

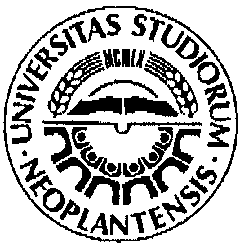
**UNIVERZITET U NOVOM SADU**

**PRIRODNO-MATEMATIČKI**

**FAKULTET**

**DEPARTMAN ZA MATEMATIKU**

**I INFORMATIKU**



Milan Balabanović, broj 347/21

Aplikacija za upravljanje transportnom firmom

(odnosno, SerboTransport)

- seminarski rad iz predmeta Skript jezici-

Novi Sad, 2022.

Table of Contents

[1. Uvod 3](#_Toc93677473)

[2. Opis programa 5](#_Toc93677474)

[Glavni program 5](#_Toc93677475)

[**Importovanje biblioteka i pratećih fajlova** 5](#_Toc93677476)

[**Prijavljivanje korisnika - login() funkcija** 6](#_Toc93677477)

[**Pokretanje programa - main() funkcija** 6](#_Toc93677478)

[Vozač i njegove funkcije 9](#_Toc93677479)

[**Kontruktorska \_\_init\_\_() funkcija** 9](#_Toc93677480)

[**getInfo() funkcija** 9](#_Toc93677481)

[**start() funkcija** 10](#_Toc93677482)

[**pomoc() funkcija** 11](#_Toc93677483)

[**prikaziGraf() funkcija** 11](#_Toc93677484)

[**nadjiAktivneRute() i aRuta() funkcije** 12](#_Toc93677485)

[**rute() funkcija** 13](#_Toc93677486)

[**zavrsi() funkcija** 15](#_Toc93677487)

[Menadžer i njegove funkcije 17](#_Toc93677488)

[**nadjiVozace() funkcija** 17](#_Toc93677489)

[**prikaziGraf() funkcija** 17](#_Toc93677490)

[**vozac() funkcija** 19](#_Toc93677491)

[**rute() funkcija** 20](#_Toc93677492)

[**nruta() funkcija** 23](#_Toc93677493)

[**posao() funkcija** 24](#_Toc93677494)

[3. Zaključak 27](#_Toc93677495)

[4. Literatura 28](#_Toc93677496)

# 1. Uvod

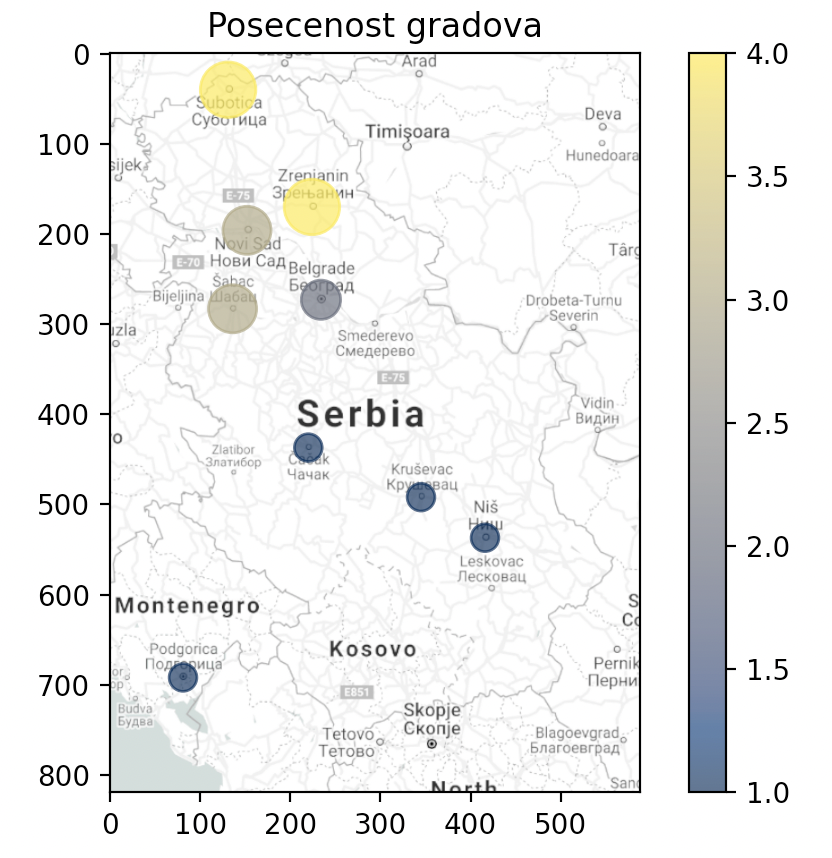


Slika 1. Logo transportne kompanije

Program je napravljen za upravljanje transportnom kompanijom zvanom SerboTransport (Slika 1). Urađen je tako da bude interaktivan i fokusiran na korisnika. Postoje dve vrste profila, vozač i menadžer, vozač ima nekoliko funkcionalnosti vezane za svoje rute, dok menadžer pored upravljanja rutama može da upravlja i vozačima.



Slika 2.Graf (tipa Pie) koji vidi vozač



Slika 3. Graf (tipa Scattered) koji vidi manadžer

Funkcionalnosti vozača su:

* Pregled svojih aktivnih ruta, uz ovu funkcionalnost dolazi i graf (tipa Pie), koji nam pokazuje koliko puta je vozac bio u kom gradu (Slika 2)
* Pregled svih svojih ruta
* Završavanje svojih aktivnih ruta

Funkcionalnosti menadžera su:

* Pregled svih ruta bilo kog vozača, uz ovu funkcionalnost dolazi i graf (tipa Scattered) koji prikazuje koliko puta je odabrani vozac bio u kom gradu (Slika 3)
* Kreiranje novih ruta za bilo kog vozača
* Pregled svih dostupnih podataka za bilo kog vozača
* Zapošljavanje i otpuštanje vozača

Svi podaci se nalaze u bazama koje su tekstualni dokumenti. Program koristi 4 baza koje su za Aktivne rute, Sve rute, Koordinate gradova i zapošljene. Podaci u tim bazama su podeljene tako da svaki zapis u bazi ima svoju liniju, a različiti podaci u liniju podeljeni su uspravnim crtama ( | ). Jedan zapis u bazi za rute bi izgledao ovako:

“Novi Sad-Zrenjanin|Pesak|3|200|4999”

Baze podataka su formatiratu:

* Aktivne rute
  + Ime fajla: AtivneRute.txt
  + Format fajla: Polazni grad-Krajnji grad|Tovar|ID vozača koji prevozi tu rutu|\_|\_

Donje linije oznacavaju prazna polja koja kasnije treba da popuni vozač kad završi rutu

* + Primer: Beograd-Nis|Pesak|3|\_|\_
* Sve rute
  + Ime fajla: Rute.txt
  + Format fajla: Polazni grad-Krajnji grad|Tovar|ID vozača koji prevozi tu rutu|Predjeni kilometri|Putni troškovi (RSD)
  + Primer: Beograd-Nis|Pesak|3|200|10000
* Koordinate gradova
  + Ime fajla: MapCoords.txt
  + Format fajla: Grad|X koordinata|Y koordinata
  + Primer: Beograd|234|273
* Zapošljeni
  + Ime fajla: Zaposljeni.txt
  + Format fajla: ID|Rank (0 je vozač, 1 je menadžer)|Ime I prezime|JMBG|Broj telefona|Korisničko ime|Lozinka
  + Primer: 2|1|Mirko Mirkovic|0312993723153|0631526724|test|sifra

Program se pokreće u terminalu tako što se promeni aktivni direktorijum da bude direktorijum u kome se nalazi **main.py** i kucanjem komande:

> python3 main.py

2. Opis programa

Program je podeljen u više fajlova:

* main.py
* Menadžer.py
* Vozac.py
* Map.png (mapa Republike Srbije koja se koristi u jednom od grafova) (Slika 3)
* folder Data u kome se nalaze baze podataka koje smo u uvodu spomenuli

Kroz kod koji sledi primećivaće se delovi koji izgledaju ovako:

\033[0;36;40m

Ovo su takozvani **ANSI escape kodovi** i njihova svrha je bojenje teksta u terminalu. Za više informacija o njima možete pogledati na veb sajtu **Ozzmaker** i **Wikipedia**, koji se u literaturi vode pod rednim brojevima 3 i 4.

Boje koje će biti korišćene u programu su:

* Crvena (\033[0;31;40m)
* Plava (\033[0;36;40m)
* Bela (\033[0;37;40m)
* Ljubičasta (\033[0;35;40m)

Takodje, ako na dnu koda primetite tri tačke (...), to znači da se kod nastavlja, ali se ne vidi u tom isečku.

## Glavni program

Glavni fajl je kičma programa. Fajl se zove **main.py** i on se pokreće komandom python3 main.py kada želimo da pokrenemo program. On u sebi sadrži nekoliko funkcija odgovornih za prijavljivanje korisnika i pokretanje glavnih funkcija.

### **Importovanje biblioteka i pratećih fajlova**

from Vozac import Vozac

from Menadzer import Menadzer

import timeit

import os

Program koristi nekoliko biblioteka. Praći fajlovi **Vozac** i **Menadzer** su prateči fajlovi u kojima su klase i one sadrže funkcije vezane za sam program. A biblioteke **timeit** i **os** se koriste za merenje vremena koje je korisnik proveo u programu i ćišćenje terminala.

### **Prijavljivanje korisnika - login() funkcija**

def login(username, passwd):

users = []

with open("Data/Zaposljeni.txt") as fp:

lines = fp.readlines()

for line in lines:

l = line.strip().split('|')

users.append({'id': l[0], 'rank': l[1], 'k\_ime': l[5], 'k\_lozinka': l[6]})

for user in users:

if username == user['k\_ime'] and passwd == user['k\_lozinka']:

return user['id'], user['rank']

return None, None

Ovo je funkcija koja kao argumente učitava **korisničko ime** i **lozinku**, zatim učitava ID, korisničko ime i lozinku svih zaposlenih iz fajla Data/Zaposljeni.txt i smesta ih u listu users. Nakon toga, prolazi kroz listu i uporedjuje korisničko ime i lozinku koje smo učitali kao argumente sa korisničkim imenom i lozinkom svakog zaposlenog iz liste, ukoliko su korisničko ime i lozinka tačni i program uspe da nadje zaposlenog, funkcija vraća ID, rank zaposlenog, ali ukoliko program ne uspe da nadje zaposlenog, vraća None, None.

### **Pokretanje programa - main() funkcija**

def main():

id = None

while id == None:

username = input('\033[0;36;40m Korisnicko ime: \033[0;37;40m')

password = input('\033[0;36;40m Unesite lozinku: \033[0;37;40m')

id, rank = login(username, password)

if id == None:

print("\033[0;31;40m Korisnicko ime ili lozinka nisu ispravni. Pokusajte ponovo.\033[0;37;40m ")

...

Za početak smo definisali promenjivu id i stavili da bude vrednosti None. Zatim smo pokrenuli jednu while petlju koja će se izvršavati sve dok id ne promeni vrednost. U toj petlji traži se od korsnika da unese korisničko ime i lozinku i ti unosi se smeštaju u promenljive username i password. Zatim se pozove funkcija login i kao argumenti se stave promeljive username i password. Pošto ta funkcija vraća podatke u formatu id,rank, mi te vraćene vrednosti smestimo u promeljive id i rank. Nakon toga, proverimo da li je vrednost promenljive id, None i ako jeste ispišemo poruku kojom obaveštavamo korisnika da je uneo korisničko ime ili lozinka koje je uneo nisu tačni, ali ako su uneti podaci tačni, while petlja se završava pošto je promenljiva id promenila vrednost.

...

os.system('clear')

print('''

\033[0;31;40m \_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_ \_\_\_\_ \_\_\_\_ \_\_ \_\_ \_ \_\_\_\_ \_\_\_\_ \_\_ \_\_\_\_ \_\_\_\_

\033[0;31;40m/ \_\_\_)( \_\_)( \_ \( \_ \ / \(\_ \_)( \_ \ / \_\ ( ( \/ \_\_\_)( \_ \ / \( \_ \(\_ \_)

\033[0;36;40m\\_\_\_ \ ) \_) ) / ) \_ (( O ) )( ) // \/ /\\_\_\_ \ ) \_\_/( O )) / )(

\033[0;37;40m(\_\_\_\_/(\_\_\_\_)(\_\_\\_)(\_\_\_\_/ \\_\_/ (\_\_) (\_\_\\_)\\_/\\_/\\_)\_\_)(\_\_\_\_/(\_\_) \\_\_/(\_\_\\_) (\_\_)

Dobrodosli u SerboTransport menadzment program. Za listu komanda napisite `\033[0;36;40mpomoc\033[0;37;40m`

''')

...

Kada se while petlja završi, program očisti terminal, ispisuje logo kompanije (slika 1) i izvršava sledeći deo koda.

...

start\_time = timeit.default\_timer()

if rank == '0':

Vozac(id).start()

elif rank == '1':

Menadzer(id).start()

else:

print('Nepoznat rank!')

stop\_time = timeit.default\_timer()

print(f'''\033[0;36;40m

Sesija gotova!

Vreme koriscenja: \033[0;35;40m{(stop\_time-start\_time):.2f}\033[0;36;40m s

Dovidjenja!\033[0;37;40m

''')

...

Ukoliko se korisnik uspešno prijavio, program je dobio korisnikom id i rank. Pokrene funkciju timeit.default\_timer()koja zabeleži broj sekundi od trenutka pokretanja programa. Nakon toga se nalazi jedno grananje koje u zavisnosti od ranka pokreće kreiramo objekat odgovarajuće klase kome prosledjujemo id i pozivamo funkciju start ili izbacuje grešku ukoliko id nije ispravan. Funkcije za pokretanje programa se nalazu u klasama Vozac i Menadzer (njihove funkcije cemo pogledati kasnije). Nakon završetka programa još jednom zovemo funkciju timeit.default\_timer()i nadjemo razliku izmedju trenutnog vremena i vremena sa početka programa, tu razliku zajedno sa porukom ispišemo i pozdravimo korisnika.

...

if \_\_name\_\_ == '\_\_main\_\_':

main()

Na samom kraju proverimo da li promenjiva \_\_name\_\_ ima vrednost \_\_main\_\_, sto označava da je taj fajl glavni fajl programa. Ukoliko jeste pokrećemo funkciju main.

## Vozač i njegove funkcije

Vozač ima svoju klasu pod imenom Vozac i nalazi se u fajlu Vozac.py. Ona u sebi sadrži funkcije glavne funkcije programa odgovorne za komande i interakciju korisnika sa programom. Kada se korisnik prijavi u svoj nalog, on dobija id, taj id se prosledjuje konstruktorskoj funkciji.

### **Kontruktorska \_\_init\_\_() funkcija**

def \_\_init\_\_(self, id):

self.id = id

self.ime = ''

self.prezime = ''

self.jmbg = ''

self.tel = ''

self.getInfo(self.id)

Kontruktorska funkcija se zove \_\_init\_\_ i kao argumente traži self i id. Promenljiva self predstavlja objekat klase i preko nje mi možemo pristupiti promenljivim i metodama te klase, ona se koristi kao argument u svakoj metodi unutar klase. Drugi argument predstavlja **id vozača** koji smo dobili od glavnog fajla **main.py** kada smo kreirali objekat klase Vozac i kao argument stavili id koji je dobio vrednost kada se vozač prijavio. Unutar konstruktorske funkcije definisemo promenljive id, ime, prezime, jmbg, tel i pozovemo metodu getInfo kojoj kao argument prosledimo id.

### **getInfo() funkcija**

def getInfo(self, id):

users = []

with open("Data/Zaposljeni.txt") as fp:

lines = fp.readlines()

for line in lines:

l = line.strip().split('|')

users.append({'id': l[0], 'rank': l[1], 'ime': l[2], 'jmbg': l[3], 'tel': l[4]})

for user in users:

if id == user['id']:

self.ime = user['ime'].split()[0]

self.prezime = user['ime'].split()[1]

self.jmbg = user['jmbg']

self.tel = user['tel']

Ovu funkciju poziva kontruktorska funkcija i kao argument dobija id. Služi da iz fajla Data/Zaposljeni.txt izvuce sve podatke o vozaču i smesti ih u promenljive. Ona otvara fajl i prolazi kroz sve linije, zatim pravi rečnik za svakog zaposlenog i smešta ga u listu users. Nakon što se lista popuni svim zaposlenima, pokrene se for petlja koja prolazi kroz svaki element niza i uporedjuje id svakog vozača iz liste sa id-jem koji je dobijen kao argument.

### **start() funkcija**

def start(self):

komanda = input('> ')

while komanda != 'izadji':

if komanda == 'pomoc':

self.pomoc()

elif komanda == 'arute':

self.aRute(self.nadjiAktivneRute(self.id))

elif komanda == 'rute':

self.sveRute(self.id)

elif komanda == 'zavrsi':

self.zavrsi(self.nadjiAktivneRute(self.id))

elif komanda == '':

pass

else:

print('\n\033[1;31;40m Nepoznata komanda.\n Napisite `pomoc` da vidite listu komandi.\033[0;37;40m\n')

komanda = input('> ')

Ovu funkciju zovemo iz glavnog fajla **main.py**. Ova funkcija nam služi za kao **elumator terminala** i u zavisnosti koju komandu korisnik upiše pokreće se odgovarajuća funkcija. Kada se ova funkcija pozove, korisniku se traži unos komande i korisnički unos se smešta u promenljivu komanda. Nakon toga pokreće se while petlja koja se izvršava sve dok komanda ne bude **"izadji"**. U while petlji se nalazi višestruko grananje za svaku moguću komandu, a ako se komanda koju je korisnik uneo ne nalazi medju mogućim komandama program ispisuje:

Nepoznata komanda.

Napisite `pomoc` da vidite listu komandi.

Na samom kraju se nalazi još jedan unos koji će se pokretati nakon svake unete komande.

### **pomoc() funkcija**

def pomoc(self):

pomoc\_vozac = '''\033[0;36;40m

pomoc .............. Ispise lisu komandi

arute .............. Pregled svih svojih aktivnih ruta

rute .............. Pregled svih svojih ruta

zavrsi .............. Zavrsite svoju aktivnu rutu

izadji .............. Zavrsite sesiju i izadjite\033[0;37;40m

'''

print(pomoc\_vozac)

Pošto postoji više komandi koje korisnik može da koristi, bilo je neophodno da se napravi funkcija koja ispisuje sve moguće komande i njihovu svrhu. U ovoj funkciji, baš ta funkionalnost je dodata. Vidimo jedan višelinijski string sa svim komandama dostupnim korisniku i funkciju print() koja to ispisuje.

### **prikaziGraf() funkcija**

def prikaziGraf(self, gradovi):

fig = plt.figure('Posecenost gradova')

ax = fig.add\_axes([0,0,1,1])

ax.axis('equal')

imena = []

posecenost = []

for grad in gradovi:

imena.append(grad['ime'])

posecenost.append(grad['posecenost'])

ax.pie(posecenost, labels = imena, autopct='%1.2f%%')

ax.set\_title('Posecenost gradova')

plt.show()

Ova funkcija služi da prikaže graf. Biblioteka korišćena za izradu grafa je **Matplotlib** i ona je importovana na samom vrhu programa linijom:

import matplotlib.pyplot as plt

Ova biblioteka u sebi ima metode za izradu različitih stilova grafova, u ovom primeru radi se o **Pie** stilu (Slika 2). Više o biblioteci možete da saznate na Matplotlib veb sajtu (Literatura, redni broj 6). Kao argument, funkciji prosledimo listu gradovi. Kreiramo figuru sa imenom **"Posecenost gradova"**, i dodamo joj ose. Zatim definišemo dve prazne liste imena i posecenost i pokrenemo for petlju koja prolazi kroz svaki grad iz liste gradovi i razdvaja imena i posećenost gradova u prethodno definisane liste. Nakon što smo razdvojili podatke, možemo iskoristiti pie() funkciju iz **Maplotlib** biblioteke i kreirati Pie graf. Za argumente funkcije stavimo liste posecenost i imena, i zaokružimo broj decimala na 2. Na kraju stavimo naslov grafa i prikazemo graf funkcijom show().

### **nadjiAktivneRute() i aRuta() funkcije**

def nadjiAktivneRute(self, id):

sveAktivneRute = []

aRute = []

with open("Data/AktivneRute.txt") as fp:

lines = fp.readlines()

for line in lines:

l = line.strip().split('|')

sveAktivneRute.append({'ruta': l[0], 'tovar': l[1], 'vozac\_id': l[2]})

for ruta in sveAktivneRute:

if id == ruta['vozac\_id']:

aRute.append(ruta)

return aRute

def aRute(self, aRute):

for ruta in aRute:

print('''

\033[0;36;40mRuta:\033[0;37;40m {}

\033[0;36;40mTovar:\033[0;37;40m {}

\033[0;36;40mVozac:\033[0;37;40m {}

\033[0;36;40mKilometraza:\033[0;37;40m ...

\033[0;36;40mPutni troskovi:\033[0;37;40m ...

'''.format(ruta['ruta'], ruta['tovar'], self.ime + ' ' + self.prezime))

Ove funkcije sam grupisao pod jednom podsekcijom zato sto rade zajedno ka istom cilju, da sakupe i prikazu aktivne rute. Funkcija nadjiAktivneRute() radi na istom principu kao i funkcija getInfo() (Strana 9). Prvo su definisane liste sveAktivneRute i aRute, nakon toga iz fajla se učitavaju sve aktivne rute svih vozača, generišu se rečnici sa podacima i ti rečnici se smeštaju u listu sveAktivneRute. Zatim se jednom for petljom, prolazi kroz popunjenu listu i uporedjuju se id vozača iz liste i id koji smo dobili kao argument, ukoliko se id-jevi poklapaju, ruta iz liste sveAktivneRute se dodaje u listu aRute. Na kraju funkcija vrati listu aRute.

Funkcija aRute(), za argument uzima listu rečnika. Ti rečnici su popunjeni odgovarajućim podacima o rutama. U funkciji se nalazi for petlja koja prolazi kroz tu listu rečnika i podatke formatira i ispisuje.

### **rute() funkcija**

def sveRute(self, id):

sveRute = []

vozaceveRute = []

gradovi = []

brojac = 0

with open("Data/Rute.txt") as fp:

lines = fp.readlines()

for line in lines:

l = line.strip().split('|')

sveRute.append({'ruta': l[0], 'tovar': l[1], 'vozac\_id': l[2], 'kilometraza': l[3], 'troskovi': l[4]})

...

Ova funkcija je velika pa će kod kod biti podeljen u više segmenata. U prvom segmentu vidimo da funkcija kao argument uzima id vozača. U funkciji prvo definišemo liste sveRute, vozaceveRute i gradovi, i promenljivu brojac. Zatim otvaramo fajl Data/Rute.txt i iz njega čitamo sve linije. Jednom for petljom prodjemo kroz sve rute i njihove podatke smestimo u rečnik. I taj rečnik upisujemo u listu sveRute.

...

for ruta in sveRute:

if id == ruta['vozac\_id']:

vozaceveRute.append(ruta)

brojac += 1

grad1, grad2 = {'ime': ruta['ruta'].split('-')[0], 'posecenost': 1}, {'ime': ruta['ruta'].split('-')[1], 'posecenost': 1}

if gradovi == []:

gradovi.append(grad1)

gradovi.append(grad2)

else:

\_grad1 = False

\_grad2 = False

for grad in gradovi:

if grad1['ime'] == grad['ime']:

grad['posecenost'] += 1

\_grad1 = True

elif grad2['ime'] == grad['ime']:

grad['posecenost'] += 1

\_grad2 = True

if \_grad1 == False:

gradovi.append(grad1)

elif \_grad2 == False:

gradovi.append(grad2)

...

U ovom delu koda kreiramo još jednu for petlju kojom prolazimo kroz listu sveRute i uporedjujemo id vozača rute i id koji smo dobili kao argument. Ako se id-jevi poklapaju rutu upisujemo u lisu vozaceveRute i brojac povećavamo za 1. Sada možemo da spremimo podatke za graf (Ako se sećate funkcije prikaziGraf(), ona je kao argument trazila listu gradovi, u ovom delu koda mi pravimo tu listu). Prvo, rutu u formatu Pozalzni grad-Krajnji grad podelimo u dva rečnika grad1 i grad2 i njihovu posećenost stavimo da bude 1. Nakon toga proverimo da li je lista gradovi prazna, ako jeste samo ubacimo ova dva grada. Ako nije prazna proverimo da li je neki od ovih gradova već u listi, ako jeste njegovu posećenost povećamo za 1, a ako nije upišemo ga u listu.

print(f' Izvozili ste {str(brojac)} ruta:')

br = 1

for ruta in vozaceveRute:

print(''' \033[0;36;40m{})\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {}rsd'''.format(br, ruta['ruta'], ruta['tovar'], ruta['kilometraza'], ruta['troskovi']))

br += 1

odgovor = input(' Da li zelite videti graf najposecenijih gradova? (da/ne): ')

if odgovor == 'da':

self.prikaziGraf(gradovi)

Poslednji deo koda ove funkcije je i najkraći. U njemu ispišemo koliko ruta ima vozač, a zatim formatiramo i ispišemo sve njegove rute. Nakon ispisanih ruta, dobijamo pitanje, da li želimo da vidimo graf. Ukoliko je odgovor "ne" funkcija se završi, ali ako je odgovor "da", pozovemo funkciju prikaziGraf() i kao argument stavimo listu gradovi.

### **zavrsi() funkcija**

S'obzirom da je i ova funkcija dosta velika biće podeljena na više delova koda.

def zavrsi(self, aRute):

print(' Vase aktivne rute:')

br = 1

for ruta in aRute:

print(''' \033[0;36;40m{})\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {}'''.format(br, ruta['ruta'], ruta['tovar']))

br += 1

ruta = input(' Koju rutu zelite zavrsiti (id rute): ')

if ruta.isdigit():

ruta = int(ruta) - 1

else:

print(" \n\033[1;31;40mID rute mora biti broj!\033[0;37;40m")

return

...

U prvom delu funkcije, ispisuju se sve aktivne rute i trazi se od korisnika da unese redni broj rute. Proverava se da li je uneta vrednost broj i ako jeste izvršavanje funkcije se nastavlje, a ko nije funkcija se prekida i ispisuje se odgovarajuća poruka korisniku.

...

kilometraza = input(' Koliko ste kilometara presli: ')

troskovi = input(' Koliko su izasli putni troskovi (gorivo, putarina...): ')

print('''

\033[0;36;40mRuta:\033[0;37;40m {}

\033[0;36;40mTovar:\033[0;37;40m {}

\033[0;36;40mVozac:\033[0;37;40m {}

\033[0;36;40mKilometraza:\033[0;37;40m {} km

\033[0;36;40mPutni troskovi:\033[0;37;40m {} rsd

'''.format(aRute[ruta]['ruta'], aRute[ruta]['tovar'], self.ime + ' ' + self.prezime, kilometraza, troskovi))

potvrda = input('Da li su podaci ispravni? (da/ne): ')

...

U drugom delu, od korisnika se traži da unese koliko je kilometara prešao i taj podatak se beleži u promenljivu kilometraza i koliko su izneli putni troškovi u dinarima i taj podatak se beleži u promenljivu troskovi. Nakon toga, program ispisuje podatke koje će uneti aktivnu rutu i pita korisnika da li su podaci ispravni.

if potvrda == 'da':

with open('Data/Rute.txt', 'a') as file:

line = aRute[ruta]['ruta'] + '|' + aRute[ruta]['tovar'] + '|' + self.id + '|' + kilometraza + '|' + troskovi

file.write('\n'+line)

a\_file = open("Data/AktivneRute.txt", "r")

lines = a\_file.readlines()

a\_file.close()

lineToDelete = aRute[ruta]['ruta'] + '|' + aRute[ruta]['tovar'] + '|' + self.id + '|\_|\_'

newFile = ''

for line in lines:

if line.strip("\n") != lineToDelete and line.strip("\n") != "":

newFile += line

if(newFile[-1:] == '\n'):

newFile = newFile[:-1]

with open('Data/AktivneRute.txt', 'w') as file:

file.write(newFile)

U poslednjem delu koda, vidimo da ako je korisnik proverio podatke i oni su ispravni, program otvara fajl Data/Rute.txt i u njega dodaje rutu, a iz fajla Data/AktivneRute.txt briše rutu na sledeći način. Prvo otvara fajl Data/AktivneRute.txt , učitava sve njegove linije i smešta ih u listu lines. Nakon toga generiše liniju koju treba da izbriše is fajla Data/AktivneRute.txt i naziva je lineToDelete i definiše novi string sa nazivom newFile. Zatim, jedna for petlja prodje kroz sve linije iz liste lines i proverava da li se ta linija poklapa sa lineToDelete. Ukoliko se ne poklapaju, ta linija se dodaje u string newFile. Na kraju se proveri da li se newFile završava sa "\n" i ako se završava, taj nastavak se briše. Ostalo je samo da se string newFile zapiše u fajl Data/AktivneRute.txt, nakon što se to uradi funkcija se prekida.

## Menadžer i njegove funkcije

Menadžer, kao i vozač, ima svoju klasu zvanu Menadzer, u njoj se nalaze sve funkcionalnosti dostupne menadžeru. Ova klasa deli neke metode sa klasom vozača i to su: \_\_init\_\_(), getInfo(), pomoc() i start(). Ove funkcije rade na istom principu kao i u klasi Vozac, razlika je jedino u funkcijama pomoc() i start() gde su zamenjane komande i funkcije koje komande pozivaju. Zbog tog razloga, ove funkcije neće biti opet detaljno opisane.

### **nadjiVozace() funkcija**

def nadjiVozace(self):

users = []

drivers = []

with open("Data/Zaposljeni.txt") as fp:

lines = fp.readlines()

for line in lines:

l = line.strip().split('|')

users.append({'id': l[0], 'rank': l[1], 'ime': l[2], 'jmbg': l[3], 'tel': l[4]})

for user in users:

if user['rank'] == '0':

drivers.append(user)

return drivers

Ova funkcija kreira listu vozača sa svim bitnim informacijama. Ona radi tako što se na početku definišu liste users i drivers, zatim se učita fajl Data/Zaposljeni.txt i za svaku liniju iz tog fajla kreira se rečnik sa informacijama o vozaču. Nakon toga se u jednoj for petlji prodje kroz svaki element liste users i ukoliko se rank zapošljenog poklapa sa 0, taj element se dodaje u listu drivers. Na kraju funkcija samo vrati listu drivers.

### **prikaziGraf() funkcija**

Ova funkcija se razlikuje od funkcije prikaziGraf() iz klase Vozac po tome što prikazuje drugu vrstu grafa. U ovoj funkciji se takodje koristi biblioteka **Matplotlib** i ona je morala da bude importovana na samom vrhu fajla Menadzer.py linijom koda:

import matplotlib.pyplot as plt

Graf koji prikazuje ova funkcija se vidi na slici 3.

def prikaziGraf(self, gradovi):

imena = []

posecenost = []

x = []

y = []

colors = []

for grad in gradovi:

imena.append(grad['ime'])

posecenost.append(int(grad['posecenost'])\*100)

with open("Data/MapCoords.txt") as fp:

lines = fp.readlines()

for line in lines:

l = line.strip().split('|')

if grad['ime'] == l[0]:

x.append(int(l[1]))

y.append(int(l[2]))

im = plt.imread("map.png")

fig, ax = plt.subplots()

im = ax.imshow(im)

ax.set\_title('Posecenost gradova')

for c in posecenost:

colors.append(c/100)

plt.scatter(x, y, c=colors, s=posecenost, alpha=0.7, cmap='cividis')

plt.colorbar()

plt.show()

Za početak definišemo liste imena, posecenost, x, y i colors. Jednom for petljom prodjemo kroz svaki element iz liste gradovi koju smo primili kao argument, iz svakog elementa izvučemo ime grada i broj koliko je puta taj grad posetio odredjeni vozač. Sada kada smo razdvojili ime grada i broj poseta, moramo pronaći koordinate gradova na grafu. To radimo tako što otvorimo fajl Data/MapCoords.txt i iz njega pročitamo sve linije, zatim jednom for petljom prodjemo kroz svaku liniju i uporedimo ime grada koje smo izdvojili iz liste gradovi sa imenom grada iz Data/MapCoords.txt fajla. Ukoliko se imena poklapaju koordinate x i y će biti dodate u odgovarajuće liste. Sada kada smo učitali sve potrebne podatke, možemo generisati graf. Prvo, učitaćemo pozadinsku sliku za graf (mapa Republike Srbije, Slika 3) i generisaćemo graf. Zatim, napravićemo jednu for petlju koja će proći kroz sve elemente liste posecenost i vrednost element / 100 smestiće u listu colors. Na kraju, iskoristićemo metode: scatter() da definišemo koji graf želimo da koristimo, colorbar() da prikažemo koje vrednosti odgovaraju kojoj boji i show() da prikažemo graf.

### **vozac() funkcija**

def vozac(self):

vozaci = self.nadjiVozace()

vozaciId = []

print(' \033[0;36;40mSvi vozaci:\033[0;37;40m')

for vozac in vozaci:

print(''' \033[0;36;40m{})\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {}'''.format(vozac['id'], vozac['ime'], vozac['jmbg'], vozac['tel']))

vozaciId.append(vozac['id'])

id = input('\n \033[0;36;40mIzaberite vozaca (njegov id):\033[0;37;40m ')

while id not in vozaciId:

id = input('\n \033[0;36;40mIzaberite vozaca (njegov id):\033[0;37;40m ')

for vozac in vozaci:

if vozac['id'] == id:

datumRodj = vozac['jmbg'][:2] + '.' + vozac['jmbg'][2:4] + '.' + ('20' if vozac['jmbg'][4:5] == '0' else '19') + vozac['jmbg'][4:6]

info = '''\033[0;36;40m

Podaci o vozacu:

\033[0;36;40mIme vozaca:\033[0;37;40m {}

\033[0;36;40mDatum rodjenja:\033[0;37;40m {}

\033[0;36;40mJMBG:\033[0;37;40m {}

\033[0;36;40mBroj telefona:\033[0;37;40m {}

'''.format(vozac['ime'], datumRodj, vozac['jmbg'], vozac['tel'])

print(info)

Ova funkcija služi da prikaže sve dostupne podatke o vozaču. Za početak se korisniku prikaže lista svih vozača, zatim se od korisnika traži da unese id vozača čije podatke želi da vidi. Kada korisnik izabere jedan od mogućih id-jeva, pokrene se jedna for petlja koja pronadje vozača i ispiše sve njegove podatke.

### **rute() funkcija**

def rute(self):

vozaci = self.nadjiVozace()

vozaciId = []

print(' \033[0;36;40mSvi vozaci:\033[0;37;40m')

for vozac in vozaci:

print(''' \033[0;36;40m{})\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {}'''.format(vozac['id'], vozac['ime'], vozac['jmbg'], vozac['tel']))

vozaciId.append(vozac['id'])

id = input('\n \033[0;36;40mIzaberite vozaca (njegov id):\033[0;37;40m ')

while id not in vozaciId:

id = input('\n \033[0;36;40mIzaberite vozaca (njegov id):\033[0;37;40m ')

gradovi = []

sveAktivneRute = []

aRute = []

brojacA = 0

with open("Data/AktivneRute.txt") as fp:

lines = fp.readlines()

for line in lines:

l = line.strip().split('|')

sveAktivneRute.append({'ruta': l[0], 'tovar': l[1], 'vozac\_id': l[2]})

for ruta in sveAktivneRute:

if id == ruta['vozac\_id']:

aRute.append(ruta)

brojacA += 1

grad1, grad2 = {'ime': ruta['ruta'].split('-')[0], 'posecenost': 1}, {'ime': ruta['ruta'].split('-')[1], 'posecenost': 1}

if gradovi == []:

gradovi.append(grad1)

gradovi.append(grad2)

else:

\_grad1 = False

\_grad2 = False

for grad in gradovi:

if grad1['ime'] == grad['ime']:

grad['posecenost'] += 1

\_grad1 = True

elif grad2['ime'] == grad['ime']:

grad['posecenost'] += 1

\_grad2 = True

if \_grad1 == False:

gradovi.append(grad1)

elif \_grad2 == False:

gradovi.append(grad2)

sveRute = []

vozaceveRute = []

brojac = 0

with open("Data/Rute.txt") as fp:

lines = fp.readlines()

for line in lines:

l = line.strip().split('|')

sveRute.append({'ruta': l[0], 'tovar': l[1], 'vozac\_id': l[2], 'kilometraza': l[3], 'troskovi': l[4]})

for ruta in sveRute:

if id == ruta['vozac\_id']:

vozaceveRute.append(ruta)

brojac += 1

grad1, grad2 = {'ime': ruta['ruta'].split('-')[0], 'posecenost': 1}, {'ime': ruta['ruta'].split('-')[1], 'posecenost': 1}

if gradovi == []:

gradovi.append(grad1)

gradovi.append(grad2)

else:

\_grad1 = False

\_grad2 = False

for grad in gradovi:

if grad1['ime'] == grad['ime']:

grad['posecenost'] += 1

\_grad1 = True

elif grad2['ime'] == grad['ime']:

grad['posecenost'] += 1

\_grad2 = True

if \_grad1 == False:

gradovi.append(grad1)

elif \_grad2 == False:

gradovi.append(grad2)

print(f'\n \033[0;36;40mVozac ima {str(brojac)} zavrsenih ruta:\033[0;37;40m')

br = 1

for ruta in vozaceveRute:

print(''' \033[0;36;40m{})\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {}rsd'''.format(br, ruta['ruta'], ruta['tovar'], ruta['kilometraza'], ruta['troskovi']))

br += 1

print(f'\n \033[0;36;40mVozac ima {str(brojacA)} aktivnih ruta:\033[0;37;40m')

br = 1

for ruta in aRute:

print(''' \033[0;36;40m{})\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {}'''.format(br, ruta['ruta'], ruta['tovar']))

br += 1

odgovor = input(' Da li zelite videti graf najposecenijih gradova? (da/ne): ')

if odgovor == 'da':

self.prikaziGraf(gradovi)

Ovo je velika funkcija koja služi da prikaže sve gotove i aktivne rute nekog vozača. Radi tako što se na početku prikažu svi vozači i od korisnika se traži da unese id vozača čije rute želi da vidi. Nakon toga učitaju se fajlovi Data/AktivneRute.txt i Data/Rute.txt, zatim se iz oba fajla prikažu sve rute koje je vozio ili koje će voziti vozač čiji je id uneo korsnik. Nakon toga, iz oba fajla se podaci o datim rutama skladište u listu gradovi, čime su spremni za prikaz na grafu. Na samom kraju ispiše se koliko ruta je vozač izvozao i postavi se pitanje korisniku da li želi da vidi graf. Ukoliko je odgovor "da", pozove se funkcija prikaziGraf() i kao argument stavi se lista gradovi.

### **nruta() funkcija**

def nruta(self):

vozaci = self.nadjiVozace()

vozaciId = []

gradovi = []

print(' \033[0;36;40mSvi vozaci:\033[0;37;40m')

for vozac in vozaci:

print(''' \033[0;36;40m{})\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {}'''.format(vozac['id'], vozac['ime'], vozac['jmbg'], vozac['tel']))

vozaciId.append(vozac['id'])

id = input('\n \033[0;36;40mIzaberite vozaca (njegov id):\033[0;37;40m ')

while id not in vozaciId:

id = input('\n \033[0;36;40mIzaberite vozaca (njegov id):\033[0;37;40m ')

with open("Data/MapCoords.txt") as fp:

lines = fp.readlines()

for line in lines:

l = line.strip().split('|')

gradovi.append(l[0])

for grad in gradovi:

print(''' \033[0;36;40m{})\033[0;37;40m {}'''.format(gradovi.index(grad)+1, grad))

grad1 = eval(input('\n \033[0;36;40mIzaberite pocetnu lokaciju (grad id):\033[0;37;40m '))-1

for grad in gradovi:

if grad != gradovi[grad1]:

print(''' \033[0;36;40m{})\033[0;37;40m {}'''.format(gradovi.index(grad)+1, grad))

grad2 = eval(input('\n \033[0;36;40mIzaberite krajnju lokaciju (grad id):\033[0;37;40m '))-1

tovar = input('\n \033[0;36;40mTovar:\033[0;37;40m ')

for vozac in vozaci:

if vozac['id'] == id:

potvrda\_poruka = '''\033[0;36;40m

\033[0;36;40mVozac:\033[0;37;40m {}

\033[0;36;40mPocetna lokacija:\033[0;37;40m {}

\033[0;36;40mKrajnja lokacija:\033[0;37;40m {}

\033[0;36;40mTovar:\033[0;37;40m {}

'''.format(vozac['ime'], gradovi[grad1], gradovi[grad2], tovar)

print(potvrda\_poruka)

potvrda = input('\n \033[0;36;40mPotvrdi novu rutu(da/ne):\033[0;37;40m ')

if potvrda == 'da':

with open('Data/AktivneRute.txt', 'a') as file:

line = gradovi[grad1] + '-' + gradovi[grad2] + '|' + tovar + '|' + id + '|\_|\_'

file.write('\n'+line)

Ova funkcija služi za kreiranje novih ruta za vozače. Radi tako što se na početku korisniku prikaže lista svih vozača i traži se od korisnika da unese id vozača za kog želi da napravi novu rutu. Kada korisnik izabere id vozača, korisnik treba da izabere **početnu i krajnju lokaciju** kao i **tovar**. Nakon što su svi podaci uneti, korisnik dobija još jednu potvrdnu poruku gde su svi podaci pregledno ispisani i od korisnika se traži potvrda da su podaci ispravni. Ukoliko su svi podaci ispravni, oni će biti dodati u fajl Data/AktivneRute.txt.

### **posao() funkcija**

def posao(self):

vozaci = self.nadjiVozace()

vozaciId = []

opcije = ['zaposli', 'otpusti']

izbor = input('\n \033[0;36;40mIzaberite opciju (zaposli/otpusti):\033[0;37;40m ')

while(izbor not in opcije):

izbor = input('\n \033[0;36;40mIzaberite opciju (zaposli/otpusti):\033[0;37;40m ')

if (izbor == 'otpusti'):

print(' \033[0;36;40mSvi vozaci:\033[0;37;40m')

for vozac in vozaci:

print(''' \033[0;36;40m{})\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {} \033[0;36;40m|\033[0;37;40m {}'''.format(vozac['id'], vozac['ime'], vozac['jmbg'], vozac['tel']))

vozaciId.append(vozac['id'])

id = input('\n \033[0;36;40mIzaberite vozaca (njegov id):\033[0;37;40m ')

while id not in vozaciId:

id = input('\n \033[0;36;40mIzaberite vozaca (njegov id):\033[0;37;40m ')

odgovor = input(' \033[0;36;40mDa li sigurni da selite da otpustite vozaca? (da/ne):\033[0;37;40m ')

if odgovor == 'da':

for vozac in vozaci:

if vozac['id'] == id:

ime = vozac['ime']

a\_file = open("Data/Zaposljeni.txt", "r")

lines = a\_file.readlines()

a\_file.close()

lineToDelete = id + '|0|' + ime

newFile = ''

for line in lines:

if lineToDelete not in line.strip("\n") and line.strip("\n") != "":

newFile += line

if(newFile[-1:] == '\n'):

newFile = newFile[:-1]

with open('Data/Zaposljeni.txt', 'w') as file:

file.write(newFile)

else:

ime = input('\n \033[0;36;40mIme i prezime:\033[0;37;40m ')

jmbg = input('\n \033[0;36;40mJMBG (13 cifara):\033[0;37;40m ')

while(len(jmbg) != 13):

jmbg = input('\n \033[0;36;40mJMBG (13 cifara):\033[0;37;40m ')

tel = input('\n \033[0;36;40mBroj telefona (+381 06X XXX XXXX):\033[0;37;40m ')

user = input('\n \033[0;36;40mKorisnicko ime:\033[0;37;40m ')

passwd = input('\n \033[0;36;40mLozinka:\033[0;37;40m ')

brojac = 0

with open("Data/Zaposljeni.txt") as fp:

lines = fp.readlines()

for line in lines:

brojac += 1

with open('Data/Zaposljeni.txt', 'a') as file:

line = str(brojac+1) + '|0|' + ime + '|' + jmbg + '|' + tel + '|' + user + '|' + passwd

file.write('\n'+line)

Ova funkcija nudi menadžeru mogućnost zapošljavanja i otpuštanja vozača. Kada se pozove, korisnik dobija opciju da li želi da **zaposli** ili **otpusti** vozača. Ukoliko se odluči da otpusti, dobiće listu svih vozača i traži se od njega da unese id vozača kog želi da otpusti. Nakon što korisnik unese id vozača, moraće da potvrdi još jednom da li zaista želi da otpusti vozača, ukoliko je siguran, taj vozač će biti uklonjen iz fajla Data/Zaposljeni.txt.

Ukoliko se odluči da zaposli, korisnik će morati da unese: ime i prezime, jmbg, broj telefona, korisničko ime i lozinku. Nakon unosa, korisniku će biti prikazani svi podaci i biće pitan da proveri podatke, ukoliko su svi podaci ispravni, biće dodati u fajl Data/Zaposljeni.txt.

# 3. Zaključak

Ovaj program je napravljen tako da bude interaktivan i fokusiran na korisnika, samim tim, kao što smo videli, primenjeno je dosta boja i formata. Program se može koristiti za širok vid primena sa izmenama koda. Način pisanja programa koji je odvojio sve funkcije zaposlenog u odgovarajuće klase, omogućuje proširanje na više rankova jednostavnim dodavanjem nove klase. To može znatno ubrzati proces proširenja programa i jednostavnosti čitanja koda. Nove funkcionalnosti korisnika se mogu lako dodati kreiranjem nove funkcije unutar odgvarajuće klase.

# 4. Literatura

1. Python Software Foundation, <https://www.python.org/>
2. Python Tutorial, <https://www.w3schools.com/python/default.asp>
3. Ozzmaker, <https://ozzmaker.com/add-colour-to-text-in-python/>
4. Wikipedia, <https://en.wikipedia.org/wiki/ANSI_escape_code>
5. Patorjk, <https://patorjk.com/software/taag/>
6. Matplotlib, <https://matplotlib.org/stable/gallery/>