Primer projektnog zadatka: studentska služba

1 Definicija projektnog zadatka

Implementirati aplikaciju za vođenje evidencije u studentskoj službi. Aplikacija treba da obezbedi rad sa sledećim entitetima:

- *student* opisan *brojem indeksa*, *imenom*, *prezimenom*, *imenom roditelja*, *datumom rođenja*, *JMBG brojem*, *adresom*, *telefonom*, *e-mailom* i *tekućom godinom studija*. Pri tome ne mogu postojati dva studenta sa identičnim brojem indeksa niti JMBG brojem. Fajl sa spiskom studenata se ažurira iz aplikacije, nakon izvršavanja odgovarajućih komandi.
- *Referent* studentske službe opisan *imenom*, *prezimenom*, *korisničkim imenom* i *lozinkom*.

Izbor formata u kome se podaci o studentima i referentima zapisuju u fajlove je deo projektnog zadatka. Nakon pokretanja aplikacije referent ima mogućnost jedino da se prijavi na sistem, pri čemu unosi korisničko ime i lozinku. Nakon uspešnog prijavljivanja na sistem aplikacija referentu treba da omogući sledeće:

- Pronalaženje studenta po broju indeksa. Nakon izvršavanja ove komande prikazuju se podaci o studentu sa datim brojem indeksa.
- Pretraživanje studenata po prezimenu. Nakon izvršavanja svake od ovih komandi prikazuju se podaci o studentima koji zadovoljavaju kriterijum pretraživanja.
- Pregledanje svih studenata sortiranih po prezimenu.
- Izmenu adrese, telefona i e-maila studenta. Pri izvršavanju ove komande, referent unosi broj indeksa studenta čiji podaci se menjaju i potom nove podatke.
- Upis studenata na prvu godinu studija, tj. upis novog studenta koji prethodno nije bio evidentiran u sistemu.
- Grupni upis studenata na narednu godinu studija. Pri tome, unose se brojevi indeksa onih studenata koji upisuju narednu godinu, i podatak o tekućoj godini studija se inkrementira za svakog od njih.

2 Rešenje zadatka

2.1 Rukovanje referentima

Jednog referenta studentske službe ćemo u programu reprezentovati rečnikom, čiji elementi odgovaraju podacima koje evidentiramo za svakog referenta. Sledeći primer predstavlja opis jednog referenta studentske službe.

```
referent1 = {
  'ime': 'Mitar',
  'prezime': 'Miric',
  'username': 'mitar',
  'password': '123'}
```

Kolekciju referenata ćemo predstaviti pomoću liste čiji elementi prestavljaju pojedinačne referente. Prema tome, svaki element liste biće rečnik sa elementima kao iz prethodnog primera. U sledećem primeru definišemo ovakvu listu sa 3 hipotetička elementa.

```
referenti = [referent1, referent2, referent3]
```

Pošto ćemo referente čuvati u tekstualnoj datoteci (po jedan referent u svakom redu), trebalo bi napraviti funkcije za konverziju rečnika koji predstavlja referenta u string i obrnuto. Podaci koji opisuju referenta u jednom redu datoteke biće razdvojeni vertikalnom crtom (`'|'`).

```
def ref2str(ref):
    return '|'.join([ref['ime'], ref['prezime'], ref['username'],
ref['password']])

def str2ref(line):
    if line[-1] == '\n':
        line = line[:-1]
    ime, prezime, username, password = line.split('|')
    ref = {
      'ime': ime,
      'prezime': prezime,
      'username': username,
      'password': password
    }
    return ref
```

Učitavanje referenata iz fajla u globalnu promenljivu `referenti` obavićemo u funkciji `loadRefs` kao u sledećem primeru. U ovoj funkciji if naredbom proveravamo da li smo slučajno iz fajla učitali prazan red (takve redove preskačemo).

```
def loadRefs():
    for line in open('referenti.txt', 'r').readlines():
        if len(line) > 1:
            ref = str2ref(line)
            referenti.append(ref)
```

Operacija prijavljivanja referenta na sistem podrazumena unošenje korisničkog imena i lozinke. Data kombinacija imena i lozinke se traži među svim referentima koje smo prethodno učitali u listu čiji elementi su rečnici. Traženje se obavlja sekvencijalno, sve dok se ne nađe traženi referent ili dok se ne iscrpe svi elementi liste. U sledećem primeru data je funkcija `login` koja obavlja ovu operaciju.

```
def login(username, password):
    for ref in referenti:
        if ref['username'] == username and ref['password'] == password:
            return True
    return False
```

Prethodno prikazane funkcije čine jedan modul koji smo smestili u datoteku `referenti.py`. Inicijalizacija ovog modula sastoji se iz inicijalizacije liste referenata (na praznu listu) i poziva funkcije `loadRefs` radi učitavanja podataka iz datoteke.

```
referenti = []
loadRefs()
```

2.2 Rukovanje studentima

Slično kao u slučaju referenata studentske službe, studente ćemo reprezentovati rečnikom čiji elementi odgovaraju podacima koje evidentiramo za svakog studenta. Sledeći primer predstavlja opis jednog studenta.

```
student1 = {
  'indeks': 'E1',
  'ime': 'Milojko',
  'prezime': 'Pantic',
  'roditelj': 'Mitar',
  'datum': '10.10.1950.',
  'jmbg': '1234567890123',
```

```
  'adresa': 'Marsala Tita 11, Beograd',
  'telefon': '011/1234-567',
  'email': 'milojko@pantic.com',
  'godina': '3'}
```

Lista ovakvih rečnika predstavljaće podatke o svim studentima upisanim na fakultet. U narednom primeru prikazana je takva lista sa tri hipotetička elementa.

```
studenti = [student1, student2, student3]
```

Konverzija rečnika u string i obrnuto – korišćena prilikom učitavanja i snimanja u datoteku – prikazana je sledećim funkcijama.

```
def str2student(line):
    if line[-1] == '\n':
        line = line[:-1]
    indeks, ime, prezime, roditelj, datum, jmbg, adresa, telefon, email, godina = line.split('|')
    stud = {
      'indeks': indeks,
      'ime': ime,
      'prezime': prezime,
      'roditelj': roditelj,
      'datum': datum,
      'jmbg': jmbg,
      'adresa': adresa,
      'telefon': telefon,
      'email': email,
      'godina': godina
    }
    return stud

def student2str(stud):
    return '|'.join([stud['indeks'], stud['ime'], stud['prezime'],
      stud['roditelj'], stud['datum'], stud['jmbg'], stud['adresa'],
      stud['telefon'], stud['email'], stud['godina']])
```

Ispis podataka o studentima na ekranu biće organizovan tabelarno, sa fiksnim širinama pojedinih kolona tabele. U tu svrhu definisane su i funkcije koje vraćaju zaglavlje tabele (`formatHeader`), string reprezentaciju jednog studenta (`formatStudent`), string reprezentaciju date liste studenata (`formatStudents`) i string reprezentaciju liste svih studenata (`formatAllStudents`).

```
def formatHeader():
    return \
      "Indeks  |Ime       |Prezime     |Ime rod.  |Datum rodj.|JMBG        |Email               |God
      "--------+----------+------------+----------+-----------+------------+--------------------+---

def formatStudent(stud):
    return "{0:8}|{1:10}|{2:12}|{3:10}|{4:10}|{5:13}|{6:20}|{7:>6}".format(
      stud['indeks'],
      stud['ime'],
      stud['prezime'],
      stud['roditelj'],
      stud['datum'],
      stud['jmbg'],
      stud['email'],
      stud['godina'])

def formatStudents(studList):
    result = ""
    for stud in studList:
        result += formatStudent(stud) + '\n'
```

```
    return result

def formatAllStudents():
    return formatStudents(studenti)
```

Učitavanje i snimanje liste studenata realizovano je na sličan način kao u prethodnom slučaju, sa referentima studentske službe. U odnosu na taj primer dodata je provera da li datoteka sa spiskom studenata postoji (funkcija checkFile). Ako bi se program pokrenuo bez te datoteke, ona će automatski biti kreirana i njen sadržaj biće inicijalno prazan.

```
from os.path import exists

def loadStudents():
    checkFile()
    for line in open('studenti.txt', 'r').readlines():
        if len(line) > 1:
            stud = str2student(line)
            studenti.append(stud)

def saveStudents():
    file = open('studenti.txt', 'w')
    for stud in studenti:
        file.write(student2str(stud))
        file.write('\n')
    file.close()

def checkFile():
    if not exists('studenti.txt'):
        open('studenti.txt', 'w').close()
```

Pronalaženje studenta sa datim brojem indeksa obavlja funkcija `findStudent` iteracijom kroz listu svih studenata koji su prethodno učitani iz datoteke. Rezultat funkcije je rečnik koji predstavlja traženog studenta (ako je student sa datim brojem indeksa pronađen) ili `None` (ako takav student nije pronađen).

```
def findStudent(indeks):
    for stud in studenti:
        if stud['indeks'] == indeks:
            return stud
    return None
```

Prethodna funkcija pronalazi studenta u listi po broju indeksa. Pretraživanje liste studenata može se obaviti i po nekom drugom obeležju studenta. Naredna funkcija, `searchStudents`, pronalazi sve studente koji imaju obeležje sa imenom jednakim parametru `field`, i vrednošću jednakom parametru `value`. Rezultat funkcije je lista studenata koji zadovoljavaju kriterijum pretrage. Ova lista može biti i prazna ako takvih studenata nema.

Pored toga, funkcija pretpostavlja da je vrednost svih elemenata u rečniku tipa string. Poređenje vrednosti obavlja se uz prethodnu konverziju u velika slova kako bi se razlike između malih i velikih slova ignorisale.

```
def searchStudents(field, value):
    result = []
    for stud in studenti:
        if stud[field].upper() == value.upper():
            result.append(stud)
    return result
```

Dodavanje novog studenta u listu studenata obavlja funkcija `addStudent`. Ažuriranje podataka o studentu, tj. zamenu elementa u listi studenata obavlja funkcija `updateStudent`. Funkcija `advanceStudent` obavlja upis studenta u narednu godinu studija. Tom prilikom se inkrementira obeležje godina. Pošto ovo obeležje sadrži podatak string tipa, radi inkrementiranja je potrebno prvo ga konvertovati u celobrojni podatak. Nakon inkrementiranja, nova vrednost se konvertuje nazad u string, i kao takva upisuje u rečnik.

```
def addStudent(stud):
    studenti.append(stud)

def updateStudent(index, stud):
    studenti[index] = stud

def advanceStudent(stud):
    stud['godina'] = str(int(stud['godina']) + 1)
```

Sortiranje liste studenata zasniva se na ideji da se prvo pronađe najmanji element u listi, i potom taj element i prvi element liste zamene mesta. Na taj način će prvi element liste biti ujedno i najmanji. Ako sada ovaj postupak ponovimo za listu skraćenu za prvi element, dobićemo drugi najmanji element na drugom mestu u listi. Ukoliko ove operacije ponovimo sve dok ne dobijemo listu dužine 1 (u kojoj je najmanji element ujedno i jedini) dobićemo sortiranu listu.

Kao primer ove operacije, posmatrajmo listu brojeva prikazanu na narednoj slici. Najmanji element te liste možemo pronaći sekvencijalnim prolaskom kroz listu od početka do kraja. U ovom primeru to je broj 6, na poziciji 3 (pozicije brojimo počevši od nule).

23	11	15	6	17	28	13

Nakon zamene nultog i trećeg elementa liste, lista će izgledati ovako:

6	11	15	23	17	28	13

Sada možemo ponovo da tražimo najmanji element liste, ali počevši od elementa sa indeksom 1. U ovom primeru to je broj 11 na poziciji 1 – samim tim, zamena mesta elementima nije neophodna jer u bi u ovom slučaju element menjao mesto sa samim sobom.

U narednom koraku tražimo najmanji element liste počevši od pozicije 2. Tada je najmanji element broj 13 koji se nalazi na poziciji 6. Nakon zamene mesta elementima sa pozicijama 2 i 6, redosled elemenata liste biće sledeći (crvenom bojom su označeni elementi koji su zamenili mesta):

6	11	13	23	17	28	15

Naredni ciklus pronalaženja najmanjeg elementa i zamene mesta počinje od indeksa 3. Mesta će zameniti elementi sa indeksima 3 i 6. Stanje nakon zamene izgleda ovako:

6	11	13	15	17	28	23

Naredni ciklus pronalaženja najmanjeg elementa i zamene mesta počinje od indeksa 4. Element sa indeksom 4 je ujedno i najmanji, pa se ne mora premeštati.

Novi ciklus počinje od indeksa 5, i tom prilikom elementi sa indeksima 5 i 6 menjaju mesta. Stanje liste je sada sledeće:

6	11	13	15	17	23	23

U narednom koraku preostaje da se analizira lista dužine 1, kojoj je prvi (jedini) element ujedno i najmanji, pa se postupak ovde završava.

Prethodno prikazani postupak se može primeniti i na sortiranje liste studenata, sa tom razlikom što elementi liste nisu celi brojevi nego rečnici koji opisuju pojedine studente. Za međusobno poređenje studenata koristi se neko od obeležja rečnika.

Funkcija `findMin` obavlja pronalaženje najmanjeg elementa u listi `studList`, poredeći obeležja sa nazivom `key`, počevši od pozicije `start`. Rezultat funkcije je indeks najmanjeg elementa u listi. Ako se funkciji prosledi prazna lista (dužine nula) rezultat funkcije će biti -1 (što označava grešku). Isti rezultat, kao signal greške, biće vraćen ako je vrednost indeksa `start` veća ili jednaka od dužine cele liste. Za slučaj da je indeks `start` pozicioniran na poslednji element liste (slučaj `n-start==1`), rezultat funkcije je upravo taj indeks. U ostalim slučajevima potrebno je pronaći najmanji element počevši od indeksa `start` do kraja liste (indeksa `n-1`). U promenljivoj `min` čuva se tekući najmanji pronađeni element, a u promenljivoj `minPos` njegov indeks u listi. Rezultat funkcije biće vrednost promenljive `minPos` kada se ispitaju svi preostali elementi liste.

```
def findMin(studList, key, start):
    n = len(studList)
    if n == 0:
        return -1
    if start >= n:
        return -1
    if n-start == 1:
        return start
    min = studList[start]
    minPos = start
    for i in range(start+1, n):
        if studList[i][key] < min[key]:
            min = studList[i]
            minPos = i
    return minPos
```

Funkcija `sort` obavlja sortiranje liste `studList`, poredeći obeležja sa nazivom `key`, počevši od indeksa `start`. Prvo se pronalazi najmanji element liste počevši od pozicije `start`. Potom element na poziciji start i najmanji element menjaju mesta. Zatim se, ukoliko indeks `start` nije došao do kraja liste (uslov `start < len(studList)-1`) ponovo poziva funkcija `sort` ali sa inkrementiranim indeksom. Funkcija `sort` će na ovaj način biti pozvana onoliko puta koliko je elemenata preostalo za testiranje do kraja liste.

```
def sort(studList, key, start):
    minPos = findMin(studList, key, start)
    if minPos == -1:
        return
    studList[start], studList[minPos] = studList[minPos], studList[start]
    if start < len(studList)-1:
        sort(studList, key, start+1)
```

Funkcija `sortStudents` služi da pojednostavi korišćenje prethodne funkcije. Jedini parametar ove funkcije je vrednost obeležja po kome će se lista sortirati. Ova funkcija će pozvati funkciju `sort` prosleđujući joj listu `studenti`, ključ po kome će lista biti sortirana (`key`), i indeks 0 (što znači da treba sortirati listu počevši od pozicije 0, tj. treba sortirati celu listu).

```
def sortStudents(key):
    sort(studenti, key, 0)
```

Prethodno prikazane funkcije čine jedan modul koji smo smestili u datoteku `studenti.py`. Inicijalizacija ovog modula sastoji se iz inicijalizacije liste studenata (na praznu listu) i poziva funkcije `loadStudents` radi učitavanja podataka iz datoteke.

```
studenti = []
loadStudents()
```

2.3 Komunikacija sa korisnikom

Pored funkcija za rukovanje referentima i studentima, potrebno je implementirati i komunikaciju našeg programa sa korisnikom. Pošto je u pitanju konzolni program, sva komunikacija odvija se putem ispisivanja stringova na konzolu i učitavanja stringova sa tastature. Novi delovi programa biće smešteni u treći modul, u datoteci `glavni.py`.

Kako će se u okviru ovog modula pozivati funkcije iz prethodna dva modula, prvo ih možemo importovati:

```
import referenti
import studenti
```

Funkcija `main` predstavlja početak izvršavanja našeg programa. Posle ispisivanja naslova programa, obavlja se prijavljivanje korisnika (tj. referenta studentske službe) na sistem (funkcija `login`). Ukoliko je prijavljivanje neuspešno (ne postoji data kombinacija korisničko ime/lozinka) izvršavanje programa

se prekida. U nastavku se započinje sentinel petlja koja će se ponavljati sve dok korisnik ne zada komandu za izlazak iz programa. Unutar tela petlje vrši se ispis menija (funkcija `menu`) na osnovu koga korisnik može da izabere komandu, izabrana komanda se dobija kao rezultat funkcije `menu` i na osnovu te vrednosti izvršava se odgovarajuća komanda (funkcije `findStudent`, `searchStudents`, `listStudents`, `updateStudent`, `addStudent` i `advanceStudents`). Sve ove funkcije nalaze se u modulu `glavni` i ne treba ih mešati sa istoimenim funkcijama iz drugih modula.

```
def main():
    print()
    print("Evidencija studenata")
    print("====================")
    print()
    if not login():
        print("\nNiste uneli postojece ime i lozinku!")
        return
    komanda = '0'
    while komanda != 'X':
        komanda = menu()
        if komanda == '1':
            findStudent()
        elif komanda == '2':
            searchStudents()
        elif komanda == '3':
            listStudents()
        elif komanda == '4':
            updateStudent()
        elif komanda == '5':
            addStudent()
        elif komanda == '6':
            advanceStudents()
    print("Dovidjenja.")
```

Funkcija `menu` će ispisati meni na ekran (poziv funkcije `printMenu`), i zahtevati unos komande od korisnika. Unos će se ponavljati u slučaju da je uneta vrednost koja nije unapred predviđena.

```
def menu():
    printMenu()
    command = input(">> ")
    while command.upper() not in ('1', '2', '3', '4', '5', '6', 'X'):
        print("\nUneli ste pogresnu komandu.\n")
        printMenu()
        command = input(">> ")
    return command.upper()
```

Funkcija `printMenu` će ispisati sadržaj menija na ekranu.

```
def printMenu():
    print("\nIzaberite opciju:")
    print("  1 - pronalazenje studenta")
    print("  2 - pretrazivanje studenata")
    print("  3 - pregled svih studenata")
    print("  4 - izmena podataka o studentu")
    print("  5 - upis novog studenta")
    print("  6 - upis nove godine studija")
    print("  x - izlaz iz programa")
```

Funkcija `login` će zatražiti od korisnika da unese korisničko ime i lozinku i potom, pozivom funkcije `login` iz modula `referenti`, proveriti da li takav korisnik postoji. Rezultat ove funkcije biće direktno preuzet od funkcije `login` iz modula `referenti`.

```
def login():
    username = input("Korisnicko ime >> ")
    password = input("Lozinka >> ")
    return referenti.login(username, password)
```

Funkcija `findStudent` će od korisnika zatražiti da unese broj indeksa studenta čiji se podaci traže, pokušati da ga pronađe u listi svih studenata (pozivom funkcije `findStudent` iz modula `studenti`) i ispisati odgovarajući sadržaj. Ukoliko je pronađen traženi student, njegovi podaci se ispisuju u tabelarnoj formi onako kako je to definisano funkcijama `formatHeader` i `formatStudent` iz modula `studenti`.

```
def findStudent():
    print("[1] Pronalazenje studenta\n")
    indeks = input("Unesite broj indeksa >> ")
    stud = studenti.findStudent(indeks)
    if stud != None:
        print(studenti.formatHeader())
        print(studenti.formatStudent(stud))
    else:
        print("Nije pronadjen student sa brojem indeksa", indeks)
```

Funkcija `searchStudents` će od korisnika zatražiti da unese prezime studenata koje želi da pronađe, pokušati da ih pronađe (pozivom funkcije searchStudents iz modula studenti) i ispisati odgovarajući sadržaj. Pošto potencijalno može biti više studenata sa istim prezimenom, rezultat ove pretrage je lista pronađenih studenata. Ispis rezultata se vrši u tabelarnom obliku ako je bilo studenata sa traženim prezimenom.

```
def searchStudents():
    print("[2] Pretrazivanje studenata\n")
    prezime = input("Unesite prezime >> ")
    studList = studenti.searchStudents('prezime', prezime)
    if len(studList) == 0:
        print("\nNema trazenih studenata.")
    else:
        print('\n')
        print(studenti.formatHeader())
        print(studenti.formatStudents(studList))
```

Funkcija `listStudents` će obaviti sortiranje svih studenata po prezimenu (pozivom funkcije `sortStudents` iz modula `studenti`) i ispisati celu listu studenata u tabelarnom formatu.

```
def listStudents():
    print("[3] Pregled svih studenata sortiranih po prezimenu\n")
    studenti.sortStudents('prezime')
    print(studenti.formatHeader())
    print(studenti.formatAllStudents())
```

Funkcija `updateStudent` će obaviti ažuriranje adrese, telefona i emaila za studenta sa datim brojem indeksa. Prvo se broj indeksa unosi sa tastature. Potom se pokušava pronalaženje studenta sa datim brojem indeksa (pozivom funkcije `findStudent` iz modula `studenti`). Ako student nije pronađen, ispisuje se poruka o grešci, a ukoliko jeste, ispisuje se trenutna vrednost njegovih podataka, sa tastature unose nove vrednosti, i na kraju se lista studenata (ovaj put sa promenjenim podacima za jednog od studenata) snima u datoteku.

```
def updateStudent():
    print("[4] Izmena podataka o studentu\n")
    indeks = input("Unesite broj indeksa >> ")
    stud = studenti.findStudent(indeks)
    if stud == None:
        print("Ne postoji student sa datim brojem indeksa.")
    else:
        print(studenti.formatHeader())
```

```
        print(studenti.formatStudent(stud))
        stud['adresa'] = input("Unesite adresu studenta >> ")
        stud['telefon'] = input("Unesite telefon studenta >> ")
        stud['email'] = input("Unesite e-mail studenta >> ")
        studenti.saveStudents()
```

Funkcija addStudent inicijalizuje novi rečnik koji predstavlja novog studenta, obavlja unos njegovih podataka sa tastature, dodaje novog studenta u listu postojećih studenata (poziv funkcije `addStudent` iz modula `studenti`), i na kraju snima sadržaj cele liste studenata u datoteku (zajedno sa novim studentom).

```
def addStudent():
    print("[5] Upis novog studenta\n")
    stud = {}
    stud['indeks'] = input("Unesite indeks >> ")
    stud['ime'] = input("Unesite ime >> ")
    stud['prezime'] = input("Unesite prezime >> ")
    stud['roditelj'] = input("Unesite ime roditelja >> ")
    stud['datum'] = input("Unesite datum rodjenja >> ")
    stud['jmbg'] = input("Unesite JMBG >> ")
    stud['adresa'] = input("Unesite adresu >> ")
    stud['telefon'] = input("Unesite telefon >> ")
    stud['email'] = input("Unesite e-mail >> ")
    stud['godina'] = '1'
    studenti.addStudent(stud)
    studenti.saveStudents()
```

Funkcija `advanceStudents` obavlja upis izabranih studenata u narednu godinu studija. Studenti koji upisuju narednu godinu se identifikuju brojem indeksa. Njihovi indeksi se unose u sentinel petlji. Ažuriranje tekuće godine studija za svakog izabranog studenta obavlja se pozivom funkcije `advanceStudent` iz modula `studenti`. Na kraju se cela lista studenata snima u datoteku.

```
def advanceStudents():
    print("[6] Upis studenata u narednu godinu studija\n")
    indeks = input("Unesite indeks studenta koji upisuje narednu godinu (<Enter> za kraj) >> ")
    while indeks != '':
        stud = studenti.findStudent(indeks)
        if stud == None:
            print("Ne postoji student sa datim brojem indeksa.")
        else:
            studenti.advanceStudent(stud)
            print("Student", indeks, "je upisan u narednu godinu.")
        indeks = input("Unesite indeks studenta koji upisuje narednu godinu (<Enter> za kraj) >> ")
    studenti.saveStudents()
```

Modul `glavni` nije od koristi drugim modulima, pa ne očekujemo da bude importovan iz nekog drugog modula. Ukoliko bi se to ipak desilo, želimo da poziv funkcije main kojom se zapravo pokreće program bude rezervisan samo za situacije kada je ovaj modul pozvan direktno pomoću komande

```
$ python3 glavni.py
```

Poziv funkcije `main` zato možemo staviti u okviru uobičajene `if` naredbe za ovakve situacije:

```
if __name__ == '__main__':
    main()
```