacke.json","r") as f:  # [{'teme': 'A', 'koordinate': [10.0, 1.1]},
    ...]
    tacke = json.load(f)
except IOError:
```

```
29    print('Error: open json file')
      exit(1)
31
try:
33    kvadrant = int(sys.argv[1])
except ValueError:
35    print('Error: invalid cast')
      exit(1)
37
# sa partial fiksiramo jedan argument: kvadrant
39  # partial vraca objekat koji se moze pozvati kao funkcija (u konkretnom slucaju
        jednog argumenta)
uslov = partial(pripada, kvadrant)
41  kvadrant_tacke = list(filter(uslov, tacke))  # listu dobijamo od filter objecta

43  # Ispis
if len(kvadrant_tacke) > 0:
45      print(min(kvadrant_tacke, key = lambda x: x['koordinate'][0]**2 + x['koordinate'
        ][1]**2))
else:
47      print("Trazena tacka ne postoji")

49  # Eventualno resavanje zadatka sortiranjem: (nije efikasno)
    # sortirane_tacke = sorted(tacke, key = lambda x: x['koordinate'][0]**2 + x['
        koordinate'][1]**2)
51  # print(sortirane_tacke)
```

4

Komponentno programiranje u programskom jeziku Python

Potrebno je imati instalirano:

(a) `Python` (verzija 3.7) https://www.python.org/

(b) `pip3` (`python3-pip`, može se instalirati preko `apt`-a za *ubuntu distribucije)

4.1 PyQt5 - Instalacija potrebnih alata

4.1.1 Instalacija PyQt5

Za distribucije zasnovane na Ubuntu distribuciji, dovoljno je pokrenuti sledeće komande:

```
$ pip3 install --user pyqt5
$ sudo apt-get install python3-pyqt5
$ sudo apt-get install pyqt5-dev-tools
$ sudo apt-get install qttools5-dev-tools
```

Slično važi i za ostale distribucije.

Kada instaliramo PyQt5 alate, možemo kreirati `.ui` fajlove koristeći Qt Designer. Kreirane `.ui` fajlove možemo zatim prevesti u Python3 kod tako što koristimo alat `pyuic5` na sledeći način:

```
$ pyuic5 ui_fajl.ui
```

Opcije (videti `man pyuic5`):

- `-o ime_fajla.py` - specificira ime generisanog fajla
- `-p` - samo prikazuje UI, bez prevoda u Python kod
- `-x` - dodatno generiše kod koji dozvoljava pokretanje generisanog Python fajla

Često ćemo pokretati `pyuic5` na sledeći način:

```
$ pyuic5 ui_fajl.ui -x -o ui_fajl.py
```

4.1.2 Instalacija dodataka za Visual Studio Code

https://marketplace.visualstudio.com/items?itemName=zhoufeng.pyqt-integration

Potrebno je podesiti lokaciju Qt Designer izvršnog fajla u podešavanjima dodatka. Ukoliko je izabrana podrazumevana lokacija prilikom instalacije Qt5, Qt alati se nalaze na lokaciji `/usr/lib/qt5/bin`. Dakle, putanja do Qt Designer-a bi u tom slučaju bila: `/usr/lib/qt5/bin/designer`.

4.2 Uvodni primeri

Zadatak 4.1 Napisati program koji ilustruje osnovni rad sa komponentama PyQt5 okruženja, integrišući par osnovnih komponenti po izboru.


```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file '1.ui'
#
# Created by: PyQt5 UI code generator 5.12.3
#
# WARNING! All changes made in this file will be lost!


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(393, 529)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayoutWidget = QtWidgets.QWidget(self.centralwidget)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(90, 20, 211, 241))
        self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setObjectName("verticalLayout")
        self.pushButton = QtWidgets.QPushButton(self.verticalLayoutWidget)
        self.pushButton.setObjectName("pushButton")
        self.verticalLayout.addWidget(self.pushButton)
        self.checkBox = QtWidgets.QCheckBox(self.verticalLayoutWidget)
        self.checkBox.setObjectName("checkBox")
        self.verticalLayout.addWidget(self.checkBox)
        self.radioButton = QtWidgets.QRadioButton(self.verticalLayoutWidget)
        self.radioButton.setObjectName("radioButton")
        self.verticalLayout.addWidget(self.radioButton)
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(96, 296, 201, 121))
        font = QtGui.QFont()
        font.setPointSize(20)
        font.setItalic(True)
        self.label.setFont(font)
```

```
        self.label.setObjectName("label")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 393, 29))
        self.menubar.setObjectName("menubar")
        self.menuInfo = QtWidgets.QMenu(self.menubar)
        self.menuInfo.setObjectName("menuInfo")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.actionAbout = QtWidgets.QAction(MainWindow)
        self.actionAbout.setObjectName("actionAbout")
        self.menuInfo.addAction(self.actionAbout)
        self.menubar.addAction(self.menuInfo.menuAction())

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.pushButton.setText(_translate("MainWindow", "Hello Push Button"))
        self.checkBox.setText(_translate("MainWindow", "Hello Check Box"))
        self.radioButton.setText(_translate("MainWindow", "Hello Radio Button"))
        self.label.setText(_translate("MainWindow", "Hello Text Label"))
        self.menuInfo.setTitle(_translate("MainWindow", "Hello Menu"))
        self.actionAbout.setText(_translate("MainWindow", "Hello Menu Item"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>393</width>
    <height>529</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QWidget" name="verticalLayoutWidget">
    <property name="geometry">
     <rect>
      <x>90</x>
      <y>20</y>
      <width>211</width>
      <height>241</height>
     </rect>
    </property>
    <layout class="QVBoxLayout" name="verticalLayout">
     <item>
      <widget class="QPushButton" name="pushButton">
       <property name="text">
        <string>Hello Push Button</string>
       </property>
      </widget>
     </item>
     <item>
```

```
      <widget class="QCheckBox" name="checkBox">
       <property name="text">
        <string>Hello Check Box</string>
       </property>
      </widget>
     </item>
     <item>
      <widget class="QRadioButton" name="radioButton">
       <property name="text">
        <string>Hello Radio Button</string>
       </property>
      </widget>
     </item>
    </layout>
   </widget>
   <widget class="QLabel" name="label">
    <property name="geometry">
     <rect>
      <x>96</x>
      <y>296</y>
      <width>201</width>
      <height>121</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>20</pointsize>
      <italic>true</italic>
     </font>
    </property>
    <property name="text">
     <string>Hello Text Label</string>
    </property>
   </widget>
  </widget>
  <widget class="QMenuBar" name="menubar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>393</width>
     <height>29</height>
    </rect>
   </property>
   <widget class="QMenu" name="menuInfo">
    <property name="title">
     <string>Hello Menu</string>
    </property>
    <addaction name="actionAbout"/>
   </widget>
   <addaction name="menuInfo"/>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
  <action name="actionAbout">
   <property name="text">
    <string>Hello Menu Item</string>
   </property>
  </action>
 </widget>
 <resources/>
 <connections/>
</ui>
```

Ubuduće nećemo prikazivati generisane .ui fajlove.

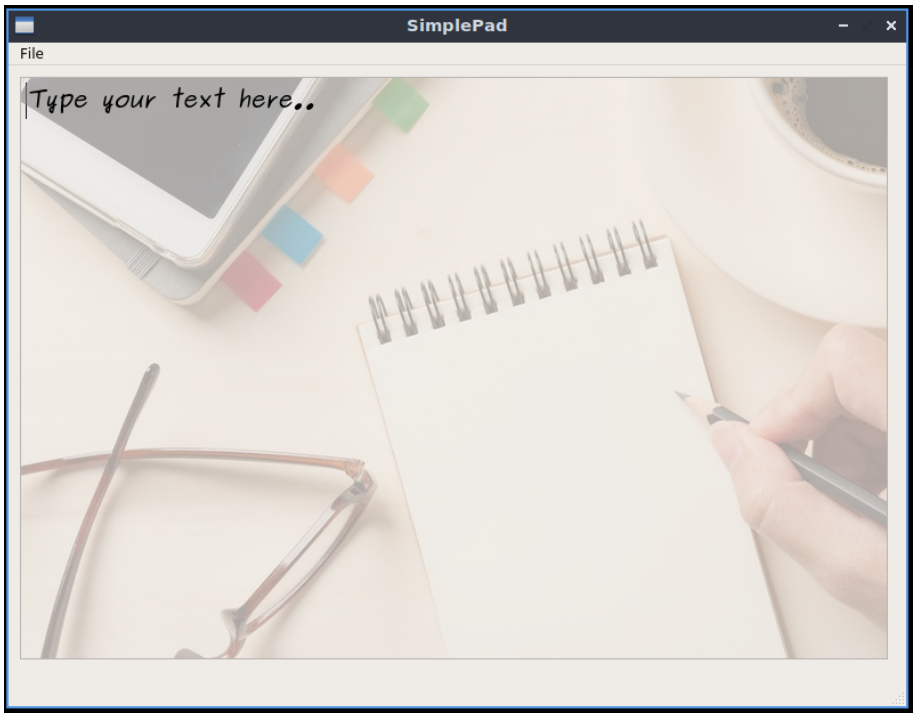
Zadatak 4.2 Napisati program koji učitava tekst korišćenjem jednolinijskog polja za unos i isti ispisuje u labelu ispod jednolinijskog polja za unos.


```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file '2.ui'
#
# Created by: PyQt5 UI code generator 5.12.3
#
# WARNING! All changes made in this file will be lost!


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(687, 386)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(66, 26, 341, 61))
        font = QtGui.QFont()
        font.setFamily("Tlwg Typo")
        font.setPointSize(14)
        font.setItalic(True)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(130, 370, 191, 71))
        font = QtGui.QFont()
        font.setPointSize(20)
        font.setItalic(True)
        self.label_2.setFont(font)
        self.label_2.setText("")
        self.label_2.setObjectName("label_2")
        self.horizontalLayoutWidget = QtWidgets.QWidget(self.centralwidget)
        self.horizontalLayoutWidget.setGeometry(QtCore.QRect(70, 100, 571, 71))
        self.horizontalLayoutWidget.setObjectName("horizontalLayoutWidget")
        self.horizontalLayout = QtWidgets.QHBoxLayout(self.horizontalLayoutWidget)
        self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.label_3 = QtWidgets.QLabel(self.horizontalLayoutWidget)
        font = QtGui.QFont()
        font.setPointSize(16)
        font.setItalic(True)
        self.label_3.setFont(font)
        self.label_3.setObjectName("label_3")
        self.horizontalLayout.addWidget(self.label_3)
        self.lineEdit = QtWidgets.QLineEdit(self.horizontalLayoutWidget)
        self.lineEdit.setObjectName("lineEdit")
        self.horizontalLayout.addWidget(self.lineEdit)
```

```
        self.label_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(190, 190, 291, 71))
        font = QtGui.QFont()
        font.setFamily("Garuda")
        font.setPointSize(26)
        font.setBold(True)
        font.setItalic(True)
        font.setUnderline(True)
        font.setWeight(75)
        self.label_4.setFont(font)
        self.label_4.setText("")
        self.label_4.setObjectName("label_4")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(270, 290, 111, 51))
        self.pushButton.setObjectName("pushButton")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 687, 29))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        self.lineEdit.textChanged['QString'].connect(self.label_4.setText)
        self.pushButton.clicked.connect(self.label_4.clear)
        self.pushButton.clicked.connect(self.lineEdit.clear)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.label.setText(_translate("MainWindow", "Hello World to Signal Handler"))
        self.label_3.setText(_translate("MainWindow", "Insert your name: "))
        self.pushButton.setText(_translate("MainWindow", "clear name"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Ubuduće nećemo prikazivati generisane .py fajlove.

Zadatak 4.3 Napisati program koji implementira *SimplePad* aplikaciju: jednostavan tekstualni editor koji podržava osnovne komande (kreiranja, otvaranja i cuvanja dokumenata), dodatno opcije kopiranja, lepljenja i isecanja delova teksta - bazne operacije za rad sa tekstom.

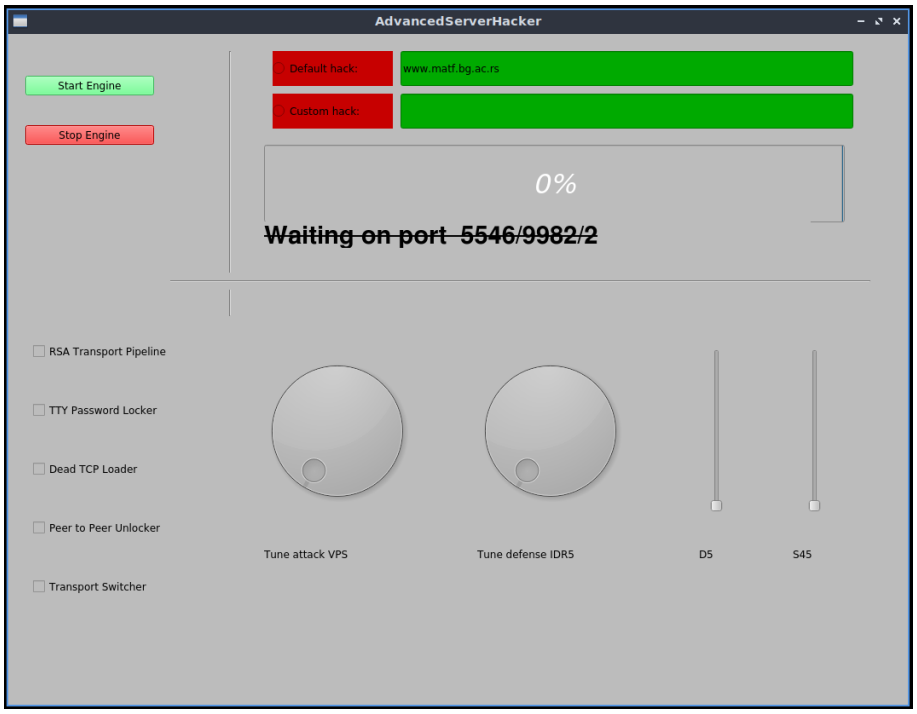



```
import ui_4_03
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QFileDialog
import sys
import os

__author__ = 'Milan Cugurovic & Ivan Ristovic, 2020.'

class SimplePadClass(ui_4_03.Ui_MainWindow, QtWidgets.QMainWindow):
    def __init__(self):
        super(SimplePadClass, self).__init__()
        self.setupUi(self)

        # Putanja do trenurno otvorenog fajla
        self.path = None

        # Akcije
        self.actionNew.triggered.connect(lambda: self.OpenMe())
        self.actionSave.triggered.connect(lambda: self.SaveMe())
        self.actionSaveAs.triggered.connect(lambda: self.SaveAsMe())

    # Azuriram naslov prozora
    def __title(self):
        self.setWindowTitle("{} - SimplePad".format(os.path.basename(self.path) if
    self.path is not None else 'Untitled'))

    # Poruka za korisnika
    def __dialog(self, msg):
        box = QMessageBox(self)
        box.setText(msg)
        box.show()

    # Cuvam na fajl sistemu
    def __save_util(self, path):
        data = self.editor.toPlainText()
        try:
            with open(path, "w") as f:
                f.write(data)
        except Exception as e:
            self.__dialog(str(e))
        else:
            self.path = path
            self.__title()
```

```
44      # ctr+O
        def OpenMe(self):
46          path, _ = QFileDialog.getOpenFileName(self, "Open file", "", "All files (*.*)
        ;")
            if path:
48              try:
                    with open(path, "r") as f:
50                      data = f.read()
                except Exception as e:
52                  self.__dialog(str(e))
                else:
54                  self.path = path
                    self.editor.setPlainText(data)
56                  self.__title()

58      # Ctr+S
        def SaveMe(self):
60          if self.path is None:
                return self.SaveAsMe()
62          self.__save_util(self.path)

64      # Ctrl+Alt+S
        def SaveAsMe(self):
66          path, _ = QFileDialog.getSaveFileName(self, "Save file", "", "All files (*.*)
        ;")
            if not path:
68              return  # cancelled
            self.__save_util(path)
70

72  if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
74      simplePad = SimplePadClass()
        simplePad.show()
76      sys.exit(app.exec_())
```

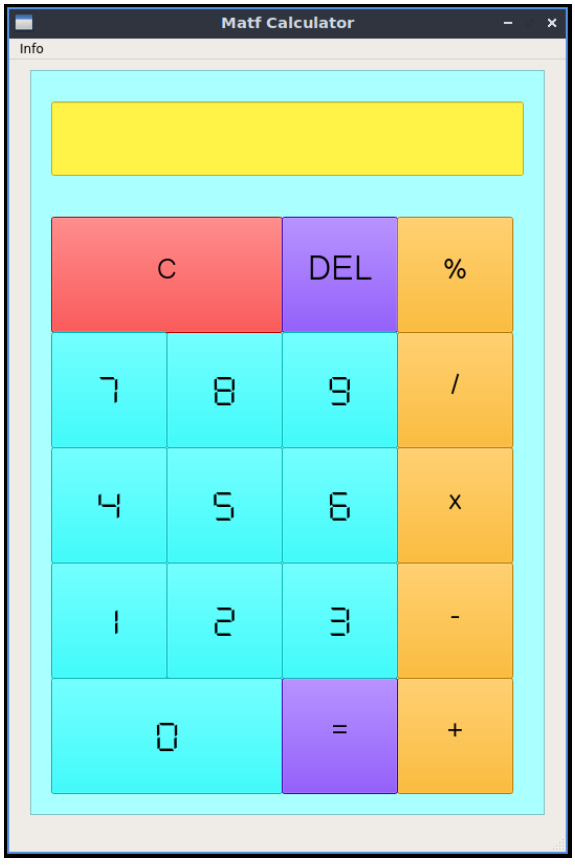
4.3 Zadaci za samostalni rad sa rešenjima

Zadatak 4.4 Napisati program koji implementira aplikaciju koja simulira hakerske napade. Aplikacija klikom na dugme za start, ažurira progres bar (od 0 do 100%), svake sekunde napredujući za po jedan procenat. Dodatno, aplikacija podržava čekiranje dodatnih opcija hakovanja, kao i prodešavanje kontinualnih parametara.



[Rešenje 4.4]

Zadatak 4.5 Implementirati digitron koji podržava osnovne aritmetičke operacije sabiranja, oduzimanja, množenja i deljenja. Implementirati brisanje unetih brojeva (back dugme) kao i resetovanje rada digitrona klikom na odgovarajuće dugme.



[Rešenje 4.5]

4.4 Rešenja

Rešenje 4.4

```
from PyQt5 import QtCore, QtGui, QtWidgets
import ui_4_04
import sys, os, random, string, time

__author__ = 'Milan Cugurovic & Ivan Ristovic, 2020.'

class HackerApp(ui_4_04.Ui_MainWindow, QtWidgets.QMainWindow):
    def __init__(self):
        super(HackerApp, self).__init__()
        self.setupUi(self)

        # Start button
        self.start.clicked.connect(self.pressed)

    def pressed(self):
        print("engine started...")
        for i in range(0, 100):
            print("...{}...".format(''.join(random.choices(string.ascii_uppercase +
    string.digits, k=random.randint(20, 40)))))
            self.progressBar.setProperty("value", i)
            if i%2==0:
                self.label_msg.setText(''.join(random.choices(string.ascii_uppercase
    + string.digits, k=random.randint(20, 40))))
            time.sleep(1)

if __name__=="__main__":
    app = QtWidgets.QApplication(sys.argv)
    hacker = HackerApp()
    hacker.show()
    sys.exit(app.exec_())
```

Rešenje 4.5

```
import ui_4_05
from PyQt5 import QtCore, QtGui, QtWidgets
import sys
import re

__author__ = 'Milan Cugurovic & Ivan Ristovic, 2020.'

class CalculatorClass(ui_4_05.Ui_Calculator, QtWidgets.QMainWindow):
    def __init__(self):
        super(CalculatorClass, self).__init__()
        self.setupUi(self)

        # Dodajem akcije za press the number
        self.pushButton_d0.clicked.connect(lambda: self.display_screen('0'))
        self.pushButton_d1.clicked.connect(lambda: self.display_screen('1'))
        self.pushButton_d2.clicked.connect(lambda: self.display_screen('2'))
        self.pushButton_d3.clicked.connect(lambda: self.display_screen('3'))
        self.pushButton_d4.clicked.connect(lambda: self.display_screen('4'))
        self.pushButton_d5.clicked.connect(lambda: self.display_screen('5'))
        self.pushButton_d6.clicked.connect(lambda: self.display_screen('6'))
        self.pushButton_d7.clicked.connect(lambda: self.display_screen('7'))
        self.pushButton_d8.clicked.connect(lambda: self.display_screen('8'))
        self.pushButton_d9.clicked.connect(lambda: self.display_screen('9'))

        # Dodajem akciju za clear
        self.pushButton_clr.clicked.connect(self.lineEdit.clear)

        # Dodajem akciju za backspace
        self.pushButton_del.clicked.connect(self.lineEdit.backspace)

        # Dodajem akcije za operacije
        self.pushButton_add.clicked.connect(lambda: self.display_operator('+'))
        self.pushButton_sub.clicked.connect(lambda: self.display_operator('-'))
```

```
34          self.pushButton_mul.clicked.connect(lambda: self.display_operator('*'))
            self.pushButton_div.clicked.connect(lambda: self.display_operator('/'))
36          self.pushButton_mod.clicked.connect(lambda: self.display_operator('%'))

38          # Dodajem finalno akciju za =
            self.pushButton_eq.clicked.connect(self.calculate)
40
        # Ispisujem broj value na ekran
42      def display_screen(self, digit):
            self.lineEdit.insert(digit)
44
        def display_operator(self, operator):
46          if len(self.lineEdit.text())>0 and self.lineEdit.text()[-1] not in ['+', '-',
         '*', '/']:
                self.lineEdit.insert(operator)
48
        def calculate(self):
50          data = str(self.lineEdit.text())
            operands = re.split('\+|\-', data)
52          operators = list(filter(lambda c: c in ['+', '-'], data))
            assert len(operators) + 1 == len(operands)
54          operands = list(map(lambda expression: self.evaluate(expression), operands))

56          result = int(operands[0])
            for i in range(0, len(operators)):
58              nextOperand = operands[i+1]
                nextOperator = operators[i]
60
                if nextOperator=='+':
62                  result += int(nextOperand)
                else:
64                  result -= int(nextOperand)
            self.lineEdit.setText(str(result))
66
        def evaluate(self, data):
68          operands = re.split('\*|/', data)
            if len(operands)==1:
70              return data
            operators = list(filter(lambda c: c in ['*', '/'], data))
72          assert len(operators) + 1 == len(operands)

74          result = int(operands[0])
            for i in range(0, len(operators)):
76              nextOperand = operands[i+1]
                nextOperator = operators[i]
78
                if nextOperator=='*':
80                  result *= int(nextOperand)
                else:
82                  result = int(result/float(nextOperand))
            return result
84
if __name__ == "__main__":
86      app = QtWidgets.QApplication(sys.argv)
        calculator = CalculatorClass()
88      calculator.show()
        sys.exit(app.exec_())
```

5

Funkcionalno programiranje

Potrebno je imati instaliran GHC kompajler na računaru. Za projekte je potrebno instalirati Haskell Stack sistem.

Literatura:

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

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

(c) https://docs.haskellstack.org/en/stable/README/

5.1 Uvod

5.1.1 Uvodni primeri

Zadatak 5.1 Napisati funkciju `main` koja ispisuje poruku `Zdravo! :)`.

```
-- Ovako pisemo jednolinijske komentare

{-
    Ovako pisemo
    viselinijske
    komentare
-}

{-
  Pretpostavljamo da smo instalirali Glasgow Haskell Compiler - ghc
  (uz njega dolazi i interpreter - ghci)

  Pokrenimo interpreter (ghci):

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

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

  Specijalne komande za interpreter pocinju sa `:`

  Interpretatoru dodajemo mogucnost izvrsavanja funkcija iz izvorne
  datoteke ime_programa.hs komandom `:load` ili krace `:l`:

  :load ime_programa.hs

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

  Prelude> main
  "Zdravo! :)"

  Ukoliko promenimo sadrzaj izvornih fajlova, mozemo naterati
```

```
  interpreter da ih ponovo ucita komandom `:r`

  Iz interpretera se izlazi komandom `:quit` ili krace `:q`
  Prelude> :quit

  Haskell programe takodje mozemo kompajlirati uz pomoc ghc-a:

  ghc ime_programa.hs

  I onda ih normalno pokrenuti:

  ./ime_programa
-}

main = print "Zdravo! :)"
```

Zadatak 5.2 Napisati funkciju `duplo n` koja računa dvostruku vrednost celog broja `n`.

```
{-
  Osnovni tipovi:
  Bool
  Char
  String
  Int
  Integer
  Float
  Double

  Tipski razredi (definisu funkcije koje neki tip mora da implementira):
  Eq - tipovi sa jednakoscu (==,/=)
  Ord - tipovi sa uredjenjem (<,<=,>,>=,...)
  Num - numericki tipovi (+,-,*,...)
  Integral - celobrojni tipovi (div, mod,...)
  Fractional - razlomacki tipovi (/, recip,...)

  Potpis funkcije (primeri):
  f :: Int -> Int                 - 1 argument
  f :: Int -> Int -> Int -> Int   - 3 argumenta

  Informacije o bilo kom objektu mozete dobiti naredbom:
  :info ime_objekta

  Tip funkcije (ili izraza) se moze dobiti naredbom
  :type izraz
-}

duplo :: Int -> Int
duplo a = a * 2

-- posto je (*) funkcija dva argumenta, mozemo da
-- iskoristimo cinjenicu da se Karijevim postupkom od (*)
-- moze dobiti funkcija jednog argumenta parcijalnom
-- primenom (*) na 2 - isto vazi za svaku funkciju
duplo' :: Int -> Int
duplo' = (*2)
```

Zadatak 5.3 Napisati funkciju `ostatak3 n` koja računa ostatak pri deljenju broja `n` brojem tri.

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

  Funkcije:
  mod, div, log, sqrt, sin, cos, tan

-}

ostatak3 n = mod n 3
```

```
13
   -- posto je mod funkcija, pise se prefiksno
15 -- mozemo funkcije pisati infiksno izmedju simbola `:
   ostatak3' n = n `mod` 3
17
   -- slicno, infiksne operatore mozemo pisati prefiksno u zagradama:
19 -- zbir a b = (+) a b
```

Zadatak 5.4 Napisati funkciju `korenCeli n` koja računa realni koren celog broja `n` korišćenjem ugrađene funkcije `sqrt`.

```
1  -- ukoliko koristimo funkcije koje su definisane za realne
   -- tipove nad argumentima koji su celobrojnog tipa potrebno
3  -- je prevesti vrednost u nadklasu Num koriscenjem funkcije
   -- fromIntegral - Haskell je striktno tipiziran
5
   korenCeli :: Int -> Double
7  korenCeli n = sqrt (fromIntegral n)
```

Zadatak 5.5 Napisati funkciju `sumaPrvih n` koja računa sumu prvih `n` prirodnih brojeva (rekurzivno, bez korišćenja formule).

```
1  -- Ne postoje petlje, jedini nacin ponavljanja je rekurzija

3  -- Blokovi se u Haskelu odvajaju indentacijom
   -- Haskell je striktan sto se tice poravnanja - blokove moramo
5  -- lepo ravnati inace cemo dobiti gresku

7  -- Uslovni izraz (mora imati else granu)
   sumaPrvih n =
9      if n == 0 then 0
       else n + sumaPrvih(n-1)
11
   -- Ogradjene jednacine (guards) - mozemo ih koristiti umesto
13 -- uslovnih izraza, bitan je redosled navodjenja, otherwise
   -- slucaj treba uvek staviti na kraju
15 sumaPrvih' n             -- primetimo da nema znaka "=" !
       | n == 0 = 0
17     | otherwise = n + sumaPrvih'(n-1)

19 -- jasno, ovaj zadatak se trivijalno moze resiti bez rekurzije
   -- ako znamo liste (vraticemo se na ovaj primer u nastavku):
21 sumaPrvih'' n = sum [1..n]
```

Zadatak 5.6 Napisati funkciju `parMax p` koja određuje veći element iz para realnih brojeva `p`.

```
1  {-
     Torke (Tuples) su uredjeni niz vrednosti nekog tipa
3    (ne nuzno istog):

5    ('a',1,[1.2, 2.3],"torka")

7    Funkcije definisane za parove (n=2):
     fst - vraca prvi element para
9    snd - vraca drugi element para
   -}
11
   parMax :: (Double, Double) -> Double
13 parMax p
     | fst p <= snd p = snd p
15   | otherwise = fst p

17 -- mozemo koristiti ugradjenu funkciju max:
   parMax' :: (Double, Double) -> Double
19 parMax' p = max (fst p) (snd p)

21 {-
     Uparivanje sablona (pattern matching)
23
```

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

  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 torki:
    prviParan (x,_) = even x
-}

parMax'' :: (Double, Double) -> Double
parMax'' (l,r) = max l r
```

Zadatak 5.7 Napisati funkciju `lista a b` koja pravi listu celih brojeva iz segmenta $[a, b]$. U slučaju da granice segmenta nisu ispravne, rezultat je prazna lista.

```
{-
  Lista je kolekcija vrednosti istog tipa:
  [element1,element2,...]

  U Haskell-u je lista osnovna struktura podataka za cuvanje
  kolekcija - lista je jednostruko ulancana i imutabilna

  Treba imati u vidu da, zbog implementacije, neke operacije su
  znatno sporije od ostalih jer pristup elementima nije konstantan
  kao kod niza npr.

  Operacije:
    - konstrukcija liste: operator (:)
      (:) :: a -> [a] -> [a]
      primer: 5 : [1,2,3]    (rezultat je [5,1,2,3])
    - konkatenacija listi: operator (++)
      (++) :: [a] -> [a] -> [a]
      primer: [1,2] ++ [3,4] (rezultat je [1,2,3,4])
    - indeksiranje liste: operator (!!)
      (!!) :: [a] -> Int -> a
      primer: [1,2,3] !! 1   (rezultat je 2)
    - raspon (range):
      [a..b]
      primer: [1..5]         (rezultat je [1,2,3,4,5])
      primer: [5,4..1]       (rezultat je [5,4,3,2,1])
      primer: [1,3..10]      (rezultat je [1,3,5,7,10])

  Neke korisne funkcije:
    - head :: [a] -> a
      vraca glavu liste
    - tail :: [a] -> [a]
      vraca rep liste
    - length :: [a] -> Int
      vraca duzinu liste
    - take :: Int -> [a] -> [a]
      uzima zadati broj elemenata sa pocetka liste
    - drop :: Int -> [a] -> [a]
      izbacuje zadati broj elemenata sa pocetka liste
    - null :: [a] -> Bool
```

```
      testira da li je lista prazna
    - elem :: Eq a => a -> [a] -> Bool
      (potpis je pojednostavljen, elem ne radi samo za liste)
      testira da li se element nalazi u listi
      elementi se porede po jednakosti, stoga elementi liste
      moraju biti u tipskom razredu Eq - operator (==)


  Niske su liste karaktera, na primer:
  niska1 = ['A','l','a','d','d','i','n']
  niska2 = "Aladdin"

  Mozemo niske tretirati kao [Char] ili kao String, nema razlike
-}

lista :: Int -> Int -> [Int]
lista a b = [a..b]

-- Napomena:
-- posto (..) nije operator vec deo sintakse, _ne mozemo_
-- ukloniti argumente tako sto napisemo:
-- lista = (..)
```

5.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 5.8 Napisati funkciju `proizvodPrvih n` koja računa proizvod prvih `n` prirodnih brojeva (rekurzivno, bez korišćenja formule). Pretpostaviti da je argument ispravan.

Primer 1

```
POKRETANJE: proizvodPrvih 5
IZLAZ:
  120
```

Primer 2

```
POKRETANJE: proizvodPrvih 30
IZLAZ:
  265252859812191058636308480000000
```

[Rešenje 5.8]

Zadatak 5.9 Napisati funkciju `prost n` koja vraća `True` ako je `n` prost, `False` inače. Pretpostaviti da je argument ispravan.

Primer 1

```
POKRETANJE: prost 5
IZLAZ:
  True
```

Primer 2

```
POKRETANJE: prost 12
IZLAZ:
  False
```

[Rešenje 5.9]

Zadatak 5.10 Napisati funkciju `nzd a b` koja računa najveći zajednički delilac brojeva `a` i `b` (koristiti Euklidov algoritam). Pretpostaviti da su argumenti ispravni.

Primer 1

```
POKRETANJE: nzd 12 15
IZLAZ:
  3
```

Primer 2

```
POKRETANJE: nzd 12 13
IZLAZ:
  1
```

[Rešenje 5.10]

Zadatak 5.11 Napisati funkciju `tipJednacine a b c` koja vraća tip kvadratne jednačine $a * x^2 + b * x + c = 0$ (`Degenerisana`, `Jedno resenje`, `Dva resenja`, `Bez resenja`).

Primer 1

```
POKRETANJE: tipJednacine 1 2 1
IZLAZ:
  "Jedno resenje"
```

Primer 2

```
POKRETANJE: tipJednacine (-1) 8 5
IZLAZ:
  Dva resenja"
```

[Rešenje 5.11]

Zadatak 5.12 Napisati funkciju `izDekadne x osn` koja prebacuje broj `x` iz dekadne u osnovu `osn` i funkciju `uDekadnu x osn` koja prebacuje broj `x` iz osnove `osn` u dekadnu osnovu. Pretpostaviti da je $osn > 1$ i $osn < 10$.

Primer 1

```
Pokretanje: izDekadne 101 2
Izlaz:
 1100101
```

Primer 2

```
Pokretanje: uDekadnu 765 8
Izlaz:
 501
```

[Rešenje 5.12]

Zadatak 5.13 Napisati funkciju `ceoDeo x` koja računa ceo deo korena pozitivnog broja `x` (bez korišćenja ugrađenih funkcija za koren i/ili stepen).

Primer 1

```
Pokretanje: ceoDeo 15
Izlaz:
 3
```

Primer 2

```
Pokretanje: ceoDeo 100
Izlaz:
 10
```

[Rešenje 5.13]

Zadatak 5.14 Napisati funkciju `harm n` koja pravi listu prvih `n` elemenata harmonijskog reda. Pretpostaviti da je argument ispravan.

Primer 1

```
Pokretanje: harm 2
Izlaz:
 [1.0,0.5]
```

Primer 2

```
Pokretanje: harm 5
Izlaz:
 [1.0,0.5,0.3333333333333333,0.25,0.2]
```

[Rešenje 5.14]

Zadatak 5.15 Napisati funkciju `delioci n` koja pravi listu svih pravih delioca pozitivnog broja `n`. Pretpostaviti da je argument ispravan.

Primer 1

```
Pokretanje: delioci 12
Izlaz:
 [2,3,4,6]
```

Primer 2

```
Pokretanje: delioci 13
Izlaz:
 []
```

[Rešenje 5.15]

Zadatak 5.16 Napisati funkciju `nadovezi l1 l2 n` koja nadovezuje na listu `l1` `n` puta listu `l2`. Pretpostaviti da je argument `n` pozitivan broj.

Primer 1

```
Pokretanje: nadovezi [1] [2] 3
Izlaz:
 [1,2,2,2]
```

Primer 2

```
Pokretanje: nadovezi [] [1,2,3] 2
Izlaz:
 [1,2,3,1,2,3]
```

[Rešenje 5.16]

5.1.3 Zadaci za vežbu

Zadatak 5.17 Napisati funkciju `sumaKvadrata n` koja računa sumu kvadrata prvih `n` prirodnih brojeva (rekurzivno, bez korišćenja formule).

Primer 1

```
Pokretanje: sumaKvadrata 5
Izlaz:
 55
```

Primer 2

```
Pokretanje: sumaKvadrata 10
Izlaz:
 385
```

Zadatak 5.18 Napisati funkciju `brojDelilaca n` koja vraća broj pravih delilaca prirodnog broja `n`.

Primer 1

```
POKRETANJE: brojDelilaca 5
IZLAZ:
 0
```

Primer 2

```
POKRETANJE: brojDelilaca 12
IZLAZ:
 4
```

Zadatak 5.19 Napisati funkciju `fib n` koja računa `n`-ti element Fibonačijevog niza. Pretpostaviti da je argument ispravan.

Primer 1

```
POKRETANJE: fib 5
IZLAZ:
 5
```

Primer 2

```
POKRETANJE: fib 12
IZLAZ:
 144
```

Zadatak 5.20 Napisati funkciju `osnova x osn1 osn2` koja prebacuje broj `x` iz osnove `osn1` u osnovu `osn2`. Pretpostaviti da su `osn1` i `osn2` brojevi veći od 1 i manji od 10.

Primer 1

```
POKRETANJE: osnova 100 10 3
IZLAZ:
 10201
```

Primer 2

```
POKRETANJE: osnova 525 8 2
IZLAZ:
 101010101
```

Zadatak 5.21 Napisati funkciju `parni n` koja pravi listu prvih `n` parnih prirodnih brojeva. Pretpostaviti da je argument ispravan.

Primer 1

```
POKRETANJE: parni 3
IZLAZ:
 [2,4,6]
```

Primer 2

```
POKRETANJE: parni 10
IZLAZ:
 [2,4,6,8,10,12,14,16,18,20]
```

Zadatak 5.22 Napisati funkciju `fibLista n` koja pravi listu prvih `n` elemenata Fibonačijevog niza. Pretpostaviti da je argument ispravan.

Primer 1

```
POKRETANJE: fibLista 5
IZLAZ:
 [1,1,2,3,5]
```

Primer 2

```
POKRETANJE: fibLista 10
IZLAZ:
 [1,1,2,3,5,8,13,21,34,55]
```

Zadatak 5.23 Napisati funkciju `jednocifreniDelioci n` koja pravi listu svih jednocifrenih delilaca prirodnog broja `n`. Pretpostaviti da je argument ispravan.

Primer 1

```
POKRETANJE: jednocifreniDelioci 15
IZLAZ:
 [1,3,5]
```

Primer 2

```
POKRETANJE: jednocifreniDelioci 40
IZLAZ:
 [1,2,4,5,8]
```

5.2 Liste, funkcije višeg reda

Zadatak 5.24 Definisati funkciju `uvecaj lista` koja svaki element celobrojne liste uvećava za jedan.

```
{-
Neke jednostavne funkcije viseg reda:
  map :: (a -> b) -> [a] -> [b]
      primenjuje funkciju (a -> b) na listu elemenata tipa a
      moze se reci da transformise listu koristeci datu funkciju ili
      da selektuje finalni rezultat (slicno SELECT-u u SQL upitima) -
      stoga se u nekim jezicima zove "transform" (C++) ili "Select" (C#)

      primer: map (\t -> t + 1) [1, 2, 3]   -- vraca [2,3,4]
      primer: map (+1) [1, 2, 3]            -- vraca [2,3,4]
      primer: map fst [(1,2), (3,4)]        -- vraca [1,3]

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

```
      filtrira listu koristeci predikat (a -> Bool) - svi elementi
      za koje taj predikat vrati True se nalaze u rezultatu

      primer: filter (\t -> t > 2) [1, 2, 3] -- vraca [3]
      primer: filter (>2) [1, 2, 3]          -- vraca [3]

  all :: (a -> Bool) -> [a] -> Bool   *
      proverava da li svi elementi date liste zadovoljavaju predikat

      * tip nije bas ovakav ako se proveri u ghci, razlog je zato sto je all
        jos vise generic - radi ne samo za liste nego za bilo koji tip
        koja pripada odredjenoj klasi (Foldable - definisani foldl i foldr
        - videti opise za foldl i foldr u nastavku teksta)

      primer: all (\t -> t < 2) [1, 2, 3] -- vraca False
      primer: all (<5) [1, 2, 3]          -- vraca True

  any :: (a -> Bool) -> [a] -> Bool
      slicno kao all samo sto vraca True ako neki od elemenata zadovoljava
      predikat

      primer: any (\t -> t < 2) [1, 2, 3] -- vraca True
      primer: any (>5) [1, 2, 3]          -- vraca False
-}

uvecaj lista = map (+1) lista

-- ili krace:
uvecaj' = map (+1)
```

Zadatak 5.25 Definisati funkciju `spoji lista1 lista2` koja pravi listu uređenih parova tako što spaja redom elemente prve liste sa elementima druge liste u parove rezultujuće liste.

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

spoji lst1 lst2 = zip lst1 lst2

-- ili krace
spoji' = zip
```

Zadatak 5.26 Definisati funkciju `pozitivni lista` koja izdvaja sve pozitivne elemente iz liste.

```
{-
  Lambda funkcije - sintaksa:
     \arg1 arg2 ... argn -> izraz

  Funkcija filter ocekuje predikat, tj funkciju tipa:
    a -> Bool
-}

pozitivni lista = filter (\x -> x > 0) lista

-- ili krace
pozitivni = filter (>0)

-- naravno, preferiramo (>0) jer je krace i citljivije
```

Zadatak 5.27 Definisati funkciju `prviNegativni lista` koja izdvaja najduži prefiks negativnih elemenata liste.

```
{-
  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 - razdvaja listu parova u liste ciji su elementi prvi, odnosno drugi elementi
     parova
  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
-}

prviNegativni lista = takeWhile (<0) lista
```

Zadatak 5.28 Definisati funkciju `sum lista` koja računa sumu elemenata celobrojne liste korišćenjem ugrađene funkcije `foldr` .

```
{-
  foldl :: (b -> a -> b) -> b -> [a] -> b     *
  foldr :: (a -> b -> b) -> b -> [a] -> b
  (izraz fold cemo koristiti kad je nebitno o kojem tacno fold-u je rec)
      pocevsi od binarne funkcije, pocetne vrednosti (akumulator) i kolekcije,
      fold primenjuje funkciju redom na elemente kolekcije i akumulator i
      rezultat primene funkcije postaje novi akumulator
      (u nekim jezicima se zove "reduce" (Python), "accumulate" (C++) mada nisu
      svuda isti parametri)

      razlike izmedju foldl i foldr:
          - foldl radi sleva na desno (normalni poredak)
          - foldr radi zdesna na levo (obrnuti poredak)
          - NAPOMENA: primetimo da nije isti redosled akumulatora i elementa!

      primer za foldl:
          funkcija: (+)   ;   acc: 0  ;  kolekcija: [1, 2, 3, 4, 5]
              koraci koje foldl radi:
              0 + 1 = 1  --  0 je acc, 1 je trenutni element,  1 postaje acc
              1 + 2 = 3  --  1 je acc, 2 je trenutni element,  3 postaje acc
              3 + 3 = 6  --  3 je acc, 3 je trenutni element,  6 postaje acc
              6 + 4 = 10 --  6 je acc, 4 je trenutni element, 10 postaje acc
              10 + 5 = 10 -- 10 je acc, 5 je trenutni element, 15 postaje acc
              -- nema vise elemenata, rezultat foldl-a je 15 - zbir liste

      postoje i varijante foldl1/foldr1 u slucajevima kada za akumulator
      zelimo da uzmemo prvi/poslednji element kolekcije

      * slicno kao all i any, i fold je genericki, radi ne samo za liste
        potpis iznad je pojednostavljen da bi bilo lakse razumevanje
-}

sum1 lista = foldl (+) 0 lista
sum2 lista = foldr (+) 0 lista

-- probati za (-), koji nije komutativan
```

Zadatak 5.29 Definisati funkciju `absSume lista_listi` koja na osnovu liste listi celih brojeva pravi listu apsolutnih suma elemenata liste listi korišćenjem kompozicije funkcija za rad sa listama.

```
{-
  Kompozicija funkcija

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

-- sum - ugradjena funkcija koja odredjuje sumu elemenata liste
-- abs - ugradjena funkcija za odredjivanje apsolutne vrednosti broja

absSume lista_listi = map (abs . sum) lista_listi
```

Zadatak 5.30 Definisati funkciju `sledbenici l` koja vraća listu sledbenika elemenata liste `l` koji su prirodni brojevi.

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

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

  to mozemo jednostavno uraditi koristeci funkcije map i filter

  map f (filter p xs)
-}

sledbenici = map (+1) (filter (>0) lista)
```

Zadatak 5.31 Definisati rekurzivnu funkciju `list_elem x l` koja proverava da li lista `l` sadrži dati element `x`. Dodatno, definisati prethodnu funkciju korišćenjem funkcija `or` i `map`.

```
list_elem :: Eq a => a -> [a] -> Bool
list_elem x = or . map (== x)

-- ili krace
list_elem' x = any (==x)
```

Zadatak 5.32 Napisati biblioteku za rad sa tačkama u 2D ravni. Definisati tipove `Tacka` (koordinate su realni brojevi) i `Putanja` (niz tačaka). Implementirati funkcije:

- `tacka x y` - konstruiše tačku sa datim koordinatama
- `putanja [t1, t2...]` - konstruiše putanju od date liste tačaka
- `duzinaPutanje p` - vraća dužinu putanje `p`
- `translirajTacku t dx dy` - translira tacku t za vektor `(dx, dy)`
- `rastojanje t1 t2` - vraća rastojanje između tačaka `t1` i `t2`
- `uKrugu r [t1, t2...]` - iz liste tačaka vraća one koje se nalaze u krugu poluprečnika `r` od tačke `(0,0)`
- `translirajPutanju p dx dz` - translira sve tačke date putanje za vektor `(dx, dy)`
- `nadovezi t p` - nadovezuje tačku `t` na kraj putanje `p`
- `spojiPutanja p1 p2` - spaja putanje `p1` i `p2` i vraća novu putanju
- `centroid [t1, t2...]` - vraća centroid liste tačaka
- `kvadrantTacke t` - vraća kvadrant tačke, 0 ako je to koordinatni početak
- `kvadrantPutanje p` - vraća kvadrant u kom se nalazi putanja ako je ona u celosti sadržana u nekom od njih, 0 inače
- `tackeUKvadrantu kv [t1, t2...]` - iz liste tačaka vraća samo one koje su u kvadrantu `kv`
- `tackeVanKvadranta kv [t1, t2...]` - iz liste tačaka vraća one koje nisu u kvadrantu `kv`
- `maksimumi [t1, t2...]` - vraća uređeni par koji predstavlja maksimume `x` i `y` koordinata tačaka iz date liste

```
module Lib where

-- Kreiramo biblioteku za rad sa grafikom u 2D

-- Pravimo alias za tip uredjenog para realnih brojeva,
-- nesto slicno kao `using` u C++-u
type Tacka = (Float, Float)
```

```haskell
-- Funkcija koja pravi tacku od datih koordinata
-- Primetimo da korisnik ne zna internu reprezentaciju
-- (proveriti sa :t tacka)
-- Naravno, sa :info Tacka vidi se definicija tipa Tacka
tacka :: Float -> Float -> Tacka
tacka x y = (x,y)

-- Funkcija koja predstavlja koordinatni pocetak
-- Ne mozemo koristiti O zato sto funkcije moraju poceti
-- malim slovom
o :: Tacka
o = (0.0, 0.0)

-- Putanja predstavlja spisak tacaka
type Putanja = [Tacka]

-- Posto je putanja lista tacaka, vracamo argument bez izmene
-- Mozemo koristiti funckiju identiteta da definisemo nasu funkciju
putanja :: [Tacka] -> Putanja
putanja = id
-- skraceno za: putanja tacke = tacke

-- Sabloni listi:
--   length [_] = 1                    -- [_] predstavlja listu sa jednim elementom
--   length lst = 1 + length (tail lst) -- lst predstavlja listu
duzinaPutanje :: Putanja -> Int
duzinaPutanje = length

-- Translira tacku za dati pomeraj
translirajTacku :: Tacka -> Float -> Float -> Tacka
translirajTacku (x,y) xt yt = tacka (x + xt) (y + yt)
-- bolje nego da vracamo par (x+xt, y+yt) jer ako promenimo
-- implementaciju tipa Tacka u nekom trenutku, ovo ce i dalje raditi!

rastojanje :: Tacka -> Tacka -> Float
rastojanje (x1, y1) (x2, y2) = sqrt $ (x1-x2)^2 + (y1-y2)^2

-- Vraca tacke koje se nalaze u krugu datog poluprecnika sa centrom u O
uKrugu :: Float -> [Tacka] -> [Tacka]
uKrugu r lst = [t | t <- lst, rastojanje o t < r]

-- Translira sve tacke date putanje za dati vektor
translirajPutanju :: Putanja -> Float -> Float -> Putanja
translirajPutanju putanja x y = map (\t -> translirajTacku t x y) putanja

-- Nadovezuje tacku na putanju
nadovezi :: Tacka -> Putanja -> Putanja
nadovezi t putanja = reverse $ t : (reverse putanja)
-- pokazujemo funkciju reverse, moze bolje naravno

spojiPutanje :: Putanja -> Putanja -> Putanja
spojiPutanje = (++)

centroid :: [Tacka] -> Tacka
centroid ts = tacka prosekX prosekY
    where prosekX = prosek $ map fst ts
          prosekY = prosek $ map snd ts
          prosek lst = (sum lst) / (fromIntegral $ length lst)

kvadrantTacke :: Tacka -> Int
kvadrantTacke (x,y)
    | x > 0 && y > 0 = 1
    | x < 0 && y > 0 = 2
    | x < 0 && y < 0 = 3
    | x > 0 && y < 0 = 4
    | otherwise = 0        -- koord. pocetak

-- Vraca kvadrant u kom se nalazi putanja ako je ona u celosti sadrzana u nekom od
-- njih, 0 inace
kvadrantPutanje :: Putanja -> Int
kvadrantPutanje lst = if istiKvadranti then head kvadranti else 0
    where kvadranti = map kvadrantTacke lst
          istiKvadranti = all (== head kvadranti) (tail kvadranti)
```

```
83 -- Vraca samo one tacke iz datog kvadranta
   tackeUKvadrantu :: Int -> [Tacka] -> [Tacka]
85 tackeUKvadrantu k = filter (\t -> kvadrantTacke t == k)
   -- skraceno za (mozemo delimicno primeniti funkciju):
87 -- tackeUKvadrantu k tacke = filter (\t -> kvadrantTacke t == k) tacke

89 tackeVanKvadranta :: Int -> [Tacka] -> [Tacka]
   tackeVanKvadranta k = filter (\t -> (not . (==k)) (kvadrantTacke t))
91 -- moze t != k, ovde samo pokazujemo kompoziciju funkcija - (.)

93 -- Vraca par maksimuma X i Y koordinata tacaka, redom
   maksimumi :: [Tacka] -> (Float, Float)
95 maksimumi lst = (maksimum $ map fst lst, maksimum $ map snd lst)
       where maksimum (x:xs) = foldl (\acc e -> if e > acc then e else acc) x xs
97     -- ekvivalentno sa:
       -- maksimum = foldl1 (\acc e -> e > if acc then e else acc)
99     -- ili, jos krace (i preferirano):
       -- maksimum = foldl1 max
```

5.2.1 Zadaci za samostalni rad sa rešenjima

Zadatak 5.33 Napisati funkcije koje određuju glavu i rep proizvoljne liste bez korišćenja ugrađenih funkcija za rad sa listama.

Primer 1

```
POKRETANJE: glava [1,2,3]
IZLAZ:
  1
```

Primer 2

```
POKRETANJE: rep [2]
IZLAZ:
  []
```

[Rešenje 5.33]

Zadatak 5.34 Napisati funkciju `parni a b` koja generiše listu parnih celih brojeva iz segmenta $[a, b]$ i funkciju `neparni a b` koja generiše listu neparnih celih brojeva iz segmenta $[a, b]$.

Primer 1

```
POKRETANJE: parni 1 10
IZLAZ:
  [2,4,6,8,10]
```

Primer 2

```
POKRETANJE: neparni [2, 7]
IZLAZ:
  [3,5,7]
```

[Rešenje 5.34]

Zadatak 5.35 Napisati funkciju `parovi a b c d` koja generiše listu parova celih brojeva (x, y), za koje `x` pripada segmentu $[a, b]$, a `y` pripada segmentu $[c, d]$.

Primer 1

```
POKRETANJE: parovi 1 3 2 4
IZLAZ:
  [(1,2),(1,4),(3,2),(3,4)]
```

Primer 2

```
POKRETANJE: parovi 1 2 3 4
IZLAZ:
  [(1,3),(1,4),(2,3),(3,4)]
```

[Rešenje 5.35]

Zadatak 5.36 Napisati funkciju `zavisnoY a b` koja generiše listu parova celih brojeva (x, y), za koje `x` pripada segmentu $[a, b]$, a `y` pripada segmentu $[x, b]$.

Primer 1

```
POKRETANJE: zavisnoY 1 3
IZLAZ:
  [(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)]
```

Primer 2

```
POKRETANJE: zavisnoY 0 2
IZLAZ:
  [(0,0),(0,1),(0,2),(1,1),(1,2),(2,2)]
```

[Rešenje 5.36]

Zadatak 5.37 Napisati funkciju `bezbedanRep l` koja ukoliko je lista `l` prazna vraća praznu listu, inače vraća rep liste `l`, koristeći: a) uslovne izraze b) ograđene jednačine c) uparivanje šablona

Primer 1

```
POKRETANJE: bezbedanRep [1,2,3]
IZLAZ:
  [2,3]
```

Primer 2

```
POKRETANJE: bezbedanRep []
IZLAZ:
  []
```

[Rešenje 5.37]

Zadatak 5.38 Napisati funkciju `savrseni n` koja pravi listu savršenih brojeva manjih od `n`. Broj je savršen ukoliko je jednak sumi svojih faktora (tj. delilaca), ne uključujući taj broj.

Primer 1

```
POKRETANJE: savrseni 50
IZLAZ:
  [6,28]
```

Primer 2

```
POKRETANJE: savrseni 1000
IZLAZ:
  [6,28,496]
```

[Rešenje 5.38]

Zadatak 5.39 Napisati funkciju `zbirPar n` koja pravi listu parova (a, b) takvih da su `a` i `b` prirodni brojevi čiji je zbir jednak `n`.

Primer 1

```
POKRETANJE: zbirPar 1
IZLAZ:
 []
```

Primer 2

```
POKRETANJE: zbirPar 10
IZLAZ:
 [(1,9),(2,8),(3,7),(4,6),(5,5),(6,4),(7,3),(8,2),(9,1)]
```

[Rešenje 5.39]

Zadatak 5.40 Napisati funkciju `poslednji l` koja određuje poslednji element proizvoljne liste `l`.

Primer 1

```
POKRETANJE: poslednji
 ["ponedeljak","nedelja"]
IZLAZ:
  "nedelja"
```

Primer 2

```
POKRETANJE: poslednji [5,4,3,2,1]
IZLAZ:
  1
```

[Rešenje 5.40]

Zadatak 5.41 Napisati funkciju `spoji l` koja spaja listu listi istog tipa `l` u jednu listu.

Primer 1

```
POKRETANJE: spoji
 [["jedan"],["tri"],["pet"]]
IZLAZ:
  ["jedan","tri","pet"]
```

Primer 2

```
POKRETANJE: spoji [[],[1],[1,2],[1,2,3]]
IZLAZ:
 [1,1,2,1,2,3]
```

[Rešenje 5.41]

Zadatak 5.42 Napisati funkciju `sufiksi l` koja pravi listu svih sufiksa proizvoljne liste `l`.

Primer 1

```
POKRETANJE: sufiksi [1,2,3]
IZLAZ:
  [[1,2,3],[2,3],[3],[]]
```

Primer 2

```
POKRETANJE: sufiksi []
IZLAZ:
  [[]]
```

[Rešenje 5.42]

Zadatak 5.43 Napisati funkciju `izbaci k l` koja izbacuje `k`-ti element iz liste `l`. U slučaju da je zadata neispravna pozicija u listi, funkcija vraća nepromenjenu listu.

Primer 1

```
POKRETANJE: izbaci 2 [1,2,3,4,5]
IZLAZ:
  [1,2,4,5]
```

Primer 2

```
POKRETANJE: izbaci (-2) [1,2,3]
IZLAZ:
  [1,2,3]
```

[Rešenje 5.43]

Zadatak 5.44 Napisati funkciju `ubaci k n l` koja ubacuje u listu `l` na poziciju `k` element `n`. U slučaju da je zadata neispravna pozicija u listi, dodati element `n` na kraj liste.

Primer 1

```
POKRETANJE: ubaci 2 5 [1,2,3]
IZLAZ:
  [1,2,5,3]
```

Primer 2

```
POKRETANJE: ubaci 10 5 [1,2,3,4,5]
IZLAZ:
  [1,2,3,4,5,5]
```

[Rešenje 5.44]

5.2.2 Zadaci za vežbu

Zadatak 5.45 Korišćenjem funkcija `and` i `map`, definisati funkciju `list_all p l` koja proverava da li svi elementi liste `l` zadovoljavaju dato svojstvo `p`.

Primer 1

```
POKRETANJE: list_all even [2,4,6]
IZLAZ:
  True
```

Primer 2

```
POKRETANJE: list_all (>3) [1,2,3,4,5]
IZLAZ:
  False
```

Zadatak 5.46 Definisati funkciju `obrni l` za obrtanje liste `l` pomoću funkcije `foldl`.

Primer 1

```
POKRETANJE: obrni [1,2,3]
IZLAZ:
  [3,2,1]
```

Primer 2

```
POKRETANJE: obrni "abcde"
IZLAZ:
  "edcba"
```

Zadatak 5.47 Definisati funkciju `delioci n` koja pravi listu delilaca datog prirodnog broja `n` korišćenjem funkcije `filter`. Korišćenjem prethodne funkcije definisati funkciju `prost n` koja proverava da li je dati prirodan broj `n` prost. Dodatno, koristeći funkcije `prost` i generisanje listi definisati funkciju `prosti n` koja određuje sve proste brojeve od 1 do `n`.

Primer 1

```
POKRETANJE: prosti 5
IZLAZ:
  [2,3,5]
```

Primer 2

```
POKRETANJE: prosti 10
IZLAZ:
  [2,3,5,7]
```

Zadatak 5.48 Definisati funkciju `sufiks l` pomoću funkcije `scanr` koja određuje sve sufikse date liste.

Primer 1

```
POKRETANJE: sufiks [1,2,3]
IZLAZ:
  [[1,2,3], [2,3], [3], []]
```

Primer 2

```
POKRETANJE: sufiks "abc"
IZLAZ:
  ["abc", "bc", č", ]
```

Zadatak 5.49 Definisati funkciju `prefiks l` koja određuje sve prefikse date liste `l` (dozvoljena je rekurzija).

Primer 1

```
POKRETANJE: prefiks [1,2,3]
IZLAZ:
  [[], [1], [1,2], [1,2,3]]
```

Primer 2

```
POKRETANJE: prefiks "abc"
IZLAZ:
  [, "a", "ab", "abc"]
```

Zadatak 5.50 Definisati rekurzivnu funkciju `ukloniDuplikate l` koja pomoću funkcija iz familije `fold` uklanja sve duplikate iz liste `l`.

Primer 1

```
Pokretanje: ukloniDuplikate [1,2,1,3]
Izlaz:
  [1,2,3]
```

Primer 2

```
Pokretanje: ukloniDuplikate "abaccddb"
Izlaz:
  "abcd"
```

Zadatak 5.51 Definisati funkciju `qsort l` sortira listu `l` u rastućem poretku. Za pivot uzeti prvi element liste.

Primer 1

```
Pokretanje: qsort [1,2,1,3]
Izlaz:
  [1,1,2,3]
```

Primer 2

```
Pokretanje: qsort "abaccddb"
Izlaz:
  "aabbccdd"
```

5.3 Korisnički tipovi, Napredni koncepti

5.3.1 Uvodni primeri

Zadatak 5.52 Definisati bulovski tip podataka.

```haskell
{-
    Algebarski tipovi

    Mozemo definisati tip podataka koristeci kljucnu rec `data`
-}

data BulovskiTip = Tacno | Netacno

-- Tacno i Netacno u ovom slucaju su "konstruktori" za tip BulovskiTip
```

Zadatak 5.53 Definisati tip podataka `Oblik` koji može biti `Kvadrat` (karakterisan stranicom), `Pravougaonik` (karakterisan dvema stranicama), `Krug` (karakterisan poluprečnikom) ili `Trougao` (karakterisan sa tri stranice).

```haskell
-- Konstruktori mogu imati argumente

data Oblik = Kvadrat Float
           | Pravougaonik Float Float
           | Krug Float
           | Trougao Float Float Float
           deriving Show

-- deriving Show znaci da nas tip Oblik pripada
-- klasi Show - koristimo cinjenicu da postoji
-- podrazumevana implementacija funkcije show
```

Zadatak 5.54 Definisati tip podataka `Zivotinja` koji može biti `Pas`, `Macka` ili `Papagaj`. Zatim definisati tip `Ljubimac` (karakterisan imenom, godinama i tipom zivotinje).

```haskell
-- Konstruktori mogu imati argumente

data Zivotinja = Pas | Macka | Papagaj

data Ljubimac = MkLjubimac { ime :: String
                           , godine :: Int
                           , tip :: Zivotinja
                           }

-- MkLjubimac je konstruktor 3 argumenta,
-- svakom dajemo ime koje mozemo koristiti kao
-- getter

-- let dzeki = MkLjubimac 3 "Dzeki" Pas
-- let starijiDzeki = dzeki { godine = 4 }
--    ovo pravi kopiju objekta sa izmenjenim poljima
```

Zadatak 5.55 Definisati tip podataka `Pravougaonik` (karakterisan dvema stranicama) i instancirati klase `Show` i `Eq` nad kreiranim tipom.

```
data Pravougaonik = MkPravougaonik { a :: Int
                                   , b :: Int
                                   }

instance Show Pravougaonik where
    show (MkPravougaonik a b) =
        "(" ++ show a ++ "," ++ show b ++ ")"

instance Eq Pravougaonik where
    (==) (MkPravougaonik a1 b1) (MkPravougaonik a2 b2) =
        a1 == a2 && b1 == b2 || a1 == b2 && b1 == a2
```

Zadatak 5.56 Napisati funckije glava i rep koje bezbedno vraćaju glavu i rep liste koristeći tip `Maybe`.

```
{-
    Maybe a

    Predstavlja opcionu vrednost (isto kao std::optional<T>,
    Optional<T> ili Nullable<T> u OOP jezicima)

    Definisan je kao (videti :info Maybe):

        data Maybe a = Nothing | Just a
-}

glava :: [a] -> Maybe a
glava []    = Nothing
glava (x:_) = Just x

rep :: [a] -> Maybe [a]
rep []     = Nothing
rep (_:xs) = Just xs
```

Zadatak 5.57 Napisati funckije glava i rep koje bezbedno vraćaju glavu i rep liste koristeći tip `Either`. U slučaju da je lista prazna, vratiti `String` sa porukom o grešci.

```
{-
    Either a b

    Predstavlja vrednosti koje mogu biti jednog ILI drugog tipa
    (nesto slicno std::variant<T1, T2...>, s tim sto je variant
    mocniji jer je sablon sa proizvoljno mnogo tipova, Either je
    fiksiran na dva)

    Definisan je kao (videti :info Maybe):

        data Either a b = Left a | Right b

    Koristi se cesto za baratanje greskama
-}

glava :: [a] -> Either String a
glava []    = Left "Prazna lista"
glava (x:_) = Right x

rep :: [a] -> Either String [a]
rep []     = Left "Prazna lista"
rep (_:xs) = Right xs
```

Zadatak 5.58 Napisati biblioteku za pomoć asistentima u arhiviranju i održavanju rezultata ispita. Definisati tipove `StepenStudija` (osnovne, master, doktorske), `Student` (karakterisan brojem indeksa, imenom, prezimenom i stepenom studija) i Rezultat (svaki student ima opcioni rezultat predstavljen brojem poena). Obezbediti da se student može ispisati na standardni izlaz i porediti po jednakosti sa ostalim studentima po broju indeksa. Implementirati funkcije:

- `rezultati studenti` - konstruiše listu rezultata za date studente, gde je svaki rezultat trenutno prazan
- `poeni student rezultati` - vraća broj poena datog studenta iz liste rezultata ili poruku ako se student ne nalazi u istoj
- `ponisti student rezultati` - poništava poene za datog studenta iz liste rezultata
- `ponistiSve rezultati` - poništava poene za sve studente iz liste rezultata
- `studije stepenStudija rezultati` - vraća samo one rezultate za studente sa datog stepena studija
- `polozeni rezultati` - vraća samo one rezultate gde je student položio ispit
- `upisi student poeni rezultati` - upisuje novi rezultat za datog studenta u listu rezultata
- `najboljih n rezultati` - vraća `n` najboljih rezultata (samo broj poena) iz liste rezultata, sortiranih opadajuće

```haskell
module Lib where

import Data.Maybe
import Data.List as List (elemIndex, sortBy)

-- Kreiramo tip koji predstavlja stepen studija
data StepenStudija = OsnovneStudije
                   | MasterStudije
                   | DoktorskeStudije
                   deriving (Show, Eq)
-- `OsnovneStudije`, `MasterStudije` i `DoktorskeStudije` su
-- konstruktori za tip `StepenStudija`.
-- Ukoliko zelimo da kreiramo objekat tipa `StepenStudija` koji
-- predstavlja `MasterStudije`, to radimo tako sto pozovemo
-- konstruktor `MasterStudije` kao bilo koju drugu funkciju.
-- Konstruktor je funkcija, u ovom slucaju bez argumenata.
-- Kazemo da StepenStudija implicitno pripada klasi Show, sto
-- znaci da moze da se pozove funkcija `show` nad objektom tog
-- tipa, pritom koristimo podrazumevanu implementaciju koja
-- ispisuje imena konstruktora

-- Napravimo tip za skracenicu, ovde je `Kratko` konstruktor
-- sa jednim argumentom
data KratkiStepenStudija = Kratko StepenStudija

-- Instancirajmo `Show` nad novim tipom
-- `Show` se sastoji samo od funkcije `show`
instance Show KratkiStepenStudija where
    show (Kratko OsnovneStudije)   = "BSc"
    show (Kratko MasterStudije)    = "MSc"
    show (Kratko DoktorskeStudije) = "PhD"

-- Jasno je da smo mogli alternativno da ne napisemo `deriving
-- Show` vec da rucno implementiramo `Show` tako da za stepen
-- studija vratimo odgovarajucu skracenicu. Ovde smo pravili novi
-- tip da bismo pokazali `deriving` i konstruktore sa argumentima

-- Definisimo studenta sa imenovanim poljima (slicno klasi u OOP)
-- Paziti na indentaciju!
data Student = MkStudent { indeks  :: String
                         , ime     :: String
                         , prezime :: String
                         , stepen  :: StepenStudija
                         }
-- U ovom slucaju, `MkStudent` je konstruktor sa 4 argumenta
-- indeks, ime, prezime i stepen su funkcije koje sluze kao
-- "geteri"
-- Studenta mozemo napraviti pozivajuci konstruktor:
--   MkStudent "123/2018" "Pera" "Peric" OsnovneStudije
-- ili, ako je s vec postojeci objekat studenta, mozemo
-- napraviti "kopiju" sa izmenjenim poljima razdvojenim zarezima
```

```
52 --   s { indeks = "124/2018" }
   --   s { indeks = "124/2018", prezime = "Peric" }
54
   -- Posto Student nije u klasi Show, definisacemo metod koji
56 -- vraca String od datog Student-a
   formatirajStudenta :: Student -> String
58 formatirajStudenta s =
       let id = indeks s
60         ip = (ime s) ++ " " ++ (prezime s)
           ss = show $ Kratko $ stepen s
62      in id ++ " : " ++ ip ++ " : " ++ ss

64 -- Instancirajmo `Show` nad studentom
   instance Show Student where
66     show = formatirajStudenta

68 -- Instancirajmo `Eq` nad studentom
   -- `Eq` se sastoji od funkcije `(==)`
70 instance Eq Student where
       s1 == s2 = (indeks s1) == (indeks s2)
72     -- naravno, moze i prefiksno:
       -- (==) s1 s2 = (indeks s1) == (indeks s2)
74

76 -- Napravimo alias za rezultate ispita
   -- Maybe predstavlja opcioni tip, definisan kao:
78 --       data Maybe a = Nothing | Just a
   -- Nesto sto je u C++-u `std::optional<T>` ili
80 -- u Javi `Optional<T>` ili u C#-u `Nullable<T>`
   -- Maybe ima jedan tipski parametar, posto mi zelimo
82 -- da cuvamo `Int` ili `Nothing` za poene na ispitu
   -- onda koristimo `Maybe Int`
84 type Rezultat = (Student, Maybe Int)

86 -- Formiramo spisak rezultata za studente
   rezultati :: [Student] -> [Rezultat]
88 rezultati = map (\s -> (s, Nothing))

90 -- Vracamo poene za studenta ili poruku ako nemamo rezultat
   -- Either je tip definisan kao:
92 --       data Either a b = Left a | Right b
   -- Dakle, imamo objekat koji moze biti tipa a ILI tipa b
94 -- Either se cesto koristi za baratanje greskama, "leva" vrednost
   -- se koristi kao specijalna
96 poeni :: Student -> [Rezultat] -> Either String (Maybe Int)
   poeni s rezultati =
98     case mi of Nothing  -> Left $ (indeks s) ++ " ne studira na fakultetu!"
                  (Just i) -> Right $ snd $ rezultati !! i
100    where mi = List.elemIndex s $ map fst rezultati

102 -- Ponisti poene za studenta
    ponisti :: Student -> [Rezultat] -> [Rezultat]
104 ponisti s rezultati = foldr azuriraj [] rezultati
        where azuriraj rez acc = if fst rez == s then (s, Nothing) : acc
106                                                 else rez : acc

108 -- Ponisti sve rezultate
    ponistiSve :: [Rezultat] -> [Rezultat]
110 ponistiSve = map (\rez -> (fst rez, Nothing))

112 -- Vraca samo rezultate sa datog stepena studija
    studije :: StepenStudija -> [Rezultat] -> [Rezultat]
114 studije ss = filter (\(s, _) -> stepen s == ss)

116 -- Vraca samo rezultate studenata koji su polozili ispit
    polozeni :: [Rezultat] -> [Rezultat]
118 polozeni = filter (\(s, mozdaRez) -> (izvuciRez mozdaRez) > 50)
        where izvuciRez Nothing    = 0
120           izvuciRez (Just rez) = rez

122 -- Funkcija koja upisuje rezultat za studenta
    -- Ako se student nalazi u rezultatima, izasao je opet - azurirati
124 -- Ako se ne nalazi, onda dodati
```

```
upisi :: Student -> Int -> [Rezultat] -> [Rezultat]
upisi s p rezultati = if elem s studenti
                      then foldr azuriraj [] rezultati
                      else (s, Just p) : rezultati
    where studenti = map fst rezultati
          azuriraj rez acc = if fst rez == s then (s, Just p) : acc
                                             else rez : acc

-- Spisak prvih n najboljih rezultata (samo poeni)
najboljih :: Int -> [Rezultat] -> [Int]
najboljih n rezultati = take n
                      $ List.sortBy (flip compare)
                    --   $ List.sortBy (\_ _ -> GT)
                      $ map (\(Just x) -> x)
                    --   $ map Data.Maybe.fromJust
                      $ filter (Nothing /=)
                      $ map snd
                      $ rezultati
```

5.3.2 Zadaci za samostalni rad sa rešenjima

Zadatak 5.59 Definisati rekurzivnu funkciju `varijacije l k` koja generiše listu koja sadrži sve varijacije sa ponavljanjem elemenata date liste `l` dužine `k`.

[Rešenje 5.59]

Primer 1

```
Pokretanje: varijacije [1,2,3] 2
Izlaz:
 [[1,1],[1,2],[1,3],[2,1],[2,2],[2,3],[3,1],[3,2],[3,3]]
```

Zadatak 5.60 Data je lista `l` koja sadrži liste ocena raznih učenika osnovne škole. Definisati funkciju `prosekOdlicni l` koja računa prosek svih odličnih učenika.

[Rešenje 5.60]

Primer 1

```
Pokretanje: prosekOdlicni [[1,2,3],[5,5,5],[4,5,5]]
Izlaz:
 4.8333335
```

Zadatak 5.61 Ana uči brojanje i razlikovanje boja koristeći kutiju punu jednobojnih kuglica. Ona prvo žmureći iz kutije izvuče određeni broj kuglica i poređa ih u niz u redosledu izvlačenja. Zatim izabere proizvoljnu boju i odredi na kojoj se sve poziciji u nizu izvučenih kuglica nalazi kuglica željene boje. Napisati funkciju `pozicije x l` koja vraća listu pozicija elementa `x` u listi `l`.

[Rešenje 5.61]

Primer 1

```
Pokretanje: pozicije "bela" ["plava","bela","bela","ljubicasta","bela"]
Izlaz:
 [1,2,4]
```

Zadatak 5.62 Učesnici nagradne igre `Sedmica` mogu proveriti dobitak putem sajta, gde se dobitna kombinacija objavljuje odmah nakon izvlačenja. Brojevi iz dobitne kombinacije se, radi jednostavnije provere pogodaka, uvek prikazuju u rastućem redosledu. Napisati funkciju `qsort l` koja rastuće sortira listu `l` algoritmom `qsort`.

[Rešenje 5.62]

Primer 1

```
Pokretanje: qsort [4,5,2,11,29,38,9]
Izlaz:
 [2,4,5,9,11,29,38]
```

Primer 2

```
Pokretanje: qsort [7,1,10,15,30,32,20]
Izlaz:
 [1,7,10,15,20,30,32]
```

Zadatak 5.63 Milan obožava da sakuplja sličice fudbalera. Da bi jednostavnije pratio koje mu sličice iz kolekcije još uvek nedostaju, čuva ih rastuće sortirane po rednom broju. Planirao je da dopuni svoju kolekciju na narednoj razmeni sličica, pa za tu priliku želi da iz svoje kolekcije izbaci sve nepotrebne duplikate koje će poneti na razmenu. Napisati funkciju `brisiPonavljanja l` koja briše sva uzastopna ponavljanja elemenata u listi `l`.

[Rešenje 5.63]

Primer 1

```
POKRETANJE: brisiPonavljanja [4,5,5,2,11,11,11]
IZLAZ:
  [4,5,2,11]
```

Primer 2

```
POKRETANJE: brisiPonavljanja [10,10,10,11]
IZLAZ:
  [10,11]
```

Zadatak 5.64 Kasirka Mica mora da ručno kuca artikle na kasi jer se pokvario skener barkodova. Pomozite Mici da taj posao obavi što brže grupisanjem artikala iste vrste na pokretnoj traci. Napisati funkciju `podlistePonavljanja l` koja grupiše sva uzastopna ponavljanja nekog elementa liste `l` u podlistu tako da rezultat bude lista listi.

[Rešenje 5.64]

Primer 1

```
POKRETANJE: podlistePonavljanja ["jabuke","jogurt","jogurt","hleb"]
IZLAZ:
  [["jabuke"],["jogurt","jogurt"],["hleb"]
```

Zadatak 5.65 Napisati funkciju `broj lista` koja vraća broj određen ciframa koje se nalaze u listi čitajući ih sa početka ka kraju liste i funkciju `brojObrnut lista` koja vraća broj određen ciframa koje se nalaze u listi čitajući ih sa kraja ka početku liste.

[Rešenje 5.65]

Primer 1

```
POKRETANJE: broj [1,0,1]
IZLAZ:
  101
```

Primer 2

```
POKRETANJE: brojObrnut [0,5,2]
IZLAZ:
  250
```

Zadatak 5.66 U toku su prijave za plesno takmičenje parova. Pored nagrade za najbolji par, dodeljuju se i pojedinačne nagrade za najboljeg ženskog i muškog takmičara. Da bi žiri jednostavnije beležio poene i odredio nagrade, potrebno je da im se dostave, pored liste parova koji se takmiče, i liste samo muških, tj. samo ženskih takmičara, odvojeno. Napisati funkciju `listaUPar lista` koja pretvara listu parova u par dve liste, tako da prva lista sadrži prve elemente svih parova, a druga druge elemente svih parova pod pretpostavkom da je prvi u paru uvek ženska osoba, a drugi muška (implementacija funkcije unzip).

[Rešenje 5.66]

Primer 1

```
POKRETANJE: listaUPar [("Ivana","Milan"), ("Ana","Jovan"), ("Anica","Petar")]
IZLAZ:
  (["Ivana","Ana","Anica"], ["Milan","Jovan","Petar"])
```

Zadatak 5.67 Nakon još jedne košarkaške sezone, potrebno je sumirati rezultate svih igrača. Svaki igrač ima jedinstveni redni broj, pod kojim se u bazi, u listi prezimena, čuva njegovo prezime, a u listi pogodaka, ostvaren broj poena u sezoni. Uparivanjem odgovarajućih podataka, napraviti za svakog igrača jedinstveni par oblika `(prezime, poeni)`. Napisati funkciju `parOdListi lista1 lista2` koja pravi listu parova od dve liste, liste prezimena i liste pogodaka, tako da prvi element svakog para bude iz prve liste, a drugi element svakog para bude iz druge liste (implementacija funkcije zip).

[Rešenje 5.67]

Primer 1

```
POKRETANJE: parOdListi ["Mikic","Peric","Jovic"] [100,76,96]
IZLAZ:
  [("Mikic",100),("Peric",76),("Jovic",96)]
```

Zadatak 5.68 Formacija igre `kolo` je polukrug sa istaknutom ulogom prvog i poslednjeg igrača. Napisati funkciju `ucesljaj mIgraci zIgraci` koja pravi jednu formaciju za kolo naizmeničnim učešljavanjem igrača iz date grupe muških i ženskih igrača, `mIgraci` i `zIgraci`, redom.

[Rešenje 5.68]

Primer 1

```
Pokretanje: ucesljaj ["Petar","Aleksa","Filip"] ["Milica","Jovana","Anica"]
Izlaz:
 ["Petar","Milica","Aleksa","Jovana","Filip","Anica"]
```

5.3.3 Zadaci za vežbu

Zadatak 5.69 Definisati alias za uređeni par `ParRazličitih a b` gde prvi i drugi element para nisu nužno istog tipa.

Zadatak 5.70 Definisati tip podataka `Krug` koji se karakteriše radijusom tipa `Float`. Definisati tip podataka `Pravougaonik` koji se definiše parom stranica (obe tipa `Float`). Instancirati `Eq` i `Show` klase nad kreiranim tipovima.

Zadatak 5.71 Aleksa i Luka žele da komuniciraju preko šifrovanih poruka koje predstavljaju listom niski. Aleksa šifruje željene podatke na sledeći način: ukoliko se niska koju šalje sastoji samo od cifara, na njen početak i kraj dodaje karakter `C`, ako se sastoji samo od malih slova, na njen početak i kraj dodaje karakter `S`, a inače na njen početak i kraj dodaje karakter `O`. Definisati sledeće funkcije koje pomažu Aleksi da pošalje Luki šifrovanu poruku:

a) `broj s` - za datu nisku `s` proverava da li su svi njeni karakteri cifre

Primer 1

```
Pokretanje: broj "123"
Izlaz:
 True
```

Primer 2

```
Pokretanje: broj ['c','a','o']
Izlaz:
 False
```

a) `mala s` - za datu nisku `s` proverava da li su svi njeni karakteri mala slova

Primer 1

```
Pokretanje: mala "pozdrav"
Izlaz:
 True
```

Primer 2

```
Pokretanje: mala ['C','a','o']
Izlaz:
 False
```

c) `sifruj ls` - datu listu niski `ls` transformiše na sledeći način: ukoliko se niska koju šalje sastoji samo od cifara, na njen početak i kraj dodaje karakter `C`, ako se sastoji samo od malih slova, na njen početak i kraj dodaje karakter `M`, a inače na njen početak i kraj dodaje karakter `O`.

Primer 1

```
Pokretanje: sifruj ["11","maj","petak"]
Izlaz:
 ["C11C","MmajM","MpetakM"]
```

Primer 2

```
Pokretanje: sifruj ["poz","Poz","poZ"]
Izlaz:
 ["MpozM","OPozO","OpoZO"]
```

Zadatak 5.72 Za razliku od Alekse i Luke, Ana i Milica imaju problem dešifrovanja podataka. Da bi Ana dešifrovala podatke koje joj Milica pošalje potrebno je da svaku nisku iz dobijene liste transformiše na sledeći način: ukoliko dobijena niska počinje cifrom, sa njenog početka izbaciti onoliko karaktera koliko ta niska ima cifara, a ukoliko dobijena niska počinje malim slovom, sa njenog početka izbaciti onoliko karaktera koliko ta niska ima malih slova. Pretpostaviti da će ovakvim dešifrovanjem Ana uvek dobiti ispravne izvorne podatke. Definisati sledeće funkcije koje pomažu Ani da dešifruje Miličine poruke:

a) `cifre s` - za datu nisku `s` vraća broj karaktera niske koji su cifre

Primer 1

```
POKRETANJE: cifre "11Maj"
IZLAZ:
 2
```

Primer 2

```
POKRETANJE: cifre ['m','a','j']
IZLAZ:
 0
```

b) `mala s` - za datu nisku `s` vraća broj karaktera niske koji su mala slova

Primer 1

```
POKRETANJE: mala "petak"
IZLAZ:
 5
```

Primer 2

```
POKRETANJE: mala ['C','a','o']
IZLAZ:
 2
```

c) `desifruj ls` - datu listu niski `ls` transformiše na sledeći način: ukoliko niska počinje cifrom, sa njenog početka izbacuje onoliko karaktera koliko ta niska ima cifara, a ukoliko dobijena niska počinje malim slovom, sa njenog početka izbacuje onoliko karaktera koliko ta niska ima malih slova.

Primer 1

```
POKRETANJE: desifruj ["aaa11","1minamaj2017","101petak"]
IZLAZ:
 ["11","maj2017","petak"]
```

Zadatak 5.73 Napisati funkciju `magicniParovi l` koja pravi listu parova čiji su prvi elementi, elementi liste prirodnih brojeva `l`, a drugi elementi odgovarajući magični brojevi elemenata liste `l`. Magičan broj prirodnog broja `n` se dobija kao proizvod njegovih cifara.

Primer 1

```
POKRETANJE: magicniParovi [12,101,154]
IZLAZ:
 [(12,2),(101,0),(154,20)]
```

Primer 2

```
POKRETANJE: magicniParovi [1000,99,111,222]
IZLAZ:
 [(1000,0),(99,81),(111,1),(222,8)]
```

Zadatak 5.74 Podaci o cenama artikala u prodavnici su zadati u obliku liste realnih brojeva. Definisati funkciju `prosek lista_cena` koja računa prosečnu cenu artikla u prodavnici.

Primer 1

```
POKRETANJE: prosek [199.99, 125, 10.99, 45.99, 123.50]
IZLAZ:
 101.09400000000001
```

5.4 Rešenja

Rešenje 5.8

```
proizvodPrvih :: Num a => Int -> a
proizvodPrvih n
      | n < 1 = 0
      | n == 1 = 1
      | otherwise = n * proizvodPrvih (n-1)
```

Rešenje 5.9

```
-- 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

-- 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
```

```haskell
-- 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)

prost :: Int -> Bool
prost n = prostTest n 2
  where prostTest n k
          | n == k = True
          | n `mod` k == 0 = False
          | otherwise = prostTest n (k+1)

-- Na narednim casovima, kad naucimo funkcije viseg reda,
-- napravicemo Eratostenovo sito :)

-- Alternativno, znajuci generatore listi (u narednim primerima),
-- mozemo da kazemo sledece: "Broj je prost ako je lista njegovih
-- pravih delilaca prazna"
prost' n = null (listaDelilaca n)
  where listaDelilaca n = [x | x <- [2..n-1], mod n x == 0]

-- Mozemo, dakle, kao i u Python-u, da pravimo nesto nalik na
-- matematicke definicije skupova
--   {x | x iz 1..n   i   n deli x}
-- "skup svih x takvih da x pripada 1..n i ...."
```

Rešenje 5.10

```haskell
nzd :: Int -> Int -> Int
nzd a b
  | b == 0 = a
  | otherwise = nzd b (a `mod` b)

-- alternativno
nzd' = gcd
```

Rešenje 5.11

```haskell
tipJednacine :: Double -> Double -> Double -> String
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"
```

Rešenje 5.12

```haskell
uDekadnu :: Int -> Int -> Int
uDekadnu x osn =
    if x == 0 then 0
    else uDekadnu (x `div` 10) osn * osn + (mod x 10)

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

Rešenje 5.13

```haskell
-- 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 :: (Ord a, Num a) => a -> a
ceoDeo x = ceoDeo' x 1
  where ceoDeo' x i
          | (i*i) > x = (i-1)
          | otherwise = ceoDeo' x (i+1)
```

Rešenje 5.14

```
-- 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
-- i na rezultat nadovezemo reciprocnu vrednost broja n
harm :: Int -> [Double]
harm n
  | n == 1 = [1.0]
  | otherwise = harm (n-1) ++ [recip n']
  where n' = fromIntegral n

-- Pritom, ovde imamo mali problem ako navedemo potpis,
-- naime da bi se radio recip, argument mora biti realan
-- broj - stoga konvertujemo n u realan broj - nesto sto
-- se prilikom automatske dedukcije samo zakljucuje
```

Rešenje 5.15

```
-- 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

-- 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

delioci n
    | n == 1 = [1]
    | otherwise = delioci' n 2
    where delioci' n k
            | k == n = []
            | n `mod` k == 0 = [k] ++ (delioci' n (k+1))
            | otherwise = delioci' n (k+1)
```

Rešenje 5.16

```
-- 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 :: [a] -> [a] -> Int -> [a]
nadovezi l1 l2 n = l1 ++ rep
  where rep = concat $ replicate n l2

-- replicate ponavlja element odredjeni broj puta - vraca
-- listu. S obzirom da je element vec lista, rezultat je
-- lista listi. Stoga, koristimo funkciju concat da ih
-- sve spojimo u jednu (flatten u nekim jezicima)
```

Rešenje 5.33

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

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

Rešenje 5.34

```haskell
parni :: Int -> Int -> [Int]
parni a b = [x | x <- [a..b], even x]

neparni :: Int -> Int -> [Int]
neparni a b = [x | x <- [a..b], odd x]
```

Rešenje 5.35

```haskell
-- Mozemo imati vise generatora

parovi :: Int -> Int -> Int -> Int -> [(Int,Int)]
parovi a b c d = [(x,y) | x <- [a..b], y <- [c..d]]

{-
obratiti paznju kako utice zamena redosleda
parovi a b c d = [(x,y) | y <- [c..d], x <- [a..b]]
-}
```

Rešenje 5.36

```haskell
-- Generatori mogu zavisiti jedni od drugih
-- kasniji generatori mogu zavisiti samo od promenljivih
-- koje su uvedene ranijim generatorima citajuci sa leva na desno
zavisnoY :: Int -> Int -> [(Int,Int)]
zavisnoY a b = [(x,y) | x <- [a..b], y <- [x..b]]
```

Rešenje 5.37

```haskell
bezbedanRep1 :: [a] -> [a]
bezbedanRep1 xs = if null xs then [] else tail xs

bezbedanRep2 :: [a] -> [a]
bezbedanRep2 xs
  | xs == [] = []
  | otherwise = tail xs

bezbedanRep3 :: [a] -> [a]
bezbedanRep3 [] = []
bezbedanRep3 (_:xs) = xs
-- bezbedanRep3 xs = tail xs
```

Rešenje 5.38

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

Rešenje 5.39

```haskell
zbirPar :: Int -> [(Int,Int)]
zbirPar n = [(a,b) | a <- [1..n], b <- [1..n], a + b == n]

-- drugi, efikasniji nacin
zbirPar' :: Int -> [(Int,Int)]
zbirPar' n = [(a,b) | a <- [1..n], b <- [n-a], b /= 0]
```

Rešenje 5.40

```haskell
poslednji :: [a] -> a
poslednji lista = lista !! poz
    where poz = length lista - 1  -- moze i: length(tail lista)

-- postoji funkcija last koja vraca poslednji element
```

Rešenje 5.41

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

{-
spoji [] = []
spoji (x:xs) = x ++ spoji xs
-}
```

Rešenje 5.42

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

Rešenje 5.43

```
izbaci :: Int -> [a] -> [a]
izbaci _ [] = []
izbaci k lst = foldr (\(i,x) acc -> if i == k then acc else x : acc) []
             $ zip [0..] lst

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

Rešenje 5.44

```
ubaci :: Int -> a -> [a] -> [a]
ubaci _ e [] = [e]
ubaci k e xs = foldr (\(i,x) acc -> if i == k then e : x : acc else x : acc) []
             $ zip [0..] xs

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

Rešenje 5.59

```
varijacije xs 0 = [[]]
varijacije xs n = concat (map (\ x -> map (x:) ys) xs)
  where ys = varijacije xs (n-1)
```

Rešenje 5.60

```
prosekOdlicni :: [[Integer]] -> Float
prosekOdlicni = prosek . filter (>= 4.5) . map prosek
    where prosek xs = realToFrac (sum xs) / fromIntegral (length xs)
```

Rešenje 5.61

```
-- 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
-- where omogucava da u okviru funkcije uvedemo novu promenljivu za neki izraz koji
    se koristi u definiciji
```

```
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
```

Rešenje 5.62

```
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]
```

Rešenje 5.63

```
-- 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)
```

Rešenje 5.64

```
-- 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)
```

Rešenje 5.65

```
-- [1,2,3] -> 321

brojObrnut :: Num a => [a] -> a
brojObrnut [] = 0
brojObrnut (x:xs) = (brojObrnut xs)*10 + x

-- [1,2,3] -> 123
-- Koriscenje funkcije brojObrnut
-- broj lista = brojObrnut (reverse lista)

-- Drugi nacin
broj [] = 0
broj (x:xs) = x*10^(length xs) + broj xs
```

Rešenje 5.66

```
-- 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
      l2 = snd rep
      rep = listaUPar xs
-}
```

Rešenje 5.67

```
-- Implementacija funkcije zip
-- [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
parOdListi [] _ = []
parOdListi _ [] = []
-- Od glava listi pravimo par koji dodajemo na pocetak rezultujuce liste za rep
parOdListi (x:xs) (y:ys) = ( (x, y) : (parOdListi xs ys))
```

Rešenje 5.68

```
-- [1,1,1] [2,2,2] -> [1,2,1,2,1,2]
-- [1,1] [2,2,2,2] -> [1,2,1,2,2,2]
-- Treba obuhvatiti slucajeve listi razlicitih duzina
-- Sledeca linija koda ne radi jer nije dozvoljeno koriscenje dzoker promenljive u
    izrazu kojim definisete vrednost funkcije
-- ucesljaj [] _ = _
ucesljaj [] lista = lista
ucesljaj lista [] = lista
ucesljaj (x:xs) (y:ys) = [x]++[y]++(ucesljaj xs ys)
```

6

Konkurentno programiranje

6.1 Jezik Scala

Scala (skr. *Scalable Language*) je moderni programski jezik kreiran od strane Martina Oderskya 2003. napravljen sa ciljem da integriše osobine objektno-orijentisanih i funkcionalnih programskih jezika.

Literatura:

(a) https://docs.scala-lang.org/tour/tour-of-scala.html

(b) https://www.tutorialspoint.com/scala/

(c) Master rad koleginice Ane Mitrović na temu primena Scala jezika u paralelizaciji raspli-nutog testiranja: http://www.racunarstvo.matf.bg.ac.rs/MasterRadovi/2016_05_04_Ana_Mitrovic/rad.pdf

Neke osnovne osobine:

(a) Scala je objektno-orijentisani jezik
Svaka vrednost je objekat, klase se mogu nasledjivati i postoje mehanizmi koji služe kao zamena za višestruko nasledjivanje

(b) Scala je funkcionalni jezik
Svaka funkcija se tretira kao vrednost i svaka vrednost je objekat - dakle, svaka funckija je objekat.

(c) Scala daje konstrukte za konkurentno i sinhronizovano procesiranje

(d) Scala je statički tipiziran jezik

(e) Scala operiše na JVM

(f) Scala može pokretati Java kod

Neke osnovne razlike u odnosu na Javu:

(a) Svi tipovi su objekti (nema primitivnih tipova)

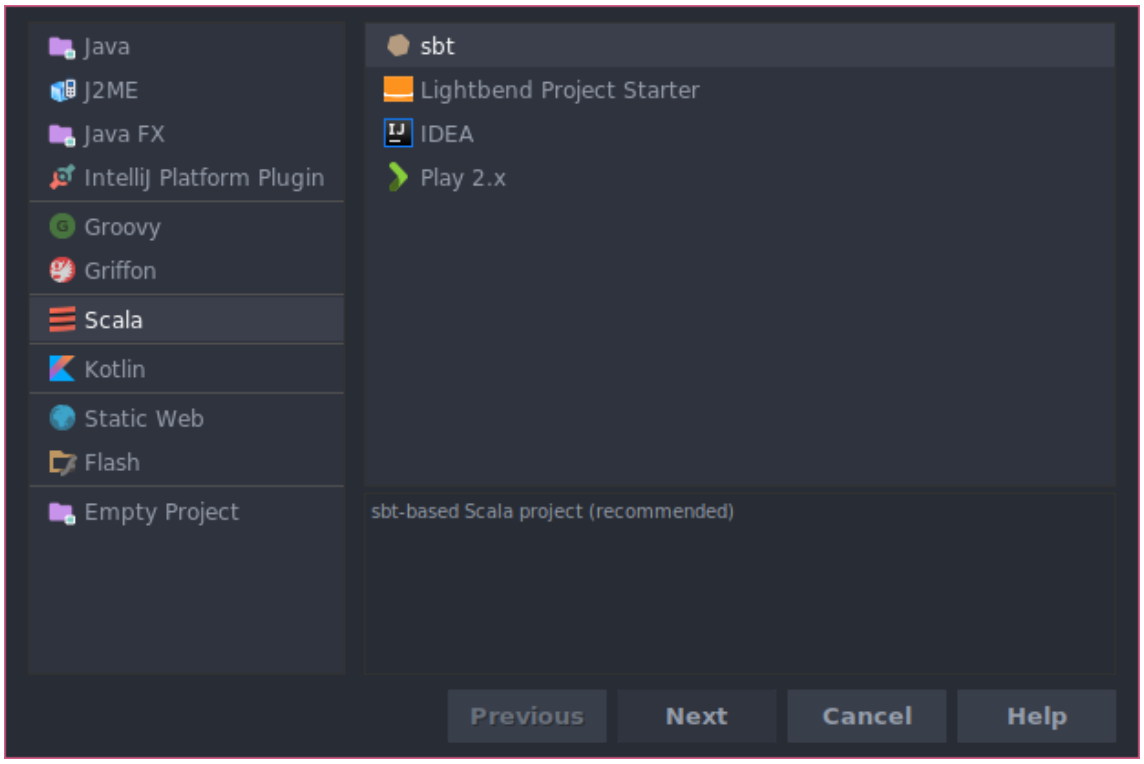
(b) *Type-inference* - tipovi se mogu dedukovati od strane prevodioca

(c) Funkcije se posmatraju kao objekti

(d) Osobine, engl. *Traits* - enkapsuliraju definicije polja i metoda

(e) Zatvorenja, engl. *Closures* - funkcije čije povratne vrednosti zavise od promenljivih deklarisanih van te funckije

(f) Ne postoje statičke klase, umesto njih se koriste objekti (singletoni)



Slika 6.1: *Početak pravljenja* `sbt` *projekta*

6.2 Instalacija potrebnih alata

Koristimo jezik `Scala` (verzija 2.12.8) http://www.scala-lang.org/.
Potrebno je imati instalirano:

(a) `IntelliJ Idea` jetbrains.com/idea/

(b) Scala plugin za `IntelliJ Idea`

6.2.1 Pravljenje projekta

Koristi se `sbt` (eng. Simple build tool) (koji će da kasnije da preuzima i konfiguriše dodatne biblioteke za nas). Na slici 6.1 je prikazano kako u okruženju odabrati `sbt` projekat.

Za pravljenje `sbt` projekta **neophodno** je imati aktivnu internet konekciju.

6.2.2 Inicijalizacija projekta

Okruženje se sada inicijalizuje, detektuju se Scala biblioteke i slično. Ovaj proces može da potraje od nekoliko sekundi do nekoliko minuta u zavisnosti od brzine mrežne konekcije i hardvera računara te treba biti strpljiv.

U donjem delu okruženja na sredini možete čitati poruke o tome šta se trenutno dešava, na primer:

```
sbt: dump project structure from sbt
```

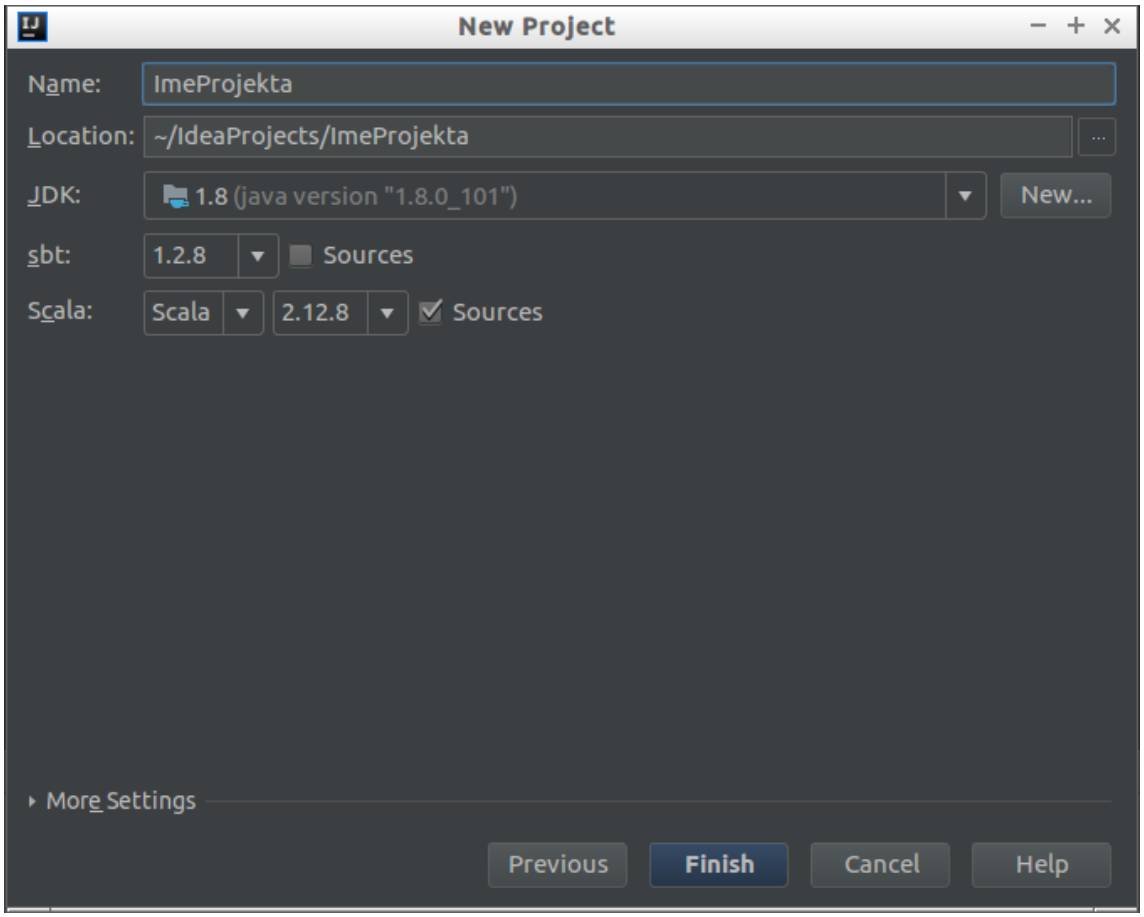
je jedan od koraka koje Intellij mora da izvrši kako bi pripremio sve za rad.

6.2.3 Konfiguracija projekta

Odaberite gde želite da se napravi projekat. Preporučeno je da imate direktorijum u kojem čuvate sve `Intellij Idea` projekte (uglavnom je to `home/korisnik/IdeaProjects`).

Za `sbt` i jezik `Scala` odaberite:

- `sbt: 1.2.max (1.2.8)`



Slika 6.2: *Kraj pravljenja* `sbt` *projekta*

- `Scala: 2.12.max (2.12.8)`

Na slici 6.2 prikazano je gde treba odabrati verzije za `sbt` i jezik `Scala`.
Kliknite `Finish`.

6.3 Uvod u jezik Scala

Zadatak 6.1 HelloWorld

```
/**
  * Viselinijske komentare u Scali pisemo ovako (isto kao i u C-u).
  *
  * */

// Jednolinijske komentare pisemo ovako

object HelloWorld {
  def main(args: Array[String]){
    //  Tacka zapeta na kraju svake naredbe je opciona
    println("Hello world! :)")
  }
}
```

Zadatak 6.2 Promenljive, niske, petlje, funkcije

```
object Uvod {
    def main(args: Array[String]) {

      println("------- Promenljive -------")
```

```
5
//    Promenljive mozemo deklarisati kao konstantne
7 //    ili promenljive cija se vrednost moze menjati.
  //    Kljucna rec val deklarise konstantu a var promenljivu.
9 //
  //    kljucnaRec imePromenljive : tipPromenljive = vrednost
11
      val c : Int = 42
13    var x : Int = 6

15    x += 6

17 //    Sledeca naredba prijavljuje gresku.
  //    `c += 4`
19
      println("Konstanta: " + c)
21    println("Promenljiva: " + x)

23    if(x > 6)
        println("Promenljiva x je veca od 6.")
25    else if(x == 6)
        println("Promenljiva x je jednaka 6.")
27    else
        println("Promenljiva x je manja od 6.")
29
      println("------- Niske -------")
31
      val crtani : String = "Maza i Lunja"
33    println("Duzina niske: '" + crtani + "' je " + crtani.length())

35 //    Viselinijske niske pisemo izmedju trostrukih navodnika:
      println(""" Likovi:
37                -Alisa
                  -Vendi
39                -Bambi
                  -Had
41            """)

43 //    Interpolacija niski:
  //    Scala nam dozvoljava da ugnjezdavamo vrednosti promenljivih u niske
45 //    dodavanjem karaktera s na pocetak niske (npr: s"Moja niska") omogucavamo
  //    ugnjezdavanje vrednosti promenljivih u nisku dodavajuci karakter $ pre imena
    promenljive
47 //    (npr s"Kolicina: $kol grama")

49    var trajanje : Int = 76
      println(s"Crtani film: '$crtani' - $trajanje minuta.")
51
      println("------- Petlje -------")
53
      var sat = 0
55    while(trajanje >= 60){
        sat += 1
57      trajanje -= 60
      }
59
      println(s"'$crtani' traju $sat sat i $trajanje minuta.")
61
  //    For petljom mozemo iterirati kroz kolekcije koristeci sintaksu
63 //
  //    for(element <- kolekcija)
65 //
  //    1 to 5 - pravi kolekciju brojeva [1,5]
67
      println("FOR - 1 to 5 ")
69    for(i <- 1 to 5)
        println(i)
71
  //    1 until 5 - pravi kolekciju brojeva [1,5)
73
      println("FOR - 1 until 5 ")
75    for(i <- 1 until 5)
        println(i)
```

```
77
//    Range(pocetak, kraj, korak) - pravi kolekciju [pocetak, kraj) sa zadatik
    korakom.
79  //    Korak moze biti i negativan npr. Range(10,0,-2)

81        println("FOR - Range(0,6,2) ")
          for(i <- Range(0,6,2))
83          println(i)


85
          println("------- Nizovi -------")
87  //    Pravimo ih na sledeci nacin:
//        var niz : Array[tip] = new Array[tip](brojElemenata)
89        var crtaci : Array[String] = new Array[String](5)
          crtaci(0) = "Petar Pan"
91        crtaci(1) = "Mulan"
          crtaci(2) = "Aladdin"
93        crtaci(3) = "Herkules"
          crtaci(4) = "Pocahontas"
95
          println("Crtaci: ")
97        for(crtac <- crtaci)
            println(crtac)
99
          println("------- Funkcije -------")
101       ispisiSortirano(crtaci)
    }
103
//    def imeFunkcije([listaArgumenata]) : povratnaVrednost = {
105 //          teloFunkcije
//    }
107
//  Povratna vrednost Unit je ekvivalentna void vrednosti
109     def ispisiSortirano(crtaci : Array[String]) : Unit = {
          println("Sortirani crtaci:")
111       for(crt <- crtaci.sortWith(poredi))
             println(crt)
113
//   Anonimne funkcije
115 //
//    ( listaArgumenata ) => { teloFunkcije }
117
          println("Sortirani crtaci koristeci anonimnu funkciju:")
119       for(crt <- crtaci.sortWith((c1 , c2) => { if(c1.compareTo(c2) < 0) false else
    true} ))
             println(crt)
121
      }
123
      def poredi(c1 : String, c2 : String) : Boolean = {
125     if(c1.compareTo(c2) < 0)
          return true
127     else
          return false
129   }
  }
```

Zadatak 6.3 Klase, objekti, nasledjivanje

```
1   //  Klase se u Scali konstruisu na sledeci nacin:
    //
3   //  class ImeKlase {
    //  teloKlase
5   //  }
    //    ili
7   //  class ImeKlase (argumentiKonstruktora) {
    //  teloKlase
9   //  }

11  class Film {
      var naslov : String = ""
13    var trajanje : Int = 0
      var godina : Int = 0
```

```
15
16    // Konstruktor
17    def this(nas : String, traj : Int, god : Int) = {
18      this()
19      this.naslov = nas
20      this.trajanje = traj
21      this.godina = god
22    }
23
24    // Metodi klase
25
26    def getNaslov() : String = {
27      // Poslednja naredba u metodu je njena povratna vrednost,
28      // tako da ne mora da se koristi `return` kljucna rec
29      return this.naslov
30      // Slicno, blok ne mora da postoji ako funkcija
31      // ima samo jednu naredbu
32    }
33
34    // Konvencija je da ukoliko je metod cist (nema sporednih efekata)
35    // i ako nema argumenata, da se onda ne pisu zagrade. IntelliJ daje
36    // upozorenje kada se ova konvencija ne prati
37    def getTrajanje() : Int = this.trajanje
38
39    def getGodina : Int = this.godina
40
41    override def toString : String = {
42      "Film " + this.naslov + ", traje " + this.trajanje + " minuta, napravljen je " +
43      this.godina + " godine"
44    }
45  }
46
47  // Nasledjivanje
48  class CrtaniFilm extends Film {
49    var animator : String = ""
50
51    def this(nas : String, traj : Int, god : Int, anim : String) = {
52      this()
53      this.naslov = nas
54      this.trajanje = traj
55      this.godina = god
56      this.animator = anim
57    }
58
59    def getAnimator : String = this.animator
60
61    override def toString : String =  "Crtani " + super.toString() + ", animator je " +
62       this.animator
63  }
64
65  // U Scali mozemo definisati takozvane 'singleton' objekte kljucnom reci object.
66  // Garantuje se da ce postojati tacno jedan 'singleton' objekat na nivou naseg
67     programa
68  // i on se najcesce koristi za implementaciju main metoda
69  object Program {
70    def main(args: Array[String]) {
71
72      val assassinsCreed = new Film("Assassin's Creed", 115, 2016)
73      val tarzan = new CrtaniFilm("Tarzan", 88 , 1999, "Walt Disney")
74
75      println(assassinsCreed)
76      println(tarzan)
77
78      //  Scala nudi brojne korisne kontejnerske i nizovske klase
79      //  kao sto su Array, ArrayBuffer, Map, HashMap, Queue, Tuple1, Tuple2, i druge
80      //  sa kojima cemo se susretati u narednim primerima
81      //  gde ce biti vise objasnjene
82    }
83  }
```

6.4 Zadaci

Zadatak 6.4 **Restoran** Pet konobara radi u restoranu, i nakon što dođu na posao, poslovođa im dodeli broj stolova koje moraju da usluže. Napisati program koji učitava sa standardnog ulaza broj neusluženih stolova u restoranu i raspoređuje ih konobarima. Nakon što usluže jedan sto, konobari šalju poruku tako da poslovođa u svakom trenutku ima uvid u brojčano stanje. Konobare implementirati kao posebne niti koje uslužuju sto (spavaju nasumično izabran broj sekundi), ispisuju redni broj stola koji su uslužili i po završetku ispisuju poruku da su završili.

[Rešenje 6.4]

Zadatak 6.5 **Množenje matrica** Napisati program koji konkurentno množi dve matrice čiji se elementi nalaze u datotekama `matrica1.txt` i `matrica2.txt` i rezultat upisuje u datoteku `matrica3.txt`. Svaka nit treba da računa vrednosti za jednu vrstu rezultujuće matrice. Format podataka u datotekama je sledeći:

```
n m
a11 a12 a13 ... a1m
a21 a22 a21 ... a2m
...
an1 an2 an3 ... anm
```

[Rešenje 6.5]

Zadatak 6.6 **Broj karaktera DNK** Napisati program koji konkurentno prebrojava koliko puta se koja baza (A, C, G, T) pojavljuje u DNK sekvenci koja se nalazi u višelinijskoj datoteci `bio_podaci.txt`. Sa standardnog ulaza učitati broj niti. Svakoj niti dodeliti određen broj linija koji ce da obrađuje a rezultate čuvati u deljenoj mapi (pogodno je koristiti klasu ConcurrentHashMap).

[Rešenje 6.6]

Zadatak 6.7 **Krediti** Bogoljub otvara banku i poseduje određeni kapital. Kao svaki dobar ekonomista on odlučuje da zaradi na davanju kredita. Napisati program koji demonstrira rad službenica i davanje kredita u Bogoljubovoj banci. Sa standardnog ulaza učitati njegov početni kapital, kamatnu stopu i broj zaposlenih službenica. U datoteci red_klijenata.txt se nalazi spisak klijenata koji čekaju na red za razgovor sa službenicom u sledećem formatu:

```
imeKlijenta potrebnaPozajmica
```

Posao svake službenice se izvršava u posebnoj niti. Službenica poziva sledećeg klijenta iz reda na razgovor. Nakon što ustanovi koliko je novca klijentu potrebno, službenica proverava da li banka trenutno poseduje tu količinu novca i daje klijentu kredit tako što umanjuje kapital banke i klijentu računa vrednost duga u skladu sa kamatom. Ukoliko banka nije u mogućnosti da izda kredit, ispisati odgovarajuću poruku. Nakon što službenice završe sa svim klijentima iz reda ispisati poruku o ukupnoj zaradi banke.

[Rešenje 6.7]

Zadatak 6.8 **Kladionica** Napisati program koji simulira proces klađenja. Kladioničari se klade na ishod pet fudbalskih utakmica uplaćujući određenu količinu novca na tiket. U datoteci `kladionicari.txt` se nalazi spisak kladioničara i njihovih tiketa u sledećem formatu:

```
imeKladionicara svotaNovca
utakmica1
rezultat
 ...
utakmica5
rezultat
...
```

Ishod može biti 1 - prva ekipa je pobedila, x - nerešeno, 2 - druga ekipa je pobedila. Svaki ishod nosi određenu kvotu kojom se množi novac koji je kladioničar uplatio. U datoteci `utakmice.txt` se nalazi spisak utakmica sa njihovim kvotama u formatu:

```
imeUtakmice
kvota1 kvotaX kvota2
...
```

Učitati podatke o utakmicama i kladioničarima. Svaki kladioničar čeka na ishod utakmica u posebnoj niti. Kladionica nakon što dobije ishod utakmica (implementirati nasumično biranje ishoda) obaveštava kladioničare da su utakmice završene. Kladioničari nakon toga računaju sopstvenu zaradu kao zbir kvota pogođenih utakmica pomnoženih sa odgovarajućim delom novca i ispisuju poruku o pogodjenom rezultatu. Na kraju ispisati ukupnu količinu novca koju kladionica treba da isplati kladioničarima.

[Rešenje 6.8]

6.5 Zadaci za vežbu sa rešenjima

Zadatak 6.9 **Zbir vektora** Napisati program koji konkurentno sabira dva vektora. Sa standardnog ulaza se učitava dimenzija vektora, elementi oba vektora i broj niti. Svakoj niti dodeliti indeks početka i kraja vektora nad kojim računa zbir a rezultat smeštati u prvi vektor. Indekse računati na osnovu dimenzije vektora i broja niti. Rezultujući vektor ispisati na standardni izlaz.

[Rešenje 6.9]

Zadatak 6.10 **Broj petocifrenih brojeva** Napisati program koji konkurentno računa broj pojavljivanja petocifrenih brojeva koji se nalaze u datotekama čija imena se učitavaju sa standardnog ulaza. Svakoj niti dodeliti po jednu datoteku nad kojom će računati. Rezultate brojanja smeštati u lokalne promenljive unutar niti i po završetku računanja za svaku datoteku ispisati broj pojavljivanja petocifrenih brojeva.

[Rešenje 6.10]

Zadatak 6.11 **Berba** Milovan ima voćnjak u kome gaji trešnje, kajsije, kruške i šljive. Došlo je vreme berbe i mora uposliti mlade studente da oberu voćnjak. U datoteci `drvoredi.txt` se nalaze podaci o drvoredima voćnjaka u sledećem formatu:

```
vrstaVoca brojStabala
```

Sa standardnog ulaza učitati broj zaposlenih studenata a iz datoteke učitati podatke o drvoredima. Svaka nit predstavlja jednog studenta. Student odlazi do jednog drvoreda skidajući ga iz reda drvoreda koje je potrebno obrati i počinje branje. Ako pretpostavimo da jedno stablo voća može roditi od 30-50 kilograma voća, student za svako stablo iz drvoreda nasumično računa broj kilograma voća koji je obran i dodaje ih u skladište. Ukoliko su svi drvoredi obrani studenti prestaju sa radom i na standardni izlaz se ispisuje ukupna količina obranog voća svake vrste.

[Rešenje 6.11]

Zadatak 6.12 **Turistička agencija** Turistička agencija FlyProgrammer organizuje nagradnu igru i daje pet vaučera od 20% popusta na cenu kupljenje karte svojim klijentima. U datoteci `ucesnici.txt` se nalaze podaci o klijentima i cenama u sledećem formatu:

```
ime prezime
cena
```

Napisati program koji simulira nagradnu igru. Svaka nit čeka na rezultate nagradne igre za jednog klijenta. Turistička agencija izvlači dobitnike slučajnom selekcijom i nakon završenog izvlačenja obaveštava niti. Niti proveravaju da li je izvučen odgovarajući klijent i ispisuje poruku o ishodu.

[Rešenje 6.12]

6.5.1 Zadaci za vežbu

Zadatak 6.13 **Množenje vektora skalarom** Napisati program koji konkurentno množi vektor skalarom. Sa standardnog ulaza učitati dimenziju i elemente vektora, skalar i broj niti. Svakoj niti dodeliti deo vektora (indekse početnog i krajnjeg elementa). Nit računa proizvod vektora skalarom za elemente u zadatom opsegu i rezultat smešta u isti vektor. Na kraju ispisati rezultat na standardni izlaz.

Zadatak 6.14 **Množenje matrice vektorom** Napisati program koji konkurentno množi matricu vektorom. Sa standardnog ulaza učitati dimenziju i elemente vektora, dimenzije i elemente matrice i broj niti. Svakoj niti dodeliti deo matrice (indekse početne i krajnje vrste). Nit računa proizvod matrice vektorom za vrste u zadatom opsegu i rezultat smešta u nov vektor. Na kraju ispisati rezultat na standardni izlaz.

Zadatak 6.15 **Transponovanje matrice** Napisati program koji konkurentno transponuje matricu. U datoteci `matrica.txt` se nalaze dimenzije matrice nakon čega slede elementi matrice. Svaka nit transponuje po jednu vrstu matrice i smešta rezultat u rezultujuću matricu. Na kraju izračunavanja rezultujuću matricu upisati u datoteku `transponovana_matrica.txt`.

Zadatak 6.16 **Matrice (dodatak)** Implementirati zadatke 6.5 i 6.9 tako da se broj niti učitava sa standardnog ulaza, i svakoj niti dodeliti broj vrsta koje će obrađivati.

Zadatak 6.17 **Funkcije nad vektorima** Napisati program koji konkurentno računa proizvod, sumu, prosečnu vrednost, broj negativnih i broj pozitivnih elemenata vektora. Sa standardnog ulaza učitavati imena datoteka u kojima se nalaze vektori. Implementirati konkurentno računanje na dva načina, koristeći paralelizaciju zadataka i paralelizaciju podataka i uporediti vremena izvršavanja (obratiti pažnju na to da je korišćenjem paralelizacije zadataka najveće moguće ubrzanje jednako najsporijem zadatku dok kod paralelizacije podataka ubrzanje zavisi od broja procesora, broja niti,...). Na kraju ispisati rezultate svih izračunavanja na standardni izlaz.

Zadatak 6.18 **Broj petocifrenih brojeva (dodatak)** Implementirati zadatak 6.10 uz sledeće izmene. Brojeve učitati iz jedne datoteke `brojevi.txt` u niz a broj niti učitati sa standardnog ulaza. Svakoj niti dodeliti deo niza brojeva koji će obrađivati a rezultat čuvati u deljenoj promenljivoj tipa `AtomicLong`.

Zadatak 6.19 **Broj karaktera DNK (dodatak)** Implementirati zadatak 6.6 tako da se rezultati brojanja baza čuvaju u deljenim promenljivama tipa `AtomicLong` i na kraju ispisati procenat pojavljivanja svake baze u lancu.

Zadatak 6.20 **Hotel** Hotel Nightlife brine o svojim klijentima i trudi se da im smanji vreme čekanja dok se sređuje soba koju su rezervisali. U datoteci `sobe.txt` nalaze se brojevi soba koje je potrebno srediti. Sa standardnog ulaza učitati broj trenutno dostupnih čistačica. Svaka nit predstavlja jednu čistačicu. Čistačica skida sobe sa reda soba koje je potrebno srediti i odlazi do nje da je sredi (ispisati redni broj sobe u koju je ušla čistačica a nit uspavati na slucajan broj sekundi). Nakon što je sredila sobu, čistačica pronalazi ostavljen bakšiš (slučajan broj od 0-500 dinara) i po dogovoru bakšiš ubacuje u zajedničku kutiju. Kada završi sa jednom sobom, čistačica odlazi do sledeće sobe (skida je sa reda) i nastavlja sve dok red ne postane prazan. Nakon što završe sa čišćenjem, čistačice otvaraju kutiju sa bakšišem i dele novac na ravne časti tj. ispisuju na standardan izlaz dobijeni bakšiš.

Zadatak 6.21 **Loto** Napisati program koji simulira izvlačenje na Loto-u. Izvlače se tri broja iz opsege [1,37] a ukupna vrednost nagradnog fonda se unosi sa standardnog ulaza. Učesnici uplaćuju srećke i pokušavaju da pogode tri broja. U datoteci `ucesnici.txt` se nalaze informacije o uplaćenim srećkama u sledećem formatu:

```
1  ime
2  broj1 broj2 broj3
3  ...
```

Učitati podatke o učesnicima. Svaki učesnik čeka na kraj izvlačenja u posebnoj niti. Izvlačenje se odvija u glavnoj niti tako što se nasumično biraju tri broja iz opsega [1,37] i po završetku se

obaveštavaju niti učesnika. Niti učesnika nakon toga porede izvučene brojeve i ukoliko se sva tri poklapaju ispisuju poruku i u ukupnu sumu novca koji je potrebno isplatiti dodaju vrednost nagradnog fonda. Ukoliko se dva od tri broja poklapaju u ukupnu sumu se dodaje 40% nagradnog fonda. Ukoliko se samo jedan broj poklapa u ukupnu sumu se dodaje 10% nagradnog fonda. Na kraju ispisati količinu novca koju je potrebno isplatiti i količinu novca koju je lutrija zaradila (ukoliko zarada postoji).

6.6 Rešenja

Rešenje 6.4 Restoran

```
import java.util.concurrent._
import java.util.Scanner

// Pravljenje niti
//
// Da bismo napravili nit potrebno je da definisemo klasu koja nasledjuje klasu Thread
// i implementiramo metod run cije izvrsavanje pocinje kada nad instancom nase klase
// pozovemo metod start.
//
// Drugi nacin je da nasa klasa implementira interfejs Runnable i implementira metod run.
// Medjutim, da bismo naglasili da zelimo da se metod run nase instance izvrsava kao posebna nit
// potrebno je da napravimo instancu tipa Thread kojoj u konstruktoru treba da prosledimo
// instancu nase klase (koja implementira metod run.

class Konobar(ime : String, brStolova : Int) extends Thread {
  override def run(){
    for(i <- 0 until brStolova){
      //  Klasa ThreadLocalRandom predstavlja generator slucajnih brojeva
      //  jedinstven na nivou jedne niti.
      //  Metod current() vraca objekat ove klase za trenutnu nit.
      Thread.sleep(ThreadLocalRandom.current().nextInt(1,10)*1000)
      println("Konobar " + ime + " je usluzio " + i + ". sto.")
    }
    println("Konobar " + ime + " je zavrsio sa posluzivanjem.")
  }
}

object Restoran {
  def main(args : Array[String]) {
    val sc : Scanner = new Scanner(System.in)

    println("Unesite broj neusluzenih stolova u restoranu: ")
    val brojStolova = sc.nextInt()

    val korak = Math.ceil(brojStolova/5.0).toInt

    //   Pravimo instance niti
    val stefan = new Konobar("Stefan", korak)
    val nikola = new Konobar("Nikola", korak)
    val filip = new Konobar("Filip", korak)
    val nebojsa = new Konobar("Nebojsa", korak)
    val djordje = new Konobar("Djordje", brojStolova - 4*korak)

    //   Ukoliko nasa klasa implementira interfejs Runnable
    //   val stefan = new Thread(new Konobar("Stefan", 10))

    //   Metod start zapocinje izvrsavanje metoda run
    stefan.start()
    nikola.start()
    filip.start()
    nebojsa.start()
    djordje.start()
    // Nevalidno, tj ne radi kako ocekujemo: stefan.run()
  }
```

```
}
```

Rešenje 6.5 Množenje matrica

```
import java.util.Scanner
import java.io.PrintWriter
import java.io.File
import scala.Array._

class Mnozilac(vrsta1 : Array[Int],
               matrica2 : Array[Array[Int]],
               rezultat : Array[Int])
  extends Thread {

  var k : Int = matrica2.length*matrica2(1).length / vrsta1.length
  var m : Int = vrsta1.length

  override def run() {
    for(i <- 0 until k)
      rezultat(i) = skProizvod(i)
  }

  def skProizvod(j : Int) : Int = {
    var res = 0
    for(i <- 0 until m)
      res += vrsta1(i)*matrica2(i)(j)
    res
  }
}

object MnozenjeMatrica {
  def main(args : Array[String]) {

    val sc1 : Scanner = new Scanner(new File("matrica1.txt"))
    val sc2 : Scanner = new Scanner(new File("matrica2.txt"))
    val pw : PrintWriter = new PrintWriter(new File("matrica3.txt"))

    // Ucitavamo dimenzije matrica
    val n = sc1.nextInt()
    val m1 = sc1.nextInt()
    val m2 = sc2.nextInt()
    val k = sc2.nextInt()

    if(m1 != m2){
      println("Greska! Dimenzije matrica se moraju poklapati!")
      return
    }

    // Funkcija ofDim[Tip](n,m) pravi visedimenzioni niz dimenzija mxn
    val matrica1 = ofDim[Int](n,m1)
    val matrica2 = ofDim[Int](m2,k)
    val rezultat = ofDim[Int](n,k)

    // Ucitavamo elemente prve matrice
    for(i <- 0 until n)
      for(j <- 0 until m1)
        matrica1(i)(j) = sc1.nextInt()

    // Ucitavamo elemente druge matrice
    for(i <- 0 until m2)
      for(j <- 0 until k)
        matrica2(i)(j) = sc2.nextInt()

    val mnozioci = new Array[Mnozilac](n)

    // Pravimo niz niti koje ce da racunaju i-tu vrstu rezultata mnozenja matrica
    for(i <- 0 until n)
      mnozioci(i) = new Mnozilac(matrica1(i), matrica2, rezultat(i))

    // Zapocinjemo izvrsavanje niti
    for(i <- 0 until n)
      mnozioci(i).start()
```

```
69
    // Cekamo da niti zavrse sa izracunavanjem
71  for(i <- 0 until n)
      mnozioci(i).join()
73
    // Upisujemo rezultujucu matricu u datoteku
75  pw.append(s"$n $k \n")
    for(i <- 0 until n){
77    for(j <- 0 until k)
        pw.append(s"${rezultat(i)(j)} ")
79    pw.append("\n")
    }
81
    pw.close()
83  }
}
```

Rešenje 6.6 Broj karaktera DNK

```
import java.util.concurrent._
2 import java.util.Scanner
import java.io.File
4 import scala.collection.mutable.ArrayBuffer

6 class Brojac(poc : Int, kraj : Int,
            linije : ArrayBuffer[String],
8           mapaKaraktera : ConcurrentHashMap[Char, Int])
  extends Thread {
10
  override def run() {
12  for(i <- poc until kraj){
      //  Racunamo broj svakog karaktera u liniji
14    val a = linije(i).count(_=='a')
      val c = linije(i).count(_=='c')
16    val g = linije(i).count(_=='g')
      val t = linije(i).count(_=='t')
18
      //  Synchronized kljucna rec obelezava kriticnu sekciju
20    //  i garantuje se da u svakom trenutku tacno jedna nit moze izvsavati naredbe
    iz bloka.
      //  Synchronized se moze koristiti na vise nacina:
22    //
      //  Metodi klase
24    //
      //  def f() = synchronized { teloFunkcije }
26    //
      //  Na ovaj nacin smo naglasili da je metod f jedne instance nase klase
    kriticna sekcija
28    //  i u svakom trenutku tacno jedna nit moze izvrsavati ovaj metod te instance.
      //
30    //  Blok instance
      //
32    //  instanca.synchronized { blok }
      //
34    //  Na ovaj nacin smo naglasili da za datu instancu u svakom trenutku
      //  tacno jedna nit moze izvrsavati naredbe bloka
36    //  Ovakvo ponasanje mozemo posmatrati i iz drugacijeg ugla.
      //  Instanca predstavlja monitor objekat.
38    //  U trenutku kada nit pozeli da izvrsava blok kriticne sekcije,
      //  ona zakljuca instancu, izvrsi kriticnu sekciju i otkljuca instancu.
40    //
      //  Treba teziti ka tome da kriticna sekcija bude sto manja
42    //  kako bismo sto vise iskoristili prednosti konkurentnog izvrsavanja
      //
44    //  TODO: Zakomentarisati synchronized blok i videti sta se desava prilikom
    pokretanja programa
      //  sa razlicitim brojem niti.
46    //  Moguci scenario je da jedna nit procita vrednost iz mape, druga nit nakon
    toga azurira mapu,
      //  a prva nit i dalje drzi vrednost koju je procitala pre azuriranja tako da
    kada ona azurira mapu
48    //  rezultat azuriranja nece biti ispravan.
```

```
      //
      mapaKaraktera.synchronized {
        mapaKaraktera.replace('a', mapaKaraktera.get('a')+a)
        mapaKaraktera.replace('c', mapaKaraktera.get('c')+c)
        mapaKaraktera.replace('g', mapaKaraktera.get('g')+g)
        mapaKaraktera.replace('t', mapaKaraktera.get('t')+t)
      }
    }
  }
}

object BrojKarakteraDNK {
  def main(args : Array[String]) {
    val sc1 : Scanner = new Scanner(new File("bio_podaci.txt"))
    val sc2 : Scanner = new Scanner(System.in)

    println("Unesite broj niti: ")
    println("Broj procesora na racunaru koji su na raspolaganju je : " + Runtime.
    getRuntime.availableProcessors())

    val brojNiti = sc2.nextInt()

    val brojaci = new Array[Brojac](brojNiti)
    //  Klasa ArrayBuffer predstavlja niz promenljive duzine
    //  Neki od korisnih metoda su:
    //  - append(e) - dodaje element na kraj niza
    //  - isEmpty() - vraca true ukolik je niz prazan, false inace
    //  - insert(i, e) - dodaje element na datu poziciju
    //  ...
    val linije = new ArrayBuffer[String]()

    while(sc1.hasNextLine)
      linije.append(sc1.nextLine())

    val brojLinija = linije.length
    println(brojLinija)

    //  Klasa ConcurrentHashMap predstavlja implementaciju mape cije su operacije
    bezbedne u kontekstu konkurentnog izvrsavanja.
    //  To znaci da u svakom trenutku tacno jedna nit moze izvrsavati operacije nad
    mapom.
    //  Medjutim, operacije citanja (get) su neblokirajuce, tako da se mogu
    preklopiti sa drugim operacijama (npr. azuriranja)
    //  i u takvim slucajevima se ne garantuje azurnost rezultata.
    //
    //  Konstruktor prima tri argumenta:
    //   - inicijalni kapacitet mape
    //   - faktor povecavanja mape
    //   - broj niti koji se pretpostavlja da ce konkurentno pristupati objektu mape
    //
    //  Neki od korisnih metoda klase ConcurrentHashMap su:
    //   - get(kljuc) - vraca element sa zadatim kljucem, odnosno null ukoliko takav
    ne postoji
    //   - put(kljuc, vrednost) - dodaje element sa zadatim parametrima
    //   - remove(kljuc) - uklanja element sa zadatim kljucem
    //   - replace(kljuc, vrednost) - postavlja vrednost elementu sa zadatim kljucem
    //   - size() - vraca velicinu mape
    //   - isEmpty() - vraca true ukoliko je mapa prazna, false inace
    //   ...
    //
    val mapaKaraktera = new ConcurrentHashMap[Char, Int](4,4,brojNiti)
    mapaKaraktera.put('a', 0)
    mapaKaraktera.put('c', 0)
    mapaKaraktera.put('g', 0)
    mapaKaraktera.put('t', 0)

    val korak = Math.ceil(brojLinija.toDouble/brojNiti.toDouble).toInt

    for(i <- 0 until brojNiti)
      brojaci(i) = new Brojac(i*korak, Math.min((i+1)*korak, brojLinija), linije,
    mapaKaraktera)

    for(b <- brojaci)
```

```
116       b.start()
118     for(b <- brojaci)
          b.join()
120
        println("Rezultati konkurentnog izvrsavanja")
122     println("A: " + mapaKaraktera.get('a'))
        println("C: " + mapaKaraktera.get('c'))
124     println("G: " + mapaKaraktera.get('g'))
        println("T: " + mapaKaraktera.get('t'))
126
        println("Pravi rezultati \nA: 1761 \nC: 1577 \nG: 1589 \nT: 1913")
128   }
    }
```

Rešenje 6.7 Krediti

```
1   import java.util.concurrent.atomic._
    import java.util.concurrent._
3   import java.util.Scanner
    import java.io.File
5
    class Sluzbenica(kamata : Int,
7                    kapital : AtomicLong,
                     redKlijenata : ConcurrentLinkedQueue[Klijent],
9                    zaduzeniKlijenti : ConcurrentLinkedQueue[Klijent])
      extends Thread {
11
      override def run() {
13      while(true){
          //    Dohvatamo sledeceg klijenta iz reda
15        var k : Klijent = redKlijenata.poll()
          //    Ukoliko takav ne postoji zavrsavamo
17        if(k == null)
            return
19
          println("Klijent " + k.getIme() + " razgovara sa sluzbenicom.")
21        Thread.sleep(ThreadLocalRandom.current().nextInt(1,10)*1000)
          //    Iako je kapital objekat AtomicLong i garantuje se atomicnost operacija
        azuriranja
23        //    mogu nastati problemi prilikom konkurentnog pristupanja i azuriranja,
          //    i zbog toga je potrebno operacije sa ovim objektom obmotati synchronized
        blokom.
25        //    Problem moze nastati u delu koda (******).
          //    Pretpostavimo da dve niti izvrsavaju ovaj deo koda konkurentno.
27        //    Prva nit procita vrednost kapitala, nakon toga druga nit procita vrednost
         kapitala
          //    pre nego sto je prva nit izmenila vrednot, odmah posle prva nit promeni
        vrednost kapitala
29        //    i postavi novu vrednot (staraVrednost - prvaPozajmica)/
          //    U ovom trenutku druga nit ima vrednost kapitala koja nije ispravna sa
        kojom dalje operise.
31        //    Druga nit promeni vrednost kapitala i postavi novu vrednost (
        staraVrednost - drugaPozajmica)
          //    sto nije realna vrednost (staraVrednost - prvaPozajmica - drugaPozajmica)
33        kapital.synchronized {
            if(k.getPozajmica > kapital.get())
35            println("Klijent " + k.getIme() + " ne moze dobiti kredit.")
            else{
37            k.setDug(k.getPozajmica*((100+kamata.toFloat)/100))
              //      ******
39            val novKapital = kapital.get() - k.getPozajmica
              kapital.set(novKapital)
41            //      ******
              println("Klijent " + k.getIme + " je dobio kredit u iznosu od "
43              + k.getPozajmica + "e odnosno sa kamatom " + k.getDug + "e.")
              zaduzeniKlijenti.add(k)
45          }
          }
47      }
      }
49  }
```

```
51 class Klijent(ime : String, pozajmica : Int) {
     var dug : Float = 0
53
     def getIme : String = ime
55   def getPozajmica : Int = pozajmica
     def getDug : Float = dug
57
     def setDug(d : Float) : Unit = {
59     dug = d
     }
61 }

63 object Banka {
     def main(args : Array[String]) {
65     //  Klasa AtomicLong predstavlja enkapsulaciju long integer vrednosti
       //  nad kojom se operacije azuriranja izvrsavaju atomicno.
67     //
       //  Neki od korisnih metoda ove klase su:
69     //   - get() - vraca trenutnu vrednost
       //   - set(v) - postavlja vrednost v
71     //   - getAndAdd(v) - atomicki dodaje vrednost v i vraca prethodnu vrednost
       //   - addAndGet(v) - atomicki dodaje vrednost v i vraca novu vrednost
73     //   - getAndIncrement() - atomicki inkrementira vrednost i vraca prethodnu
       vrednost
       //   - incrementAndGet() - atomicki inkrementira i vraca novu vrednost
75     //   - getAndDecrement() - atomicki dekrementira vrednost i vraca prethodnu
       vrednost
       //   - decrementAndGet() - atomicki dekrementira i vraca novu vrednost
77     //   - compareAndSet(ocekivanaVrednost, novaVrednost) - postavlja novu vrednost
       //   ukoliko je stara jednaka ocekivanoj
79     //
       //   U paketu java.util.concurrent.atomic postoje i druge korisne klase kao sto
       su
81     //   AtomicBoolean, AtomicIntegerArray, AtomicInteger itd.
       val sc1 : Scanner = new Scanner(System.in)
83
       println("Unesite pocetni kapital banke: ")
85     val kapital = new AtomicLong(sc1.nextLong())
       val sacuvanKapital : Float = kapital.get()
87
       println("Unesite kamatnu stopu: ")
89     val kamata = sc1.nextInt()

91     println("Unesite broj sluzbenica u ekspozituri: ")
       val sluzbenice = new Array[Sluzbenica](sc1.nextInt())
93
       val sc2 : Scanner = new Scanner(new File("red_klijenata.txt"))
95
       //   Klasa ConcurrentLinkedQueue predstavlja implementaciju reda
97     //  cije su operacije bezbedne u kontekstu konkurentnog izvrsavanja.
       //
99     //   Neki od korisnih metoda su:
       //  - add(e) - dodaje element u red
101    //  - poll() - skida element sa pocetka reda i vraca ga kao rezultat
       //  - peek() - vraca elelent sa pocetka reda (ne skida ga)
103    //  - remove(e) - ukljanja element e iz reda
       //  - isEmpty() - vraca true ukoliko je red prazan
105    //  ...
       //
107    val redKlijenata = new ConcurrentLinkedQueue[Klijent]()
       val zaduzeniKlijenti  = new ConcurrentLinkedQueue[Klijent]()
109
       while(sc2.hasNextLine)
111      redKlijenata.add(new Klijent(sc2.next(), sc2.nextInt()))

113    // .indices() vraca Range svih indeksa niza
       for(i <- sluzbenice.indices)
115      sluzbenice(i) = new Sluzbenica(kamata, kapital, redKlijenata, zaduzeniKlijenti)

117    for(s <- sluzbenice)
         s.start()
119
```

```
    for(s <- sluzbenice)
      s.join()

    //  Iteriramo kroz red zaduzenih klijenata i racunamo ukupno zaduzenje
    var ukupnoZaduzenje : Float = 0
    val iterator = zaduzeniKlijenti.iterator()
    while(iterator.hasNext)
      ukupnoZaduzenje += iterator.next().getDug

    println(s"Banka je zaradila ${ukupnoZaduzenje-sacuvanKapital}e.")
  }
}
```

Rešenje 6.8 Kladionica

```
import java.util.concurrent._
import java.util.Scanner
import java.io.File
import scala.collection.mutable.HashMap
import scala.collection.mutable.ArrayBuffer

class Kladionicar(ime : String,
                  novac : Int,
                  tiket : HashMap[String, Char],
                  utakmice : HashMap[String, (Float, Float, Float, Char)])
  extends Thread {

  var zarada : Float = 0

  override def run(){
    //  Cekamo da se odigraju sve utakmice
    //  Funkcije wait(), notify() i notifyAll()
    //  moraju biti zakljucane unutar bloka synchronized
    utakmice.synchronized {
      utakmice.wait()
    }

    var pogodjeno = 0
    var ukupnaKvota : Float = 0
    //  Racunamo ukupnu zaradu
    for(t <- tiket)
      if(t._2 == utakmice(t._1)._4){
        println(ime + " je pogodio utakmicu " + t._1 + " - " + utakmice(t._1)._4)
        pogodjeno += 1
        if(utakmice(t._1)._4 == '1')
          ukupnaKvota += utakmice(t._1)._1
        else if(utakmice(t._1)._4 == 'x')
          ukupnaKvota += utakmice(t._1)._2
        else if(utakmice(t._1)._4 == '2')
          ukupnaKvota += utakmice(t._1)._3
      }
    if(pogodjeno != 0)
      zarada = ukupnaKvota * novac/pogodjeno
  }

  def getIme : String = ime
  def getZarada : Float = zarada
}

object Kladionica {
  def main(args : Array[String]) {
          val sc1 : Scanner = new Scanner(new File("utakmice.txt"))
          val sc2 : Scanner = new Scanner(new File("kladionicari.txt"))
    //  Klasa HashMap iz paketa scala.collection.mutable
    //  predstavlja implementaciju mape koja se moze azurirati (eng. mutable)
    //
    //  Neke od korisnih funkcija su:
    //  -put(k,v) - dodaje vrednost u mapu sa zadatim kljucem
    //  -size - vraca velicinu mape
    //  -contains(k) - vraca true ukoliko postoji element sa zadatim kljucem, false
    inace
    //  ...
```

```
    //
    //  Takodje mozemo iterirati kroz elemente mape for petljom
    //
    //  Klasa Tuple4 je jedna u nizu klasa koje implementiraju torke (Tuple1, Tuple2,
     Tuple3,...)
    //  koje se mogu azurirati.
    //  Elementima torke pristupamo na sledeci nacin - torka._1, torka._2, torka._3,
    torka._4

    val utakmice = new HashMap[String, (Float, Float, Float, Char)]()

    //  Rezultate utakmica postavljamo da budu karakter '-'
    //  kako bismo naglasili da se utakmice jos nisu odigrale
    while(sc1.hasNextLine){
      utakmice.put(sc1.nextLine(),(sc1.nextFloat(), sc1.nextFloat(), sc1.nextFloat(),
     '-'))
      sc1.nextLine()
    }
    val kladionicari = new ArrayBuffer[Kladionicar]()

    while(sc2.hasNextLine) {
      val ime = sc2.next()
      val novac = sc2.nextInt()
      val tiket = new HashMap[String, Char]()
      for(i <- 0 until 5){
        sc2.nextLine()
        tiket.put(sc2.nextLine(), sc2.next()(0))
      }
      kladionicari.append(new Kladionicar(ime, novac, tiket, utakmice))
    }

    for(k <- kladionicari)
      k.start()

    println("Cekamo da se utakmice odigraju.")
    Thread.sleep(5000)

    //  Racunamo rezultate utakmica
    val res = Array('1','x', '2')
    for(u <- utakmice)
      utakmice(u._1) = (u._2._1,
        u._2._2,
        u._2._3,
        res(ThreadLocalRandom.current().nextInt(0, 3))
      )

    //  Ulazimo u kriticnu sekciju
    //  i obavestavamo niti koje cekaju
    utakmice.synchronized {
      utakmice.notifyAll()
    }

    for(k <- kladionicari)
      k.join()

    var isplata : Float = 0
    for(k <- kladionicari){
      isplata += k.getZarada
      println(k.getIme + " ceka na isplatu " + k.getZarada + " dinara.")
    }
    println("Ukupno kladionica treba da isplati " + isplata + " dinara.")
  }
}
```

Rešenje 6.9 Zbir vektora

```
import java.util.Scanner

class Sabirac(poc : Int,
              kraj : Int ,
              vektor1 : Array[Float],
              vektor2 : Array[Float])
```

```
  extends Thread {
  // Svaka nit racuna zbir svog dela vektora [poc, kraj)
  // i rezultat smesta u prvi vektor.
  override def run() {
    for(i <- poc until kraj)
      vektor1(i) += vektor2(i)
  }
}

object ZbirVektora {
  def main(args : Array[String]){

    val sc = new Scanner(System.in)

    println("Unesite dimenziju vektora: ")
    val n = sc.nextInt()

    val vektor1 : Array[Float] = new Array(n)
    val vektor2 : Array[Float] = new Array(n)

    println("Unesite elemente prvog vektora: ")
    for(i <- 0 until n)
      vektor1(i) = sc.nextFloat()

    println("Unesite elemente drugog vektora: ")
    for(i <- 0 until n)
      vektor2(i) = sc.nextFloat()

    println("Unesite broj niti: ")
    val brojNiti = sc.nextInt()

    val niti = new Array[Sabirac](brojNiti)

    val korak = Math.ceil(n/brojNiti.toDouble).toInt

    //  Pravimo niti i zadajemo im indekse - granice
    for(i <- 0 until brojNiti)
      niti(i) = new Sabirac(i*korak, Math.min((i+1)*korak,n),vektor1,vektor2)

    //  Pokrecemo racunanje
    for(i <- 0 until brojNiti)
      niti(i).start()

    //  Cekamo da niti zavrse sa racunanjem
    for(i <- 0 until brojNiti)
      niti(i).join()

    //  Kada sve niti zavrse sa racunanjem ispisujemo rezultat
    print("Zbir vektora je: \n[")
    for(i <- 0 until n-1)
      print(vektor1(i) + ", ")
    println(vektor1(n-1) + "]")
  }
}
```

Rešenje 6.10 Broj petocifrenih brojeva

```
import java.io._
import java.util.Scanner
import scala.collection.mutable.ArrayBuffer

class BrojacPetocifrenih(dat : String) extends Thread {

  var rezultat : Int = 0
  // Citamo brojeve iz datoteke i povecavamo lokalni brojac
  // ukoliko je broj petocifren
  override def run() {
    val sc : Scanner = new Scanner(new File(dat))

    while(sc.hasNextInt()){
      val broj = sc.nextInt()
      if(broj >= 10000 && broj <= 99999)
```

```
16          this.rezultat+=1
17        }
18      }
19
20      def getRezultat: Int = rezultat
21      def getDatoteka: String = dat
22    }
23
24    object BrojPetocifrenih {
25      def main(args : Array[String]){
26
27        val sc = new Scanner(System.in)
28
29        val brojaci = ArrayBuffer[BrojacPetocifrenih]()
30        var kraj = false
31        var odg = ""
32        var dat = ""
33
34        while(!kraj) {
35          println("Da li zelite da zadate ime datoteke koja ce biti obradjena (y/n)?")
36          odg = sc.next()
37          if(odg.toLowerCase() == "n")
38            kraj = true
39          else{
40            println("Unesite ime datoteke: ")
41            dat = sc.next()
42            brojaci.append(new BrojacPetocifrenih(dat))
43          }
44        }
45
46        //   Zapocinjemo izvrsavanje
47        for(brojac <- brojaci)
48          brojac.start()
49
50        //   Pozivom metoda join cekamo sve brojace da zavrse sa izracunavanjem
51        for(brojac <- brojaci)
52          brojac.join()
53
54        //   Citamo rezultate brojanja svake niti
55        for(brojac <- brojaci)
56          println(s"Datoteka ${brojac.getDatoteka} sadrzi ${brojac.getRezultat}
57        petocifrenih brojeva.")
58      }
59    }
```

Rešenje 6.11 Berba

```
1   import java.util.concurrent.atomic._
2   import java.util.concurrent._
3   import java.util.Scanner
4   import java.io.File
5
6   class Berac(drvoredi : ConcurrentLinkedQueue[(String, Int)],
7               skladiste : AtomicIntegerArray) extends Thread {
8
9     override def run() {
10
11      while(true) {
12        //  Dohvatamo dvored za berbu iz reda
13        val drvored = drvoredi.poll()
14        //  Ukoliko nema vise drvoreda za berbu zavrsavamo
15        if(drvored == null)
16          return
17        println("Berac bere drvo " + drvored._1 )
18        Thread.sleep(ThreadLocalRandom.current().nextInt(1,10)*1000)
19        //  Dodajemo kilograme voca koje smo obrali u skladiste
20        for(_ <- 0 until drvored._2){
21          val obrano = ThreadLocalRandom.current().nextInt(30, 50)
22          if(drvored._1 == "tresnje")
23            skladiste.getAndAdd(0, obrano)
24          else if(drvored._1 == "kruske")
25            skladiste.getAndAdd(1, obrano)
```

```
        else if(drvored._1 == "kajsije")
          skladiste.getAndAdd(2, obrano)
        else if(drvored._1 == "sljive")
          skladiste.getAndAdd(3, obrano)
      }
    }
  }
}

object Berba {
  def main(args : Array[String]) {
    val sc1 : Scanner = new Scanner(new File("drvoredi.txt"))
    val sc2 : Scanner = new Scanner(System.in)

    //  Pravimo skladiste voca koje imamo u vocnjaku
    //  i postavljamo inicijalne kolicine voca.
    //  Klasa AtomicIntegerArray sadzi niz integer vrednosti
    //  nad kojima se operacije izvrsavaju atomicno.
    //  Slicno kao kod klasa AtomicInteger, AtomicLong i dr.
    val skladiste = new AtomicIntegerArray(4)
    skladiste.set(0,0)
    skladiste.set(1,0)
    skladiste.set(2,0)
    skladiste.set(3,0)
    //  Pravimo red drvoreda za berbu
    //  Svaki drvored je jedan par (voce, brojStabala).
    val drvoredi = new ConcurrentLinkedQueue[(String, Int)]()
    while(sc1.hasNextLine)
      drvoredi.add((sc1.next(), sc1.nextInt()))

    println("Unesite broj beraca: ")
    val brojBeraca = sc2.nextInt()

    val beraci = new Array[Berac](brojBeraca)
    for(i <- 0 until brojBeraca)
      beraci(i) = new Berac(drvoredi, skladiste)

    for(b <- beraci)
      b.start()

    for(b <- beraci)
      b.join()

    println("Tresanja je obrano: " + skladiste.get(0) + " kilograma.")
    println("Krusaka je obrano: " + skladiste.get(1) + " kilograma.")
    println("Kajsija je obrano: " + skladiste.get(2) + " kilograma.")
    println("Sljiva je obrano: " + skladiste.get(3) + " kilograma.")
  }
}
```

Rešenje 6.12 Turistička agencija

```
import java.util.concurrent._
import java.util.Scanner
import java.io.File

class Ucesnik(ime : String,
              cena : Int,
              dobitnici : Array[String])
  extends Thread {

  override def run() {
    //  Cekamo dok se ne zavrsi izvlacenje.
    //
    //  Metod wait() suspenduje nit, oslobadja kriticnu sekciju obmotanu synchronized
     blokom
    //  i dozvoljava drugim nitima koje su zakljucale isti objekat da udju u kriticnu
     sekciju
    //  sve dok neka druga nit ne pozove nad istim objektom metod notifyAll()
    //  cime se obavestavaju sve niti koje cekaju sa metodom wait()
    //  da mogu nastaviti sa radom
    //
```

```
    dobitnici.synchronized {
      dobitnici.wait()
    }
    for(d <- dobitnici)
      if(d == ime){
        println("Cestitamo " + ime +
          "!!! Osvojili ste popust od 20% na cenu karte. Vasa karta sada kosta " +
    cena*0.8 + "e.")
        return
      }
    println("Nazalost " + ime +
      " niste osvojili popust, vise srece drugi put. Vasa karta kosta " + cena + "e."
    )

  }

  def getIme: String = ime
}

object TuristickaAgencija {
  def main(args : Array[String]) {
    val sc : Scanner = new Scanner(new File("ucesnici.txt"))

    val dobitnici = new Array[String](5)
    val n = sc.nextInt()
    sc.nextLine()
    val ucesnici = new Array[Ucesnik](n)
    for(i <- 0 until n){
      ucesnici(i) = new Ucesnik(sc.nextLine(), sc.nextInt(), dobitnici)
      sc.nextLine()
    }

    for(u <- ucesnici)
      u.start()

    println("Izvlacenje je u toku.")
    Thread.sleep(5000)
    //   Ulazimo u kriticnu sekciju, i racunamo dobitnike nagradnih popusta
    dobitnici.synchronized {
      val izvuceniIndeksi = ThreadLocalRandom.current().ints(0, n).distinct().limit
    (5).toArray
      for(j <- 0 until izvuceniIndeksi.length)
        dobitnici(j) = ucesnici(izvuceniIndeksi(j)).getIme

      //   Kada su izracunati dobitnici, obavestavamo niti koje cekaju
      //   i izlazimo iz kriticne sekcije
      dobitnici.notifyAll()
    }

  }
}
```

7

Distribuirano programiranje

7.1 O Apache Spark-u

Apache Spark je radni okvir (eng. framework) za distribuirano programiranje. Pruža interfejse za programiranje (eng. API) u jezicima Java, Scala, Python i R.

Svaka Spark aplikacija se sastoji od glavnog (eng. *driver*) programa koji pokreće funkciju `main` i izvršava paralelne operacije na povezanom klaster računaru. Pristupanje klaster računaru se vrši pomoću objekta kontekst tipa `SparkContext`. Prilikom konstrukcije kontekst objekta, potrebno je definisati koji klaster računar će se koristiti. Spark aplikacija može koristiti lokalni računar kao simulaciju klaster računara (svaka procesorska jedinica će simulirati jedan čvor klaster računara) ili neki udaljeni klaster računar.

Spark koristi posebne kolekcije podataka koje se mogu obrađivati paralelno (eng. *RDD - Resilient Distributed Datasets*). Paralelne kolekcije se mogu napraviti od već postojećih kolekcija u programu ili se mogu učitati iz spoljašnjeg sveta. Nad paralelnim kolekcijama možemo izvršavati dva tipa operacija *transformacije* i *akcije* (slika 7.1). Transformacije transformišu kolekciju na klaster računaru i prave novu paralelnu kolekciju. Sve transformacije su lenje, što znači da rezultat izračunavaju u trenutku kada on postane potreban.

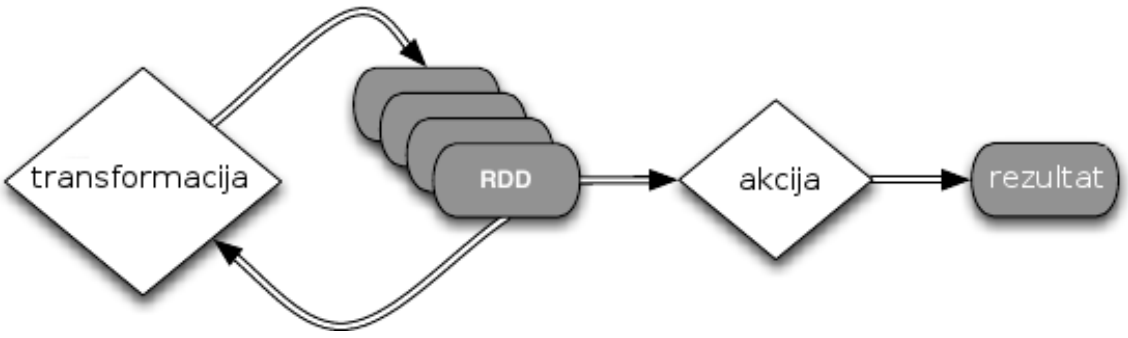
Akcije su operacije koje se izvršavaju na klaster računaru nad paralelnim kolekcijama i njihov rezultat (izračunata vrednost) se vraća na lokalni računar.

Spark može da sačuva paralelne kolekcije u memoriji čvorova klaster računara i na taj način ubrzati izvršavanje narednih operacija.

Spark pruža mogućnost korišćenja deljenih podataka u vidu emitovanih promenljivih (eng. *broadcast variables*) i akumulatora (eng. *accumulators*).

Neke od funkcija transformacija su:

- `map(f)` - vraća novu kolekciju koja se dobija tako što se primeni funkcija f nad svakim elementom postojeće kolekcije
- `filter(f)` - primenjuje funkciju f nad svim elementima kolekcije i vraća novu kolekciju koja sadrži one elemente za koje je funkcija f vratila true



Slika 7.1: *Operacije nad paralelnim podacima*

- `flatMap(f)` - slična je funkciji map, razlika je to što primena funkcije f nad nekim elementom kolekcije može da vrati 0 ili više novih elemenata koji se smeštaju u rezultujuću kolekciju
- `groupByKey()` - poziva se nad kolekcijom parova (kljuc, vrednost) i vraća kolekciju parova (kljuc, Iterable<vrednost>) tako što grupiše sve vrednosti sa istim ključem i smešta ih u drugi element rezultujućeg para
- `reduceByKey(f)` - poziva se nad kolekcijom parova (kljuc, vrednost) i vraća kolekciju parova (kljuc, nova_vrednost) , nova_vrednost se dobija agregiranjem svih vrednosti sa istim ključem koristeću zadatu funkciju agregacije f
- `aggregateByKey(pocetna_vrednost)(f1, f2)` - poziva se nad kolekcijom parova (kljuc, vrednost) i vraća kolekciju parova (kljuc, nova_vrednost) , nova_vrednost se dobija agregiranjem pocetne vrednosti i svih vrednosti sa istim ključem koristeću zadatu funkciju agregacije f1 u svakom čvoru klaster računara, a funkcija f2 agregira vrednosti izračunate u čvorovima klaster računara u jednu vrednost - nova_vrednost
- `sortByKey()` - poziva se nad kolekcijom parova (kljuc, vrednost) i vraća novu kolekciju sortiranu po ključu
- `cartesian(druga_kolekcija)` - spaja kolekciju sa drugom kolekcijom i vraća kolekciju svih parova (vrednost_iz_prve_kolekcije,vrednost_iz_druge_kolekcije)
- `zip(druga_kolekcija)` - spaja kolekciju sa drugom kolekcijom spajajući elemente na istim pozicijama i vraća kolekciju parova (vrednost_iz_prve_kolekcije, vrednost_iz_druge_kolekcije)

Neke od funkcija akcija su:

- `reduce(f)` - agregira elemente kolekcije koristeći funkciju f i vraća rezultat agregacije
- `collect()` - pretvara paralelnu kolekciju u niz (koji se nalazi na lokalnom računaru)
- `count()` - vraća broj elemenata kolekcije
- `countByKey()` - poziva se nad kolekcijom parova (kljuc, vrednost), za svaki ključ broji koliko ima elemenata sa tim ključem i vraća neparalelnu kolekciju (kljuc, broj_elemenata)
- `first()` - vraća prvi element kolekcije
- `take(n)` - vraća prvih n elemenata kolekcije
- `takeSample(sa_vracanjem, n, seed)` - vraća prvih n nasumično izabranih elemenata kolekcije (sa ili bez vraćanja), seed predstavlja početnu vrednost generatora slučajnih brojeva
- `takeOrdered(n[, poredak])` - vraća prvih n elemenata sortirane kolekcije (koristeći prirodan poredak kolekcije ili zadati poredak)
- `saveAsTextFile(ime_direktorijuma)` - upisuje kolekciju u datoteke koje se nalaze u zadatom direktorijumu
- `foreach(f)` - poziva funkciju f nad svim elementima kolekcije (uglavnom se koristi kada funkcija f ima neke sporedne efekte kao što je upisivanja podataka u datoteku ili slično)

Konfiguraciju Spark aplikacije možemo podešavati dinamički prilikom pokretanja aplikacije na klaster računaru. Potrebno je upakovati aplikaciju zajedno sa svim njenim bibliotekama u .jar datoteku koristeći neki od alata (Maven [1], SBT [2] i sl.) i instalirati Spark upravljač na klaster računaru (Standalone [3], Mesos [4], Yarn [5] i sl.). Aplikaciju možemo pokrenuti pomoci `spark-submit` skripta koji se nalazi u `bin` direktorijumu instaliranog Spark alata i konfigurisati dinamički. Na primer:

[1] http://maven.apache.org/
[2] http://www.scala-sbt.org/
[3] http://spark.apache.org/docs/latest/spark-standalone.html
[4] http://spark.apache.org/docs/latest/running-on-mesos.html
[5] http://spark.apache.org/docs/latest/running-on-yarn.html

```
./bin/spark-submit --class Main --master local --num-executors 20
   Aplikacija.jar
```

Parametri koji se najčešće koriste prilikom konfiguracije su:

- `--master url` - URL klaster racunara
- `--class ime_klase` - glavna klasa naše aplikacije
- `--num-executors n` - broj čvorova koji izvršavaju našu aplikaciju (eng. executors)
- `--executor-cores n` - broj zadataka koje jedan čvor može izvršavati istovremeno
- `--executor-memory n` - veličina hip memorije svakog čvora

7.2 Uputstvo za Apache Spark

Uputstvo prikazuje kako konfigurisati projekat u okruženju `Intellij Idea` da koristi biblioteku `Apache Spark`.

Literatura:

- spark.apache.org/docs/0.9.1/scala-programming-guide.html

7.2.1 Potreban softver i alati

Potrebno instalirati:

- Intellij Idea (jetbrains.com/idea/)
- Java JDK8 (oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)
- Plugin `Scala` za `Intellij Idea` (instalacija iz okruženja)

7.2.2 Pravljenje projekta

Potrebno je napraviti `sbt` projekat na isti način kao i u prethodnim odeljcima (videti uputstvo za Konkurentno programiranje u delu 6.2.1).

7.2.3 Dodavanje biblioteke u `build.sbt`

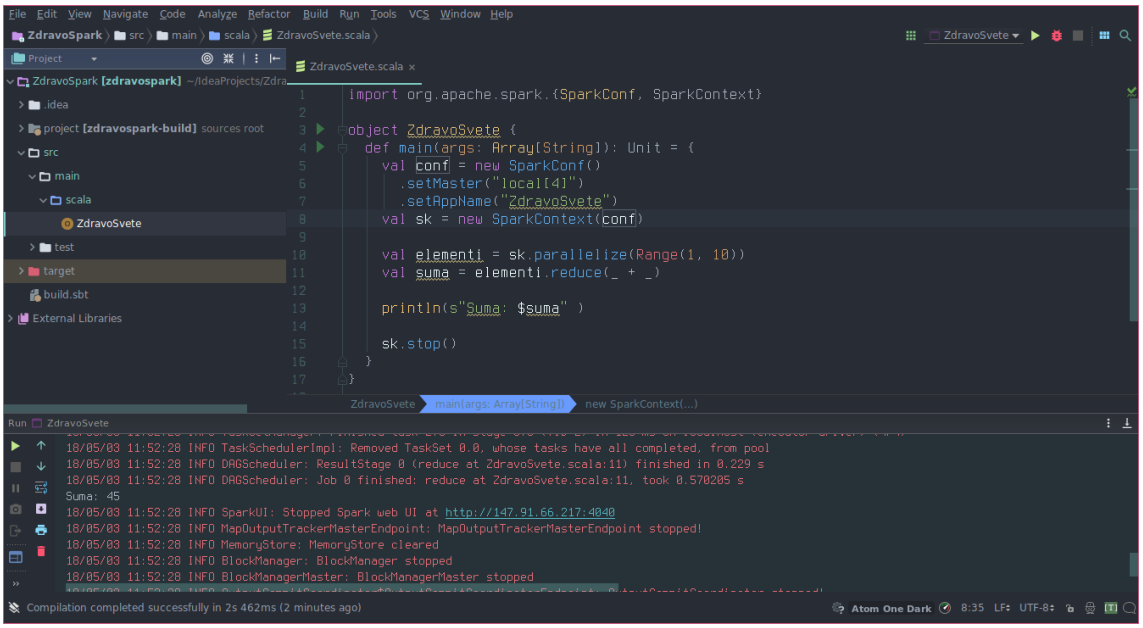
Potrebno je da sistemu `sbt` definišemo da naš projekat koristi spoljnu biblioteku. U datoteci `build.sbt` možemo definisati zavisnost (eng. *dependency*) od spoljne biblioteke.

Potrebno je dodati sledeći kod u datoteku `build.sbt`

```
libraryDependencies ++= {
  val sparkVer = "2.4.0"
  Seq(
    "org.apache.spark" %% "spark-core" % sparkVer
  )
}
```

Vaš `build.sbt` bi trebao imati sledeći oblik:

```
name := "ZdravoSpark"
version := "0.1"
scalaVersion := "2.12.8"
libraryDependencies ++= {
  val sparkVer = "2.4.0"
  Seq(
    "org.apache.spark" %% "spark-core" % sparkVer
  )
}
```



Slika 7.2: *Pokretanje Spark programa*

Pri čemu atribut `name` zavisi od imena projekta koje ste originalno odabrali.

Sačuvajte izmene. Kada Vas okruženje pita da ažurira projekat jer je izmenjena datoteka `build.sbt` prihvatite izmene. Alat `sbt` će u skladu sa Vašim izmenama preduzeti odgovarajuće akcije. U ovom slučaju to će biti preuzimanje biblioteke `spark-core` sa odgovarajućeg repozitorijuma i njeno uključivanje u Vaš projekat.

7.2.4 Pokretanje programa

Nakon što je sbt pripremio okruženje za rad, možemo da pristupimo Spark biblioteci. Na slici 7.2 je prikazano pokretanje Spark programa.

7.3 Zadaci sa rešenjima

Zadatak 7.1 **Parni kvadrati** Napisati program koji učitava ceo broj n veći od 2 i ispisuje sve kvadrate parnih brojeva počev od broja 2 do n.

[Rešenje 7.1]

Zadatak 7.2 **Broj petocifrenih** Napisati program koji ispisuje broj petocifrenih brojeva koji se nalaze u datoteci *brojevi.txt* (svaka linija sadrzi jedan broj).

[Rešenje 7.2]

Zadatak 7.3 **Skalarni proizvod** Napisati program koji racuna skalarni proizvod dva vektora (pretpostavimo da su vektori uvek zadati ispravno tj. iste su dužine) i ispisuje ih na izlaz. Vektori se nalaze u datotekama *vektor1.txt* i *vektor2.txt* u formatu:

```
1 a1, a2, a3, a4, ... an
```

[Rešenje 7.3]

Zadatak 7.4 **Broj pojavljivanja reči** Napisati program koji za svaku reč iz knjige (datoteka *knjiga.txt*) broji koliko se puta ona pojavljuje i rezultat upisuje u datoteku.

[Rešenje 7.4]

Zadatak 7.5 **Uređaji transakcije** U datoteci *uredjaji.txt* se nalaze podaci o kupljenim uređajima u formatu:

```
1 marka_uredjaja ostali_podaci
```

Napisati program koji izdvaja podatke o svim transakcijama jedne marke i upisuje ih u posebnu datoteku sa nazivom *ime_marke.txt*.

[Rešenje 7.5]

Zadatak 7.6 **Log poruke** Datoteka *log.txt* sadrži podatke koji su generisani pokretanjem Java programa. Napisati program koji izdvaja poruke koje se odnose na pakete jezika Java grupisane po tipu poruke i ispisuje ih na izlaz.
Poruke mogu biti informacione, upozorenja ili greške.
Format poruka je:

```
1 tip ostatak_poruke
```

Tip moze biti `[warn]`, `[info]` ili `[error]`.

[Rešenje 7.6]

Zadatak 7.7 **Uspešna preuzimanja** U datoteci *mavenLog.txt* se nalaze podaci o započetim/uspešnim preuzimanjima paketa prilikom pokretanja Maven alata (upravljač zavisnostima) u formatu:

```
1  Downloading: ostatak_poruke
```

ili

```
1  Downloaded: ostatak_poruke
```

Napisati program koji računa procenat uspešnih preuzimanja paketa u odnosu na započeta preuzimanja.

[Rešenje 7.7]

Zadatak 7.8 **Pokloni** Bliži se Božić i firma želi da pokloni svojim zaposlenim programerima tri paketića. Sin direktora firme je budući programer i želi da napravi program koji će simulirati izvlačenje troje dobitnika paketića. Kako još uvek nije dobro savladao programiranje, pomozimo mu tako što ćemo napisati program koji nasumicno bira tri programera i ispisuje njihova imena, prezimena i email.
Podaci o zaposlenima se nalaze u datoteci *zaposleni.txt* u formatu:

```
1 ime prezime pol identifikator IP_adresa_racunara datum_zaposlenja
    sifra_pozicije plata
```

Šifra pozicije programera je `IT_PROG`.

[Rešenje 7.8]

Zadatak 7.9 **Prosečna temperatura** U datoteci *temperaturaBoston.txt* se nalaze podaci o prosečnim temperaturama u Bostonu od 1995 do 2016 godine u Farenhajtima. Napisati program koji ispisuje prosečne temperature u Bostonu za svaku godinu od 1995 do 2016 godine posebno u Celzijusima.

Format podataka je:

```
1  mesec dan godina temperatura
```

[Rešenje 7.9]

7.4 Zadaci za vežbu

Zadatak 7.10 Napisati program koji učitava ceo broj n i ispisuje faktorijele svih brojeva od 1 do n.

Zadatak 7.11 Napisati program koji učitava ceo broj n i ispisuje sumu prvih n elemenata harmonijskog reda (podsetimo se, harmonijski red je red oblika 1 + 1/2 + 1/3 + 1/4 ...).

Zadatak 7.12 Napisati program koji racuna zbir dva vektora (pretpostavimo da su vektori uvek zadati ispravno tj. iste su dužine) i ispisuje ih na izlaz.

Vektori se nalaze u datotekama *vektor1.txt* i *vektor2.txt* u formatu:

```
a1, a2, a3, a4, ... an
```

Zadatak 7.13 Napisati program koji racuna broj pojavljivanja svake cifre 0-9 u datoteci *knjiga.txt* i ispisuje rezultat sortiran po ciframa.

Zadatak 7.14 Napisati program koji prebrojava sve poruke o greškama koje se odnose na Spark alat i rezultat ispisuje na izlaz. Poruke o greeskama se nalaze u datoteci *log.txt* u formatu:

```
tip ostatak_poruke
```

Tip poruke o grešci je `[error]`.

Zadatak 7.15 Napisati program koji izdvaja dane sa najvišom temperaturom u Bostonu za svaku godinu posebno, počev od 1995 do 2016 i rezultat upisuje u direktorijum *MaxTemp*. Podaci o temperaturama se nalaze u datoteci *temperaturaBoston.txt* u formatu:

```
mesec dan godina temperatura
```

Zadatak 7.16 Napisati program koji izdvaja podatke o veličini preuzetih paketa koji se odnose na Apache server i rezultat upisuje u direktorijum *ApacheDownloaded*. Podaci o preuzimanjima se nalaze u datoteci *mavenLog.txt* u formatu:

```
Downloaded: putanja podaci_o_velicini
```

Zadatak 7.17 Ekonomski analitičari žele da analiziraju tržište tehničke robe sa nekim rasponom cena. Napisati program koji učitava ime marke i bira nasumično pet poruka o transakcijama koje se odnose na tu marku. Podaci o transakcijama se nalaze u datoteci *uredjaji.txt* u formatu:

```
marka_uredjaja ostali_podaci
```

Zadatak 7.18 Napisati program koji računa prosečnu platu programera u firmi. Podaci o zaposlenima se nalaze u datoteci *zaposleni.txt* u formatu:

```
ime prezime pol identifikator IP_adresa_racunara datum_zaposlenja
   sifra_pozicije plata
```

Šifra pozicije programera je `IT_PROG`.

7.5 Rešenja

Rešenje 7.1 Parni kvadrati

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD._

object ParniKvadrati {
```

```
  def main(args: Array[String]) = {

    println("Unesite broj n:")
    val n = Console.readInt()
    /**
     * Podesavamo konfiguraciju Spark okruzenja
     * tako sto dajemo ime aplikaciji
     * i dodeljujemo joj potencijalno 4 cvora
     * (u nasem slucaju procesorska jezgra)
     * */
    val konf = new SparkConf()
      .setAppName("ParniKvadrati")
      .setMaster("local[4]")

    /**
     * Ako Spark izbacuje gresku da je heap size manji od potrebnog,
     * otkomentarisati narednu liniju i proslediti odgovarajucu vrednost
     * */
    // konf.set("spark.testing.memory", "2147480000")

    /**
     * Pravimo objekat Spark konteksta
     * koji pokrece i upravlja Spark okruzenjem
     * */
    val sk = new SparkContext(konf)


    /**
     * Ukoliko zelimo da podesavamo parametre dinamicki
     * (broj cvorova koji izvrsavaju nasu aplikaciju,
     * velicina hip memorije i sl.)
     * potrebno je da inicijalizujemo spark kontekst na sledeci nacin
     *
     * val sk = new SparkContext(new SparkConf());
     *
     * cime naznacavamo da ce se parametri konfiguracije
     * podesiti dinamicki (koriscenjem spark-submit skripte).
     * */

    val niz = (2 to n by 2).toArray

    /**
     * Pravimo niz tipa RDD[Integer] od niza tipa Array[Integer]
     * */
    val nizRDD = sk.parallelize(niz)

    /**
     * Pravimo niz kvadrata parnih brojeva (uzimamo prvih 10)
     * i rezultat pretvaramo u niz tipa Array[Integer]
     * */
    val nizKvadrata = nizRDD.map(x => x*x)
                          /* .take(10) */
                             .collect()

    /**
     * Zaustavljamo Spark okruzenje
     * */
    sk.stop()

    /**
     * Ispisujemo rezultujuci niz
     * */
    println("Niz kvadrata parnih brojeva: ")
    println(nizKvadrata.mkString(", "))
  }
}
```

Rešenje 7.2 Broj petocifrenih

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
```

```
import org.apache.spark.rdd.RDD._


object BrojPetocifrenih {

  def main(args: Array[String]){

      val konf = new SparkConf()
        .setAppName("BrojPetocifrenih")
        .setMaster("local[4]")

      val sk = new SparkContext(konf)

      /**
       * Otvaramo datoteku i njen sadrzaj cuvamo
       * u nizu tipa RDD[String].
       * Elementi niza su pojedinacne linije iz datoteke.
       * */
      val datRDD = sk.textFile("brojevi.txt")

      /**
       * Filtriramo niz tako da nam ostanu samo petocifreni brojevi
       * i prebrojavamo ih.
       * */
      val brojPetocifrenihBrojeva = datRDD.filter(_.length() == 5)
                                          .count()

      sk.stop()

      println("Petocifrenih brojeva ima: ")
      println(brojPetocifrenihBrojeva)
  }
}
```

Rešenje 7.3 Skalarni proizvod

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD._

object SkalarniProizvod {

   def main(args: Array[String]){

      val konf = new SparkConf()
        .setAppName("SkalarniProizvod")
        .setMaster("local[4]")

      val sk = new SparkContext(konf)

      /**
       * Otvaramo datoteku i ucitavamo vektor.
       * */
      val vek1RDD = sk.textFile("vektor1.txt")
                      /**
                      * Razdvajamo elemente vektora iz niske koristeci separator ", "
                      * */
                      .flatMap(_.split(", "))
                      /**
                      * Kastujemo niske u tip Integer.
                      * */
                      .map(_.toInt)

      val vek2RDD = sk.textFile("vektor2.txt")
                      .flatMap(_.split(", "))
                      .map(_.toInt)

      /**
       * Spajamo nizove vektora A i B funkcijom zip
       * i pravimo jedan niz parova (tipa Tuple)
       * tako da svaki par sadrzi element vektora A i element vektora B
       *  (a1,b1), (a2,b2), ... (an, bn).
```

```
37        * */
          val skProizvod = vek1RDD.zip(vek2RDD)
39                              /**
                                * Mnozimo elemente para.
41                              * */
                                .map(par => par._1 * par._2)
43                              /**
                                * Pomnozene elemente parova sabiramo.
45                              * */
                                .reduce((a, b) => a+b)
47        sk.stop()

49        println("Skalarni proizvod je: ")
          println(skProizvod)
51    }
}
```

Rešenje 7.4 Broj pojavljivanja reči

```
import org.apache.spark.SparkConf
2 import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD._
4
object BrojPojavljivanjaReci {
6
   def main(args: Array[String]){
8
      val konf = new SparkConf()
10      .setAppName("BrojPojavljivanjaReci")
        .setMaster("local[4]")
12
      val sk = new SparkContext(konf)
14
      val knjigaRDD = sk.textFile("knjiga.txt")
16
      /**
18     * Ucitavamo linije i razlazemo ih separatorom " " tako da dobjemo niz reci
       */
20    val reciBr = knjigaRDD.flatMap(_.split(" "))
                            /**
22                          * Od svake reci pravimo par (rec, 1).
                            */
24                          .map(rec => (rec , 1))
                            /**
26                          * Sabiramo sve vrednosti drugog elementa para
                            * grupisane po prvom elementu para
28                          * koji nam predstavlja kljuc.
                            */
30                          .reduceByKey((_+_))
                            /**
32                          * Sortiramo reci leksikografski.
                            */
34                          .sortByKey()
                            /**
36                          * Cuvamo ih u datotekama koje se nalaze u direktorijumu
                            * BrojPojavljivanjaReci
38                          */
                            .saveAsTextFile("BrojPojavljivanjaReci")
40
      sk.stop()
42  }
}
```

Rešenje 7.5 Uređaji transakcije

```
1 import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
3 import org.apache.spark.rdd.RDD._
import java.io._
5
```

```
object UredjajiTransakcije {

  def main(args: Array[String]){

      val konf = new SparkConf()
        .setAppName("UredjajiTransakcije")
        .setMaster("local[4]")

      val sk = new SparkContext(konf)

      val transakcije = sk.textFile("uredjaji.txt")
                          /**
                           * Razdvajamo podatke o uredjajima
                           * i pravimo parove transakcija
                           * (marka, ostali_podaci)
                           * */
                          .map(linija => {
                              val niz = linija.split(" ")
                              (niz(0), niz.drop(1).mkString(" "))
                            })
                          /**
                          * Grupisemo ih po marki tako da za svaku marku
                          * cuvamo niz obavljenih transakcija
                          * (tj. niz koji sadrzi ostale podatke za svaku transakciju)
                          * */
                          .groupByKey()
                          /**
                           * Prolazimo kroz niz parova (marka, niz_transakcija)
                           * i u datoteku ime_marke.txt
                           * upisujemo podatke o transakcijama.
                           *
                           * Parametar t u foreach konstrukciji predstavlja jedan par
                           * (marka, niz_transakcija).
                           * */
                          .foreach(t => {
                              val dat = new PrintWriter(new File(t._1.toLowerCase() +
    ".txt" ))

                              dat.write("---" + t._1 + "---\n")

                              t._2.foreach(por => {
                                  dat.append(por + "\n")
                              })

                              dat.close()
                          })

      sk.stop()
   }
}
```

Rešenje 7.6 Log poruke

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD._

object LogPoruke {

  def main(args: Array[String]){

      val konf = new SparkConf()
        .setAppName("LogPoruke")
        .setMaster("local[4]")

      val sk = new SparkContext(konf)

      val poruke = sk.textFile("log.txt")
                     /**
                      * Filtriramo podatke tako da nam ostanu
                      * samo linije koje predstavljaju
                      * upozorenja, informacije ili greske i odnose se na javu.
```

```
                        * */
21                      .filter(linija =>
                                  (linija.contains("[warn]")
23                                 || linija.contains("[info]")
                                   || linija.contains("[error]"))
25                                 && (linija.contains("java")))
                        /**
27                       * Pravimo parove (tip_poruke, poruka).
                         * */
29                      .map(linija => {
                             val niz = linija.split(" ")
31                           (niz(0), niz.drop(1).mkString(" "))
                        })
33                      /**
                         * Grupisemo poruke po njihovom tipu (kljuc).
35                       * tako da dobijemo niz parova (tip_poruke, niz_poruka).
                         * */
37                      .groupByKey()
                        /**
39                       * Za svaki tip racunamo broj poruka tog tipa
                         * tako sto od parova (tip_poruke, niz_poruka)
41                       * pravimo par (tip_poruke, broj_poruka)
                         * */
43                      .map( por => (por._1, por._2.size))
                        .collect()
45
        println("Informacije o log porukama koje se odnose na Javu: ")
47
        poruke.foreach( por => println(" " + por._1 + ": " + por._2 ))
49
        sk.stop()
51    }
}
```

Rešenje 7.7 Uspešna preuzimanja

```
1  import org.apache.spark.SparkConf
   import org.apache.spark.SparkContext
3  import org.apache.spark.rdd.RDD._

5  object UspesnaPreuzimanja {

7    def main(args: Array[String]){

9        val konf = new SparkConf()
           .setAppName("UspesnaPreuzimanja")
11         .setMaster("local[4]")

13       val sk = new SparkContext(konf)

15       /**
          * Ucitavamo podatke i smestamo ih u kes memoriju radi brzeg pristupanja.
17        * */
         val preuzimanja = sk.textFile("mavenLog.txt")
19                           .cache()
         /**
21        * Racunamo broj zapocetih preuzimanja.
          * */
23       val zapoceta = preuzimanja.filter(_.contains("Downloading:"))
                                   .count()
25       /**
          * Racunamo broj zavrsenih preuzimanja.
27        * */
         val zavrsena = preuzimanja.filter(_.contains("Downloaded:"))
29                                 .count()

31       sk.stop()

33       println("%.2f".format(zavrsena*100.0/zapoceta) + " procenata zapocetih
       preuzimanja je zavrseno.")
     }
35 }
```

Rešenje 7.8 Pokloni

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD._
import scala.compat._

object RandomZaposleni {

  def main(args: Array[String]){

      val konf = new SparkConf()
        .setAppName("RandomZaposleni")
        .setMaster("local[4]")

      val sk = new SparkContext(konf)

      /**
       * Pronalazimo liniju koja sadrzi radnika programera,
       * razdvajamo podatke o jednom radniku
       * i cuvamo njegovo ime, prezime i email nalog.
       * Nakon toga nasumicno biramo 3 programera.
       *
       * Funkcija takeSample kao argumente prihvata:
       * - indikator da li zelimo izbora sa vracanjem
       * - broj uzoraka
       * - pocetnu vrednost (seed) za slucajni generator
       * */
      val triProgramera = sk.textFile("zaposleni.txt")
                            /**
                             * Pronalazimo liniju koja sadrzi radnika programera,
                             * */
                            .filter(_.contains("IT_PROG"))
                            /**
                            * Razdvajamo podatke o jednom radniku
                            * i cuvamo njegovo ime, prezime i email nalog.
                            * */
                            .map(linija => {
                                val niz = linija.split(" ")
                                (niz(0), niz(1), niz(3))
                              })
                            /**
                            * Nasumicno biramo 3 programera.
                            * Prvi parametar (false) oznacava
                            * izbor bez vracanja a poslednji parametar
                            * predstavlja pocetnu vrednost (seed)
                            * generatora slucajnih brojeva.
                            * */
                            .takeSample(false, 3, Platform.currentTime)

      println("Tri zaposlena radnika u IT sektoru su: ")
      triProgramera.foreach(prog => {
        println("Ime i prezime: " + prog._1 + " " + prog._2 + "\n  Email: " + prog._3
    .toLowerCase()+"@firma.com")
      })

      sk.stop()
   }
}
```

Rešenje 7.9 Prosečna temperatura

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD._

object ProsecnaTemperatura {

   def main(args: Array[String]){

      val konf = new SparkConf()
        .setAppName("ProsecnaTemperatura")
```

```
11          .setMaster("local[4]")

13        val sk = new SparkContext(konf)

15        val tempRDD = sk.textFile("temperatureBoston.txt")
                        /**
17                       * Pravimo torke (kljuc, vrednost) takve da je
                         * kljuc = godina
19                       * vrednost = (mesec, dan, temperatura)
                         * */
21                      .map(linija => {
                          val niz = linija.split(" ")
23                        (niz(3),(niz(1), niz(2), niz(4).toFloat))
                        })
25                      /**
                         * Grupisemo torke po njihovom kljucu i za svaki kljuc racunamo
27                       * sumu svih temperatura i broj temperatura koje smo sabrali.
                         *
29                       * Funkcija aggregateByKey(pocetnaVrednostAkumulatora)(f1, f2)
                         * obradjuje niz parova (kljuc, vrednost)
31                       * grupise ih po kljucu, akumulira vrednosti
                         * (inicijalna vrednost akumulatora
33                       * se prosledjuje kao parametar)
                         * i kao rezultat vraca niz parova (k, akumuliranaVrednost).
35                       *
                         * Funkcija f1(akumulator, vrednost) se primenjuje
37                       * nad svim vrednostima koje se nalaze u jednom cvoru
                         * i rezultat se smesta u akumulator tog cvora
39                       * Funkcija f2(akumulator1, akumulator2) se primenjuje
                         * nad svim izracunatim akumulatorima pojedinacnih cvorova
41                       * i rezultat se smesta u globalni akumumulator.
                         *
43                       * U nasem slucaju, akumulator ce da sadrzi dve vrednosti
                         * (sumaTemp, brojTemp) sa nulom kao pocetnom vrednoscu.
45                       * Funkcija f1(akumulator, vrednost)
                         * sabira vrednost (temperaturu)
47                       * sa sumaTemp iz akumulatora, a brojTemp povecava za 1.
                         * Funkcija f2(akumulator1, akumulator2)
49                       * sabira obe vrednosti ova dva akumulatora.
                         *
51                       * Kao rezultat primene ove funkcije dobicemo niz parova
                         * (kljuc, (sumaTemp, brojTemp))
53                       * gde nam je kljuc godina
                         * u kojoj zelimo da izracunamo prosecnu temperaturu.
55                       * */
                        .aggregateByKey((0.0, 0))((ak, vr) => (ak._1 + vr._3 , ak._2 +
    1 ),
57                                                (a1 ,a2) => (a1._1 + a2._1, a1._2 + a2.
    _2))
                        /**
59                       * Pravimo niz parova (kljuc, prosecnaTemperatura)
                         * */
61                      .map( st => (st._1, st._2._1/st._2._2))
                        .sortByKey()
63                      .collect()
                        .foreach( st => println("Godine "
65                                              + st._1
                                                + " prosecna temperatura je iznosila "
67                                              + "%.2f".format(((st._2-32)/1.8))
                                                + " celzijusa. "))
69
        sk.stop()
71    }
  }
```

8

Logičko programiranje

8.1 Jezik Prolog

Prolog (eng. *PROgramming in LOGic*) je deklarativan programski jezik namenjen rešavanju zadataka simboličke prirode. Prolog se temelji na teorijskom modelu logike prvog reda. Početkom 1970-ih godina Alain Kolmerauer (eng. *Alain Colmerauer*) i Filipe Rousel (eng. Philippe Roussel) na Univerzitetu u Marselju (eng. University of Aix-Marseille), zajedno sa Robertom Kovalskim (eng. *Robert Kowalski*) sa Odeljka Veštačke Inteligencije (eng. *Department of Artifical Intelligence*) na Univerzitetu u Edinburgu (eng. *University of Edinburgh*), razvili su osnovni dizajn jezika Prolog.

8.2 Instalacija BProlog-a

U okviru kursa će biti korišćena distribucija Prologa pod nazivom `BProlog`. `BProlog` se može preuzeti sa zvanične Veb strane http://www.picat-lang.org/bprolog/.

Potrebno je preuzeti adekvatnu verziju za Vaš sistem i otpakovati je. U dobijenom direktorijumu će postojati izvršiva datoteka `bp` kojom se može pokrenuti `BProlog` interpreter. Preporučeno je dodati `bp` u `PATH` kako bi `BProlog` bio dostupan iz komandne linije.

Na primer, pretpostavimo da imamo 64bitni Linux. Potrebno je preuzeti datoteku `bp81_linux64.tar.gz` i smestiti je u direktorijum po izboru, na primer `/home/korisnik/Downloads`. Potom treba izvršiti sledeće naredbe:

```
cd ~/Downloads
tar -xvf bp81_linux64.tar.gz
sudo mv BProlog /opt
sudo ln -s /opt/BProlog/bp /usr/bin/bprolog
```

Nakon toga, `BProlog` interpreter se iz konzole može pokrenuti komandom `bprolog`.

8.3 Uvod

8.3.1 Uvodni primeri

Zadatak 8.1 U bazu znanja uneti informacije o životinjama i njihovim odnosima po pitanju veličine. Napisati pravilo koje omogućava da se proveri koja je od dve životinje veća, kao i da generiše sve životinje za koje je neka životinja veća.

```
% ovo je jednolinijski komentar

/*
ovo je viselinijski
komentar
*/

/*
```

```
Programi se cuvaju sa ekstenzijom .pro ili .pl (na sistemima gde ekstenzije imaju globalno znacenje, kao sto je MS-Windows da ne bi bilo mesanja sa Perl programima treba uvek koristiti .pro).
Interpreter se pokrece komandom bp. Naredbe:
help -- pomoc
compile('ime_programa') -- prevodi program i pravi izvrsni fajl ukoliko nema gresaka
load('ime_izvrsnog_fajla') -- uvozi izvrsni fajl
cl('ime_programa') -- compile + load
halt ili control+D -- za izlazak iz interpretera

Termovi: konstante, promenljive ili kompozitni termovi.
--- Konstante: atomi i brojevi.
------ Atomi: stringovi od najvise 1000 karaktera koji pocinju malim slovom ('abc', 'a01', 'b_cd', 'l122k', ...).
------ Brojevi: celi i realni.
--- Promenljive: imena pocinju velikim slovom ili podvlakom (_). Specijalna, anonimna promenljiva: '_'.
--- Kompozitni (slozeni) termovi ili strukture: oblika f(t1, ..., tn) gde je f neka funkcija arnosti n ( 0<n<32768), a t1, ..., tn termovi.

Program:  sekvenca Hornovih klauza. Postoje tri tipa Hornovih klauza: cinjenice, pravila i upiti.
--- Cinjenice: atomicna formula oblika p(t1, ..., tn) gde je p predikat arnosti n, a t1, ..., tn termi. One opisuju svojstva i relacije izmedju objekata. Primer:
  zivotinja(slon).
  veci(zebra,vuk).
--- Pravila: imaju sledecu formu
  H :- B1, ..., Bn. (n>0)
H, B1, ..., Bn su atomicne formule. H se zove GLAVA pravila, dok je sve sa desne strane :- TELO pravila. Citamo ih kao implikaciju sa desna na levo: vazi H, ako vaze B1, ..., Bn ("," u pravilu zamenjuje logicko "i").
--- Cinjenice i pravila cine BAZU ZNANJA.
--- Upiti: konstrukcije kojima korisnik komunicira sa bazom znanja. Za ovo je neophodan interpetator kome se postavlja upit. Primer:
  ?- veci(slon, zec).
  true.
  ?- zivotinja(veverica).
  false.
*/

/* cinjenice, svojstva */
zivotinja(slon).
zivotinja(vuk).
zivotinja(zec).
zivotinja(zebra).

/* cinjenice, odnosi */
veci(slon,vuk).
veci(vuk,zec).
veci(slon,zebra).
veci(zebra,vuk).
veci(slon,zec).

/*
upiti sa promenljivama:
-- daje jedno resenje, ako zelimo da prikaze jos resenja kucamo ; nakon prikazanog ? a za prekid control+C

?- veci(slon, X).
X = vuk ? ^C

| ?- veci(slon, X).
X = vuk ?;
X = zebra ?;
X = zec
yes

| ?- veci(X, Y).
X = slon
Y = vuk ?;
X = vuk
Y = zec ?;
X = slon
```

```
Y = zebra ?;
X = zebra
Y = vuk ?;
X = slon
Y = zec
yes
*/

/*
pravilo : za neko X i Y vazi je_veci(X,Y) ako postoji Z tako da vazi veci(X,Z) i veci
    (Z,Y)
*/
je_veci(X,Y):- veci(X,Z), veci(Z,Y).
/*
| ?- je_veci(X,Y)
X = slon
Y = zec ?;
X = slon
Y = vuk ?;
X = zebra
Y = zec ?;
no
*/
```

Zadatak 8.2 Unifikacija. Jednakost.

```
/*

Provera tipa:

atom(X) - da li je term X atom
atomic(X) - da li je term X atom ili broj
number(X) -  da li je term X broj
float(X) ili real(X) - da li je term X realan broj
integer(X) - da li je term X ceo broj
var(X) -  da li je term X slobodna promenljiva
nonvar(X) - da li term X nije promenljiva

Primeri:
| ?- atom('abc')
yes
| ?- atomic(5)
yes

Unifikacija:

=   unifikabilni
\=  nisu unifikabilni
==  identicno jednaki termovi
\== nisu identicno jednaki termovi

*/

uni(X, Y):- X = Y.

/*
| ?- uni(4,5)
no
| ?- uni(4,X)
X = 4
yes
*/

jed(X, Y):- X == Y.

/*
| ?- jed(4,X)
no
| ?- jed(4,5)
no
| ?- jed(4,4)
yes
*/
```

Zadatak 8.3 Aritmetički operatori. Operatori *is* i *cut*.

```
/*
is   aritmeticko izracunavanje
=:=  aritmeticki jednaki
=\=  aritmeticki nisu jednaki
<, =<, >, >=
+, -, *, /, // (celobrojno deljenje), div, mod, ** (stepenovanje)
*/

/*
Ako je X promenljiva, tada se njoj dodeljuje vrednost koju ima term Y (mora biti
    poznata vrednost), a ukoliko X nije promenljiva, X is Y se svodi na X =:= Y
*/
op1(X, Y):- X is Y.
/*
Termovima X i Y moraju biti poznate vrednosti, inace ce prijaviti gresku.
*/
op2(X, Y):- X =:= Y.

/*
| ?- op1(3,4)
no
| ?- op1(4,4)
yes
| ?- op1(X,4)
X = 4
yes
| ?- op1(4,X)
*** error
| ?- op2(4,4)
yes
| ?- op2(4,2)
no
| ?- op2(4,X)
*** error
| ?- op2(X,4)
*** error
*/

/*
apsolutna vrednost, prvi argument je broj za koji trazimo apsolutnu vrednost, a drugi
     promenljiva gde se smesta rezultat

losa implementacija, za pozitivne brojeve oba pravila prolaze

| ?- abs1(1,X)
X = 1 ?;
X = -1
yes
| ?- abs1(-1,X)
X = 1
yes

*/
abs1(X, X):- X >= 0.
abs1(X, Y):- Y is -X.

/*
dobre implementacije abs2 i abs3

| ?- abs2(1,X)
X = 1 ?
yes
| ?- abs2(-1,X)
X = 1
yes
*/
abs2(X, X):- X >= 0.
abs2(X, Y):- X < 0, Y is -X.
```

```
/*
koriscenje operatora "cut" koji se oznacava sa "!" uklanjamo alternativne klauze,
    tako da se u slucaju pozitivnih nece primeniti drugi predikat cim uspe prvi

| ?- abs3(1,X)
X = 1
yes
| ?- abs3(-1,X)
X = 1
yes

*/
abs3(X, X):- X >= 0, !.
abs3(X, Y):- Y is -X.
```

Zadatak 8.4 Rekurzivni predikat, primer porodičnog stabla.

```
% porodicno stablo

% svojstva
musko(mihajlo).
musko(stevan).
musko(petar).
musko(mladen).
musko(rajko).
zensko(milena).
zensko(milica).
zensko(jelena).
zensko(senka).
zensko(mina).
zensko(maja).

% odnosi
roditelj(mihajlo,milica).
roditelj(mihajlo,senka).
roditelj(milena,rajko).
roditelj(maja,petar).
roditelj(maja,mina).
roditelj(stevan,mladen).
roditelj(stevan,jelena).
roditelj(milica,mladen).
roditelj(milica,jelena).

% pravila
majka(X,Y):- roditelj(X,Y), zensko(X).
otac(X,Y):- roditelj(X,Y), musko(X).
brat(X,Y):- musko(X), majka(Z,X), majka(Z,Y), X\==Y.
sestra(X,Y):- zensko(X), majka(Z,X), majka(Z,Y), X\==Y.
ujak(X,Y):- brat(X,Z), majka(Z,Y).
tetka(X,Y):- sestra(X,Z), majka(Z,Y).

% rekurzivno pravilo
% roditelj je predak
predak(X,Y):- roditelj(X,Y).
% roditelj pretka je takodje predak
predak(X,Y):- roditelj(X,Z), predak(Z,Y).
```

Zadatak 8.5 Napisati Prolog predikate:

- `prestupna` koji određuje da li je godina prestupna
- `brdana` koji određuje koliko dana ima prosleđeni mesec

```
% godina je prestupna ako je deljiva sa 4 i nije deljiva sa 100 ili je deljiva sa 400
prestupna(X):- X mod 4 =:= 0, X mod 100 =\= 0.
prestupna(X):- X mod 400 =:= 0.

/*
anonimna promenljiva _ se koristi da oznaci da nam vrednost koja se prosledi za
    godinu nije bitna, moze biti bilo sta, ali tu vrednost ne koristimo

```

```
| ?- brdana(januar, _, X)
X = 31
yes
| ?- brdana(januar, 2017, X)
X = 31
yes
*/
brdana(januar, _, 31).
brdana(februar, X, 28):- not(prestupna(X)).
brdana(februar, X, 29):- prestupna(X).
brdana(mart,_,31).
brdana(april,_,30).
brdana(maj,_,31).
brdana(jun,_,30).
brdana(jul,_,31).
brdana(avgust,_,31).
brdana(septembar,_,30).
brdana(oktobar,_,31).
brdana(novembar,_,30).
brdana(decembar,_,31).
```

8.3.2 Zadaci za samostalni rad sa rešenjima

Zadatak 8.6 Napisati sledeće predikate:

a) `maksimum(A, B, M)` - određuje maksimum za dva broja `A` i `B`

b) `suma(N, S)` - za dati prirodan broj `N` računa sumu prvih `N` brojeva

c) `sumaParnih(N, S)` - za dati paran prirodan broj `N` računa sumu parnih brojeva od 2 do `N`

d) `proizvod(N, P)` - za dati prirodan broj `N` računa proizvod prvih `N` prirodnih brojeva

e) `proizvodNeparnih(N, P)` - za dati neparan prirodan broj `N` računa proizvod neparnih brojeva od 1 do `N`

f) `cifre(N)` - ispisuje cifre prirodnog broja `N` rečima

[Rešenje 8.6]

8.3.3 Zadaci za vežbu

Zadatak 8.7 Napisati sledeće predikate:

a) `sumaCifara(N, SC)` - određuje sumu cifara prirodnog broja `N`

b) `brojCifara(N, BC)` - određuje broj cifara prirodnog broja `N`

c) `maxCifra(N, MC)` - određuje maksimalnu cifru prirodnog broja `N`

d) `sumaKvadrata(N, SK)` - računa sumu kvadrata prvih `N` prirodnih brojeva

e) `fakt(N, F)` - računa faktorijel prirodnog broja `N`

g) `sumaDel(X, D)` - računa sumu pravih delilaca broja `X`

Zadatak 8.8 Ako su date činjenice oblika:

- `ucenik( SifraUcenika, ImeUcenika, Odeljenje)`
- `ocene( SifraUcenika, SifraPredmeta, Ocena)`
- `predmet( SifraPredmeta, NazivPredmeta, BrojCasova)`

Napisati sledeće predikate:

a) `bar2PeticeSifra(S)` - određuje šifru `S` učenika koji ima bar dve petice iz različitih predmeta

b) `bar2PeticeIme(X)` - određuje ime `X` učenika koji ima bar dve petice iz različitih predmeta

c) `odeljenjePetice(X,Y)` - određuje odeljenje `X` u kome postoje bar dve petice iz predmeta sa šifrom `Y`

Zadatak 8.9 Ako su date činjenice oblika:

- `film(NazivFilma, ZanrFilma, ImeReditelja, SifraGlumca)`
- `glumac(SifraGlumca, ImeGlumca, GodRodj, MestoRodj)`

Napisati sledeće predikate:

a) `filmskiUmetnik(X)` - `X` je filmski umetnik ako je reditelj nekog filma i igra u nekom filmu

b) `glumacBarDva(X)` - određuje ime glumca `X` koji igra u bar dva različita filma

c) `opstiGlumac(X)` - određuje ime glumca `X` koji igra u bar dva filma različitog žanra

d) `zanrovskiGlumac(X,Y)` - određuje ime glumca `X` koji igra u filmu žanra `Y`

8.4 Liste

8.4.1 Uvodni primeri

Zadatak 8.10 Osnovni pojmovi i predikati za rad sa listama.

```
/*
Lista - niz uredjenih elemenata, tj. termova.
Lista moze biti:
  [] - prazna
  .(G,R) - struktura, gde je G ma koji term i naziva se glava liste, a R lista i
    naziva se rep liste

Primeri:
  [] - prazna
  .(a, []) - jednoclana lista, gde je a bilo koji term
  .(a, .(b, [])) - dvoclana lista, gde su a i b termi ...

Zapis pomocu zagrada (prvi element predstavlja glavu, a ostali cine listu koja je rep
    ):
  [a,b,c] <=> .(a, .(b, [c]))

Zapis liste u kom su jasno razdvojeni glava i rep (pogodan za unifikaciju): [G|R].

Primeri unifikacije listi:
[X, Y, Z] [jabuka, kruska, banana]  -----> X = jabuka, Y = kruska, Z = banana
[racunar] [X|Y]                     -----> X = racunar, Y = []
[maja, ana, jovana] [X, Y|Z]          -----> X = maja, Y = ana, Z = [jovana]

*/

% predikat proverava da li element pripada listi (ako joj pripada jednak je glavi ili
     nekom elementu iz repa liste)
sadrzi(X, [X|_]):- !.
sadrzi(X, [G|R]):- G \== X, sadrzi(X, R).

% drugi nacin, predikat kao disjunkcija:
% sadrzi(X, [G|R]):- G == X; sadrzi(X, R).

% predikat koji racuna duzinu liste (prazna je duzine nula, nepraznu dekomponujemo na
     glavu i rep, pa je duzina liste = 1 + duzina repa)
duzina([], 0).
duzina([G|R], L):- duzina(R,L1), L is L1+1.

% predikat racuna sumu elemenata liste brojeva
```

```
suma([], 0).
suma([G|R], S):- number(G), suma(R, S1), S is S1+G.

% predikat racuna aritmeticku sredinu elemenata liste brojeva
arsr([],0).
% ako ne koristimo sablon za nepraznu listu [G|R], moramo proveriti da li je K nula
    jer se sada L moze unifikovati sa []
arsr(L, A):- duzina(L, K), K =\= 0, suma(L, S), A is S/K.

% predikat ucitava listu duzine N ciji elementi mogu biti proizvoljni termovi
% za negativno N ne ucitava listu
ucitaj(N,_) :- N < 0, !.
% za nulu vraca praznu listu
ucitaj(0, []).
% read(X) ucitanu vrednost sa ulaza dodeljuje promenljivoj X
% ako je N > 1, lista ima glavu i rep, ucitamo glavu, pa pozovemo predikat za
    ucitavanje repa
ucitaj(N, [G|R]):- N >= 1, write('unesi element '), read(G), nl, M is N-1, ucitaj(M,R
    ).

/*
prilikom unosa vrednosti obavezna je tacka kao oznaka kraja ulaza za read
| ?- ucitaj(3,L)
unesi element | 5.

unesi element | 4.

unesi element | 3.

L = [5,4,3]
yes

| ?- ucitaj(3,L)
unesi element | [1,2,3].

unesi element | 4.

unesi element | [].

L = [[1,2,3],4,[]]
yes

*/
```

8.4.2 Zadaci za samostalni rad sa rešenjima

Zadatak 8.11 Napisati sledeće predikate:

a) `dodajPocetak(X, L, NL)` - dodaje `X` na početak liste `L`

b) `dodajKraj(X, L, NL)` - dodaje `X` na kraj liste `L`

c) `obrisiPrvi(L, NL)` - briše prvi element, tj. glavu liste

d) `obrisiPoslednji(L, NL)` - briše poslednji element liste

e) `obrisi(X, L, NL)` - briše sva pojavljivanja elementa `X` u listi `L`

f) `obrisiPrvo(X, L, NL)` - briše samo prvo pojavljivanje elementa `X` u listi `L`

g) `obrisiK(L, K, NL)` - briše `K`-ti element liste `L`

[Rešenje 8.11]

Zadatak 8.12 Napisati predikat `podeli(L, L1, L2)` koji deli listu `L` na dve liste, listu pozitivnih elemenata `L1` i listu negativnih elemenata `L2`.

[Rešenje 8.12]

Zadatak 8.13 Napisati predikat `dupliraj(L, NL)` koji od date liste `L` formira novu listu `NL` tako što svaki negativan element duplira, tj. dva puta upisuje u novu listu.

[Rešenje 8.13]

Zadatak 8.14 Napisati predikat `zameni(X, Y, L, NL)` koji od date liste `L` formira novu listu `NL` zamenom elemenata `X` i `Y`.

[Rešenje 8.14]

Zadatak 8.15 Napisati predikat `pretvori(L, X)` koji za datu listu cifara `L` formira broj određen tim ciframa.

[Rešenje 8.15]

Zadatak 8.16 Napisati predikat `maxEl(L, X)` koji određuje maksimalni element liste `L`.

[Rešenje 8.16]

Zadatak 8.17 Napisati predikate za sortiranje liste rastuće:

a) `insertionSort(L, SL)` - insertion sort algoritam se zasniva na ubacivanju redom svakog elementa liste na svoje pravo mesto (mesto u sortiranoj listi)

b) `mergeSort(L, SL)` - merge sort algoritam se zasniva na dekompoziciji liste, tj. listu delimo na dva jednaka dela, te delove sortiramo i posle toga ih objedinjujemo

[Rešenje 8.17]

8.4.3 Zadaci za vežbu

Zadatak 8.18 Napisati predikat `parNepar(L, L1, L2)` koji deli listu `L` na dve liste, listu parnih elemenata `L1` i listu neparnih elemenata `L2`.

Zadatak 8.19 Napisati predikat `podeli(L, N, L1, L2)` koji deli listu `L` na dve liste `L1` i `L2`, pri čemu je zadata dužina prve liste `L1`.

Zadatak 8.20 Napisati predikat `ogledalo(L1, L2)` koji proverava da li je lista `L1` jednaka obrnutoj listi liste `L2`.

Zadatak 8.21 Napisati predikat `interval(X, Y, L)` koji kreira listu `L` koja sadži sve cele brojeve iz intervala zadatog sa prva dva argumenta.

Zadatak 8.22 Napisati predikat `skalar(L1, L2, S)` koji određuje skalarni proizvod dva vektora, tj. listi brojeva `L1` i `L2`.

Zadatak 8.23 Napisati predikat `sortirana(L)` koji proverava da li je lista `L` sortirana, bilo opadajuće ili rastuće.

Zadatak 8.24 Napisati predikat `spoji(L1, L2, L)` koji spaja dve rastuće sortirane liste `L1` i `L2` u treću tako da i ona bude sortirana rastuće.

8.5 Razni zadaci

8.5.1 Zadaci sa rešenjima

Zadatak 8.25 Napisati predikate `nzd(N, M, NZD)` i `nzs(N, M, NZS)` koji određuju najveći zajednički delilac i najmanji zajednički sadržalac prirodnih brojeva `N` i `M` redom.

[Rešenje 8.25]

Zadatak 8.26 Ako su date činjenice oblika:

- `stan(Porodica, KvadraturaStana)`
- `clan(Porodica, BrojClanova)`

Napisati predikat `poClanu(Porodica, Prosek)` koji određuje prosečan broj kvadrata stana po članu porodice koja živi u njemu.

[Rešenje 8.26]

Zadatak 8.27 Ako je data baza znanja:

- `automobil(SifraAutomobila, NazivAutomobila)`
- `vlasnik(ImeVlasnika, SifraAutomobila)`
- `brziSifra(SX, SY)` - automobil šifre `SX` je brži od automobila šifre `SY`

Napisati predikate:

a) `brziNaziv(X, Y)` - automobil naziva `X` je brži od automobila naziva `Y`

b) `imaAutomobil(X)` - `X` je vlasnik nekog automobila

c) `imaBrzi(X, Y)` - `X` je vlasnik bržeg automobila od onog čiji je vlasnik `Y`.

[Rešenje 8.27]

Zadatak 8.28 Napisati predikat `savrsen(N)` koji proverava da li je prirodan broj `N` savršen, tj. da li je jednak sumi svojih pravih delilaca. U slučaju da se prosledi neispravan argument, predikat treba da ispiše poruku o grešci i prekine program.

[Rešenje 8.28]

Zadatak 8.29 Napisati predikat `izbaci3(N, X)` koji iz prirodnog broja `N` izbacuje sve cifre manje 3. U slučaju da se prosledi neispravan argument, predikat treba da prekine program.

[Rešenje 8.29]

Zadatak 8.30 Napisati predikate za liste brojeva:

a) `duplikati(L, L1)` - izbacuje duplikate iz liste `L`

b) `unija(L1, L2, L)` - određuje uniju listi `L1` i `L2`

c) `presek(L1, L2, L)` - određuje presek listi `L1` i `L2`

d) `razlika(L1, L2, L)` - određuje razliku listi `L1` i `L2`

[Rešenje 8.30]

Zadatak 8.31 Napisati program koji rešava sledeću zagonetku. Postoji pet kuća, svaka različite boje u kojoj žive ljudi različitih nacionalnosti koji piju različita pića, jedu različita jela i imaju različite kućne ljubimce. Važi sledeće:

- Englez živi u crvenoj kući
- Španac ima psa
- kafa se pije u zelenoj kući
- Ukrajinac pije čaj
- zelena kuća je odmah desno uz belu
- onaj koji jede špagete ima puža
- pica se jede u žutoj kući
- mleko se pije u srednjoj kući

- Norvežanin živi u prvoj kuci s leva
- onaj koji jede piletinu živi pored onoga koji ima lisicu
- pica se jede u kući koja je pored kuće u kojoj je konj
- onaj koji jede brokoli pije sok od narandze
- Japanac jede suši
- Norvežanin živi pored plave kuće

Čija je zebra, a ko pije vodu?

[Rešenje 8.31]

Zadatak 8.32 Napisati program koji rešava sledeću zagonetku. Svakog vikenda, Milan čuva petoro komšijske dece. Deca se zovu Kata, Lazar, Marko, Nevenka i Ognjen, a prezivaju Filipović, Grbović, Hadžić, Ivanović i Janković. Svi imaju različit broj godina od dve do šest. Važi sledeće:

- jedno dete se zove Lazar Janković
- Kata je godinu dana starija od deteta koje se preziva Ivanović koje je godinu dana starije od Nevenke
- dete koje se preziva Filipović je tri godine starije od Marka
- Ognjen je duplo stariji od deteta koje se preziva Hadžić

Kako se ko zove i koliko ima godina?

[Rešenje 8.32]

8.5.2 Zadaci za vežbu

Zadatak 8.33 Napisati predikat `uzastopni(X, Y, Z, L)` koji proverava da li su prva tri argumenta uzastopni elementi u listi `L`.

Zadatak 8.34 Napisati predikat `kompresuj(L, KL)` koji u datoj listi `L` eliminiše uzastopne duplikate.

Zadatak 8.35 Napisati predikat `prefiksi(L, P)` koji određuje sve liste koje su prefiksi date liste `L`.

Zadatak 8.36 Napisati predikat `sufiksi(L, S)` koji određuje sve liste koje su sufiksi date liste `L`.

Zadatak 8.37 Napisati predikat `opadajuce(N, L)` koji za dat prirodan broj `N` formira listu brojeva od `N` do `1`.

Zadatak 8.38 Napisati predikat `form(N, L)` kojim se formira lista od prirodnih brojeva deljivih sa 5 i manjih od datog prirodnog broja `N`.

Zadatak 8.39 Napisati program koji rešava sledeću zagonetku. Četiri žene se zovu Petra, Milica, Lenka i Jovana, a prezivaju Perić, Mikić, Lazić i Jović. One imaju četiri kćerke koje se takodje zovu Petra, Milica, Lenka i Jovana. Važi sledeće:

- nijedna majka nema prezime koje počinje istim slovom kao ime
- nijedna kćerka nema prezime koje počinje istim slovom kao ime
- nijedna kćerka se ne zove kao majka
- majka koja se preziva Perić se zove isto kao Miličina kćerka
- Lenkina kćerka se zove Petra

Odrediti imena majki i kćerki.

Zadatak 8.40 Napisati program koji rešava sledeću zagonetku. Četiri para je došlo na maskenbal:

- Markova zena se maskirala kao macka
- dva para su stigla pre Marka i njegove žene, a jedan muskarac je bio maskiran u medveda
- prvi koji je stigao nije bio Vasa, ali je stigao pre onoga koji je bio maskiran u princa
- žena maskirana u vešticu (nije Bojana) je udata za Peru, koji se maskirao kao Paja patak
- Marija je došla posle Laze, a oboje su stigli pre Bojane
- žena maskirana u Ciganku je stigla pre Ane, pri čemu nijedna od njih nije udata za muškarca maskiranog u Betmena
- žena maskirana u Snežanu je stigla posle Ivane

Odrediti kako je bio obučen koji par.

Zadatak 8.41 Izračunavanje vrednosti aritmetičkog izraza korišćenjem listi možete pogledati ovde:
https://rosettacode.org/wiki/Arithmetic_evaluation#Prolog

Zadatak 8.42 Implementacije raznih problema u Prologu možete pogledati ovde:
https://rosettacode.org/wiki/Category:Prolog

9

Programiranje ograničenja - Prolog

Potrebno je imati instaliran B-Prolog na računaru.

Literatura:

(a) http://www.picat-lang.org/bprolog/

(b) http://www.picat-lang.org/bprolog/download/manual.pdf

9.1 Programiranje ograničenja

9.1.1 Uvodni primeri

Zadatak 9.1 Osnovni pojmovi i postavka problema.

```
/*

Programiranje ogranicenja nad konacnim domenom:
1) generisanje promenljivih i njihovih domena
2) generisanje ogranicenja nad promenljivima
3) instanciranje promenljivih ili obelezavanje

Definisanje domena (D) promenljivih:
-- X in D ili X :: D -> promenljiva X uzima samo vrednosti iz konacnog domena D
-- Vars in D ili Vars :: D -> sve promenjive iz liste Vars uzimaju samo vrednosti iz
    konacnog domena D
Domen se definise kao interval u obliku Pocetak..Korak..Kraj (Korak je opcion i
    ukoliko se ne navede, podrazumeva se da je Korak = 1)
Primeri za domen:
1..10  -> 1,2,3,4,5,6,7,8,9,10
1..2..10 -> 1,3,5,7,9

Osnovni predikati za ogranicenja

1) opsta:
-- alldifferent(Vars) ili alldistinct(Vars) - sve vrednosti razlicite u listi termova
     Vars
-- atmost(N,L,V) - najvise N elemenata iz skupa L jednako sa V (N mora biti Integer,
    V term, a L lista termova)
-- atleast(N,L,V) - najmanje N elemenata iz skupa L jednako sa V (N mora biti Integer
    , V term, a L lista termova)
-- excatly(N,L,V) - tacno N elemenata iz skupa L jednako sa V (N mora biti Integer, V
     term, a L lista termova)

2) aritmeticka:
-- E1 R E2 gde su E1 i E2 aritmeticki izrazi, a R iz skupa {#=, #\=, #>=, #>, #=<,
    #<}
-- min(L) - minimalni element iz liste termova L
-- max(L) - maksimalni element iz liste termova L
```

```
-- max(E1, E2)/min(E1, E2) -- manji/veci od izraza E1 i E2
-- sum(L) - suma elemenata liste termova L

Instanciranje i prikazivanje promenljivih: labeling(Vars).
Funkcija labeling se moze pozvati i sa razlicitim opcijama u obliku labeling(Options, Vars)
gde je Options lista opcija. Ukoliko se pozove sa labeling(Vars), podrazumevano je Options = []. Neke od opcija:
-- minimize(E) - trazi instance za Vars pri kojima je vrednost celobrojnog izraza E minimalna
-- maximize(E) - trazi instance za Vars pri kojima je vrednost celobrojnog izraza E maksimalna

*/

/*
Primer: X pripada skupu {1,2,3}, Y skupu {2,4,6,8,10}, Z skupu {5,6,7,8} i vazi Z>=Y
*/

primer(Vars) :- Vars = [X, Y, Z], % generisanje promenljivih
    X :: 1..3,    % definisanje domena
    Y :: 2..2..10,
    Z :: 5..8,
    Z #>= Y,    % ogranicenje
    labeling(Vars).   % instanciranje

/*
| ?- primer(Vars).
Vars = [1,2,5] ?;
Vars = [1,2,6] ?;
Vars = [1,2,7] ?;
Vars = [1,2,8] ?;
Vars = [1,4,5] ?;
Vars = [1,4,6] ?;
Vars = [1,4,7] ?;
Vars = [1,4,8] ?;
Vars = [1,6,6] ?;
Vars = [1,6,7] ?;
Vars = [1,6,8] ?;
Vars = [1,8,8] ?;
...
*/

/*
Primer: ispisati sve brojeve od 1..100 koji su puni kvadrati
 */

puni(Vars) :- Vars = [X],
           Vars :: 1..100,
           Y*Y #= X ,
           labeling(Vars).

/*
| ?- puni(Vars)
Vars = [1] ?;
Vars = [4] ?;
Vars = [9] ?;
Vars = [16] ?;
Vars = [25] ?;
Vars = [36] ?;
Vars = [49] ?;
Vars = [64] ?;
Vars = [81] ?;
Vars = [100] ?;
no

*/
```

9.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 9.2 Napisati program koji pronalazi petocifren broj `ABCDE` za koji je izraz `A+2*B-3*C+4*D-5*E`

minimalan i `A`, `B`, `C`, `D` i `E` su različite cifre.

[Rešenje 9.2]

Zadatak 9.3 Dati su novčići od 1, 2, 5, 10, 20 dinara. Napisati program koji pronalazi sve moguće kombinacije tako da zbir svih novčića bude 50 i da se svaki novčič pojavljuje bar jednom u kombinaciji.

[Rešenje 9.3]

Zadatak 9.4 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 9.4]

Zadatak 9.5 Napisati program koji pronalazi sve vrednosti promenljivih X, Y, 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 $Z \in \{1, 10, 20, ..., 100\}$

[Rešenje 9.5]

Zadatak 9.6 Napisati program koji dodeljuje različite vrednosti različitim karakterima tako da suma bude zadovoljena:

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

[Rešenje 9.6]

Zadatak 9.7 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 $W \in \{1, 8, 15, 22, ..., 1000\}$.

[Rešenje 9.7]

Zadatak 9.8 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 9.8]

Zadatak 9.9 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 9.9]

Zadatak 9.10 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 9.10]

Zadatak 9.11 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 9.11]

Zadatak 9.12 Napisati program koji raspoređuje n dama na šahovsku tablu dimenzije nxn tako da se nikoje dve dame ne napadaju.

[Rešenje 9.12]

Zadatak 9.13 Napisati program koji za ceo broj n 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 $i = 0, 1, ..., n-1$.

[Rešenje 9.13]

9.1.3 Zadaci za vežbu

Zadatak 9.14 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 9.15 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 9.16 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 9.17 Uraditi sve zadatke koji su pobrojani ovde:
http://www.primepuzzle.com/leeslatest/alphameticpuzzles.html

Zadatak 9.18 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 9.19 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 9.20 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?

9.2 Rešenja

Rešenje 8.6

```
% maksimum dva broja
% I nacin:
maksimum(A,B,M):- A>=B, M is A.
maksimum(A,B,M):- A<B, M is B.
% II nacin bez trece promenjive:
% maksimum(A,B,A):- A>=B.
% maksimum(A,B,B):- A<B.

% suma prvih N prirodnih brojeva
suma(1,1).
suma(N,S):- N>1, N1 is N-1, suma(N1,S1), S is S1+N.

% suma parnih prirodnih brojeva od 2 do N
% moze se dodati provera N mod 2 =:= 0 u pravilu, ali i bez toga sam prepoznaje za
    neparne da je netacan upit jer rekurzijom dodje do sumaParnih(1,S) sto je netacna
     cinjenica u bazi
sumaParnih(2,2).
sumaParnih(N,S):- N>2, N1 is N-2, sumaParnih(N1,S1), S is S1+N.

% proizvod prvih N prirodnih brojeva
proizvod(1,1).
proizvod(N,P):- N>1, N1 is N-1, proizvod(N1,P1), P is P1*N.

% proizvod neparnih prirodnih brojeva od 1 do N
proizvodNeparnih(1,1).
proizvodNeparnih(N,P):- N>1, N1 is N-2, proizvodNeparnih(N1,P1), P is P1*N.

% ispis cifara unetog prirodnog broja N
cifra(0, nula).
cifra(1, jedan).
cifra(2, dva).
cifra(3, tri).
cifra(4, cetiri).
cifra(5, pet).
cifra(6, sest).
cifra(7, sedam).
cifra(8, osam).
cifra(9, devet).

% ukoliko nije prirodan broj, cut operatorom sprecavamo poziv poslednjeg predikata
cifre(N):- N < 1, !.

% ukoliko je jednocifren svodi se na poziv predikata cifra
% write(t) gde je t neki term, ispisuje term
% nl (newline) - ispisuje se novi red
% obratiti paznju na upotrebu cut operatora ! - sprecavamo poziv poslednjeg predikata
     za jednocifrene
cifre(N):- N > 1, N < 10, cifra(N, C), write(C), nl, !.

% ukoliko nije jednocifren, racunamo tekucu cifru koju ispisujemo i ostatak broja za
    koji se ponovo poziva predikat
cifre(N):- N1 is (N // 10), cifre(N1), N2 is (N mod 10), cifra(N2, C), write(C), nl.
```

Rešenje 8.11

```
% dodaje element na pocetak  liste
dodajPocetak(X, L, [X|L]).

% dodaje element na kraj liste
dodajKraj(X, [], [X]).
dodajKraj(X, [G|R], [G|LR]):- dodajKraj(X, R, LR).

% brise prvi element liste
% za praznu listu ce uvek vratiti no jer ne moze unifikovati sa sablonom [G|R], a
    mozemo napraviti i sami za taj slucaj da je predikat netacan
% fail je uvek netacan pa ce nam ovako definisan predikat za slucaj prazne liste
    vratiti no
```

```
obrisiPrvi([], _):- fail.
obrisiPrvi([_|R], R).

% brise poslednji element liste
obrisiPoslednji([],_):- fail.
% bitan cut operator jer se jednoclana moze upariti sa sablonom [G|R]
obrisiPoslednji([_],[]):- !.
obrisiPoslednji([G|R], [G|R1]):- obrisiPoslednji(R, R1).

% brise element X iz liste ako postoji (svako pojavljivanje elementa X)
obrisi(_, [], []).
obrisi(X, [X|R], R1):- obrisi(X, R, R1), !.
obrisi(X, [G|R], [G|R1]):- G \== X, obrisi(X, R, R1).

% brise element X iz liste ako postoji (samo prvo pojavljivanje elementa X)
obrisiPrvo(X, [], []).
obrisiPrvo(X, [X|R], R):- !.
obrisiPrvo(X, [G|R], [G|R1]):- G \== X, obrisiPrvo(X, R, R1).

% brise K-ti element liste, brojimo od 1, ako je K vece od duzine liste, treci
    argument je jednak prvom
obrisiK([], K, []):- K > 0.
obrisiK([G|R], 1, R):- !.
obrisiK([G|R], K, [G|R1]):- K>1, K1 is K-1, obrisiK(R, K1, R1).

% druga varijanta predikata: brise K-ti element liste, broji od 1, ali ukoliko zadata
     lista nema K-ti element, predikat vraca no kao odgovor i nema unifikacije za
    treci argument
% obrisiK([], K, []):- fail.
% obrisiK([G|R], 1, R):- !.
% obrisiK([G|R], K, [G|R1]):- K>1, K1 is K-1, obrisiK(R, K1, R1).
```

Rešenje 8.12

```
% podeli1 - deli listu na dve liste - listu pozitivnih i listu negativnih elemenata
% L1 - lista pozitivnih, L2 - lista negativnih
podeli([], [], []).
podeli([G|R], [G|R1], L2):- G >= 0, podeli(R, R1, L2), !.
podeli([G|R], L1, [G|R2]):- G < 0, podeli(R, L1, R2).
```

Rešenje 8.13

```
% svaki negativan element dodajemo dva puta u novu listu, a pozitivne samo jednom
dupliraj([], []).
dupliraj([G|R], [G,G|R1]):- G<0, dupliraj(R, R1), !.
dupliraj([G|R], [G|R1]):- G>=0, dupliraj(R, R1).
```

Rešenje 8.14

```
% menja elemente X i Y u listi
zameni(X, Y, [], []).
zameni(X, Y, [X|R], [Y|R1]):- zameni(X, Y, R, R1), !.
zameni(X, Y, [Y|R], [X|R1]):- zameni(X, Y, R, R1), !.
zameni(X, Y, [G|R], [G|R1]):- G \== X, G \== Y, zameni(X, Y, R, R1).
```

Rešenje 8.15

```
% potreban nam je dodatni predikat za izdvajanje poslednjeg elementa liste
izdvojPoslednji([G], G, []):- !.
izdvojPoslednji([G|R], X, [G|R1]):- izdvojPoslednji(R, X, R1).

% formira broj od date liste cifara
pretvori([], 0):- !.
pretvori(L, X):- izdvojPoslednji(L, Poslednji, Ostatak),
    pretvori(Ostatak, Y),
    X is Poslednji + 10 * Y.

```

```
/*
| ?- pretvori([7,0,7], X)
X = 707
yes
| ?- pretvori([0,2,3], X)
X = 23
yes
| ?- pretvori([], X)
X = 0
yes

*/
```

Rešenje 8.16

```
% maksimalni element liste
maxEl([X], M):- M is X, !.
% pozivamo za rep (idemo u dubinu), pa poredimo maksimalni element repa i glavu liste
maxEl([G|R], X):- maxEl(R, Y), G < Y, X is Y, !.
maxEl([G|R], X):- maxEl(R, Y), G >= Y, X is G.
```

Rešenje 8.17

```
/*

Insertion sort algoritam se zasniva na ubacivanju redom svakog elementa liste na
    svoje pravo
mesto.

IH (Induktivna hipoteza) - umemo da sortiramo listu od n-1 elementa.
IK (Induktivni korak) - n-ti element ubacujemo na odgovarajucu lokaciju.

*/
insertionSort([], []).
insertionSort([G|R], SL):- insertionSort(R, S1), ubaciS(G, S1, SL).

% ubacuje sortirano element X u listu
ubaciS(X, [], [X]).

% ubaciS(X, [G|R], [X, G|R]):- X=<G.
% ubaciS(X, [G|R], [G|SL]):- X>G, ubaciS(X, R, SL).
/*
 alternativno sa cut operatorom ne moramo
 da pisemo X>G u drugom predikatu, jer
 kada stavimo cut operator kod prvog, cim
 on uspe drugi predikat se nece ni pokusavati
*/
ubaciS(X, [G|R], [X, G|R]):- X=<G, !.
ubaciS(X, [G|R], [G|SL]):- ubaciS(X, R, SL).

/*

Merge sort algoritam se zasniva na dekompoziciji. Naime, ulaznu listu delimo na dva
    jednaka
dela, te delove sortiramo i posle toga ih objedinjujemo.

IH - umemo da sortiramo liste velicine n/2.
IK - u linearnom vremenu objedinjujemo dve sortirane liste od po n/2 elemenata.

*/
mergeSort([], []).
mergeSort([X], [X]).
mergeSort(N, SL):- podeli(N, L, R),
     mergeSort(L, L1),
     mergeSort(R, R1),
     objedini(L1, R1, SL).

% delimo tako sto prvi element stavljamo u prvu particiju
% drugi u drugu pri cemu su particije dobijene
% vec rekurzivno podelom repa
```

```
podeli([], [], []).
podeli([X], [X], []):- !.
podeli([G1, G2|R], [G1|R1], [G2|R2]):- podeli(R, R1, R2).

/*
Primer: ako objedinjujemo [1,3,5,7] i [5,6,7,8,9] znamo da 1 prethodi svima pa mozemo
     rekurzivno da objedinimo [3,5,7] sa [5,6,7,8,9] i dodamo 1 na pocetak tako
    objedinjene liste
*/
objedini(L, [], L).
objedini([], L, L).
objedini([G1|R1], [G2|R2], [G1|R]):- G1<G2, objedini(R1, [G2|R2], R), !.
objedini([G1|R1], [G2|R2], [G2|R]):- G1>=G2, objedini([G1|R1], R2, R).
```

Rešenje 8.25

```
% poredjenjem glava listi zakljucujemo u kom redosledu dodajemo elemente u spojenu
    listu
spoji([], L, L):- !.
spoji(L, [], L):- !.
spoji([G1|R1], [G2|R2], [G1|R]):- G1<G2, spoji(R1, [G2|R2], R), !.
spoji([G1|R1], [G2|R2], [G2|R]):- G1>=G2, spoji([G1|R1], R2, R).
```

Rešenje 8.26

```
/*
za odredjivanje nzd i nzs dva prirodna broja koristicemo Euklidov algoritam

bitno je obezbediti da se pogram ispravno ponasa - kada nadje jedno resenje treba
    obezbediti prekid koriscenjem cut operatora u 1. klauzi nzdPom, jer bi u
    suprotnom program zasao u skup negativnih brojeva (pa bi nastao beskonacan ciklus
    ) ili pokusaj deljenja sa nulom (sto bi prijavilo gresku)
*/
nzd(A,B,NZD):- A>=B, B>0, nzdPom(A,B,NZD),!.
nzd(A,B,NZD):- A<B, A>0, nzdPom(B,A,NZD).

nzdPom(N, 0, N):- !.
nzdPom(N, M, NZD):- P is N mod M, nzdPom(M, P, NZD).

% nzs dobijamo mnozenjem oba prirodna broja i deljenjem sa nzd
nzs(N, M, NZS):- nzd(N, M, P), NZS is N*M//P.

/*
| ?- nzd(15,9,NZD)
NZD = 3
yes
| ?- nzd(12,8,NZD)
NZD = 4
yes
| ?- nzs(13,17,NZS)
NZS = 221
yes
*/
```

Rešenje 8.27

```
% cinjenice
stan(petrovic, 76).
stan(ciric, 93).
stan(aleksic, 55).
stan(lisic, 123).
stan(peric, 67).

clan(ciric, 3).
clan(peric, 5).
clan(aleksic, 2).
clan(lisic, 3).
clan(petrovic, 4).

```

```
% koristimo operator za realno deljenje / (// je za celobrojno, ne mesati ova dva
    operatora) da bi se dobio tacan rezultat
poClanu(Porodica, Prosek):- stan(Porodica, X), clan(Porodica, Y), Prosek is X/Y.
```

Rešenje 8.28

```
% baza znanja
automobil(a1, audi).
automobil(h1, honda).
automobil(m1, mercedes).
automobil(m2, mercedes).
automobil(c1, citroen).
automobil(c2, citroen).

vlasnik(milan, h1).
vlasnik(maja, m1).
vlasnik(nemanja, m2).
vlasnik(aleksandar, a1).
vlasnik(andjela, c1).
vlasnik(petar, c2).

brziSifra(a1, c1).
brziSifra(m1, c1).
brziSifra(m2, h1).
brziSifra(a1, c2).

brziNaziv(X, Y):- automobil(SX, X), automobil(SY, Y), brziSifra(SX, SY).

imaAutomobil(X):- vlasnik(X, _).

imaBrzi(X, Y):- vlasnik(X, S1), vlasnik(Y, S2), brziSifra(S1, S2).
```

Rešenje 8.29

```
% fail ce prouzrokovati da 2. klauza uvek bude netacna jer zelimo da se poziv
    predikata za neispravan argument vidi kao netacna cinjenica u bazi
provera(N):- N > 0.
provera(N):- N =< 0, write('Broj nije prirodan'), nl, fail.

% predikat sumaPom za treci argument ima kandidata delioca broja N koji postepeno
    uvecavamo (pocinjemo od jedinice koja deli svaki broj)
sumaDelilaca(N, S):- sumaPom(N, S, 1).
% dovoljno je ici do N/2, suma je inicijalno 0, tako da na primer za N==1 vraca 0
sumaPom(N, 0, I):- I>N//2.
% ako I deli, dodajemo ga na tekucu sumu i pozivamo za sledeceg kandidata delioca
sumaPom(N, S, I):- I=<N//2, N mod I=:=0, I1 is I+1, sumaPom(N, S1, I1), S is S1+I.
% ako I ne deli, ne dodajemo ga na tekucu sumu vec samo pozivamo za sledeceg
    kandidata delioca
sumaPom(N, S, I):- I=<N//2, N mod I=\=0, I1 is I+1, sumaPom(N, S, I1).

% ako provera prodje, tada se izvrsava predikat sumaDelilaca i proverava da li je
    dobijena suma jednaka broju N, ako provera ne prodje, ispisace se poruka i
    prekinuti program
savrsen(N):- provera(N), sumaDelilaca(N, S), N=:=S.

/*
| ?- savrsen(6)
yes
| ?- savrsen(-6)
Broj nije prirodan
no
*/
```

Rešenje 8.30

```
% maksimum dva broja
% I nacin:
maksimum(A,B,M):- A>=B, M is A.
maksimum(A,B,M):- A<B, M is B.
```

```
% II nacin bez trece promenjive:
% maksimum(A,B,A):- A>=B.
% maksimum(A,B,B):- A<B.

% suma prvih N prirodnih brojeva
suma(1,1).
suma(N,S):- N>1, N1 is N-1, suma(N1,S1), S is S1+N.

% suma parnih prirodnih brojeva od 2 do N
% moze se dodati provera N mod 2 =:= 0 u pravilu, ali i bez toga sam prepoznaje za neparne da je netacan upit jer rekurzijom dodje do sumaParnih(1,S) sto je netacna cinjenica u bazi
sumaParnih(2,2).
sumaParnih(N,S):- N>2, N1 is N-2, sumaParnih(N1,S1), S is S1+N.

% proizvod prvih N prirodnih brojeva
proizvod(1,1).
proizvod(N,P):- N>1, N1 is N-1, proizvod(N1,P1), P is P1*N.

% proizvod neparnih prirodnih brojeva od 1 do N
proizvodNeparnih(1,1).
proizvodNeparnih(N,P):- N>1, N1 is N-2, proizvodNeparnih(N1,P1), P is P1*N.

% ispis cifara unetog prirodnog broja N
cifra(0, nula).
cifra(1, jedan).
cifra(2, dva).
cifra(3, tri).
cifra(4, cetiri).
cifra(5, pet).
cifra(6, sest).
cifra(7, sedam).
cifra(8, osam).
cifra(9, devet).

% ukoliko nije prirodan broj, cut operatorom sprecavamo poziv poslednjeg predikata
cifre(N):- N < 1, !.

% ukoliko je jednocifren svodi se na poziv predikata cifra
% write(t) gde je t neki term, ispisuje term
% nl (newline) - ispisuje se novi red
% obratiti paznju na upotrebu cut operatora ! - sprecavamo poziv poslednjeg predikata za jednocifrene
cifre(N):- N > 1, N < 10, cifra(N, C), write(C), nl, !.

% ukoliko nije jednocifren, racunamo tekucu cifru koju ispisujemo i ostatak broja za koji se ponovo poziva predikat
cifre(N):- N1 is (N // 10), cifre(N1), N2 is (N mod 10), cifra(N2, C), write(C), nl.
```

Rešenje 8.31

```
% pomocni predikat za proveru pripadnosti elementa listi
sadrzi(X, [X|_]):- !.
sadrzi(X, [G|R]):- G \== X, sadrzi(X, R).

% izbacivanje duplikata iz liste
duplikati([], []).
duplikati([G|R], [G|R1]):- not(sadrzi(G, R)), duplikati(R,R1), !.
duplikati([_|R], R1):- duplikati(R,R1).

% pomocni predikat za spajanje dve liste
spoji([], L, L).
spoji([G|R1], L2, [G|R3]):- spoji(R1, L2, R3).

% unija listi - jedna ideja: spajamo liste pa uklanjamo duplikate
% za vezbu implementirati ovaj predikat tako da se duplikati uklanjaju pri samom spajanju listi
unija(L1, L2, L):- spoji(L1, L2, L3), duplikati(L3, L).

% presek listi
presek([], _, []).
presek([G|R1], L2, [G|R3]):- sadrzi(G, L2), presek(R1, L2, R3), !.
```

```
presek([_|R1], L2, L):- presek(R1, L2, L).

% razlika listi L1 i L2
razlika([], _, []).
razlika([G1|R1], L2, R3):- sadrzi(G1, L2), razlika(R1, L2, R3), !.
razlika([G|R1], L2, [G|R3]):- razlika(R1, L2, R3).
```

Rešenje 8.32

```
/*
strukturama oblika k(boja, nacionalnost, jelo, pice, kucniLjubimac) opisujemo date
    cinjenice, a u listi L su kuce poredjane jedna pored druge, tako da po redosledu
    u listi imamo informaciju da li je kuca desno od neke druge kuce i da li su kuce
    jedna pored druge
*/

/*
pomocni predikat koji proverava da li je X clan liste, njemu prosledjujemo strukturu
    k sa poznatim vrednostima iz teksta i opisujemo kakva kuca treba da bude u listi
*/
clan(X, [X|_]).
clan(X, [_|R]):- clan(X,R).

% predikat smesta u listu L kuce koje zadovoljavaju uslove iz teksta, tj. predikat L
    unifikuje sa resenjem zagonetke
% u listu ubacujemo cinjenice koje su vezane za raspored kuca i to samo one koje
    jednoznacno odredjuju poziciju kuce u listi
kuce(L):- L = [ k(_,norvezanin,_,_,_),
    k(plava,_,_,_,_),
    k(_,_,_,mleko,_),
    k(_,_,_,_,_),
    k(_,_,_,_,_) ],
    % dodajemo kakve sve kuce treba da budu u listi
    clan(k(crvena, englez,_,_,_),L),
    clan(k(_, spanac,_,_,pas),L),
    clan(k(zelena,_,_,kafa,_),L),
    clan(k(_, ukrajinac,_,caj,_),L),
    % kada informacija daje relaciju za neke dve kuce iz liste koristimo predikate
    desno i pored
    desno(k(zelena,_,_,kafa,_),k(bela,_,_,_,_),L),
    clan(k(_,_,spagete,_,puz),L),
    clan(k(zuta,_,pica,_,_),L),
    pored(k(_,_,piletina,_,_),k(_,_,_,_,lisica),L),
    pored(k(_,_,pica,_,_),k(_,_,_,_,konj),L),
    clan(k(_,_,brokoli,narandza,_),L),
    clan(k(_,japanac,susi,_,_),L),
    % medju datim informacijama se ne pominje zebra niti voda, ali posto je krajnje
    pitanje vezano za ove pojmove, moramo dodati da takvi clanovi treba da postoje u
    resenju zagonetke
    clan(k(_,_,_,_,zebra),L),
    clan(k(_,_,_,voda,_),L).

% proverava da li su kuce X i Y jedna pored druge u listi L
pored(X,Y,[X,Y|_]).
pored(X,Y,[Y,X|_]).
pored(X,Y,[_|R]):- pored(X,Y,R).

% proverava da li je kuca X desno od kuce Y u listi L
desno(X,Y,[Y,X|_]).
desno(X,Y,[_|R]):- desno(X,Y,R).

% predikat zagonetka daje odgovor na pitanje cija je zebra, a ko pije vodu, tako sto
    prvo trazi resenje zagonetke pa iz njega izdvaja samo potrebne clanove
zagonetka(X,Y):- kuce(L), clan(k(_,X,_,_,zebra),L), clan(k(_,Y,_,voda,_),L) .

/*
resenje:

| ?- kuce(L)
L = [k(zuta,norvezanin,pica,voda,lisica),k(plava,ukrajinac,piletina,caj,konj),k(
    crvena,englez,spagete,mleko,puz),k(bela,spanac,brokoli,narandza,pas),k(zelena,
    japanac,susi,kafa,zebra)]
```

```
52
odgovor na pitanje:
54
| ?- zagonetka(X,Y)
56 X = japanac
Y = norvezanin
58
*/
```

Rešenje 9.2

```
% funkciji labeling prosledjujemo zahtev za minimizaciju trazenog izraza u listi
2 petocifren :-   Vars = [A,B,C,D,E],
    % definisemo domen
4   A :: 1..9,
    B :: 0..9,
6   C :: 0..9,
    D :: 0..9,
8   E :: 0..9,
    % dodajemo uslov
10  alldifferent(Vars),
    % prilikom obelezavanja prosledjujemo i uslov minimizacije
12  labeling([minimize(A+2*B-3*C+4*D-5*E)],Vars),
    % prevodimo dobijene vednosti u broj
14  Broj is 10000*A+1000*B+100*C+10*D+E,
    % ispisujemo resenje
16  write(Broj), nl.
```

Rešenje 9.3

```
% promenljive oznacavaju broj novcica u kombinaciji redom za A - 1 din, B - 2 din, C
    - 5 din, D - 10 din, E - 20 din.
2 kombinacije(Vars) :- Vars = [A,B,C,D,E],
    % definisemo domen
4   A :: 1..50,
    B :: 1..25,
6   C :: 1..10,
    D :: 1..5,
8   E :: 1..2,
    % dodajemo uslov
10  A+2*B+5*C+10*D+20*E #= 50,
    labeling(Vars),
12  % ispisujemo resenje
    write(A+2*B+5*C+10*D+20*E = 50), nl.
```

Rešenje 9.4

```
magicni(Vars):- Vars=[X1,X2,X3,X4,X5,X6,X7,X8,X9],
2   % domen za sve je isti
    Vars :: 1..9,
4   % uslov razlicitosti
    alldifferent(Vars),
6   % uslovi za zbirove
    X1+X2+X3#=15,
8   X4+X5+X6#=15,
    X7+X8+X9#=15,
10      X1+X4+X7#=15,
        X2+X5+X8#=15,
12      X3+X6+X9#=15,
        X1+X5+X9#=15,
14      X3+X5+X7#=15,
        labeling(Vars),
16  write(X1), write(' '), write(X2), write(' '), write(X3), nl,
    write(X4), write(' '), write(X5), write(' '), write(X6), nl,
18  write(X7), write(' '), write(X8), write(' '), write(X9), nl.
/*
20 | ?- magicni(Vars).
2 7 6
22 9 5 1
```

```
4 3 8
Vars = [2,7,6,9,5,1,4,3,8] ?;
2 9 4
7 5 3
6 1 8
Vars = [2,9,4,7,5,3,6,1,8] ?;
4 3 8
9 5 1
2 7 6
...
*/
```

Rešenje 9.5

```
pronadji(Vars):- Vars=[X,Y,Z],
    % definisemo domen
    X :: 1..90,
    Y :: 2..2..60,
    Z :: 1..10..100,
    % definisemo ogranicenja
    Z #=< X,
    2*X+Y*X+Z #=< 34,
    % instanciramo promenljive
    labeling(Vars).

/*
| ?- pronadji(Vars)
Vars = [1,2,1] ?;
Vars = [1,4,1] ?;
Vars = [1,6,1] ?;
Vars = [1,8,1] ?;
Vars = [1,10,1] ?;
...

*/
```

Rešenje 9.6

```
jednakost(Vars):- Vars=[T,W,O,F,U,R],
    Vars :: 0..9,
    % iskljucujemo za T i F nulu iz domena
    T#\=0,
    F#\=0,
    alldifferent(Vars),
    2*(T*100+W*10+O) #= F*1000+O*100+U*10+R,
    labeling(Vars),
    write(' '), write(T), write(W), write(O), nl,
    write('+'), write(T), write(W), write(O), nl,
    write('----------'), nl,
    write(F), write(O), write(U), write(R), nl.
/*
| ?- jednakost(Vars)
 734
+734
----------
1468
Vars = [7,3,4,1,6,8] ?;
 765
+765
----------
1530
...

*/

/*
BITNO: Ukoliko se na kraju predikata doda fail, ispisace se sva resenja pri pozivu,
    inace staje cim pronadje jedno resenje, pa sledece dobijamo kucanjem ; za
    nastavljanje pretrage
kao u prethodnom primeru
```

```
*/

jednakostSvi(Vars):- Vars=[T,W,O,F,U,R],
    Vars :: 0..9,
    % iskljucujemo za T i F nulu iz domena
    T#\=0,
    F#\=0,
    alldifferent(Vars),
    2*(T*100+W*10+O) #= F*1000+O*100+U*10+R,
    labeling(Vars),
    write(' '), write(T), write(W), write(O), nl,
    write('+'), write(T), write(W), write(O), nl,
    write('------'), nl,
    write(F), write(O), write(U), write(R), nl,
    % odvajamo resenja jer zbog fail ne staje kod prvog
    % resenja vec ce nastaviti pretragu
    write('----------'), nl, fail.
```

Rešenje 9.7

```
pronadji(Vars):- Vars=[X,Y,Z,W],
    % definisemo domen
    X :: 1..10,
    Y :: 1..2..51,
    Z :: 10..10..100,
    W :: 1..7..1002,
    % definisemo ogranicenja
    2*W #=< X,
    3+Y #=< Z,
    X-11*W+Y+11*Z #=< 100,
    % instanciramo promenljive
    labeling(Vars).
/*
Ovo je primer sistema nejednacina bez resenja:

| ?- pronadji(Vars)
no

*/
```

Rešenje 9.8

```
% A1, B1, C1, D1, E1 predstavljaju brojeve na glavnoj dijagonali
% A2, B2, C1, D2, E2 predstavljaju brojeve na glavnoj dijagonali
dijagonale(Vars):- Vars=[A1,B1,C1,D1,E1,A2,B2,D2,E2],
    % domen za sve je isti
    Vars :: 1..9,
    % uslov razlicitosti
    alldifferent(Vars),
    % uslovi za zbirove
    A1+B1+C1+D1+E1#=25,
    A2+B2+C1+D2+E2#=25,
    % uslovi za poredak
    A1#<B1, B1#<C1, C1#<D1, D1#<E1,
    A2#<B2, B2#<C1, C1#<D2, D2#<E2,
        labeling(Vars),
    write(A1), write('     '), write(A2), nl,
    write(' '), write(B1), write('   '), write(B2), nl,
    write('   '), write(C1), nl,
    write(' '), write(D2), write('   '), write(D1), nl,
    write(E2), write('     '), write(E1), nl.
```

Rešenje 9.9

```
/*
 H - broj komada hleba, K - broj komada kifli

 H>=0
 K>=0
```

```
 S obzirom na to da imamo 20kg brasna na raspolaganju, mozemo napraviti:
 - najvise 20000/120 kifli
 - najvise 20000/300 hleba

 K <= 20000/120 ~ 166
 H <= 20000/300 ~ 66

 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 <= 66, K <= 100

*/

pekara(Vars) :- Vars = [H, K],
    H :: 0..66,
    K :: 0..100,

/*
 Ogranicenja vremena:
 - vreme potrebno za mesenje jednog hleba je 10 min,
   tj. za mesenje H komada hleba potrebno je 10*H minuta
 - vreme potrebno za mesenje jedne kifle je 12 min,
   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
*/

    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
*/
    300*H + 120*K #=< 20000,


/*
 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 - ovo dodajemo prilikom obelezavanja
*/
    labeling([maximize(7*H+9*K)], Vars),
    Zarada is 7*H+9*K,
    write('Maksimalna zarada od '), write(Zarada), write(' dinara se ostvaruje za '),
    write(H), write(' komada hleba i '), write(K), write(' komada kifli.'), nl.
```

Rešenje 9.10

```
heksagon(Vars):- Vars=[A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S],
    % domen za sve je isti
    Vars :: 1..38,
    % uslov razlicitosti
    alldifferent(Vars),
/*
 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
*/
    A+B+C#=38,
    D+E+F+G#=38,
    H+I+J+K+L#=38,
        M+N+O+P#=38,
        Q+R+S#=38,
% Dodajemo ogranicenja za svaku od glavnih dijagonala
    H+M+Q#=38,
    D+I+N+R#=38,
    A+E+J+O+S#=38,
        B+F+K+P#=38,
        C+G+L#=38,
% Dodajemo ogranicenja za svaku od sporednih dijagonala
    A+D+H#=38,
    B+E+I+M#=38,
    C+F+J+N+Q#=38,
        G+K+O+R#=38,
        L+P+S#=38,

        labeling(Vars),
    write('  '), write(A), write(' '), write(B), write(' '), write(C), nl,
    write(' '), write(D),write(' '), write(E), write(' '),write(F),write(' '), write(
    G), nl,
    write(H), write(' '),write(I),write(' '), write(J),write(' '), write(K),write(' '
    ), write(L), nl,
    write(' '), write(M),write(' '), write(N), write(' '),write(O),write(' '), write(
    P), nl,
    write('  '), write(Q),write(' '), write(R), write(' '),write(S), nl.
```

Rešenje 9.11

```
/*
kompanija ima 250 zaposlenih radnika
za sve njih organizuje dodatnu obuku
ako je E promenjiva za Elixir, a D za Dart
mora da vazi (gruba procena) E<=250, D<=250 i E + D = 250
*/

kompanija(Vars) :- Vars = [E, D],
    Vars :: 0..250,

    % dodajemo uslov za ukupan broj radnika
    E+D#=250,

    % dodajemo ogranicenje za broj projekat sati
    150*E+170*D#=<51200,

    % dodajemo ogranicenje za raspoloziva sredstva
    100*E+105*D#=<26000,

/*
ukupna zarada se dobija kada od ostvarene dobiti preko broja projekat/sati oduzmemo
    gubitak za placanje kurseva radnicima, tj. funkcija koju treba maksimizovati je:

150*5*E + 170*6*D - (100*E + 105*D) --> ovo dodajemo kod obelezavanja
*/
    labeling([maximize(150*5*E + 170*6*D - (100*E + 105*D))],Vars),
    Zarada is (150*5*E + 170*6*D - (100*E + 105*D)),
    write('Maksimalna zarada je '), write(Zarada),
    write(', broj radnika koje treba poslati na kurs Elixir je '), write(E),
    write(', a broj radnika koje treba poslati na kurs Dart je '), write(D),nl.
```

Rešenje 9.12

```
/*
```

```
Kraljica se moze pomerati u pravoj liniji vertikalno, horizontalno ili dijagonalno,
    za bilo koliko slobodnih polja. Resenje zahteva da nikoje dve dame ne dele istu
    vrstu, kolonu ni dijagonalu. Problem je opstiji od poznatog problema osam
    sahovskih dama jer u ovom problemu treba N dama postaviti na sahovsku tablu
    dimenzije NxN koje se zadaje kao ulazni podatak.

*** U resenju se koristi petlja FOREACH. Opsti oblik:

  foreach(E1 in D1, . . ., En in Dn, LocalVars, Goal)

- Ei - moze biti bilo koji term, najcesce promenljiva
- Di - lista ili opseg celih brojeva oblika L..R (L i R su ukljuceni)
- LocalVars (opciono) - lista lokalnih promenljivih
- Goal - predikat koji se poziva

Primeri:

| ?- foreach(I in [1,2,3], write(I))
123
yes

Domen za X i Y u ovom slucaju mora biti iste kardinalnosti
| ?- foreach((X,Y) in ([a,b], 1..2), writeln(X==Y))
a==1
b==2
yes

| ?- foreach(X in [a,b], Y in [1,2], writeln(X==Y))
a==1
a==2
b==1
b==2
yes
*/

% Indeksi niza Qs I i J predstavljaju kolone u kojima su kraljice, a elementi niza Qs
    [I] i Qs[J] predstavljaju vrste u kojima se nalaze kraljice
kraljice(N):- length(Qs,N),   % Qs je niz, tj. lista od n promenljivih
    Qs :: 1..N,
% I je implicitno razlicito od J -> razlicite kolone
% Qs[I] treba da bude razlicito od Qs[J] -> razlicite vrste
% apsolutna vrednost razlike vrsta treba da bude razlicita od apsolutne vrednosti
    razlike kolona -> razlicite dijagonale
    foreach(I in 1..N-1, J in I+1..N,
    (Qs[I] #\= Qs[J], abs(Qs[I]-Qs[J]) #\= J-I)),
    labeling(Qs),
    writeln(Qs), fail.
% stavljanjem predikata fail na kraj, dobicemo sve moguce kombinacije za kraljice
% bez fail, program staje posle prvog pronadjenog rezultata
```

Rešenje 9.13

```
/*
Primer magicne sekvence za N=5:
[2,1,2,0,0]

Izvrsavanje programa za gornji primer (1 sa desne strane kad je uslov S[J]#=I
    ispunjen, 0 kad nije):
sum([(S[J]#=I) : J in 1..N])#=S[I+1])
I=0
   S[1]==0 0
   S[2]==0 0
   S[3]==0 0
   S[4]==0 1
   S[5]==0 1
   ------
   ukupno 2
   S[1]==2 tacno! (imamo dve nule)
I=1
   S[1]==1 0
   S[2]==1 1
   S[3]==1 0
```

```
   S[4]==1 0
   S[5]==1 0
   ------
   ukupno 1
   S[2]==1 tacno! (imamo jednu jedinicu)
I=2
   S[1]==2 1
   S[2]==2 0
   S[3]==2 1
   S[4]==2 0
   S[5]==2 0
   ------
   ukupno 1
   S[3]==2 tacno! (imamo dve dvojke)
I=3
   S[1]==3 0
   S[2]==3 0
   S[3]==3 0
   S[4]==3 0
   S[5]==3 0
   ------
   ukupno 0
   S[4]==0 tacno! (imamo 0 trojki)

I=4
   S[1]==4 0
   S[2]==4 0
   S[3]==4 0
   S[4]==4 0
   S[5]==4 0
   ------
   ukupno 0
   S[5]==0 tacno! (imamo 0 cetvorki)

*/

/*

Zadavanje listi u obliku: [T : E1 in D1, . . ., En in Dn, LocalVars, Goal]
- za svaku kombinacija vrednosti E1,...,En, ako predikat Goal uspe, T se
dodaje u listu
- LocalVars i Goal su opcioni argumenti

*/
magicna(N):- length(S,N),
    S :: 0..N-1,
/* dodajemo ogranicenja:
element na poziciji S[I+1] treba da bude jednak broju elemenata liste koji su jednaki
     sa I, taj broj dobijamo sumiranjem liste ciji su elementi poredjenja na
    jednakost elementa liste sa trazenim brojem I
*/
    foreach(I in 0..N-1,
    sum([(S[J] #= I) : J in 1..N]) #= S[I+1]),
    labeling(S),
    writeln(S), fail. % da bismo dobili sve, a ne samo jednu magicnu sekvencu
    dodajemo fail predikat na kraj
```