NAZIV PROJEKTA:

Mreža centara za regionalne kratke studijske programe u zemljama Zapadnog Balkana

(Network of centers for regional short study programs in the countries of the Western Balkans - WBNET)

Erasmum+ projekt br.101128813

2023-2026

IT KURS - PREDMET

OSNOVE PROGRAMIRANJA – PYTHON

(BASICS OF PROGRAMMING – PYTHON)

Una Drakulić MA, Melisa Haurdić MA

una.drakulic@unbi.ba melisa.haurdic@unbi.ba

Univerzitet u Bihaću

Tehnički fakultet

Sadržaj

1 UVOD U PYTHON

Za one koji su programirali u jezicima kao što su C/C++, Java, C# i sličnim programskim jezicima, tri su ključne razlike između tih jezika i Pythona:

- U Pythonu varijable nemaju oznaku tipa iako pojam tipa podatka postoji. To znači da nekoj varijabli možemo pridružiti vrijednost bilo kojeg tipa. Varijabla u Pythonu ne mora uvijek sadržavati vrijednost istog tipa, to jest, u jednom trenutku može sadržavati, recimo, znakovni niz, a u drugom cijeli broj.

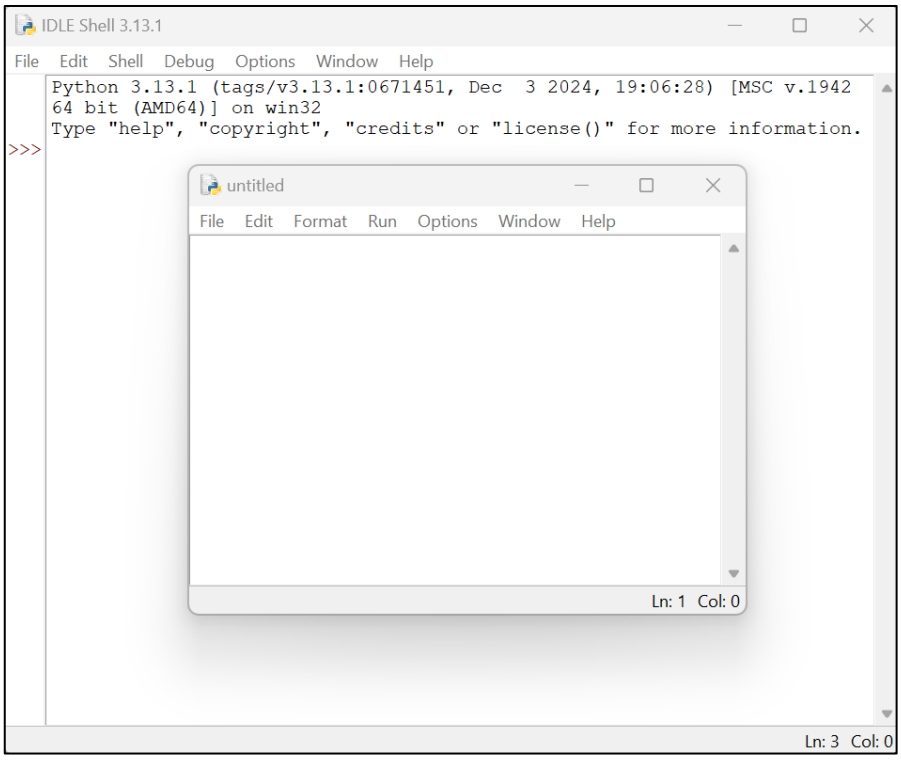
- U Pythonu memorija se oslobađa automatski. Drugim riječima, ne postoji naredba kojom se eksplicitno uklanja ili dealocira objekt iz memorije. To je omogućeno sakupljačem smeća (engl. garbage collector) sistemom, sistemom za automatsko upravljanje memorijom koji je dio Pythonovog izvršnog sistema. To znači da programer ne mora voditi računa o tome na kojem mjestu u programu treba neki objekt obrisati, niti mora misliti na potencijalno „curenje memorije“ (engl. memory leak), što je jedan od čestih problema s jezicima koji nemaju automatsko upravljanje memorijom. U Pythonu automatsko upravljanje memorijom radi na sličan način kao i u jezicima Java i C#.

- Program pisan u Pythonu ne prevodi se na prirodni kôd (engl. native code) dotičnog računala nego se izvršava pomoću drugog programa koji se zove interpreter. Iz tog razloga programi pisani u Pythonu izvršavaju se sporije od onih pisanih u jezicima koji izvorni kôd prevode na prirodni, kao što su C i C++. Zbog toga i zbog načina izvršavanja Python nije idealan jezik za izradu programa kod kojih je brzina izvršavanja važan aspekt, kao što su sistemski programi (na primjer, operacijski sistemi), programi za upravljanje hardverskim komponentama (engl. driver), razni kontroleri i sl. Međutim, u zadnje vrijeme Python se sve više upotrebljava za programiranje ugrađenih sistema (engl. embedded systems), čime je primjena tog jezika ušla u prostor koji je nekad bio uglavnom rezerviran za jezike C i C++.

Jedna, možda i najznačajnija prednost Pythona u odnosu na druge popularne programske jezike je ta da za njega postoji, pored njegovih standardnih biblioteka, i velik broj biblioteka za mnoga područja primjene, kao što su analiza podataka, web , mašinsko učenje i umjetna inteligencija, mrežno programiranje i mnoge druge. Ovo posljednje, umjetna inteligencija, područje je u kojem je Python trenutno najpopularniji programski jezik. Dalje, s obzirom da je Python programski jezik otvorenog koda (engl. Open source) za njega postoje nebrojeni izvori informacija, od knjiga i časopisa, do online izvora kao što su YouTube i mnoge web stranice. Službena web stranica za Python je: http://www.python.org.

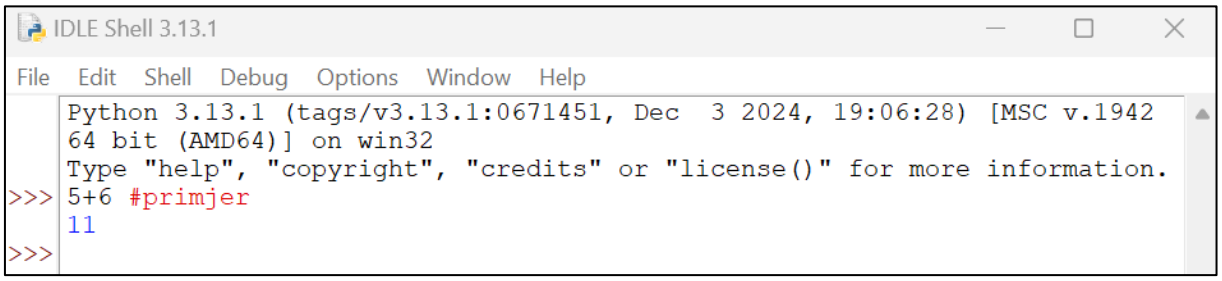
Python se isporučuje sa standardnim razvojnim okruženjem koje se zove IDLE (Integrated Development and Learning Environment). Nakon pokretanja IDLE-a dobije se komandna linija u koju se može unositi programski kôd. Nakon unosa, po pritisku tipke Enter ili Return ispod same linije, dobije se rezultat unesenog kôda.

Na slici 1 je prikazan IDLE Shell 3.13.1 sa otvornim fajlom za pisanje novog kôda.



Slika 1. IDLE Shell 3.13.1 sa otvornim fajlom za pisanje novog kôda

Najjednostavniji primjer sabiranja dva unesena broja i ispis rezultata je prikazan na slici 2.



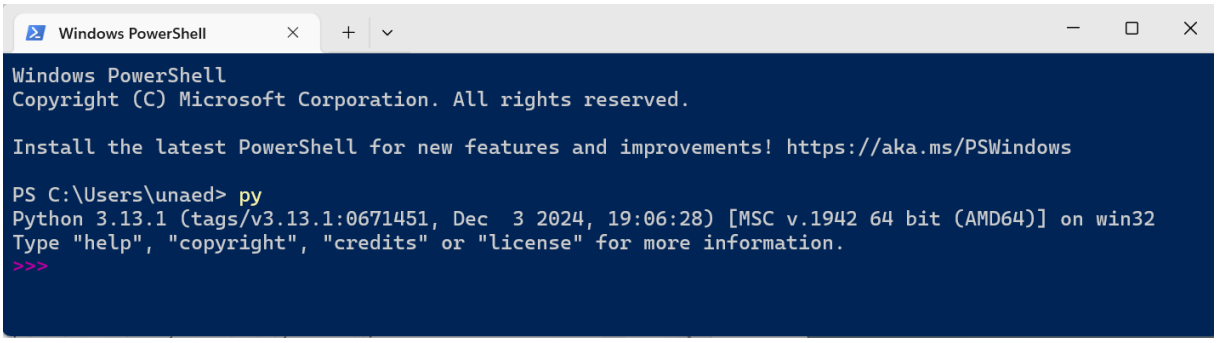
Slika 2. Primjer sabiranja dva broja koristeći IDLE Shell

Na slici 2 se može vidjeti da pisanje komentara u Pythonu označava sa oznakom # gdje se tekst iza oznake smatra komentarom i ne izvršava prilikom pokretanja kôda. Komentari koji obuhvaćaju više redova i koji se stavljaju između jednostrukih ili dvostrukih navodnika.

Iako je rad u komandnoj liniji koristan za kratke programe te razna isprobavanja i testiranja, za veće programe on je nepraktičan, a i to mu nije glavna namjena. U IDLE Shell može se odabrati stavka izbornika *File/NewFile*, nakon čega će se otvoriti uređivač teksta (slika 1). Tako uneseni program može se pokrenuti stavkom izbornika *Run/Run Module*. Prije pokretanja programa taj program se mora pohraniti u neku datoteku sa nastavkom .py. Svaka datoteka s kôdom pisanim u Pythonu mora imati nastavak .py i naziva se modulom. To znači da se svaki program pisan u Pythonu mora sastojati od barem jednog modula. Modul iz kojeg pokrećemo program glavni je modul i njegova datoteka često se zove main.py, iako to nije obavezno.

Također, treba se napomenuti da se Python ne mora nužno koristiti pomoću okruženja IDLE Shell već se može koristiti i preko komandne linije operacijskog sistema na kojem radimo. U sistemu Windows, na primjer, upisivanjem naredbe py (pod uvjetom da je Python instaliran i da je putanja

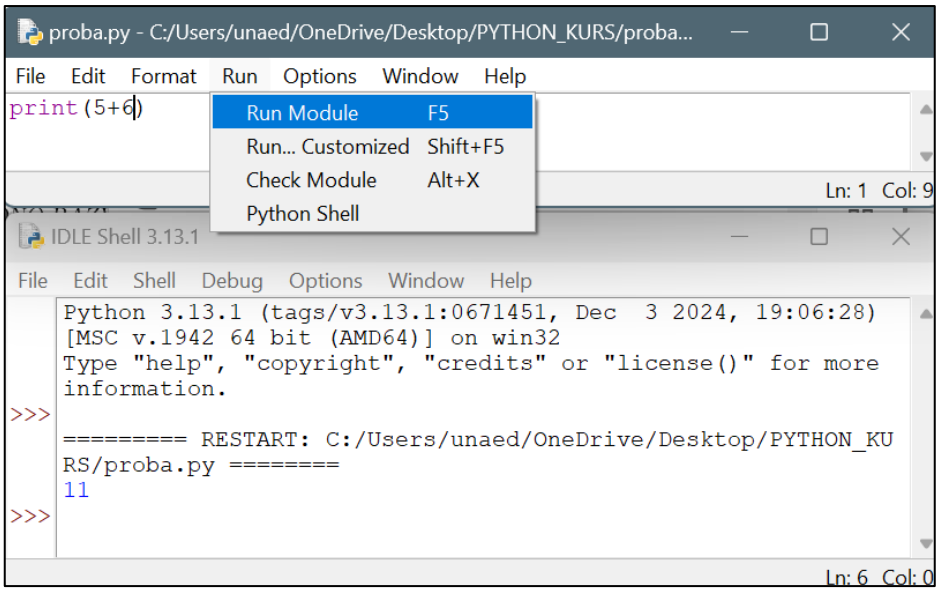
u varijabli PATH ispravno postavljena) dobijemo sličnu komandnu liniju kao i u IDLE Shell-u, slika 3.



Slika 3. Pisanje Python kôda koristeći Windows PowerShell

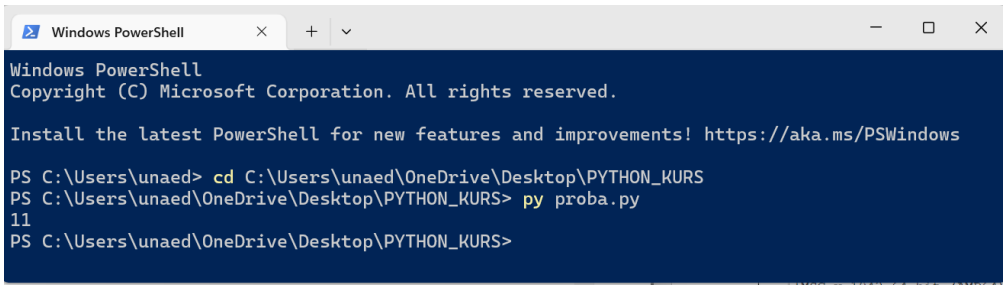
Iz ovoga se može izaći pritiskom tipki Ctrl+Z+Enter (ili Return) ili naredbom exit().

Ako želimo pokrenuti program izrađen u Pythonu koji se nalazi u nekom modulu, ime tog modula možemo dodijeliti naredbi py kao parametar, čime će ga Python interpreter izvršiti. Pretpostavimo da se u modulu proba.py nalazi sljedeći jednostavni program:



Slika 4. Pokretanje modula proba.py koristeći IDLE Shell

Taj program se može pokrenuti i na sljedeći način:

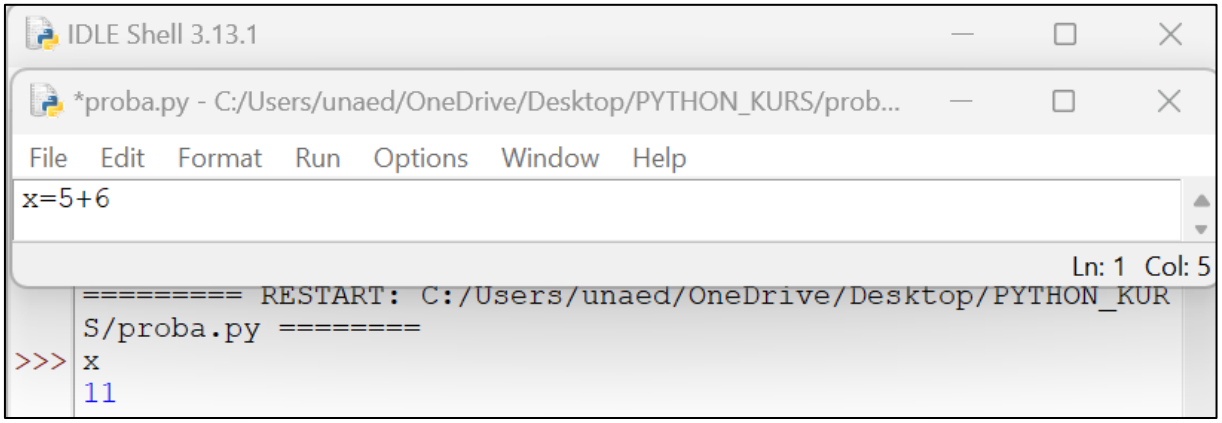


Slika 5. Pokretanje modula proba.py koristeći Windows PowerShell

U narednom poglavlju opisani su osnovni pojmovi u Python programskom jeziku.

2 OSNOVNI POJMOVI

Prije nego što pređemo na sâm programski jezik treba jasno definirati pojam izraza, odnosno razliku između izraza i naredbe. To je važno da bismo razumjeli neke osnovne konstrukte u Pythonu, ali i drugim programskim jezicima. Izraz je bilo koji konstrukt programskog jezika čije izvršavanje vraća neku vrijednost kao rezultat. Na primjer 5 + 6 je izraz, jer daje neki rezultat, tj. vrijednost 11. Također, na primjer 5 < 6 je logički izraz, čiji rezultat u ovom slučaju bude 1 ili True. S druge strane, naredbe nemaju definiran rezultat. Primjerice, naredba pridruživanja x = 5 + 6 nema definiran rezultat, a to znači da ona ne može biti dio nekog većeg izraza. To možemo vidjeti i u IDLE-u, gdje se nakon unosa gornje naredbe ispod komandne linije ne ispisuje ništa. Naredba pridruživanja jednostavno pridružuje rezultat izraza s desne strane varijabli s lijeve strane. U ovom primjeru, ako želimo vidjeti šta je pridruženo varijabli x možemo unijeti izraz x:

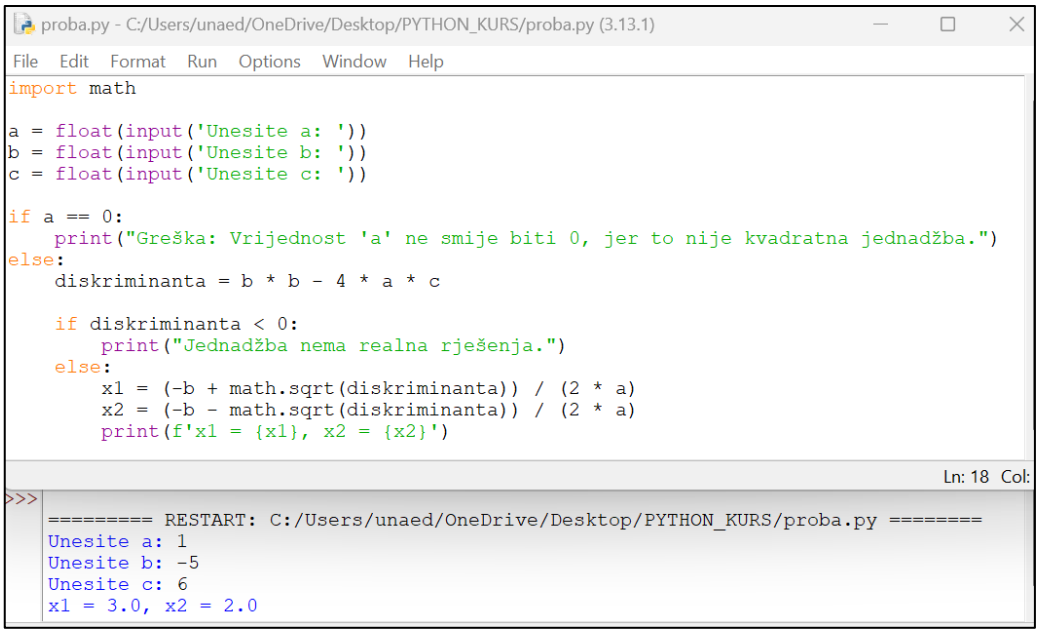


Slika 6. Pozivanje vrijednosti x iz modula proba.py

Kod naredbe pridruživanja prvo se odredi vrijednost izraza s desne strane, a zatim se ta vrijednost pridruži imenu varijable s lijeve strane. Kada varijabli pridružimo neku vrijednost, ona joj ostaje pridružena sve dok toj varijabli ne pridružimo neku drugu vrijednost.

Funkcija koju ćemo vrlo često koristiti je *print*. Ona služi za ispis rezultata svih izraza (uključujući i one trivijalne kao što su konstante ili varijable) koji su joj zadani, s lijeva na desno (slika 4). Ova se funkcija može upotrebljavati s različitim vrstama (tipovima) podataka.

Primjer korištenja naredbe print za različite tipove podataka, rješavanje kvadratne jednačine (slika 7).



Slika 7. Primjer rješavanja kvadratne jednačine

Ovaj Python program rješava kvadratnu jednadžbu oblika $ax^2 + bx + c = 0$ koristeći matematičku formulu za rješenja. Program koristi ugrađeni **math** paket kako bi izračunao kvadratni korijen diskriminante ($\Delta = b^2 - 4ac$) pomoću funkcije math.sqrt(). Prvo provjerava da $a \neq 0$ kako bi spriječio dijeljenje s nulom, a zatim osigurava da diskriminanta nije negativna kako bi izbjegao grešku pri računanju kvadratnog korijena. Ako su svi uvjeti ispunjeni, ispisuje rješenja x_1 i x_2.

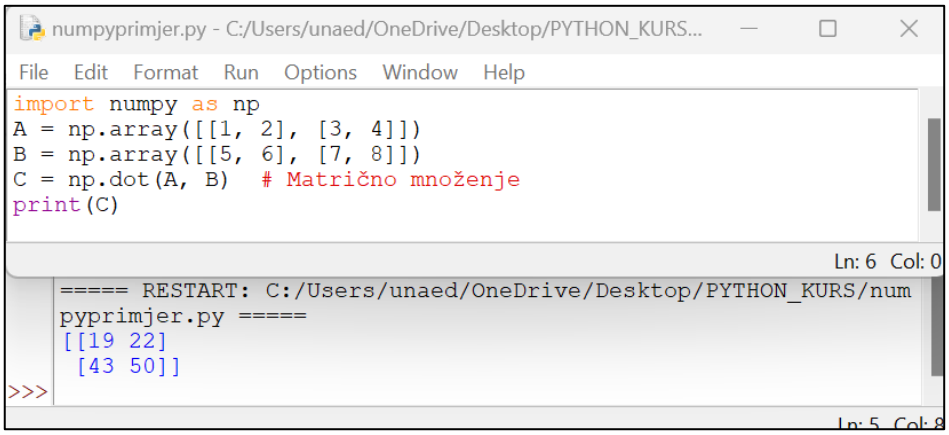
Math je ugrađeni Python paket koji pruža matematičke funkcije i konstante, poput trigonometrijskih funkcija, eksponencijalnih i logaritamskih operacija, te rada s kvadratnim korijenima. Koristi se kada su potrebne precizne matematičke operacije koje standardni operatori ne podržavaju. Pošto je *math* dio standardne Python biblioteke, ne treba ga posebno instalirati – dovoljno ga je uvesti u program pomoću naredbe *import math*. Primjeri korisnih funkcija u *math* paketu:

- math.sqrt(x) – vraća kvadratni korijen broja x
- math.pow(x, y) – potenciranje (x^y)
- math.sin(x), math.cos(x), math.tan(x) – trigonometrijske funkcije
- math.pi – konstanta π

Python je danas jedan od najčešće korištenih programskih jezika u znanstvenim i inženjerskim primjenama, ponajviše zbog svoje bogate kolekcije paketa (biblioteka) koje omogućuju jednostavnu i efikasnu implementaciju različitih algoritama. Pored opisanog *math* paketa, neke od najkorištenijih su:

2.1. *NumPy* paket za numeričke proračune

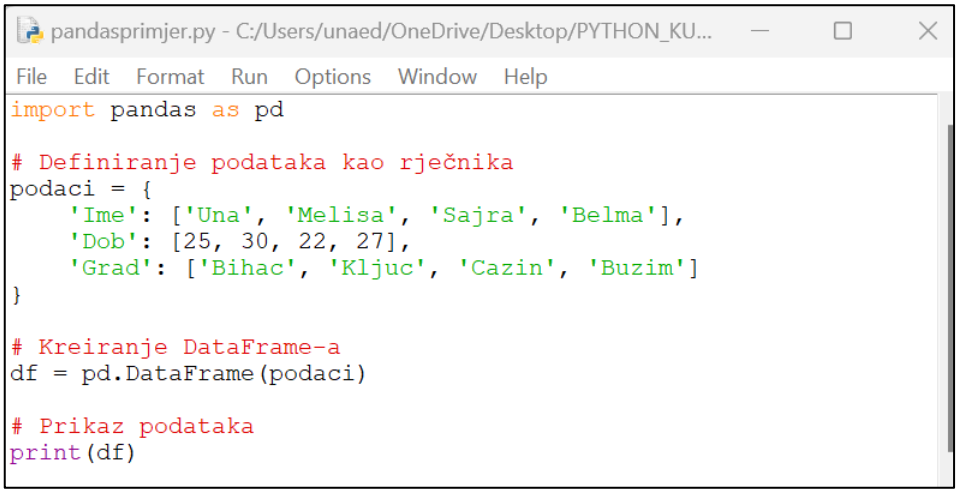
NumPy (Numerical Python) je osnovni paket za numeričke izračune u Pythonu. Omogućava rad s višedimenzionalnim nizovima (*arrays*), vektorskim i matričnim operacijama, kao i linearnom algebrom. Prednost NumPy-ja u odnosu na standardne Python liste leži u njegovoj optimizaciji za brze vektorske operacije i paralelizaciju. Osnove funkcionalnosti su: numpy.array za efikasno rukovanje višedimenzionalnim nizovima, numpy.linalg za operacije linearne algebre, numpy.random za generiranje pseudo-slučajnih brojeva i druge. Način instalacije paketa je korištenje naredbe: pip install numpy, a primjer upotrebe je prikazan na slici 8.

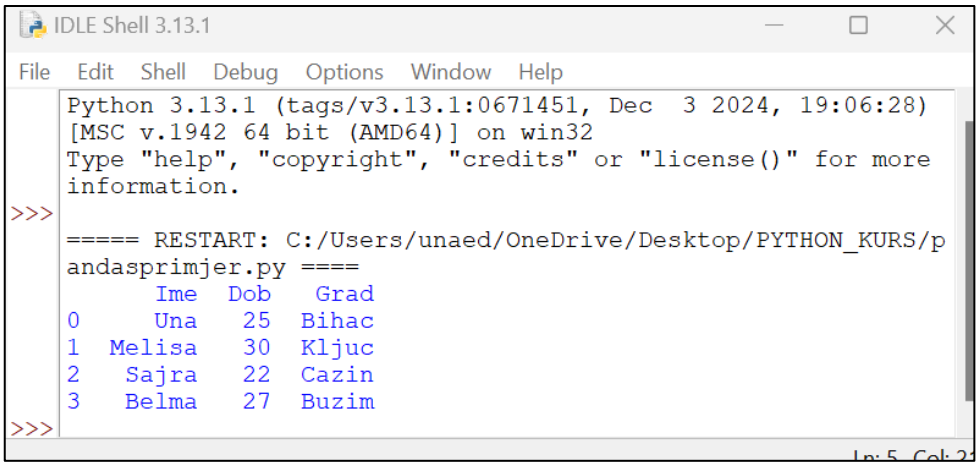


Slika 8. Primjer upotrebe numpy paketa

2.2. *Pandas* paket za obradi i analizu podataka

Pandas je vrlo učinkovit paket za manipulaciju i analizu podataka. Omogućava rad s podacima u tabličnom obliku kroz *DataFrame* strukturu, koja nudi fleksibilne metode za filtriranje, grupiranje i agregaciju podataka. Osnovne funkcionalnosti su: učitavanje podataka iz CSV, Excel i SQL formata, manipulacija podacima (grupiranje, filtriranje, spajanje), rad sa vremenskim serijama i druge. Način instalacije paketa je korištenje naredbe: pip install pandas, a primjer upotrebe je prikazan na slici 9.

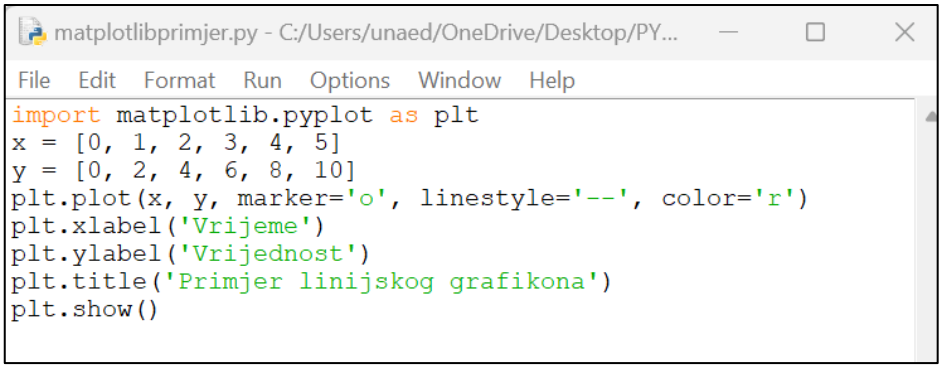


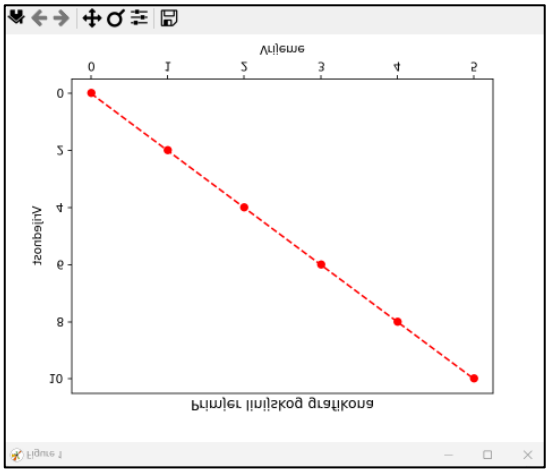


Slika 9. Primjer upotrebe pandas paketa

2.3. Matplotlib paket za vizualizaciju podataka

Matplotlib je standardni paket za crtanje grafova u Pythonu. Omogućava prikaz podataka u obliku linijskih, stupčastih, raspršenih (scatter) i histogram grafova. Često se koristi zajedno s Pandasom i NumPy-jem za analizu podataka. Osnovne funkcionalnosti su: prikaz vremenskih serija, histogrami, 3D grafovi, prilagodba osi i oznaka, interaktivni prikazi podataka i druge. Način instalacije paketa je korištenje naredbe: pip install matplotlib, a primjer upotrebe je prikazan na slici 10.



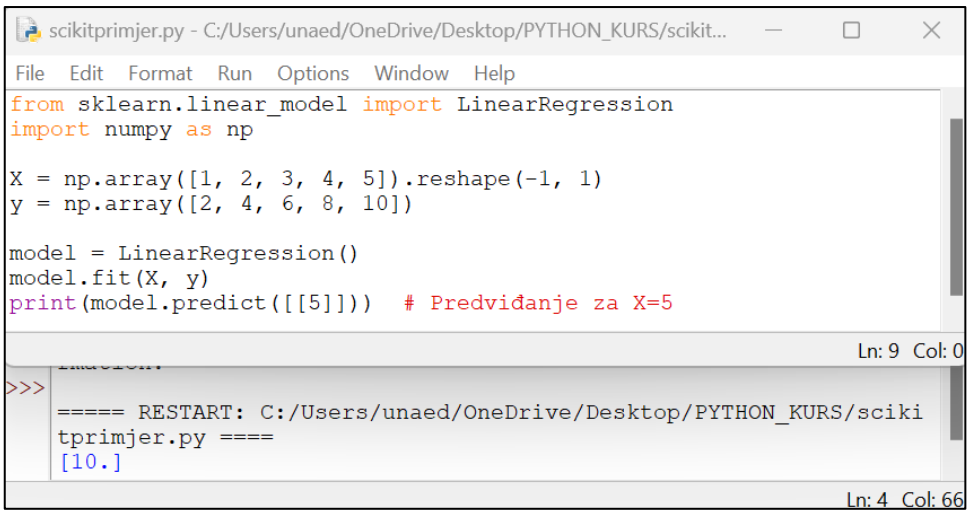


Slika 10. Primjer upotrebe matplotlib paketa

Ova naredba uvozi modul *pyplot* iz paketa *Matplotlib* i dodjeljuje mu alias (skraćenicu) plt kako bismo ga lakše koristili u kodu. *Pyplot* sadrži funkcije koje omogućuju brzo kreiranje i prilagodbu grafova u Pythonu. Naredba *plt.plot (x, y, ...)* crta linijski graf na temelju podataka iz x i y na listi, naredba *marker='o'* crta tačke na svakoj koordinati, naredba *linestyle='- -'* koristi isprekidanu liniju između tačaka, naredba *color='r'* liniju isrtava u crvenoj boji. Nazivi osa se dodaju naredbama *plt.xlabel(' naziv'),* tako i za ostale ose, a naziv grafa naredbom *plt.title('naziv').* Za prikaz grafa se koristi naredba *plt.show().*

2.4. Scikit-learn paket za mašinsko učenje

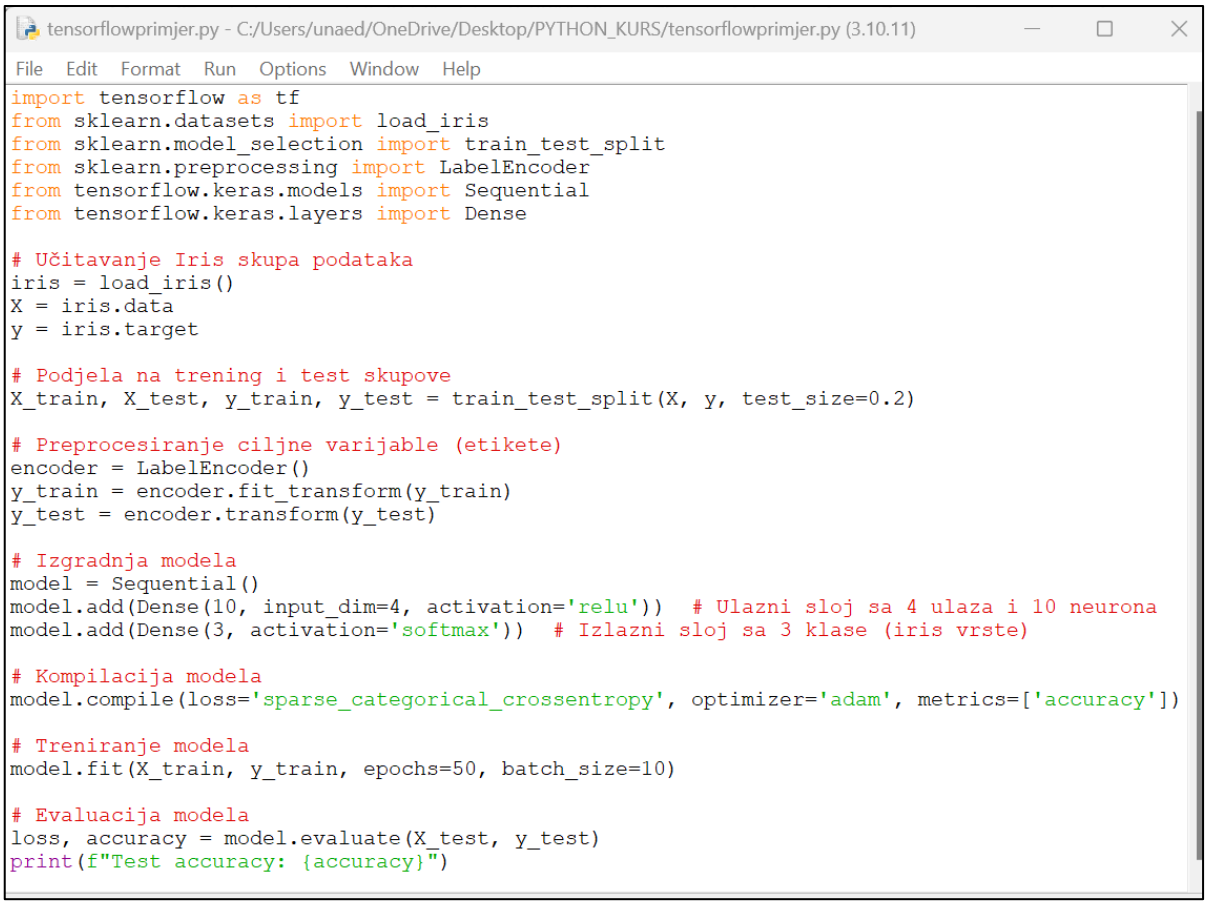
Scikit-learn je jedan od najpopularnijih paketa za klasične algoritme mašinskog učenja. Omogućava implementaciju regresijskih, klasifikacijskih i klastering algoritama s optimiziranom obradom podataka. Osnovne funkcionalnosti su: linearna i logička regresija, SVM (support vector machine), K-means klasteriranje, predprocesiranje podataka i redukcija dimenzionalnosti, podjela podataka na skupove za treniranje i testiranje i druge. Način instalacije paketa je korištenje naredbe: pip install scikit-learn, a primjer upotrebe je prikazan na slici 11.

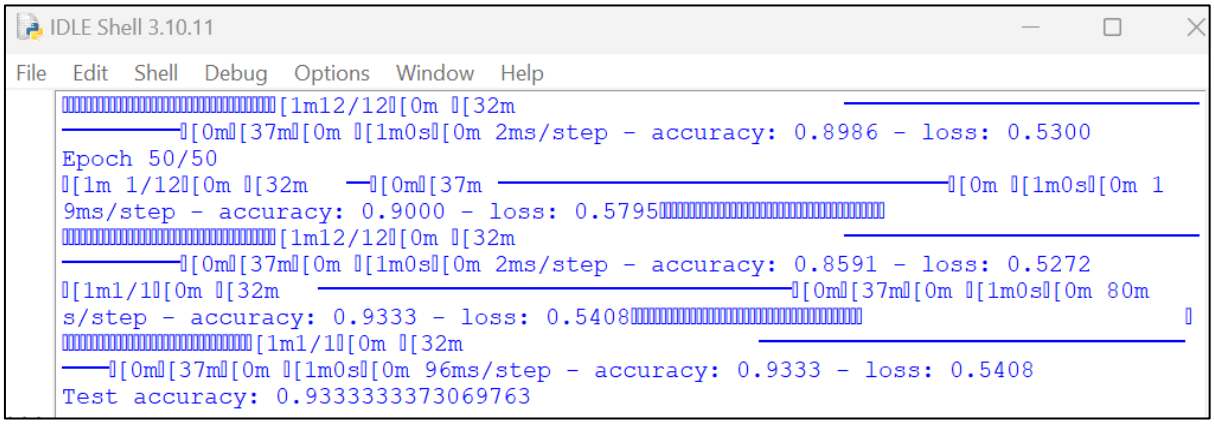


Slika 11. Primjer upotrebe scikit-learn paketa

2.5. TensorFlow/PyTorch paket za duboko učenje

Tensorflow/PyTorch je paket za napredne primjene neuronskih mreža. Omogućava razvoj i treniranje složenih modela dubokog učenja uz podršku za GPU akceleraciju. Osnovne funkcionalnosti su: definiranje i treniranje neuronskih mreža, optimizacija i prilagodba modela, računalna vizija i obrada prirodnog jezika i druge. Način instalacije paketa je korištenje naredbe: pip install tensorflow, pip install torch, a primjer upotrebe je prikazan na slici 12.





Slika 12. Primjer upotrebe Tensorflow *paketa*

Objašnjenje koda:

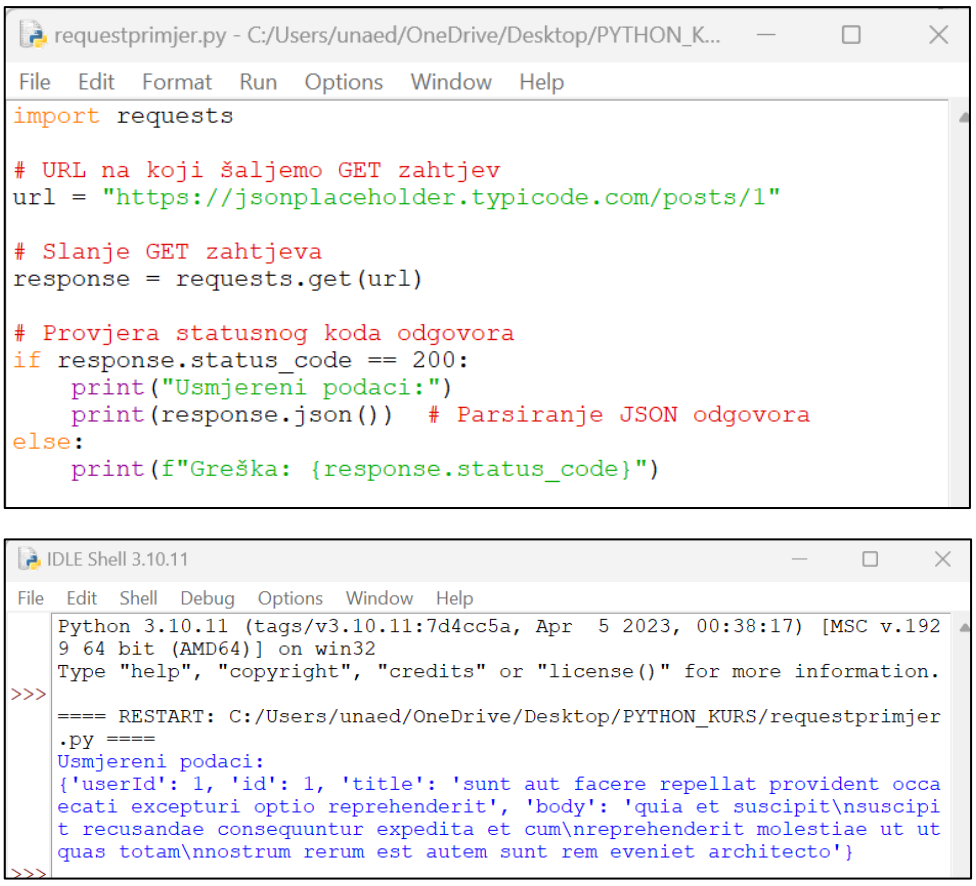
1. Učitavanje skupa podataka: Koristi se Iris dataset, koji je popularan za zadatke klasifikacije. Skup podataka ima 4 parametra i 3 klase (tri vrste iris cvijeta).
2. Podjela skupa podataka: Skup podataka se dijeli na trening i test skupove u omjeru 80/20.
3. Preprocesiranje ciljne varijable: Korišten je LabelEncoder kako bi se oznake (koje su u obliku brojeva 0, 1, 2) pretvorile u numeričke vrijednosti koje model može obraditi.

4. Izgradnja modela: Model je jednostavan neural network s jednim skrivenim slojem (10 neurona) i izlaznim slojem s 3 klase. Korištena je softmax funkcija aktivacije u izlaznom sloju za višeklasnu klasifikaciju.

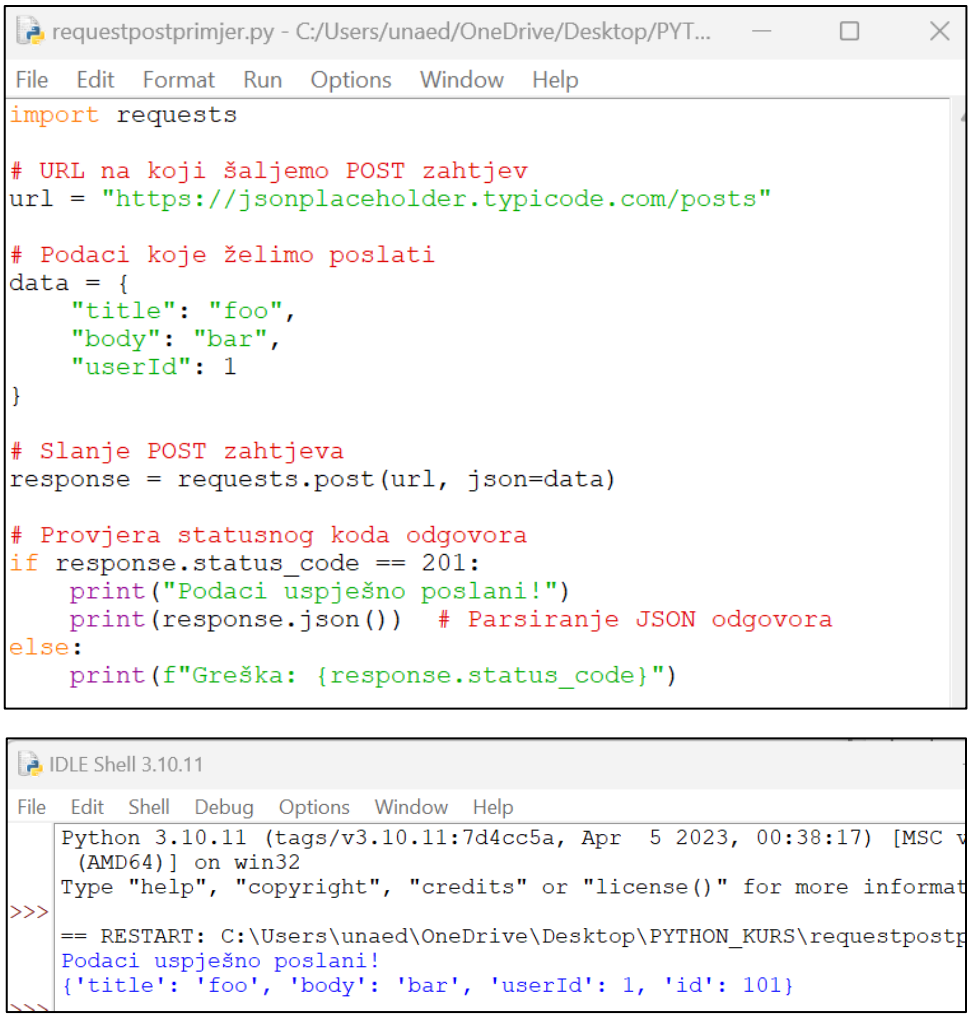
5. Treniranje i evaluacija: Model se trenira s Adam optimizatorom i sparse categorical crossentropy funkcijom gubitka za višeklasnu klasifikaciju. Na kraju, model se evaluira na testnim podacima i ispisuje tačnost.

2.6. Requests – HTTP paket za zahtjeve

Requests je jednostavan, ali moćan paket za slanje HTTP zahtjeva i dohvaćanje podataka s interneta. Često se koristi za rad s REST API-jima i automatizaciju dohvaćanja podataka. Omogućava slanje GET i POST zahtjeva, autentifikaciju i rad sa API-jima, preuzimanje podataka u JSON formatu i druge. Način instalacije paketa je korištenje naredbe: pip install requests, a primjer upotrebe je prikazan na slikama 13 i 14.



Slika 13. Primjer upotrebe request *paketa – GET zahtjev*



Slika 14. Primjer upotrebe request *paketa – POST zahtjev*

Objašnjenje koda:

1. GET zahtjev: Na JSONPlaceholder šaljemo GET zahtjev za dohvat podataka s određenim ID-em (/posts/1). Očekujemo JSON odgovor koji zatim parsiramo pomoću response.json().

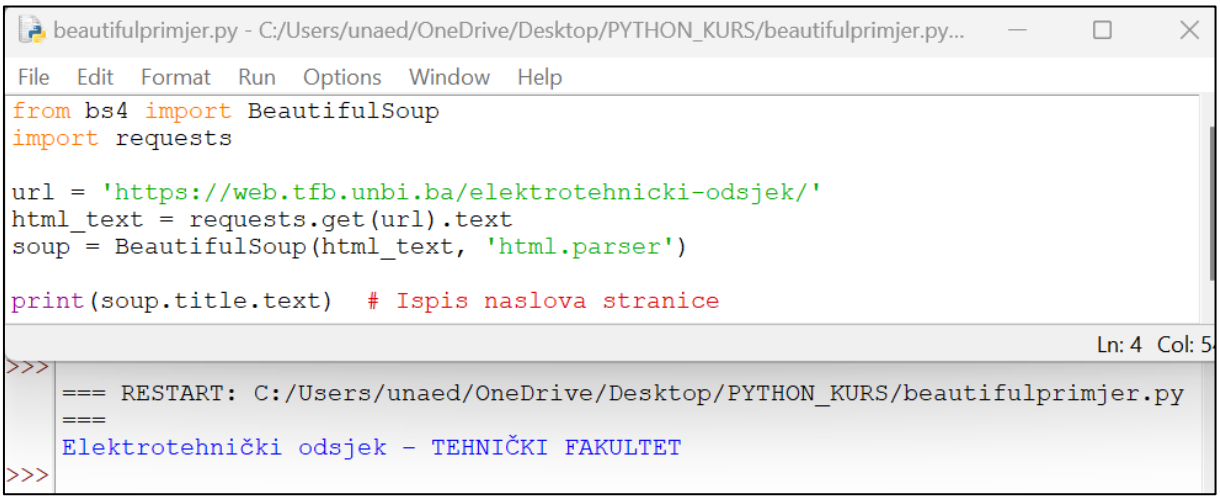
2. POST zahtjev: Za slanje podataka na API, koristimo requests.post() s URL-om i podacima koje želimo poslati. Podaci se šalju u JSON formatu (json=data). Ako je zahtjev uspješan, odgovor će imati status kod 201, što označava da su podaci uspješno stvoreni.

Odgovori:

- response.status_code: Vraća statusni kod HTTP odgovora, koji označava uspjeh ili grešku zahtjeva.
- response.json(): Ako je odgovor u JSON formatu, koristi se za parsiranje odgovora u Python rječnik.

2.7. BeautifulSoup – Web scraping

BeautifulSoup je paket koji omogućava parsiranje i ekstrakciju podataka iz HTML i XML dokumenata. Najčešće se koristi za automatizirano prikupljanje informacija s web stranica. Omogućava analizu HTML sadržaja, pretragu specifičnih elemenata (naslovi, linkovi ...), automatsko prikupljanje podataka i druge. Način instalacije paketa je korištenje naredbe: pip install beautifulsoup4, a primjer upotrebe je prikazan na slici 15.



Slika 15. Primjer upotrebe beautifulsoup4 paketa

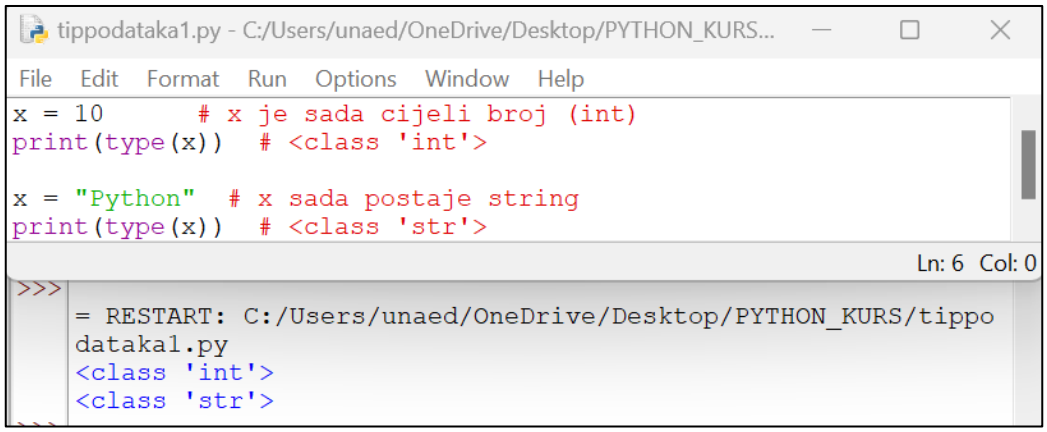
U narednom poglavlju opisani su tipovi podataka u Python programskom jeziku.

3 TIPOVI PODATAKA

Za razliku od programskih jezika kao što su Java, C++ ili C#, gdje varijable moraju imati unaprijed određeni tip podatka kao što su int, float, double ili char, Python pripada skupini dinamički tipiziranih programskih jezika, što znači da tip varijable nije fiksiran prilikom deklaracije.

U jezicima sa statičkom tipizacijom, poput C++-a ili Jave, programer mora eksplicitno navesti tip podatka koji će određena varijabla sadržavati, a svaka promjena tipa zahtijeva eksplicitnu konverziju ili ponovno definiranje varijable. S druge strane, u Pythonu varijabla može u bilo kojem trenutku sadržavati vrijednost bilo kojeg tipa, a sam interpreter automatski određuje tip varijable na temelju dodijeljene vrijednosti.

Na primjer, možemo definirati varijablu x i prvo joj dodijeliti cijeli broj, a zatim promijeniti vrijednost u niz znakova, što bi u jezicima sa statičkom tipizacijom zahtijevalo dodatne konverzije ili redefiniciju, slika 16.



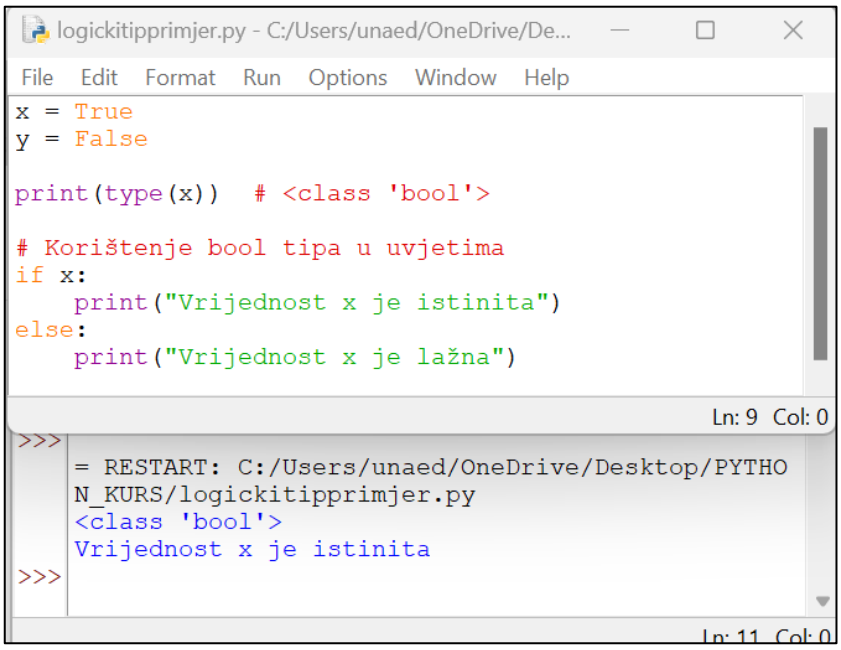
Slika 16. Primjer konverzije tipa podatka

Kao i u drugim programskim jezicima tip podatka jedan je od temeljnih pojmova u Pythonu i u ovom dijelu ćemo opisati sve tipove podataka koji su ugrađeni u ovaj jezik. Python ima nekoliko ugrađenih tipova podataka od kojih su nam ovdje najvažniji sljedeći:

- logički tip: *bool* (s vrijednostima True i False)
- cjelobrojni tip: *int*
- realni tip: *float*
- kompleksni tip: *complex*
- znakovni niz: *str*
- dva uređena niza vrijednosti: *list* i *tuple*
- dvije neuređene kolekcije vrijednosti: *dict* i *set*

3.1. Logički tip

Logički tip (bool) koristi se za predstavljanje istinitosnih vrijednosti, koje mogu biti **True** (istina) ili **False** (laž). Ovaj tip se najčešće koristi u uvjetnim izrazima i logičkim operacijama (slika 17).

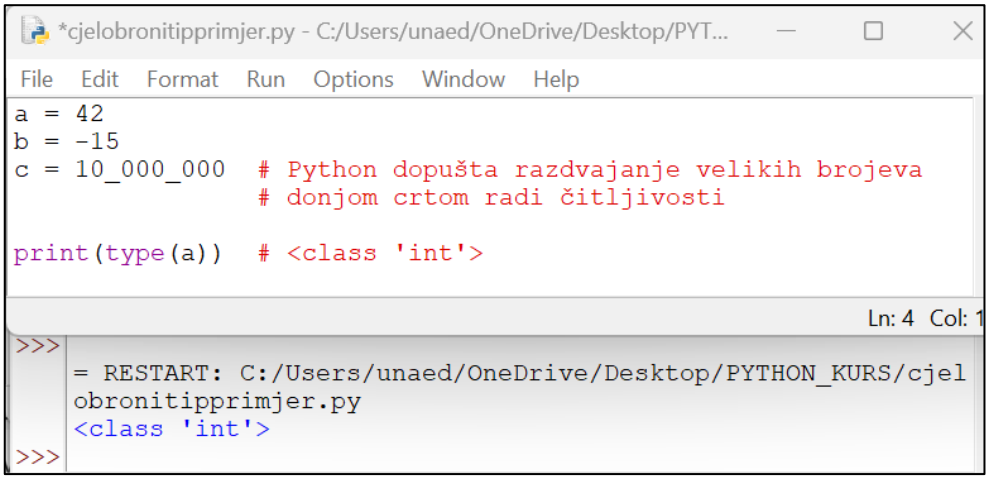


Slika 17. Primjer logičkog tipa podatka

U Pythonu, sve vrijednosti različite od 0, None, "" (prazan string), [] (prazna lista) i {} (prazan rječnik) smatraju se **True**, dok se navedene smatraju **False**.

3.2. Cjelobrojni tip

Cjelobrojni tip (*int*) koristi se za pohranu cijelih brojeva, bez decimalnog dijela. Python može raditi s vrlo velikim cijelim brojevima bez potrebe za posebnom deklaracijom (slika 18).

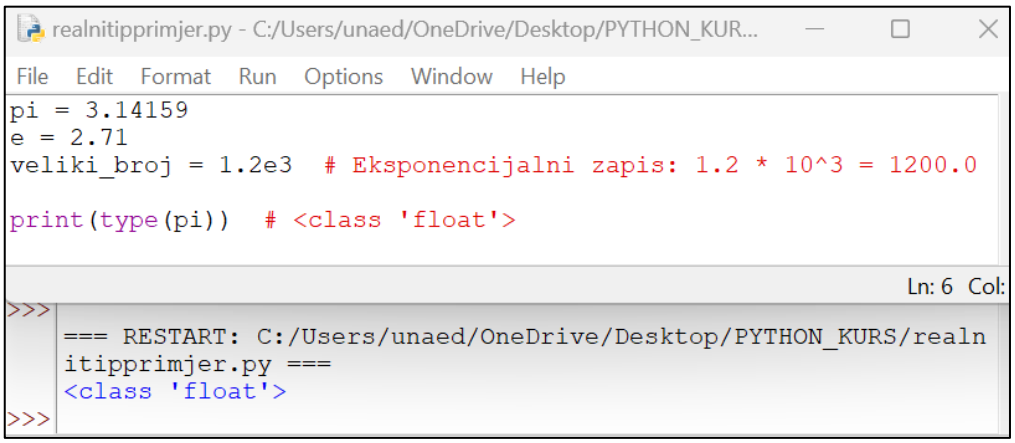


Slika 18. Primjer cjelobrojnog tipa podatka

Python automatski pretvara *int* u *float* ako je potrebno, primjerice u dijeljenju (/), gdje rezultat uvijek bude *float*, čak i ako su oba operanda cijeli brojevi.

3.3. Realni tip

Tip *float* koristi se za pohranu realnih brojeva s decimalnim dijelom, uključujući i eksponencijalni zapis, (slika 19).

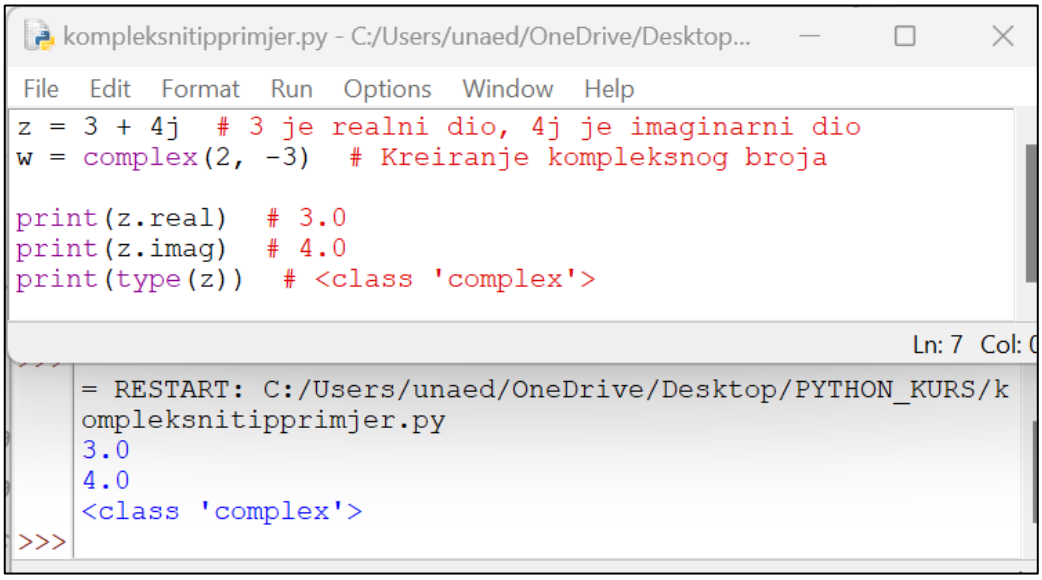


Slika 19. Primjer realno tipa podatka

Python omogućava automatsku konverziju između *int* i *float* kada je to potrebno.

3.4. Kompleksni tip

Python podržava kompleksne brojeve, koji se sastoje od realnog i imaginarnog dijela. Imaginarnim dijelom upravlja **j** (umjesto i kao u matematici), (slika 20).

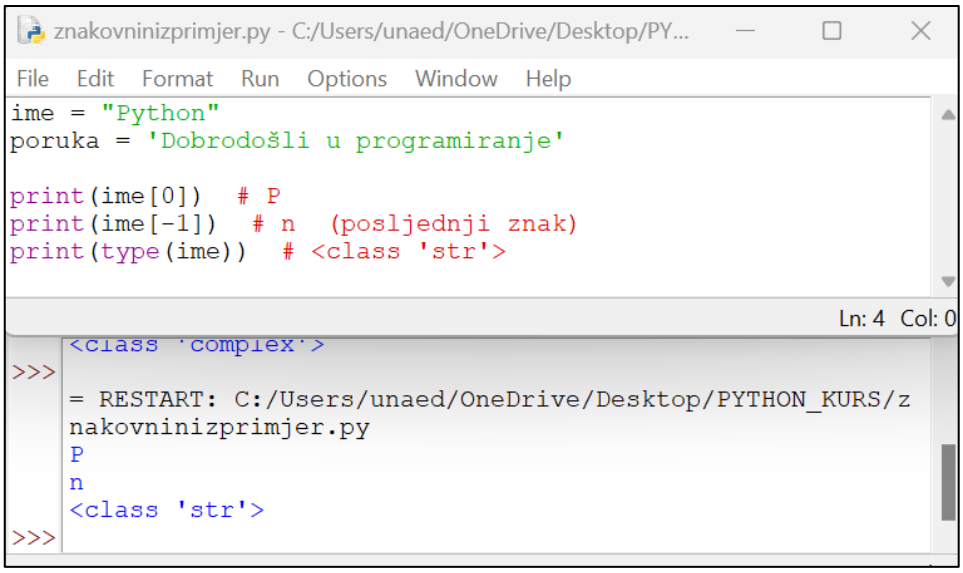


Slika 20. Primjer kompleksnog tipa podatka

Kompleksni brojevi se rijetko koriste u općem programiranju, ali su korisni u znanstvenim proračunima i signalnoj obradi.

3.5. Znakovni niz

Tip *str* koristi se za pohranu tekstualnih podataka (nizova znakova). Može se definirati jednostrukim ('...') ili dvostrukim ("...") navodnicima, (slika 21).

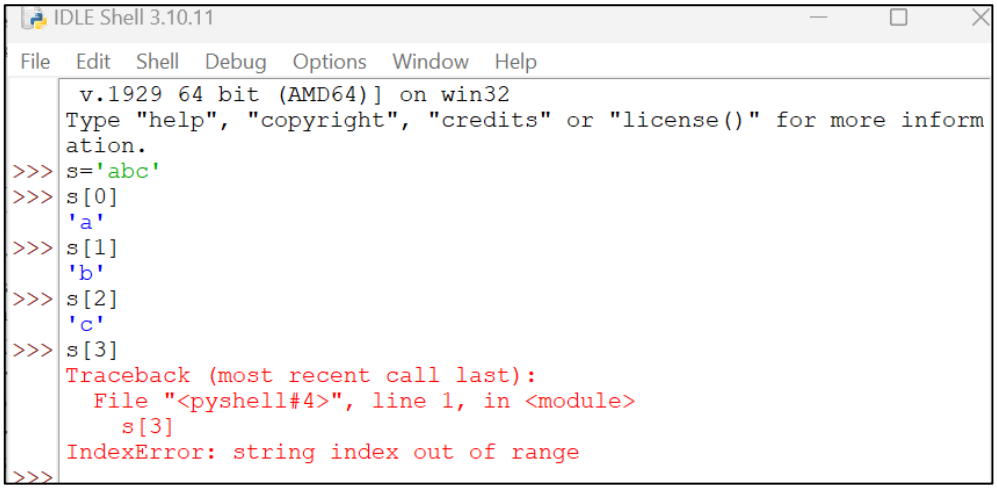


Slika 21. Primjer znakovnog tipa podatka

Znakovni nizovi su nepromjenjivi (*immutable*), što znači da ih nije moguće mijenjati nakon stvaranja, već je potrebno stvoriti novi niz.

Znakovni niz tekstualni je tip podatka koji se sastoji od uređenog niza znakova, na primjer: s='abc' ili s="abc". Neke česte operacije sa znakovnim nizovima su sljedeće: indeksiranje, dohvaćanje duljine (broj znakova), spajanje dvaju znakovnih nizova, segmentiranje, dobivanje položaja zadanog znaka, provjera pripadnosti nekog znaka znakovnom nizu i zamjena dijela znakovnog niza drugačijim sadržajem.

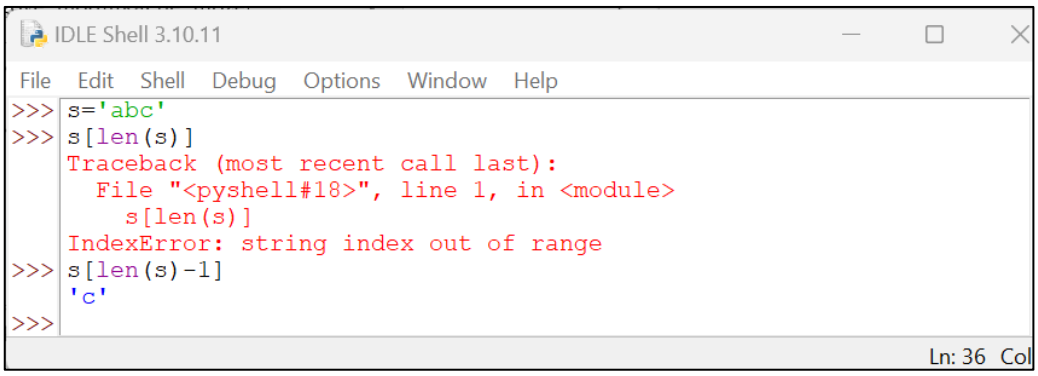
S obzirom da je znakovni niz uređeni niz znakova, njegovim znakovima možemo pristupiti preko indeksa, počevši od 0, slika 22.



Slika 22. Primjer pristupanja znakovima preko indeksa

Ovdje se varijabli s dodjeljuje vrijednost "abc", što znači da se ona sastoji od tri znaka: 'a', 'b' i 'c'. Izraz [0] pristupa prvom znaku niza, jer indeksi u Pythonu počinju od 0. Vrijednost s[0] je 'a', izraz [1] pristupa znaku niza 'b', izraz [2] pristupa znaku niza 'c', a izraz [3] uzrokuje grešku (IndexError), jer niz "abc" sadrži samo tri znaka (indeksi 0, 1 i 2). Budući da ne postoji znak s indeksom 3, Python generira grešku. Također, indeks može biti i negativan, u kojem slučaju označava poziciju s desna. Na primjer, da bismo dohvatili posljednji znak možemo zadati na primjer s[-1].

Važna osobina znakovnih nizova je ta da se oni ne mogu modificirati, tj. nepromjenjivi su. Ako neki znakovni niz želimo promijeniti, umjesto njegove modifikacije možemo napraviti novi znakovni niz koji sadrži odgovarajuće promjene. Ako želimo dobiti podatak o posljednjem znaku u zadanom znakovnom nizu možemo koristiti funkciju *len*: npr >>> len (s), slika 23.



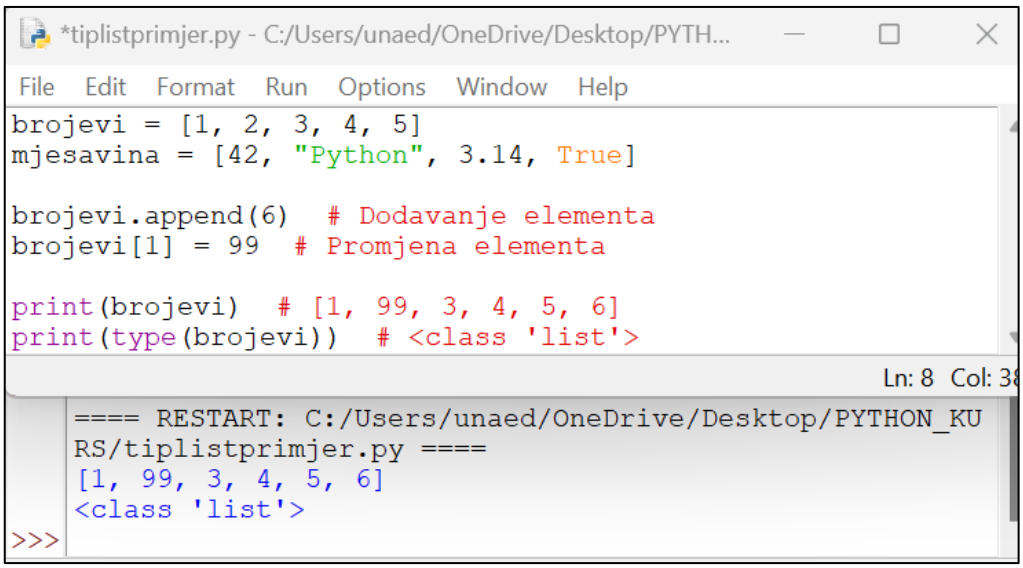
Slika 23. Primjer određivanja posljednjeg znaka u znakova u nizu

Na slici 23 se može vidjeti da naredba *s[len(s)]* rezultira greškom, jer indeksiranje u Pythonu počinje sa 0, pa je ispravan način korištenje naredbe *s[len(s)-1]*.

3.6. Dva uređena niza

3.6.1. Lista (*list*)

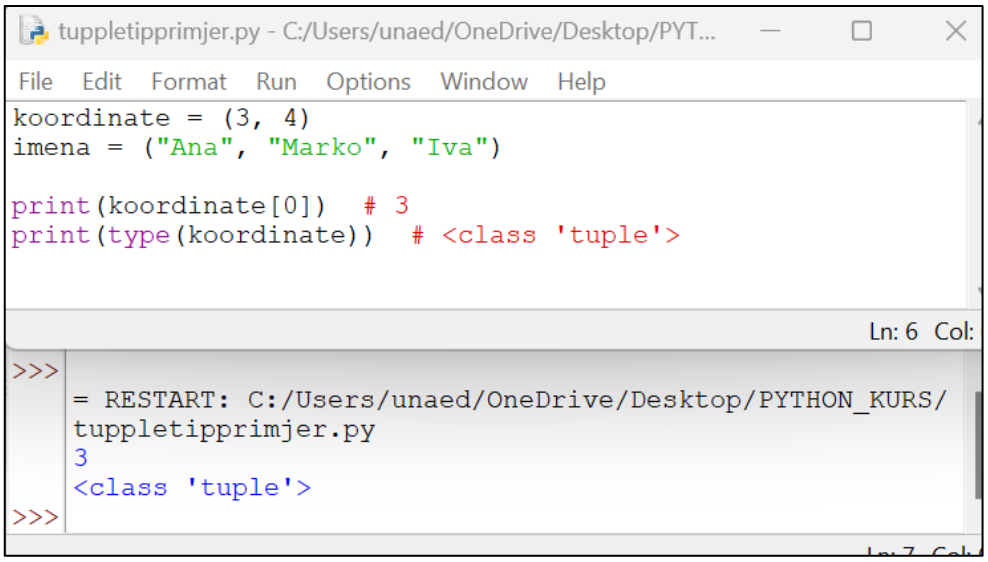
Liste su promjenjive (*mutable*) kolekcije elemenata koje mogu sadržavati različite tipove podataka, (slika 24).



Slika 24. Primjer list tipa podatka

3.6.2. Torka (*tuple*)

Torke su slične listama, ali su nepromjenjive (*immutable*), što znači da nakon stvaranja ne mogu biti izmijenjene (slika 25).

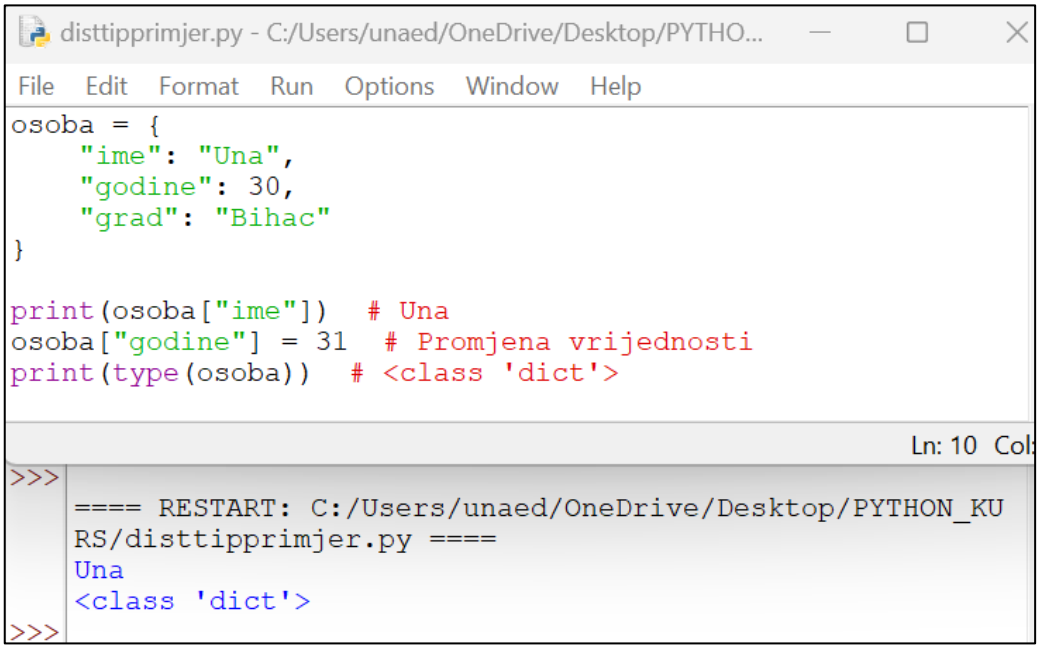


Slika 25. Primjer tuple tipa podatka

3.7. Dvije neuređene kolekcije vrijednosti

3.7.1. Rječnik (*dict*)

Rječnik (*dictionary*) je kolekcija ključeva i vrijednosti, gdje svaki ključ mora biti jedinstven. Koristi se za brzi dohvat podataka, (slika 26).

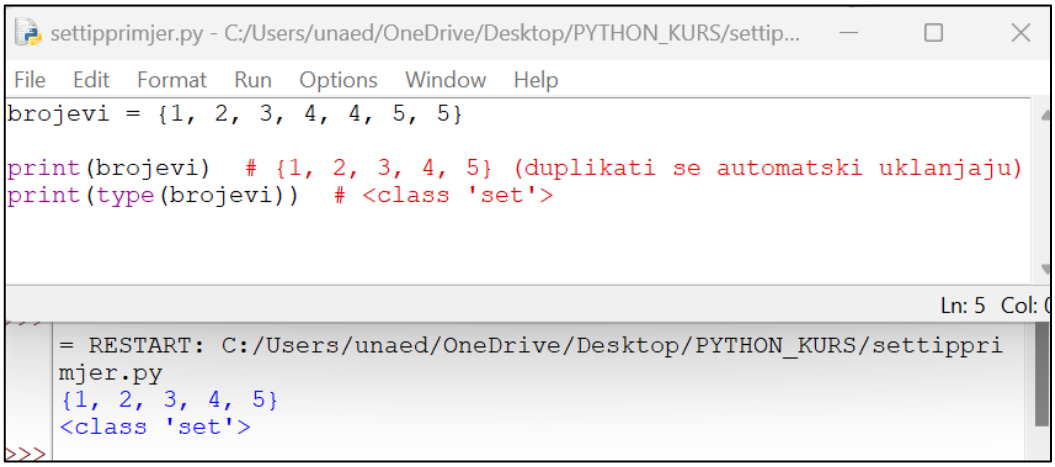


Slika 26. Primjer dict tipa podatka

Rječnici su izuzetno korisni kada radimo s podacima koji imaju strukturu ključ-vrijednost, poput JSON formata.

3.7.2. Skup (*set*)

Skup (set) je kolekcija jedinstvenih elemenata, što znači da ne može sadržavati duplikate, (slika 27).



Slika 27. Primjer set tipa podatka

Setovi su korisni za provjeru jedinstvenih vrijednosti i izvođenje operacija skupova, poput presjeka (&) i unije (|).

Python omogućava rad s različitim tipovima podataka, od jednostavnih numeričkih i logičkih vrijednosti do složenih kolekcija podataka poput lista, torki i rječnika. Razumijevanje razlika između promjenjivih (list, dict, set) i nepromjenjivih (tuple, str) tipova ključno je za pisanje učinkovitog i sigurnog koda.

4 FUNKCIJE

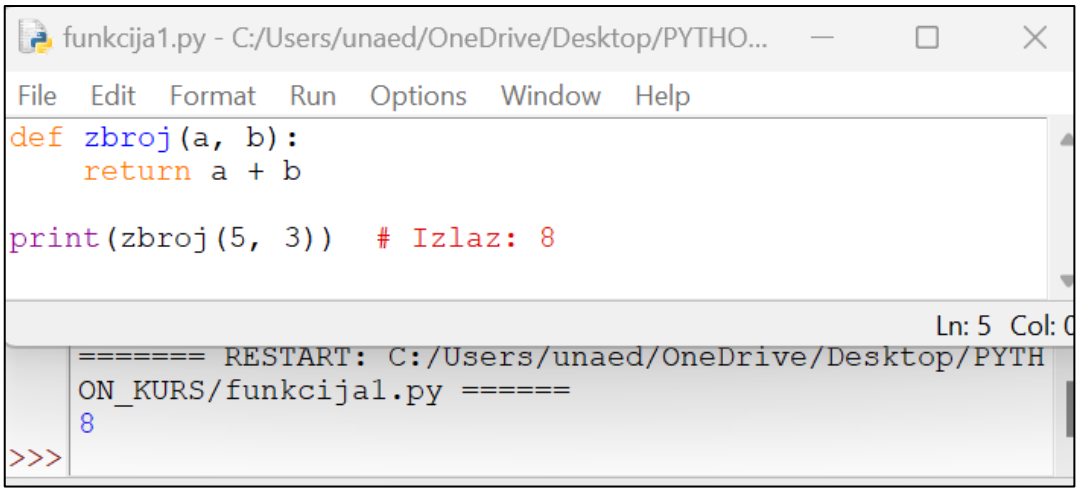
Python podržava različite vrste funkcija koje omogućuju organizaciju i ponovnu upotrebu koda.

4.1. Standardne (definirane) funkcije

Ove funkcije definiramo pomoću ključne riječi *def*. Sintaksa je sljedeća:

```
def ime_funkcije(parametri):
    """ Dokumentacijski string (opcionalno) """
    naredbe
    return vrijednost    # (opcionalno)
```

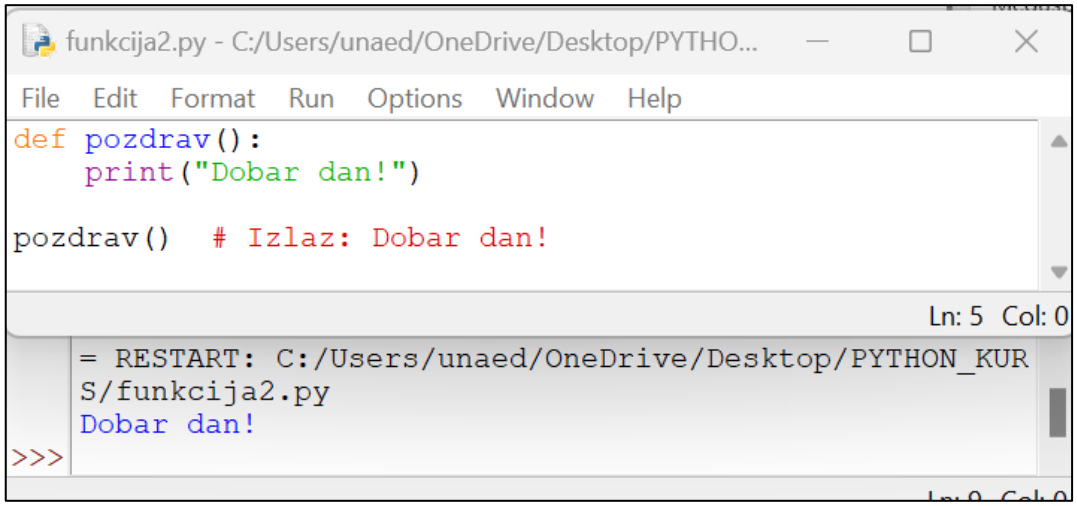
Primjer, slika 28.



Slika 28. Primjer standardne funkcije sa def

4.2. Funkcije bez parametara

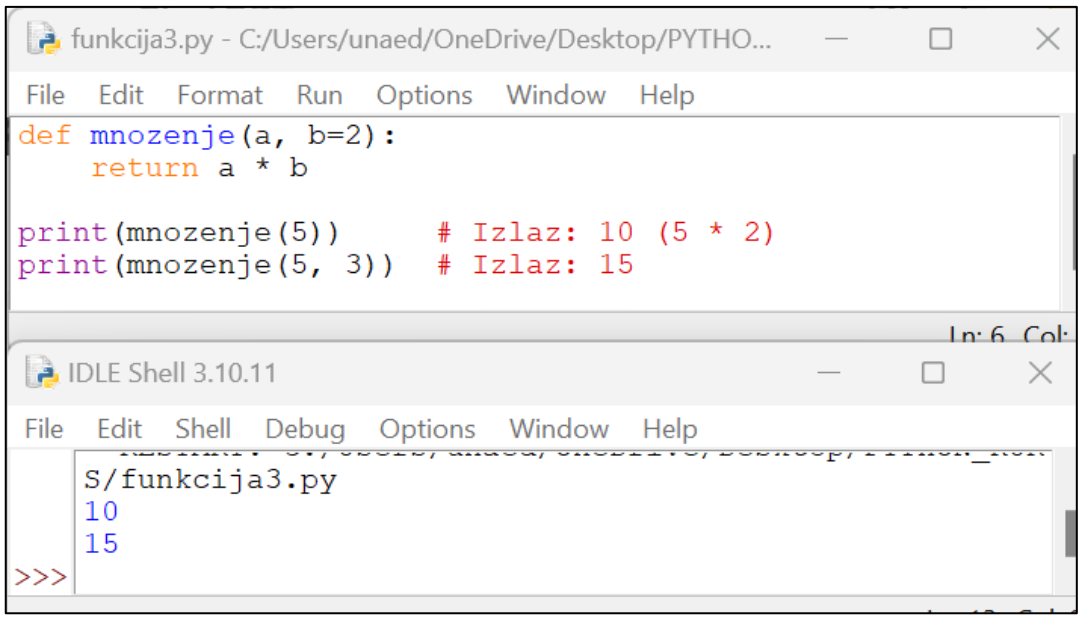
Ako funkcija ne prima parametre, može se jednostavno pozvati po imenu, slika 29.



Slika 29. Primjer funkcije bez parametara

4.3. Funkcije sa podrazumijevanim vrijednostima parametara

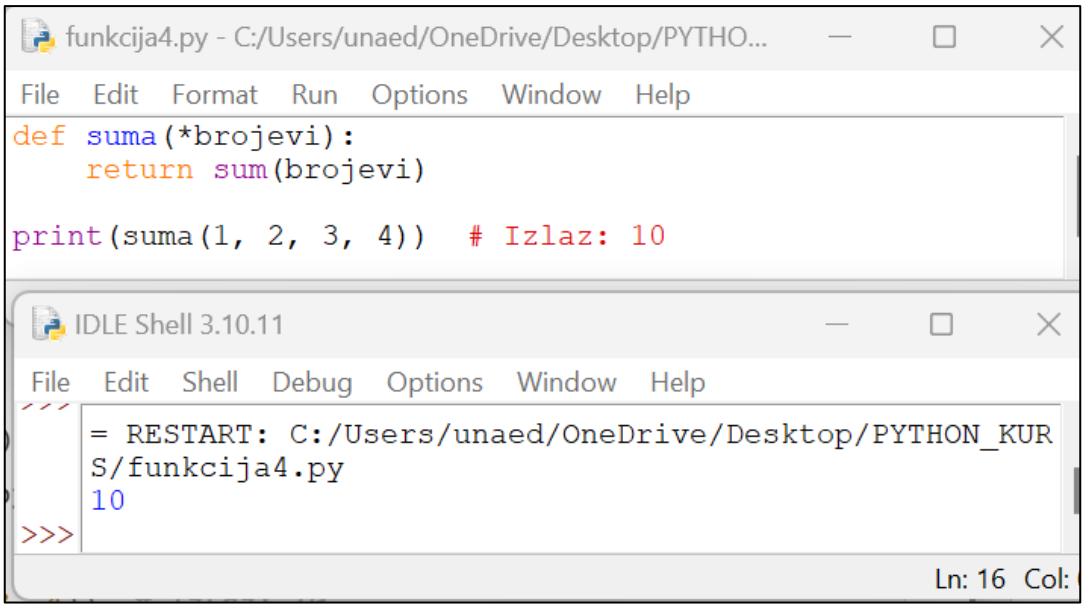
Ako se argument ne proslijedi, koristi se podrazumijevana vrijednost, primjer slika 30.



Slika 30. Primjer funkcije sa podrazumijevanim vrijednostima parametara

4.4. Funkcije sa proizvoljnim brojem argumenata (**args*)

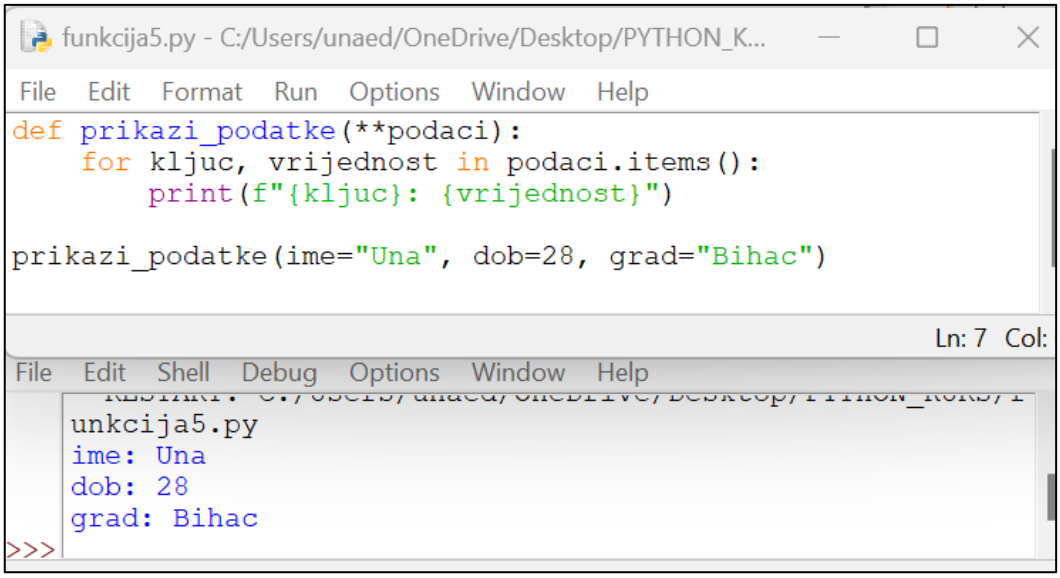
Omogućuje prosljeđivanje više argumenata koji se tretiraju kao tuple, primjer slika 31.



Slika 31. Primjer funkcije sa proizvoljnim brojem argumenata (**args*)

4.5. Funkcije sa proizvoljnim imenovanim argumentima (**kwargs)

Omogućuje prosljeđivanje imenovanih argumenata kao rječnik (dict), primjer 32.



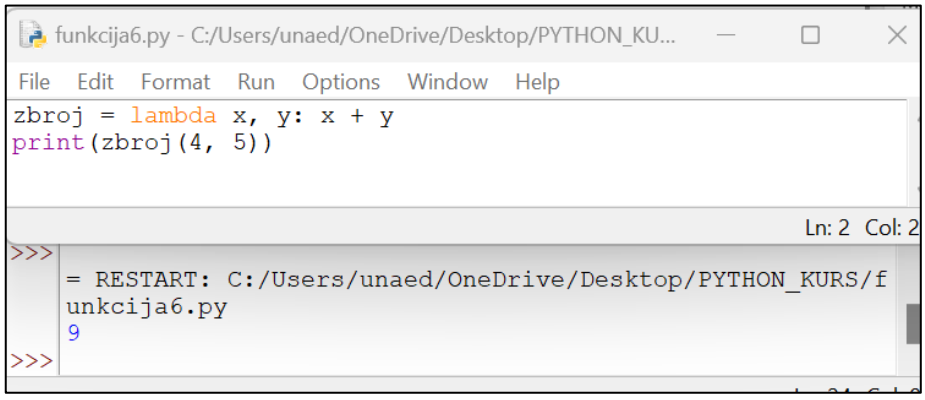
*Slika 32. Primjer funkcije sa proizvoljnim imenovanim argumentima (**kwargs)*

4.6. Lambda (anonimne) funkcije

Kompaktne funkcije koje se definiraju pomoću lambda izraza. Sintaksa je:

```
lambda argumenti: izraz
```

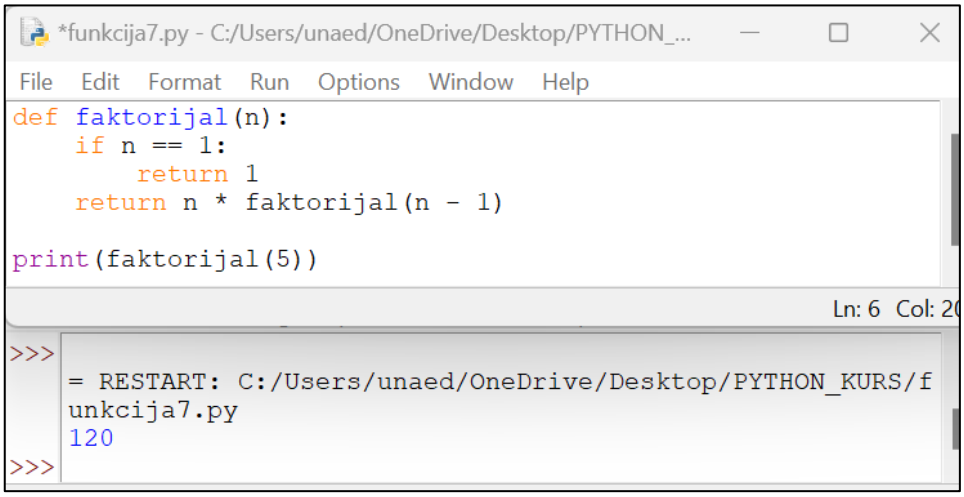
Primjer slika 33.



Slika 33. Primjer lambda (anonimne) funkcije

4.7. Rekurzivne funkcije

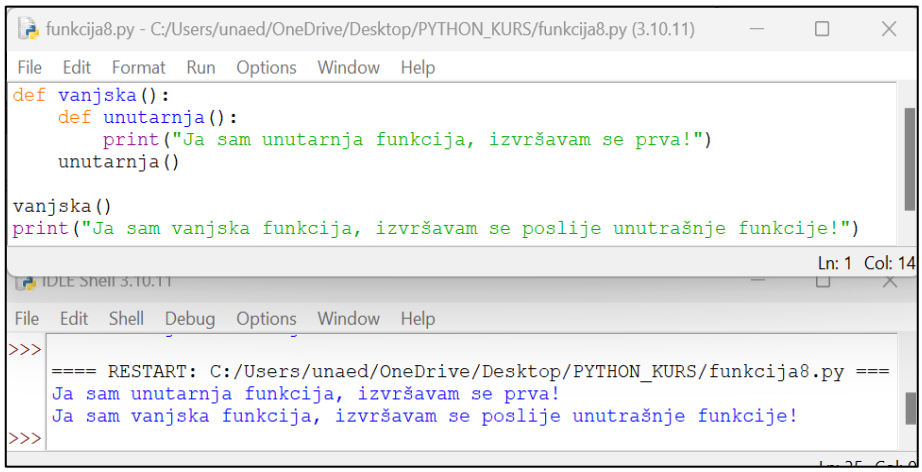
Funkcije koje pozivaju same sebe, primjer slika 34.



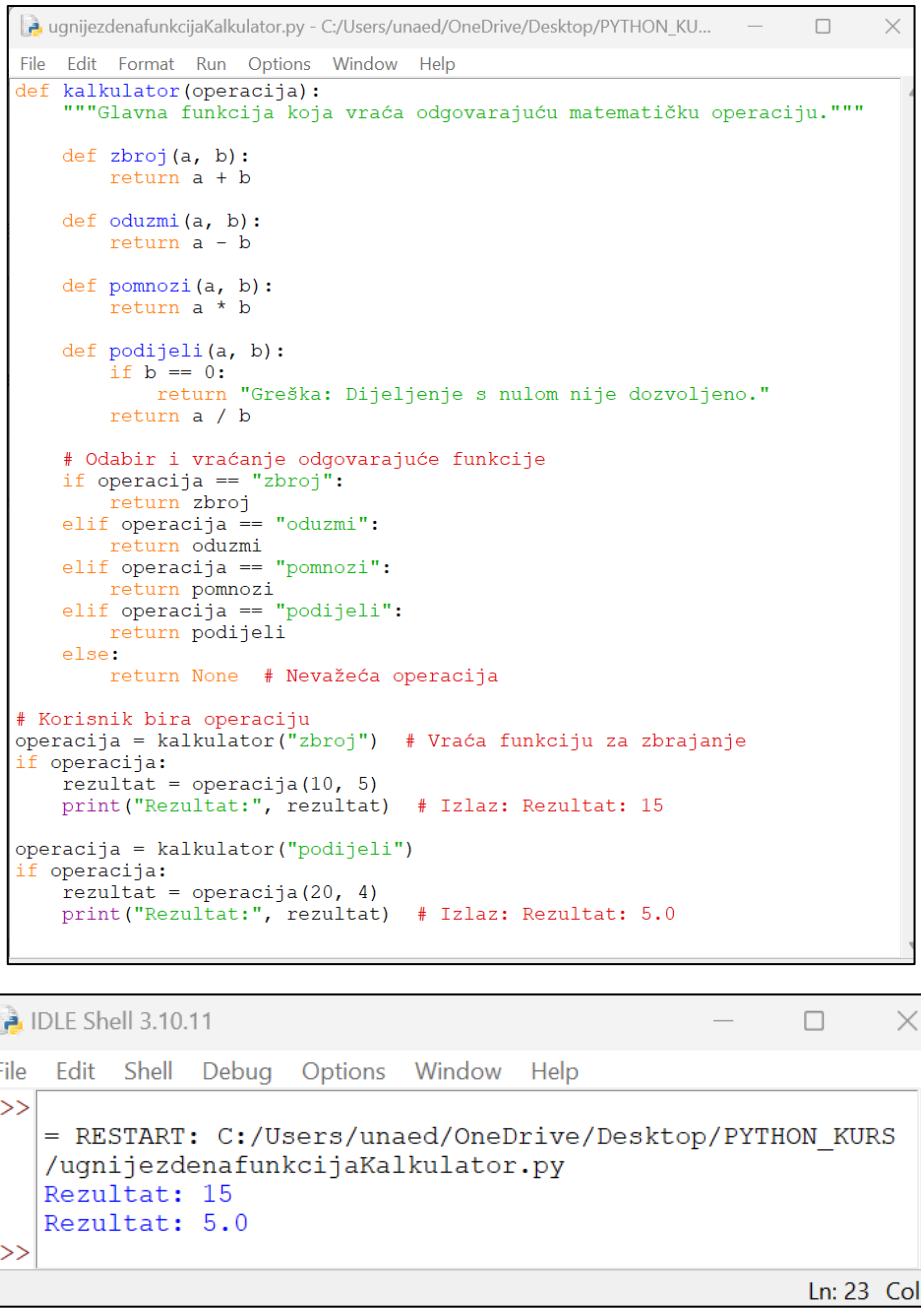
Slika 34. Primjer rekurzivne funkcije

4.8. Funkcije unutar funkcija (ugniježdene funkcije)

Funkcija definirana unutar druge funkcije, primjer slika 35 i 36.



Slika 35. Primjer ugniježdene funkcije



Slika 36. Primjer ugniježđene funkcije

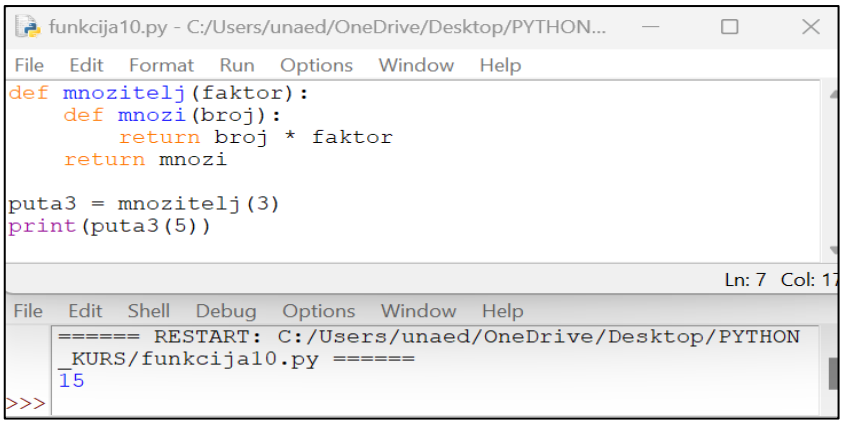
4.9. Funkcije kao argumenti drugih funkcija

U Pythonu, funkcije se mogu prosljeđivati kao argumenti drugim funkcijama, primjer slika 37.

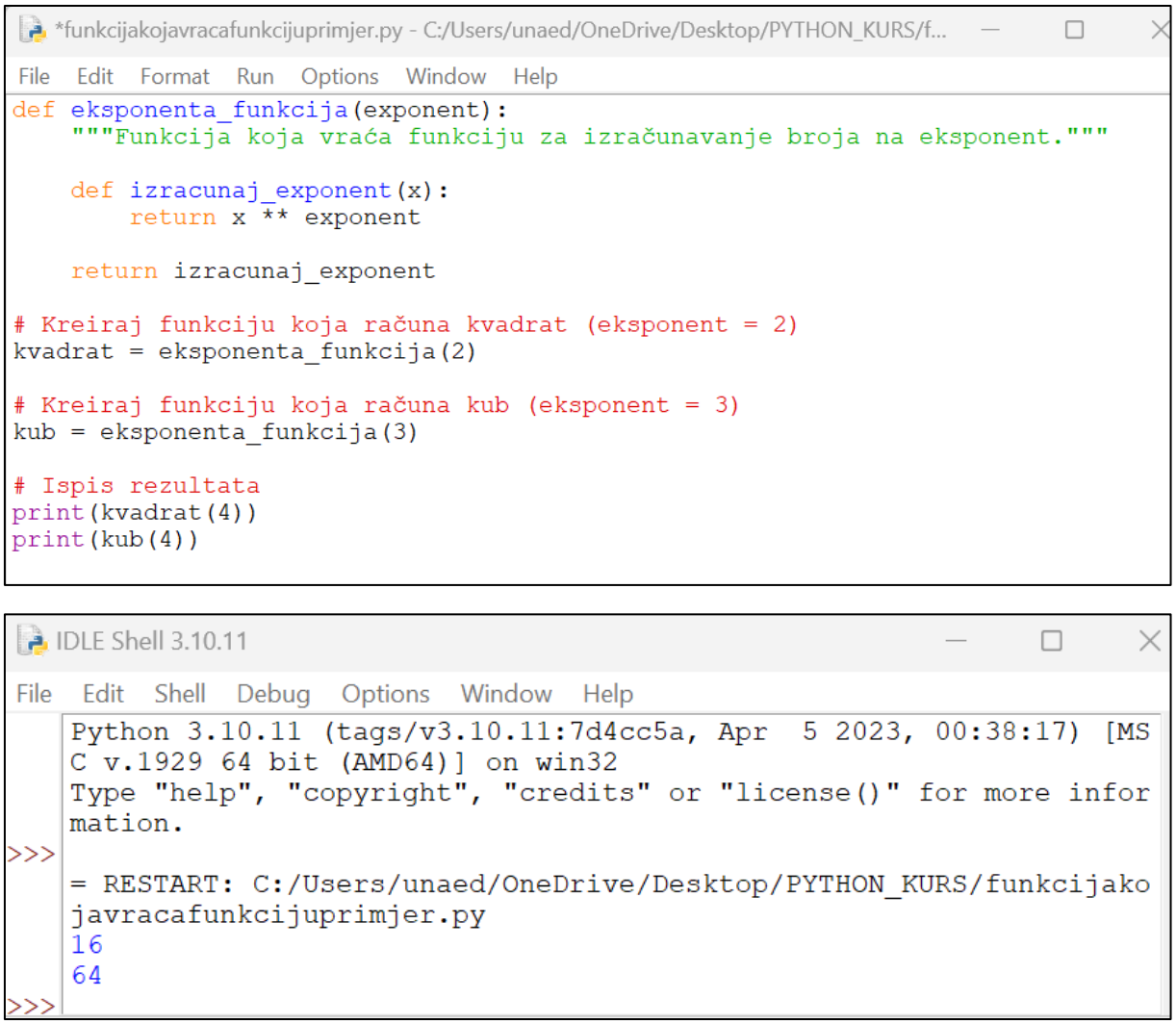
Slika 37. Primjer funkcije kao argument druge funkcije

4.10. Funkcije koje vraćaju druge funkcije

Funkcija može vratiti drugu funkciju, primjer slike 38 i 39.



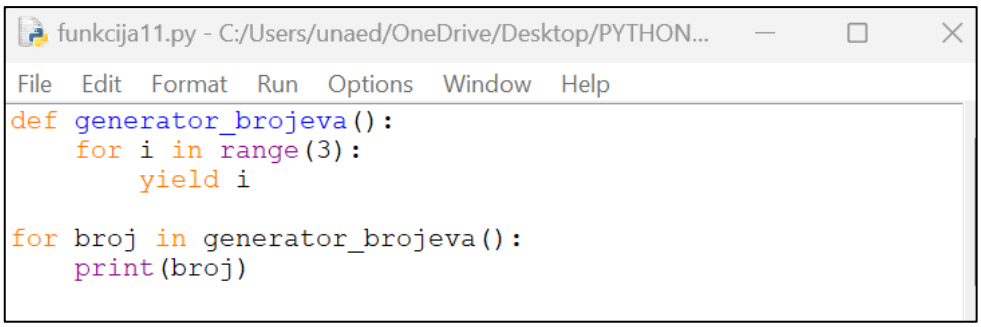
Slika 38. Primjer funkcije koja vraća drugu funkciju



Slika 39. Primjer funkcije koja vraća drugu funkciju

4.11. Generator funkcije (yield)

Umjesto return, koriste yield za stvaranje iteratora, primjer slika 40 i 41.



```
File  Edit  Shell  Debug  Options  Window  Help
>>>
= RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS
/funkcija11.py
0
1
2
>>>
```

Slika 40. Primjer generator funkcije

generatorgunkcijaprimjer.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/generatorgu...

File Edit Format Run Options Window Help

```
def pozdrav_generator(jezik):
    """Vraća funkciju koja generira pozdrav na određenom jeziku."""

    def hrvatski(ime):
        return f"Bok, {ime}!"

    def engleski(ime):
        return f"Hello, {ime}!"

    def njemacki(ime):
        return f"Hallo, {ime}!"

    def talijanski(ime):
        return f"Ciao, {ime}!"

    def spanski(ime):
        return f"¡Hola, {ime}!"

    # Vraćanje odgovarajuće funkcije
    if jezik == "hr":
        return hrvatski
    elif jezik == "en":
        return engleski
    elif jezik == "de":
        return njemacki
    elif jezik == "it":
        return talijanski
    elif jezik == "es":
        return spanski
    else:
        return lambda ime: f"Nepoznat jezik, {ime}!"  # Default poruka

# Testiranje s različitim jezicima
jezici = ["hr", "en", "de", "it", "es", "fr"]  # Dodali smo i nepoznat jezik "fr"

for jezik in jezici:
    odabrani_pozdrav = pozdrav_generator(jezik)
    print(odabrani_pozdrav("Una"))
```

IDLE Shell 3.10.11

File Edit Shell Debug Options Window Help

```
= RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/generat
orgunkcijaprimjer.py
Bok, Una!
Hello, Una!
Hallo, Una!
Ciao, Una!
¡Hola, Una!
Nepoznat jezik, Una!
>>>
```

Slika 41. Primjer generator funkcije

Ako bi htjeli da omogućimo korisniku da odabere na kojem jeziku da ispiše ime, onda bi kod izgledao kao na slici 42.

generatorgunkcijaprimjer2.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/generatorgunkcijaprimjer2.py (3.10.11)

File Edit Format Run Options Window Help

```
def pozdrav_generator(jezik):
    """Vraća funkciju koja generira pozdrav na određenom jeziku."""

    def hrvatski(ime):
        return f"Bok, {ime}!"

    def engleski(ime):
        return f"Hello, {ime}!"

    def njemacki(ime):
        return f"Hallo, {ime}!"

    def talijanski(ime):
        return f"Ciao, {ime}!"

    def spanski(ime):
        return f"¡Hola, {ime}!"

    # Vraćanje odgovarajuće funkcije
    if jezik == "hr":
        return hrvatski
    elif jezik == "en":
        return engleski
    elif jezik == "de":
        return njemacki
    elif jezik == "it":
        return talijanski
    elif jezik == "es":
        return spanski
    else:
        return lambda ime: f"Nepoznat jezik, {ime}!"  # Default poruka

# Korisnik unosi jezik
print("Dostupni jezici: hr (hrvatski), en (engleski), de (njemački), it (talijanski), es (španjolski)")

odabrani_jezik = input("Unesite kod jezika za pozdrav (npr. 'hr', 'en'): ").strip().lower()
ime = input("Unesite vaše ime: ").strip()

# Pozivamo funkciju za odabrani jezik
odabrani_pozdrav = pozdrav_generator(odabrani_jezik)
print(odabrani_pozdrav(ime))
```

IDLE Shell 3.10.11

File Edit Shell Debug Options Window Help

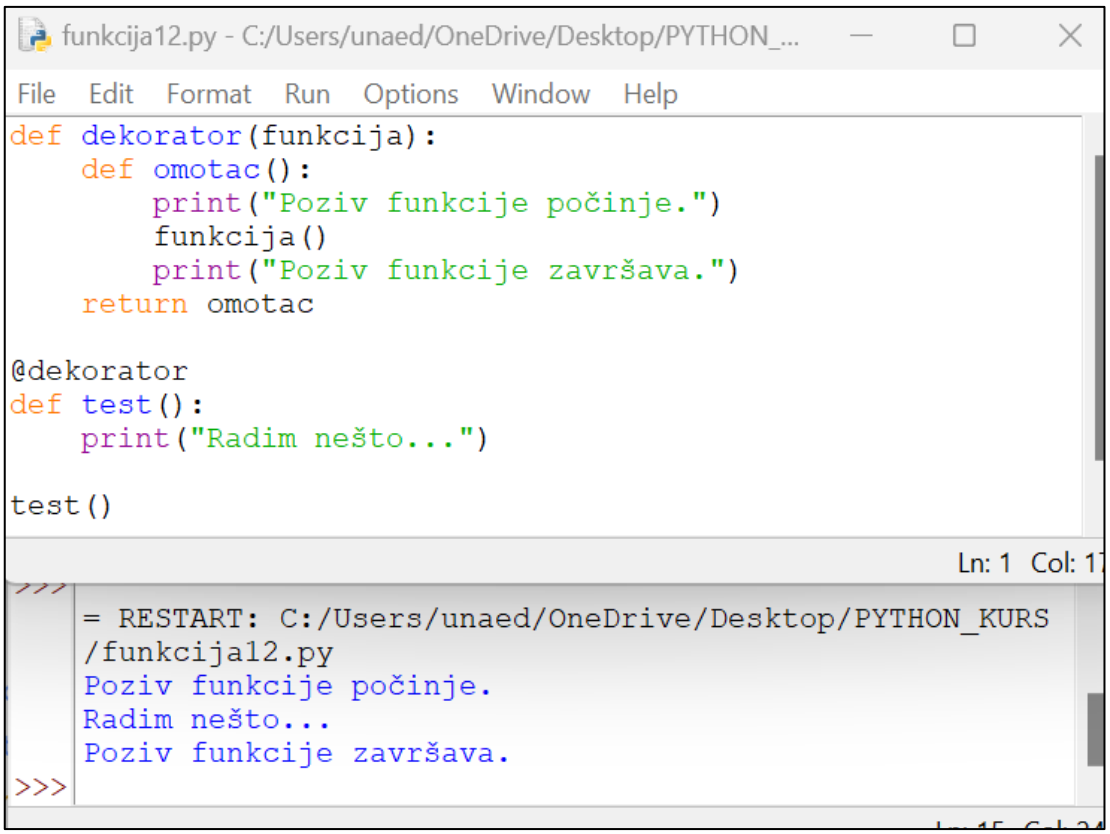
```
>>> 
= RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/generat
orgunkcijaprimjer2.py
Dostupni jezici: hr (hrvatski), en (engleski), de (njemački),
it (talijanski), es (španjolski)
Unesite kod jezika za pozdrav (npr. 'hr', 'en'): es
Unesite vaše ime: Una
¡Hola, Una!
>>> 
```

Ln: 38 Col:

Slika 42. Primjer generator funkcije

4.12. Funkcije dekoratori

Dekodatori su funkcije koje mijenjaju ponašanje drugih funkcija, primjer slika 43.



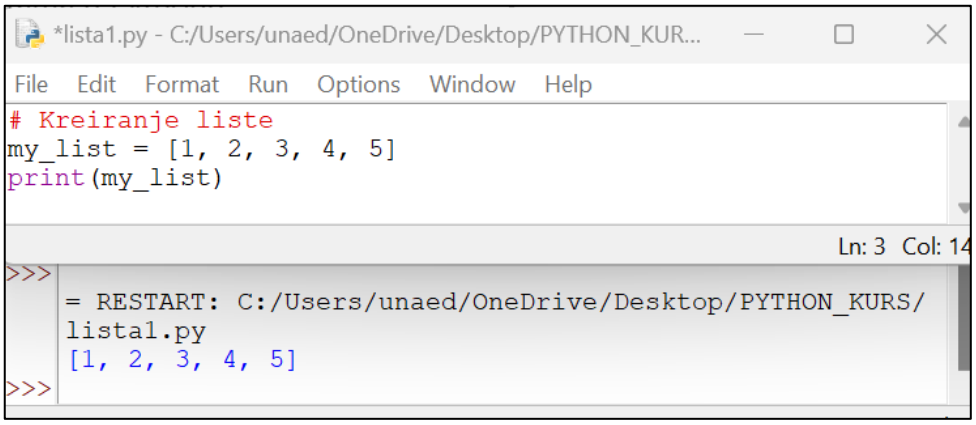
Slika 43. Primjer funkcije dekorator

5 LISTE

Liste u Pythonu su jedan od osnovnih tipova podataka i omogućuju pohranu sekvencijalnih podataka. Liste su uređene kolekcije podataka, što znači da se elementi čuvaju u određenom redoslijedu, i može im se pristupiti koristeći indekse.

5.1. Kreiranje liste

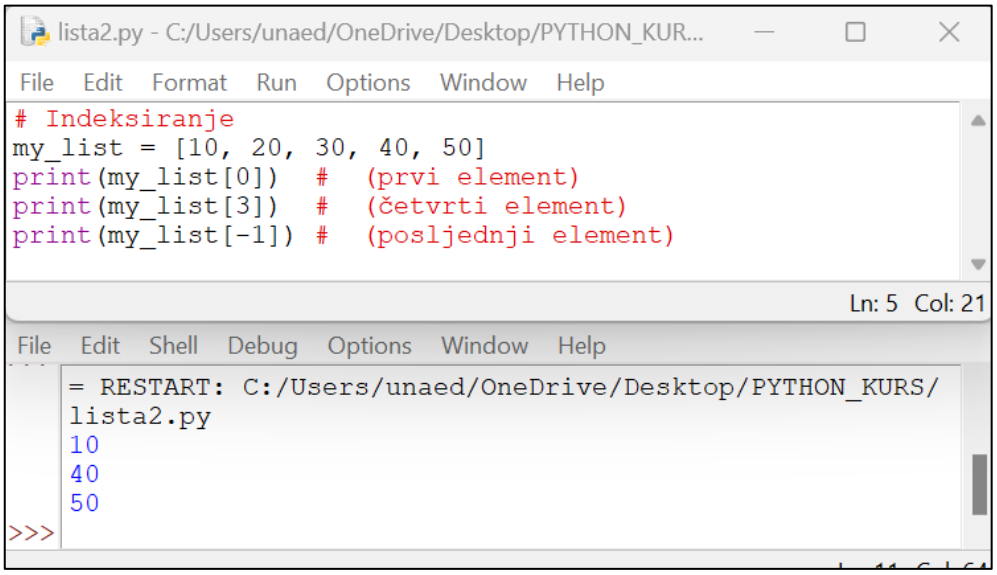
Lista se može kreirati pomoću uglatih zagrada [ ], a elementi unutar liste su odvojeni zarezom, slika 44.



Slika 44. Primjer kreiranja liste

5.2. Indeksiranje lista

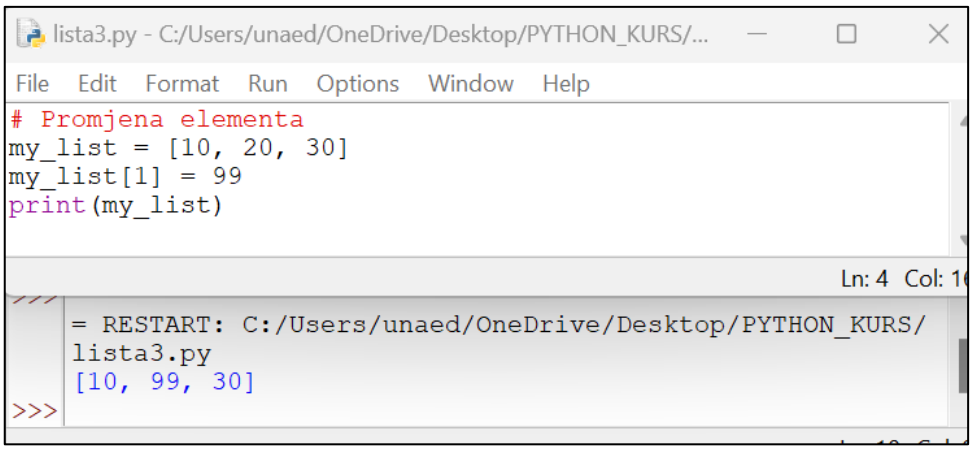
Indeksiranje lista u Pythonu počinje od **0**. Može se pristupiti svakom elementu pomoću indeksa, slika 45.



Slika 45. Primjer indeksiranja liste

5.3. Promjena elemenata u listi

Listama u Pythonu mogu se mijenjati elementi, slika 45.

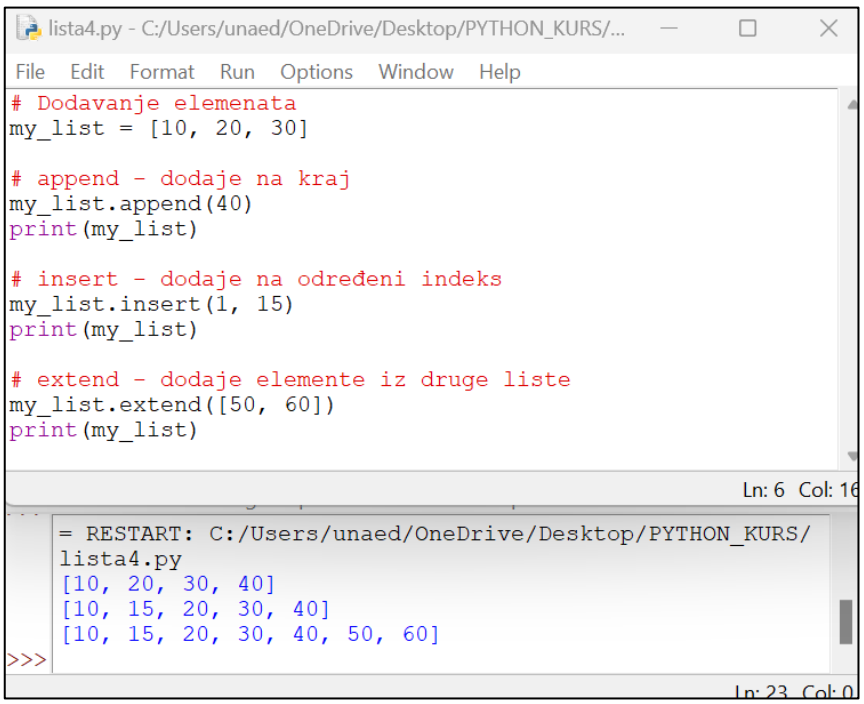


Slika 46. Primjer promjene elemenata u listi

5.4. Dodavanje elemenata u listu

Postoji nekoliko načina za dodavanje novih elemenata u listu, slika 47.

- *append ()* – dodaje element na kraj liste
- *insert ()* – umetanje elementa na određenu poziciju
- *extend ()* – spaja dvije liste

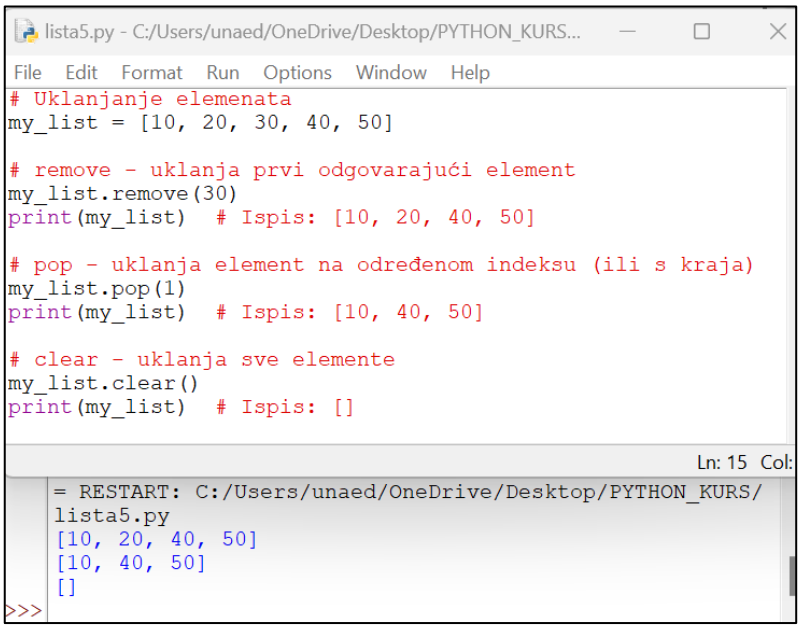


Slika 47. Primjer dodavanja elemenata u listu

5.5. Uklanjanje elemenata iz liste

Za uklanjanje elemenata možete koristiti (slika 48):

- *remove ()* – uklanja prvi element koji odgovara vrijednosti
- *pop ()* – uklanja element s određenog indeksa (ili s kraja ako nije naveden indeks)
- *clear ()* – uklanja sve elemente iz liste

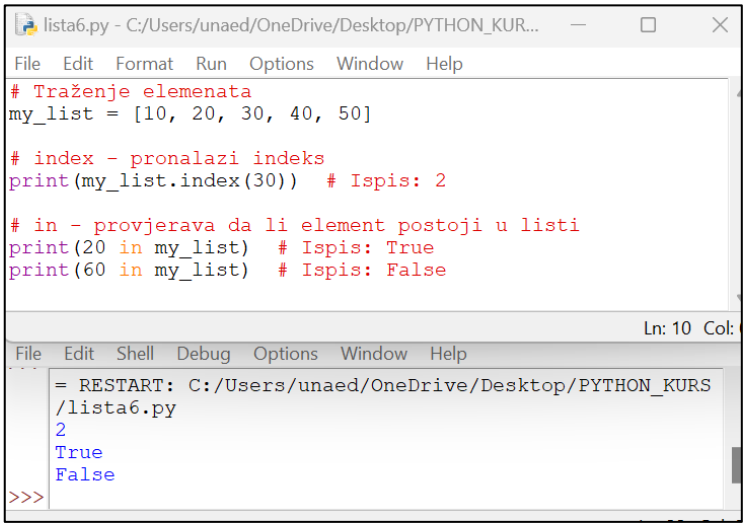


Slika 48. Primjer uklanjanja elemenata iz liste

5.6. Traženje elemenata u listi

Za pronalaženje elementa u listi možete koristiti (slika 49):

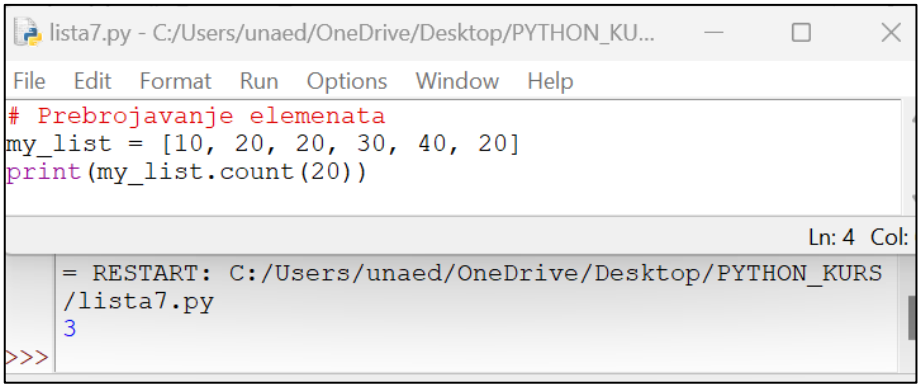
- *indeks ()* – vraća indeks prvog pojavljivanja elementa
- *in* – provjerava je li element u listi



Slika 49. Primjer traženja elemenata u listi

5.7. Prebrojavanje elemenata u listi

Koristi se metoda *count()* za brojanje koliko puta se element pojavljuje u listi, slika 50.

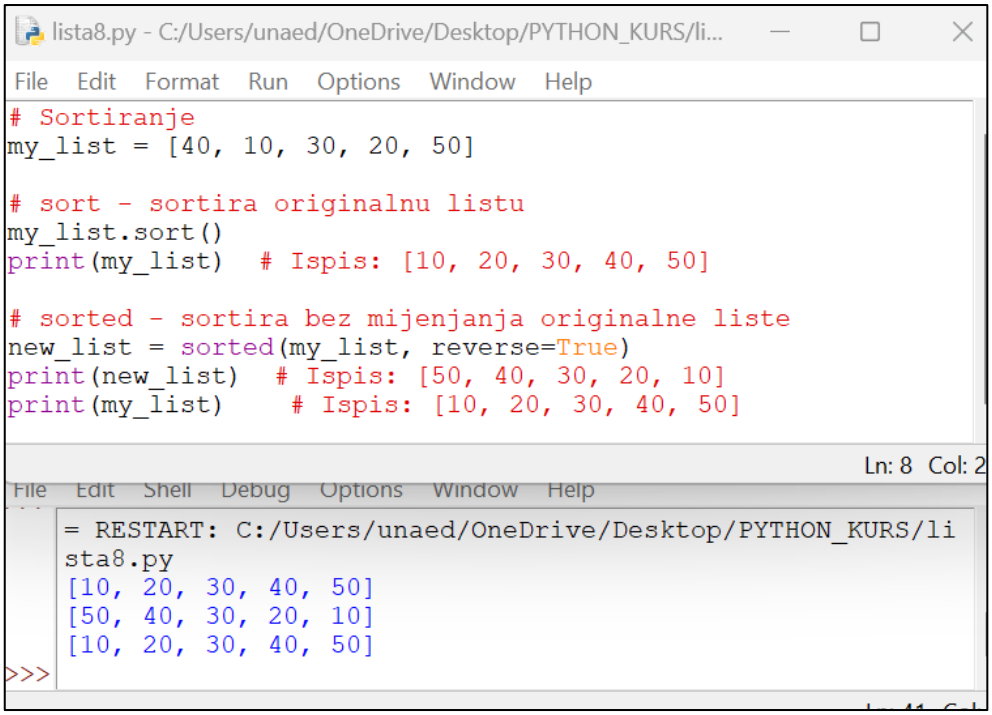


Slika 50. Primjer prebrojavanja elemenata u listi

5.8. Sortiranje liste

Metode za sortiranje, (slika 51):

- *sort()* – mijenja originalnu listu
- *sorted()* – vraća novu sortiranu listu bez mijenjanja originalne

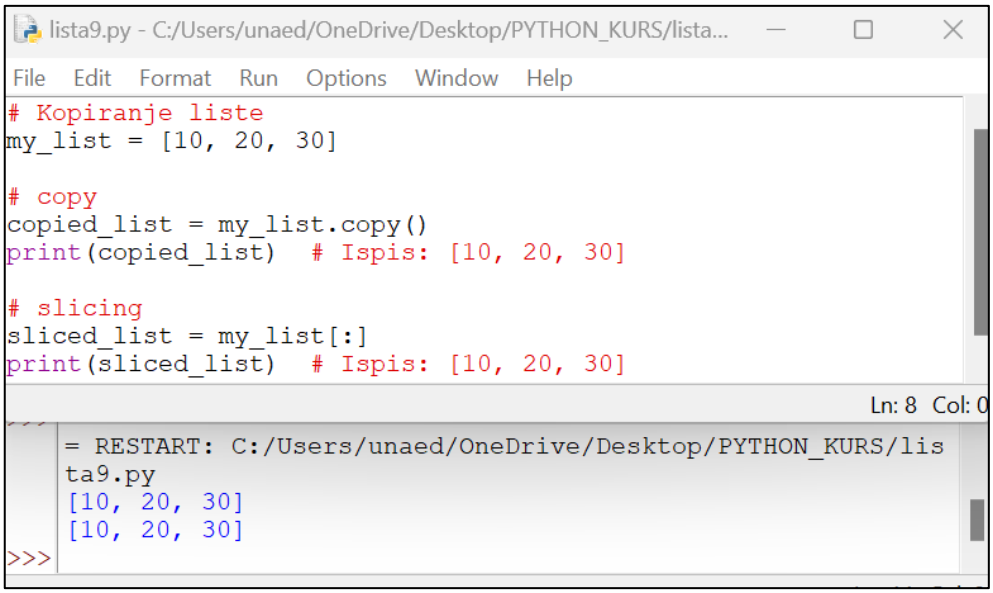


Slika 51. Primjer sortiranja liste

5.9. Kopiranje liste

Za kopiranje liste koristi, (slika 52):

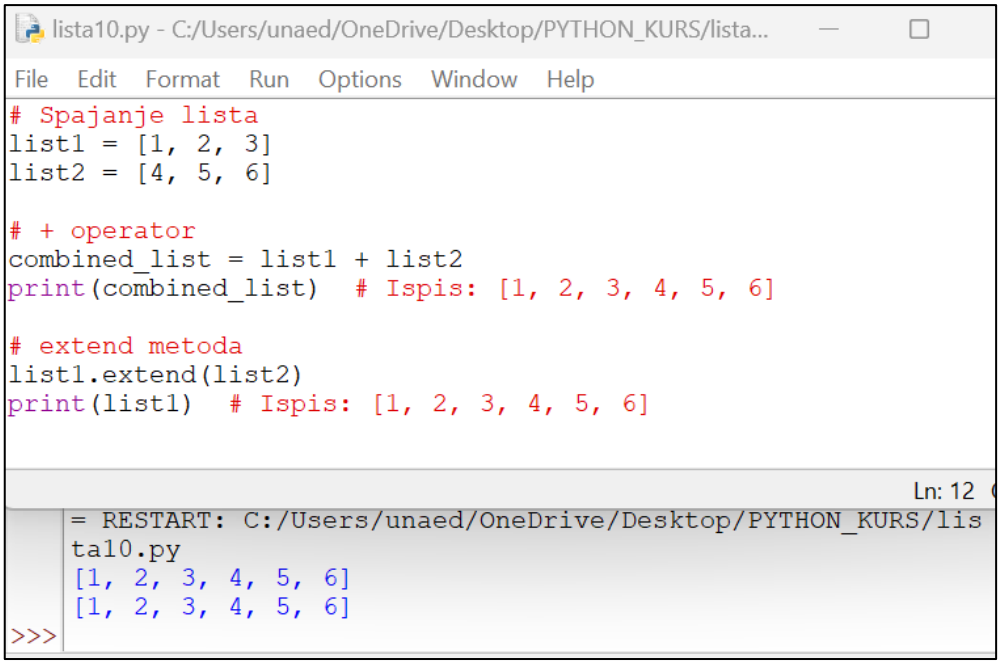
- *copy()* – za kopiranje cijele liste.
- *slicing* – pomoću sintakse [:].



Slika 52. Primjer kopiranja liste

5.10. Spajanje lista

Listu se može spojiti pomoću **+** operatora ili **extend()** metode, slika 53.



Slika 53. Primjer spajanja lista

5.11. List Comprehension ili generiranje listi

List Comprehension (ili generiranje listi) je način za stvaranje novih lista koristeći kraću i efikasniju sintaksu u odnosu na klasične for petlje. Koristi se za generiranje lista na temelju postojećih podataka u jednoj liniji koda, čineći kod čitljivijim i bržim.

List Comprehension je moćna tehnika u Pythonu koja omogućava brzo kreiranje lista s manje koda. Koristi se za:

- Generiranje novih listi pomoću for petlje.
- Filtriranje podataka pomoću if uvjeta.
- Transformaciju elemenata pomoću izraza unutar liste.

Osnovna sintaksa je:

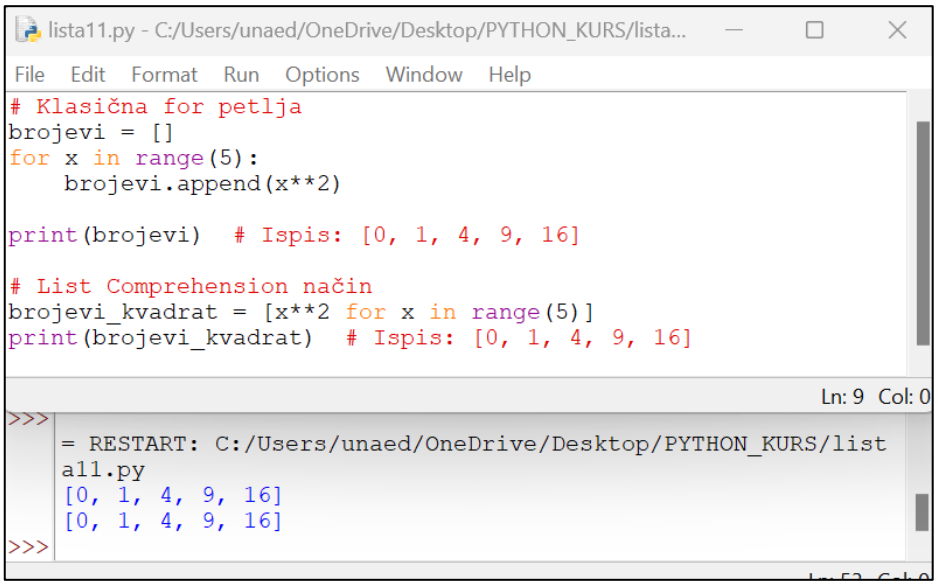
```
nova_lista=[izraz for element in kolekcija if uvijet]
```

Gdje je:

- **izraz** – operacija koja se želi primijeniti na svaki element
- **for element in kolekcija** – iteracija kroz kolekciju (lista, range, string, itd.)
- **if uvjet** *(opcionalno)* – filtriranje elemenata koji zadovoljavaju uvjet

5.11.1. Generiranje liste brojeva

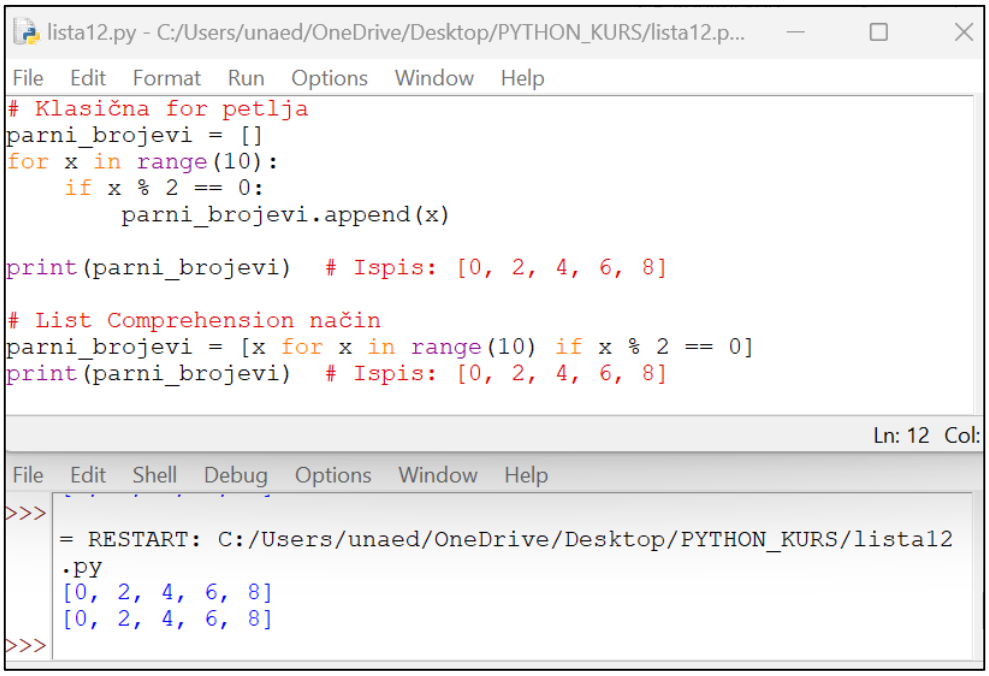
Ekvivalent for petlji, ali u kraćoj formi, slika 54.



Slika 54. Primjer generiranja liste brojeva

5.11.2. Filtriranje parnih brojeva

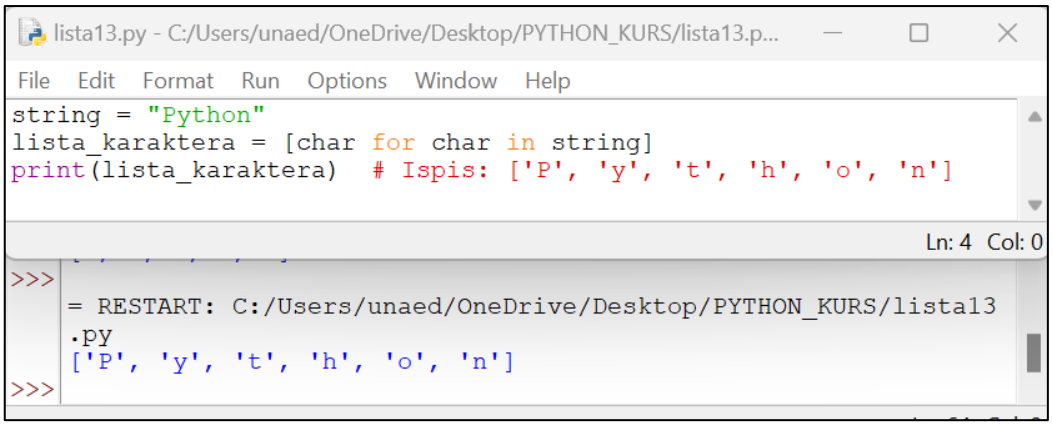
Dodavanje **if** uvjeta, slika 55.



Slika 55. Primjer filtriranja liste brojeva

5.11.3. Konverzija stringa u listu karaktera

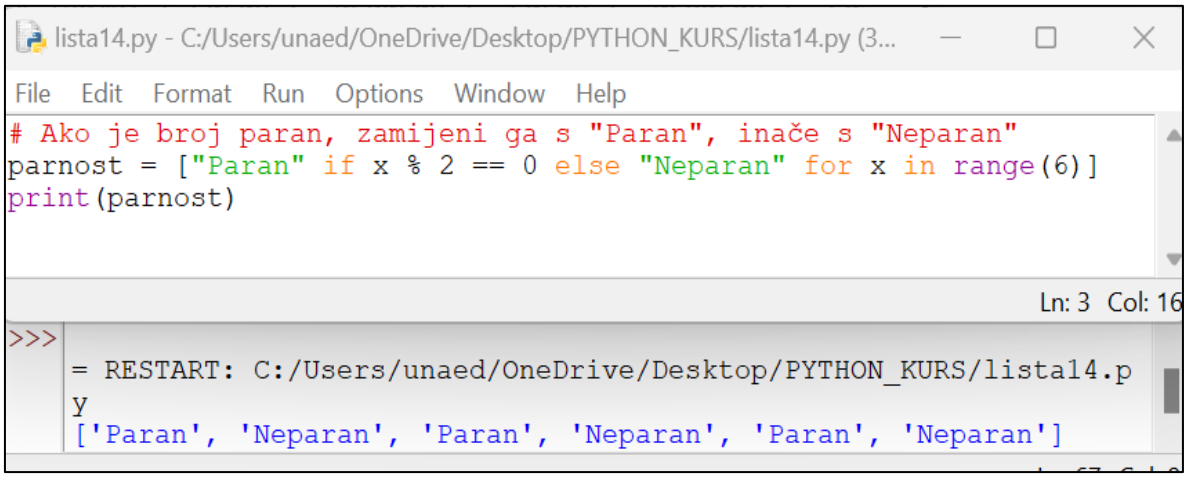
Primjer konverzije stringa u listu karaktera je prikazan na slici 56.



Slika 56. Primjer konverzije stringa u listu karaktera

5.11.4. Zamjena if – else u List Comprehension

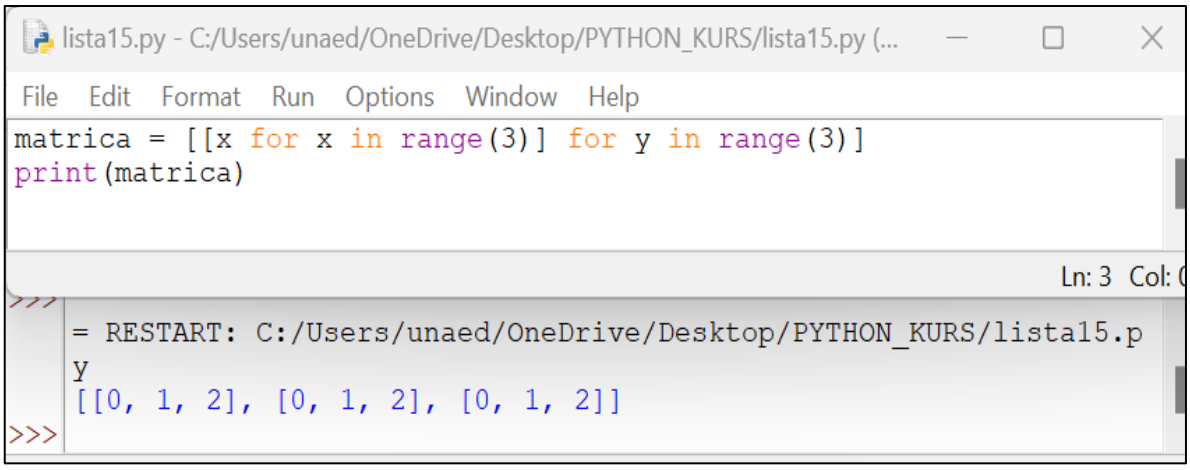
Primjer za zamjenu if – else u List Comprehension je dat na slici 57.



Slika 57. Primjer zamjene if – else u List Comprehension

5.11.5. Ugniježdeni List Comprehension

Primjer za Ugniježdeni List Comprehension je dat na slici 58.



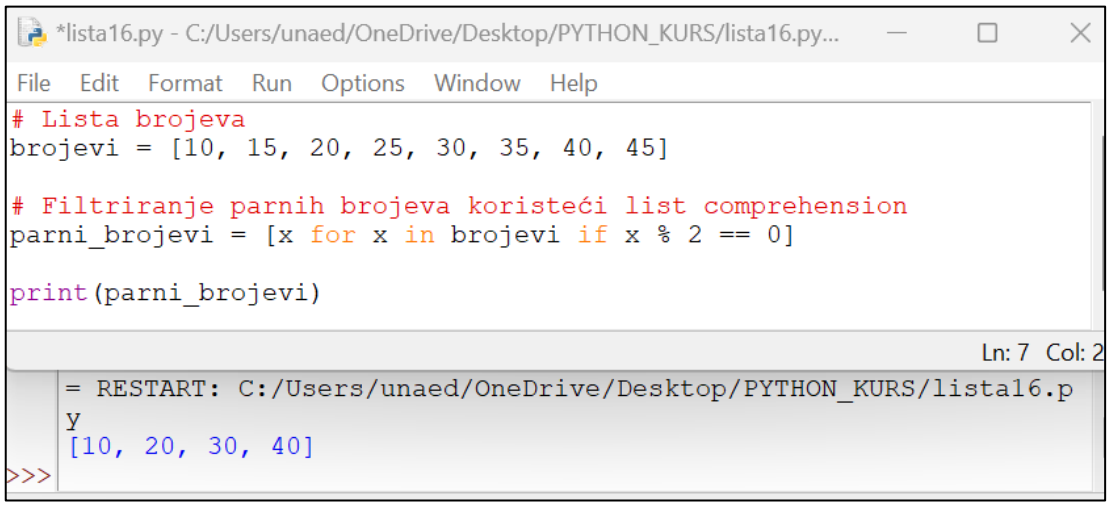
Slika 58. Primjer za Ugniježdeni List Comprehension

Liste su vrlo korisne u Pythonu i koriste se u različitim situacijama, od obrade podataka do rada s bazama podataka i analiza.

Primjeri korištenja lista u Pythonu su:

- *Filtriranje brojeva*-izdvajanje parnih brojeva iz liste

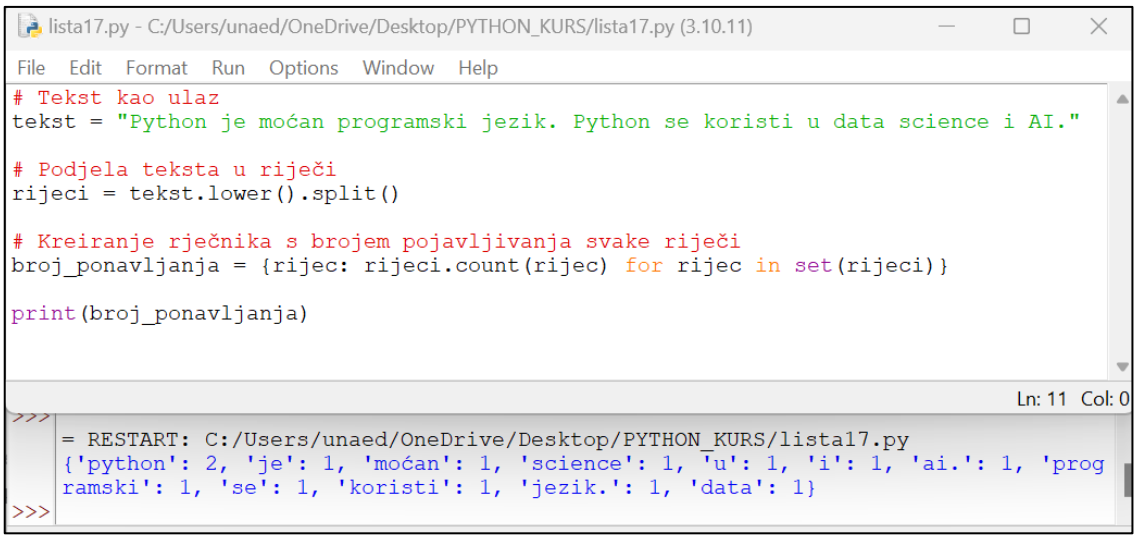
Kada se radi sa velikim skupovima podataka, često se trebaju izdvojiti elementi sa određenim specifikacijama, primjer slika 59.



Slika 59. Primjer izdvajanja parnih brojeva iz liste

- *Rad sa tekstom* – brojanje pojavljivanja riječi u tekstu

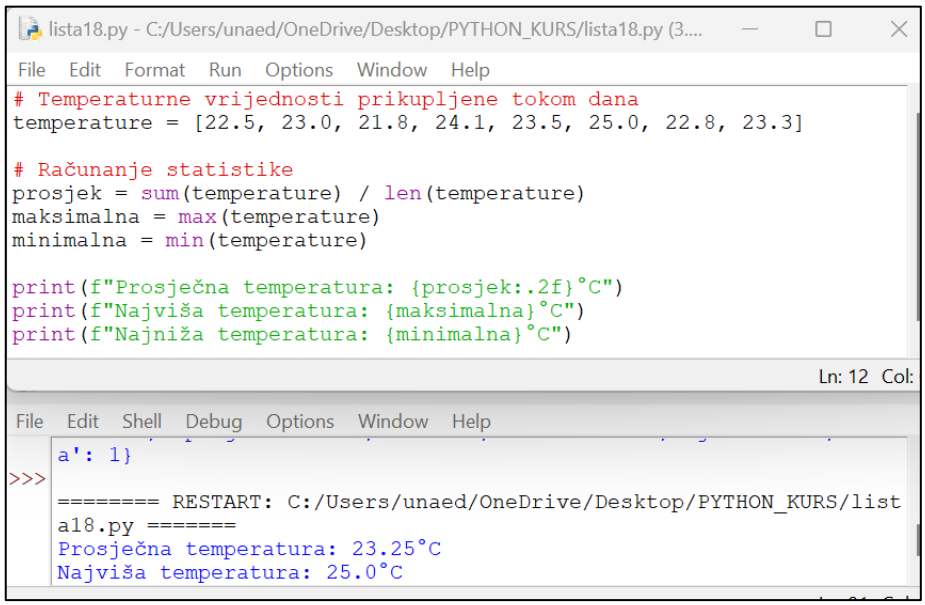
U analizi teksta, često se treba saznati koliko puta se neka riječ pojavljuje, primjer slika 60.



Slika 60. Primjer rada sa tekstom u listi

❖ *Analiza podataka* – prosječna temperatura senzora

Kada se prikupljaju podaci iz senzora (npr. temperatura), često se treba izračunati prosjek i pronaći najveća i najmanja vrijednost, primjer slika 61.



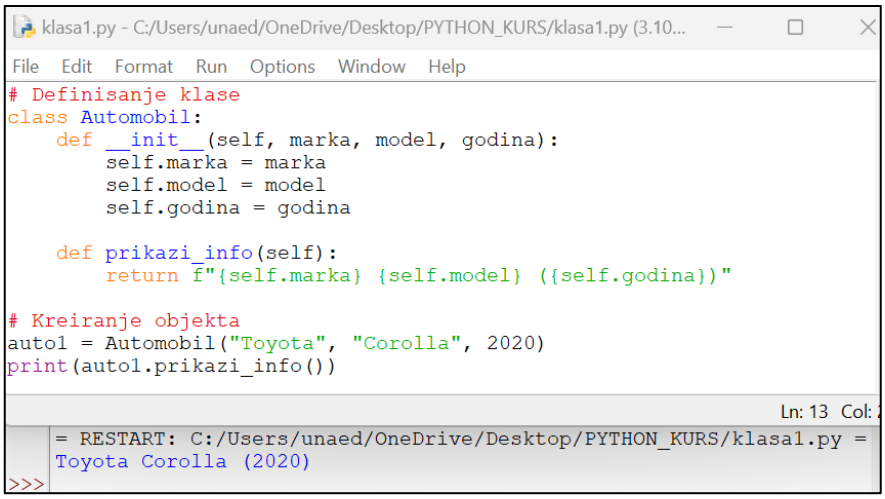
Slika 61. Primjer rada sa tekstom u listi

6 KLASE

Klase su temelj objektno orijentisanog programiranja (OOP) u Pythonu. Omogućavaju organizaciju koda u objekte koji sadrže podatke (atribute) i metode (funkcije koje rade s tim podacima).

6.1. Kreiranje osnovne klase

Klasa se definiše pomoću ključne riječi *class*, a objekti se kreiraju iz nje, primjer slika 62.

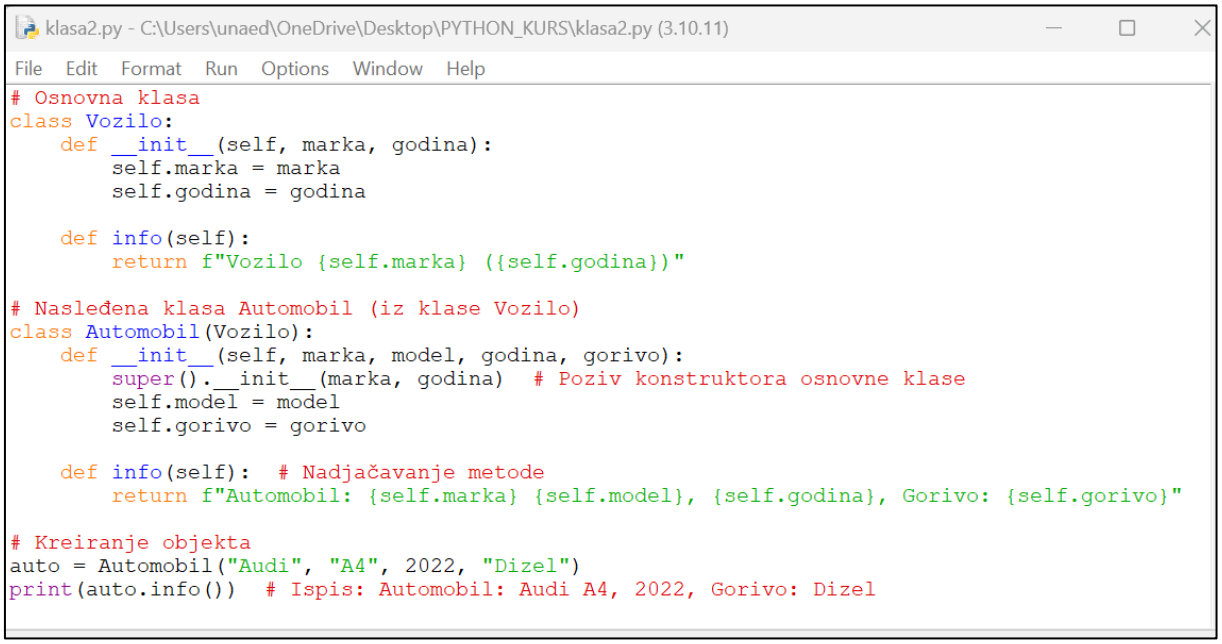


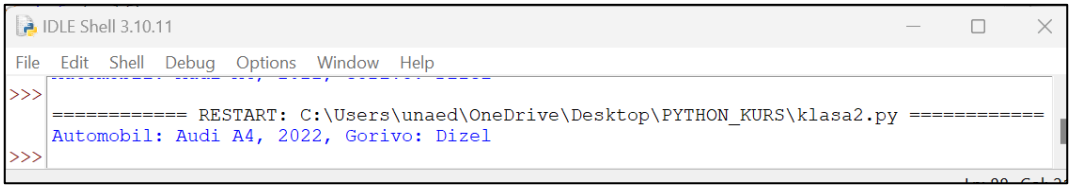
Slika 62. Primjer kreiranja objekta

- __init__ je konstruktor koji inicijalizuje atribute prilikom kreiranja objekta
- self označava referencu na instancu klase
- Metoda prikazi_info() vraća string sa podacima o automobilu

6.2. Nasljeđivanje (Inheritance)

Nasljeđivanje omogućava stvaranje nove klase koja preuzima osobine postojeće klase, čime se izbjegava dupliranje koda, primjer slika 63.



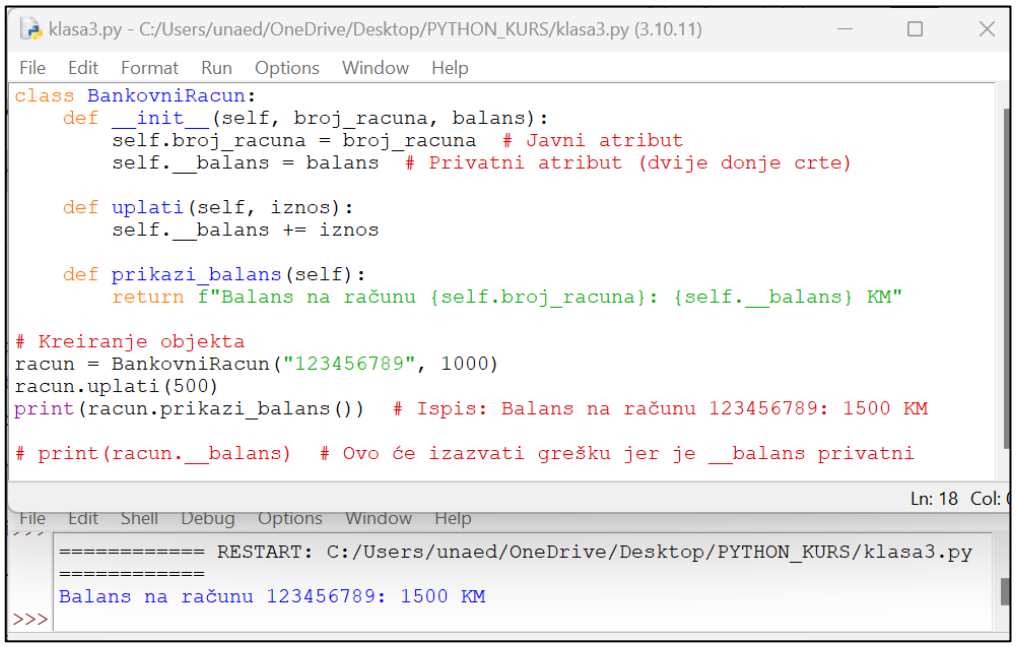


Slika 63. Primjer nasljeđivanja

- Klasa Automobil nasljeđuje klasu Vozilo, što znači da preuzima njene metode i atribute.
- super().__init__(marka, godina) poziva konstruktor osnovne klase.
- info() metoda je nadjačana i sada daje specifičniji ispis.

6.3. Enkapsulacija (Private i Protected atributi)

Enkapsulacija ograničava pristup podacima unutar klase, štiteći ih od vanjske izmjene primjer slika 64.

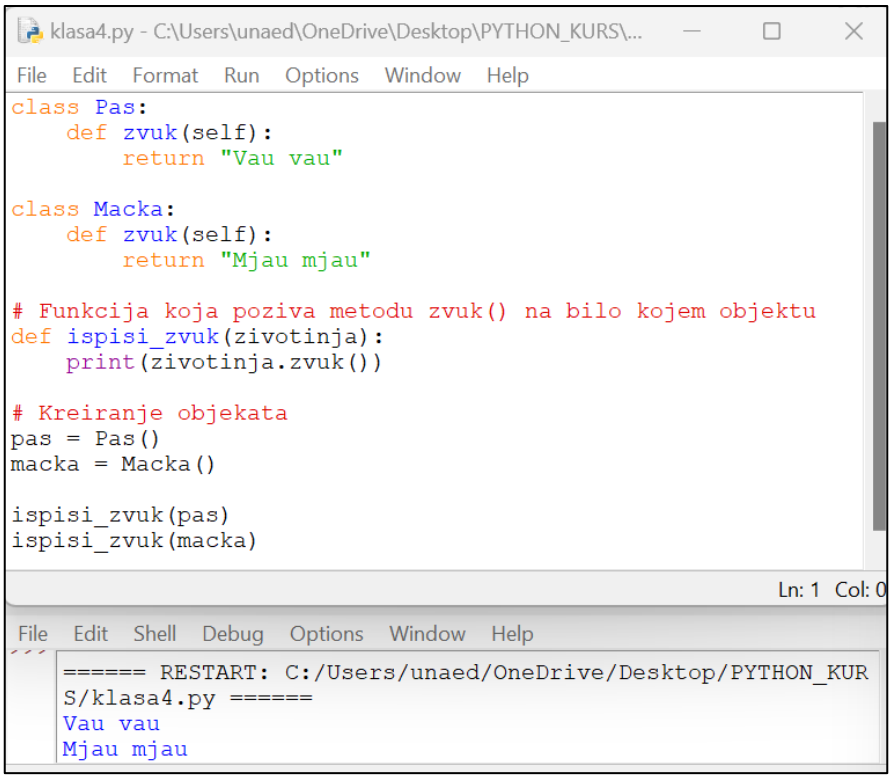


Slika 64. Primjer enkapsulacije

- __balans je privatni atribut i ne može mu se direktno pristupiti izvan klase.
- uplati() i prikazi_balans() metode omogućavaju manipulaciju

6.4. Polimorfizam

Polimorfizam (metode s istim imenom u različitim klasama), omogućava korištenje istih metoda u različitim klasama sa različitim ponašanjem, primjer slika 65.

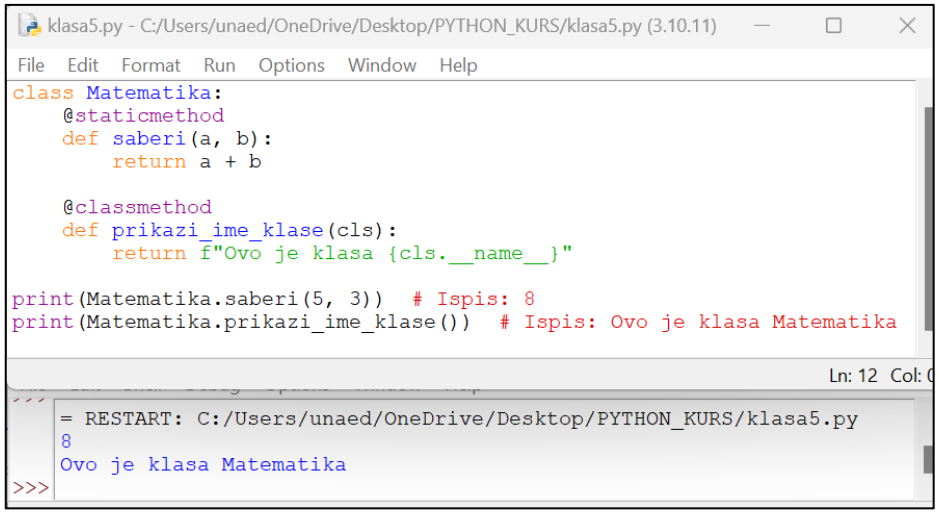


Slika 65. Primjer polimorfizma

- Metoda zvuk() postoji u obje klase (Pas, Macka), ali se ponaša drugačije.
- Funkcija ispisi_zvuk() prima bilo koji objekat koji ima metodu zvuk(), bez obzira na njegovu klasu.

6.5. Statičke metode i metode klase

Statičke metode ne koriste *self* i ne zavise od instance klase. Metode klase koriste *cls* i rade sa svim instancama klase, primjer slika 66.



Slika 66. Primjer statičke metode i metode klase

- @staticmethod označava metodu koja ne zavisi od instance (self).
- @classmethod koristi cls, što omogućava rad s klasom, a ne samo instancom.

Klase se koriste u raznim aplikacijama, od web programiranja do vještačke inteligencije i obrade podataka.

Primjeri korištenja lista u Pythonu su:

- *Web programiranje* (Django, Flask)

U web razvoju, klase se koriste za modeliranje baze podataka, rukovanje korisnicima, autentifikaciju i rukovanje HTTP zahtjevima. Primjer model korisnika u Django aplikaciji slika 67.

KLASA6.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/KLASA6.py...

File Edit Format Run Options Window Help

```
from django.db import models

class Korisnik(models.Model):
    ime = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    datum_registracije = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.ime} - {self.email}"
```

Slika 67. Primjer definiranja Django u Pythonu za rad sa bazom podataka

- Korisnik je klasa koja predstavlja tabelu u bazi podataka
- CharField i EmailField su atributi koji definišu tipove podataka
- __str__ metoda omogućava prikaz korisnika kao string

- *Vještačka inteligencija i mašinsko učenje*

Klase se koriste za kreiranje neuronskih mreža, rad sa podacima i modelovanje algoritama mašinskog učenja. Primjer definiranje neuronske mreže pomoću TensorFlow-a slika 68.

klasa7.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/klasa7.py (3...

File Edit Format Run Options Window Help

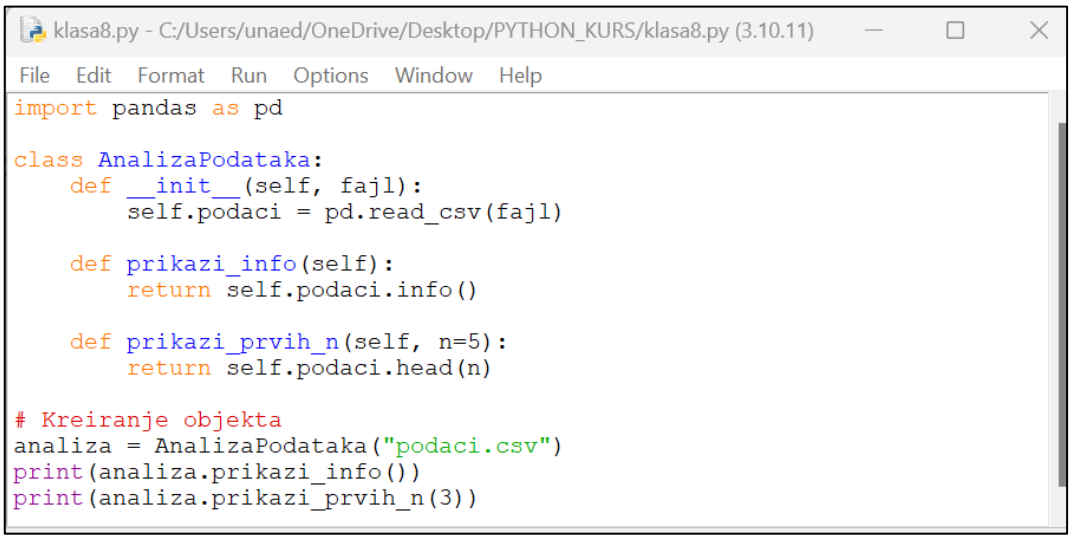
```
import tensorflow as tf
from tensorflow import keras

class Mreza(keras.Model):
    def __init__(self):
        super(Mreza, self).__init__()
        self.sloj1 = keras.layers.Dense(64, activation='relu')
        self.sloj2 = keras.layers.Dense(10, activation='softmax')

    def call(self, ulaz):
        x = self.sloj1(ulaz)
        return self.sloj2(x)

# Kreiranje modela
model = Mreza()
```

Slika 68. Primjer definiranja neuronske mreže

- *Mreza* je klasa koja nasleđuje keras.Model
- call metoda definiše protok podataka kroz slojeve neuronske mreže

❖ *Obrada podataka*

Klase se koriste za organizaciju i analizu podataka u nauci i analitici, primjer analize podataka je dat na slici 69.

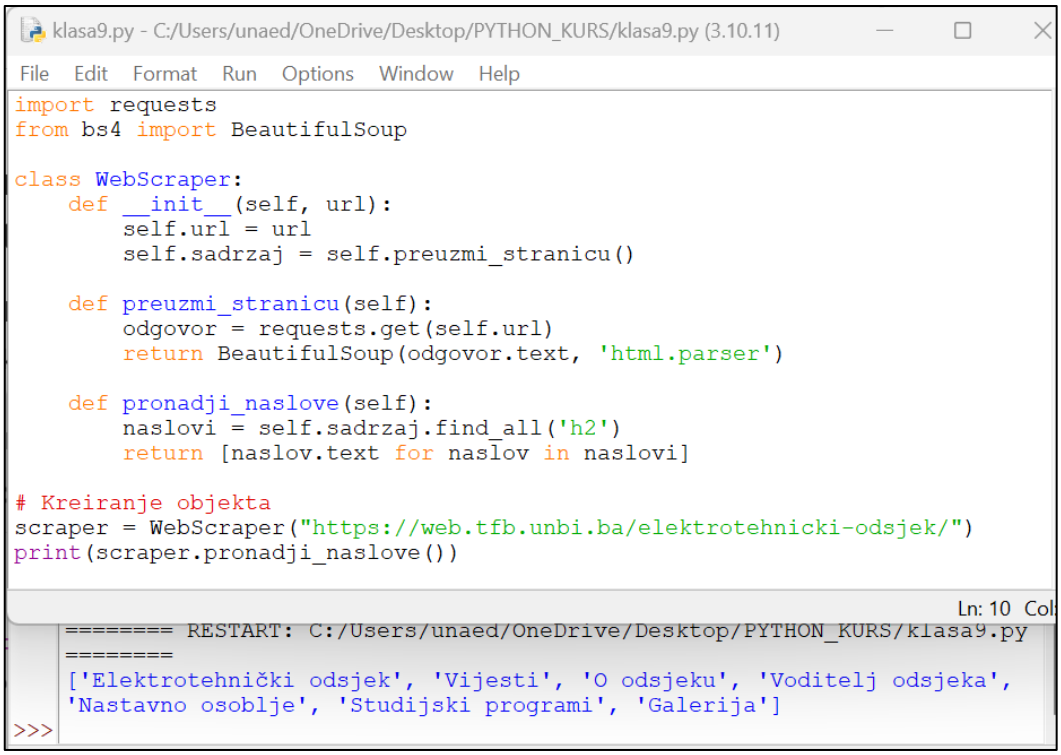


Slika 69. Primjer obrade podataka

- AnalizaPodataka učitava CSV fajl i omogućava analizu podataka.
- prikazi_info() daje informacije o datasetu.
- prikazi_prvih_n(n) prikazuje prvih n redova.

❖ *Automatizacija zadataka*

Klase se koriste za web scraping, automatizaciju unosa podataka i testiranje web aplikacija. Primjer automatskog preuzimanja podataka sa web stranice koristeći *BeautifulSoup* je dat na slici 70.



Slika 70. Primjer preuzimanja podataka

- WebScraper preuzima sadržaj stranice i izdvaja h2 naslove
- requests.get() dohvaća HTML sadržaj stranice
- BeautifulSoup omogućava pretragu elemenata u HTML-u

❖ *Igranje igrica i simulacije*

Klase se koriste za pravljenje video igrica, simulacija i sistema za učenje pomoću pojačanja. Paketi koji se najčešće koriste su Pygame, OpenAI Gym. Primjer kreiranja jednostavne igrice koristeći Pygame je dat na slikama 71, 72 i 73.

klasa10.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/klasa10.py (3.10.11)

File Edit Format Run Options Window Help

```
import pygame

# Inicijalizacija Pygame-a
pygame.init()

# Postavljanje dimenzija prozora
WIDTH, HEIGHT = 800, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Pong")

# Boje
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

# Dimenzije lopatica i loptice
PADDLE_WIDTH, PADDLE_HEIGHT = 10, 100
BALL_SIZE = 10

# Početne pozicije lopatica
paddle1 = pygame.Rect(50, HEIGHT//2 - PADDLE_HEIGHT//2, PADDLE_WIDTH, PADDLE_HEIGHT)
paddle2 = pygame.Rect(WIDTH - 50 - PADDLE_WIDTH, HEIGHT//2 - PADDLE_HEIGHT//2, PADDLE_WIDTH, PADDLE_HEIGHT)

# Početna pozicija loptice i brzina
ball = pygame.Rect(WIDTH//2 - BALL_SIZE//2, HEIGHT//2 - BALL_SIZE//2, BALL_SIZE, BALL_SIZE)
ball_speed_x = 4
ball_speed_y = 4

# Brzina lopatica
paddle_speed = 5

# Glavna petlja igre
running = True
while running:
    pygame.time.delay(10)  # Osvježavanje igre

    # Provjera događaja (event handling)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Kontrole za igrače
    keys = pygame.key.get_pressed()
    if keys[pygame.K_w] and paddle1.top > 0:
        paddle1.y -= paddle_speed
    if keys[pygame.K_s] and paddle1.bottom < HEIGHT:
        paddle1.y += paddle_speed
```

```
    # Pomicanje loptice
    ball.x += ball_speed_x
    ball.y += ball_speed_y

    # Sudar loptice sa zidovima (gore i dolje)
    if ball.top <= 0 or ball.bottom >= HEIGHT:
        ball_speed_y = -ball_speed_y

    # Sudar loptice s lopaticama
    if ball.colliderect(paddle1) or ball.colliderect(paddle2):
        ball_speed_x = -ball_speed_x

    # Reset loptice ako izađe izvan ekrana (poen za protivnika)
    if ball.left <= 0 or ball.right >= WIDTH:
        ball.x = WIDTH//2 - BALL_SIZE//2
        ball.y = HEIGHT//2 - BALL_SIZE//2
        ball_speed_x = -ball_speed_x  # Promjena smjera nakon svakog gola

    # Crtanje elemenata igre
    screen.fill(BLACK)  # Osvježavanje ekrana
    pygame.draw.rect(screen, WHITE, paddle1)
    pygame.draw.rect(screen, WHITE, paddle2)
    pygame.draw.ellipse(screen, WHITE, ball)
    pygame.draw.aaline(screen, WHITE, (WIDTH//2, 0), (WIDTH//2, HEIGHT))  # Linija po sredini terena

    # Ažuriranje ekrana
    pygame.display.flip()

pygame.quit()
```

Slika 71. Primjer kreiranja igrice Pong

klasa11.py - C:\Users\unaed\OneDrive\Desktop\PYTHON_KURS\klasa11.py (3.10.11)

File Edit Format Run Options Window Help

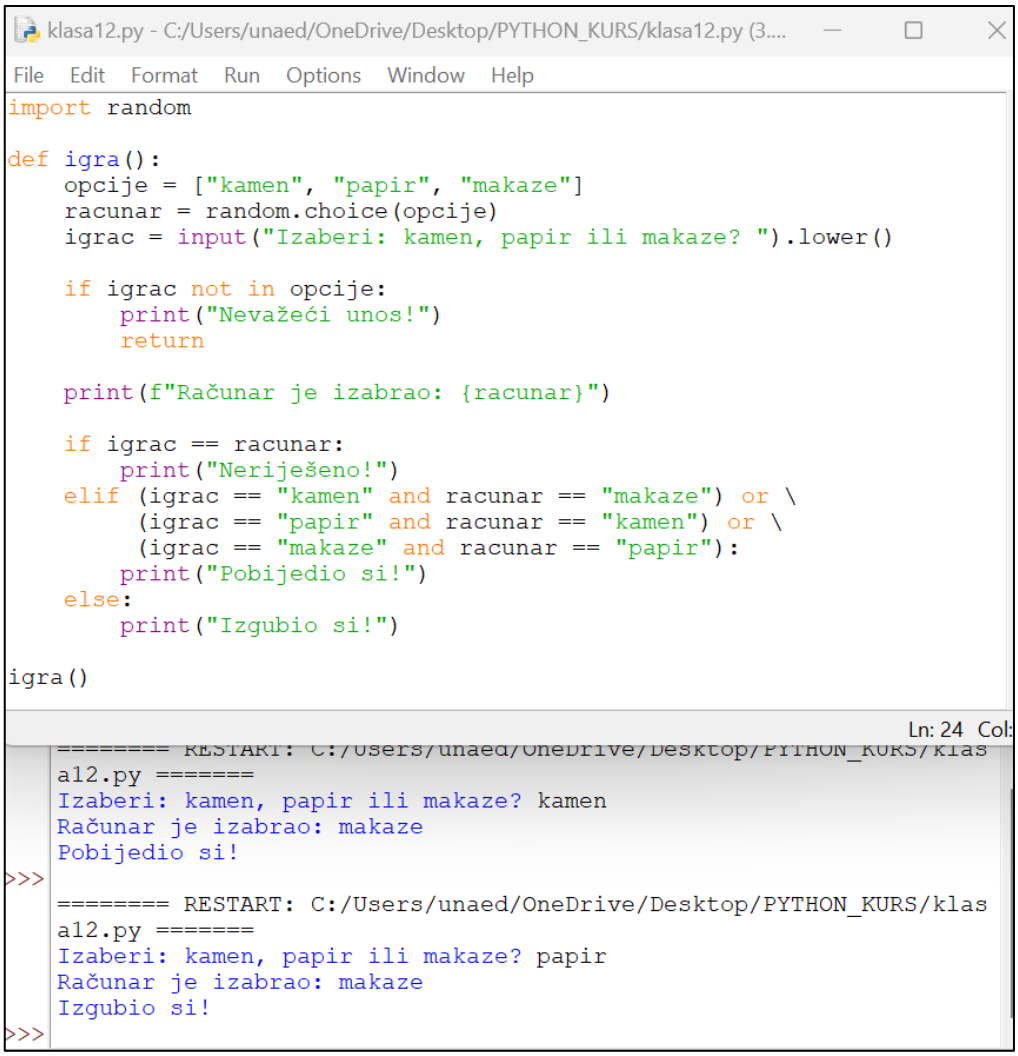
```
import random

def pogodi_broj():
    broj = random.randint(1, 5)  # Računar bira broj
    pokusaji = 0

    while True:
        pokusaj = int(input("Pogodi broj (1-5): "))
        pokusaji += 1

        if pokusaj < broj:
            print("Previše malo! Pokušaj ponovo.")
        elif pokusaj > broj:
            print("Previše veliko! Pokušaj ponovo.")
        else:
            print(f"Čestitam! Pogodio si broj {broj} u {pokusaji} pokušaja.")
            break

pogodi_broj()
```

IDLE Shell 3.10.11

File Edit Shell Debug Options Window Help

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/klasa11.py =
======
Pogodi broj (1-5): 4
Previše veliko! Pokušaj ponovo.
Pogodi broj (1-5): 2
Previše malo! Pokušaj ponovo.
Pogodi broj (1-5): 3
Čestitam! Pogodio si broj 3 u 3 pokušaja.
>>>
```

Slika 72. Primjer kreiranja igrice Pogađanja broja



Slika 73. Primjer kreiranja igrice Kamen, papir, makaze

Klase u Pythonu omogućavaju modularnost, ponovnu upotrebu koda i organizaciju složenih sistema kroz objektno orijentirano programiranje (OOP). Koriste se u raznim domenama, uključujući razvoj softvera, web aplikacije, vještačku inteligenciju, obradu podataka i simulacije.

Korištenjem klasa možemo:

- *Strukturirati kod* – Grupiranjem podataka i funkcionalnosti u objekte.
- *Smanjiti ponavljanje koda* – Naslađivanjem i višekratnom upotrebom klasa.
- *Povećati čitljivost i održavanje koda* – Jasnim razdvajanjem funkcionalnosti kroz enkapsulaciju.
- *Efikasno modelirati stvarne sisteme* – Simulacijom stvarnih entiteta kroz objekte i metode.

U savremenom programiranju, klase su neizostavan dio aplikacija i omogućavaju razvoj fleksibilnih, skalabilnih i efikasnih programskih rješenja.

7 UPRAVLJAČKE NAREDBE

Upravljačke naredbe su ključni element svakog programskog jezika, pa tako i u Pythonu, jer omogućavaju kontrolu toka izvršavanja programa. Kroz naredbe kao što su if, elif, else, for, while, break, continue, i pass, Python omogućava uslovnu i petljastu kontrolu, kao i mogućnost upravljanja tokom izvršavanja programa na temelju različitih uvjeta.

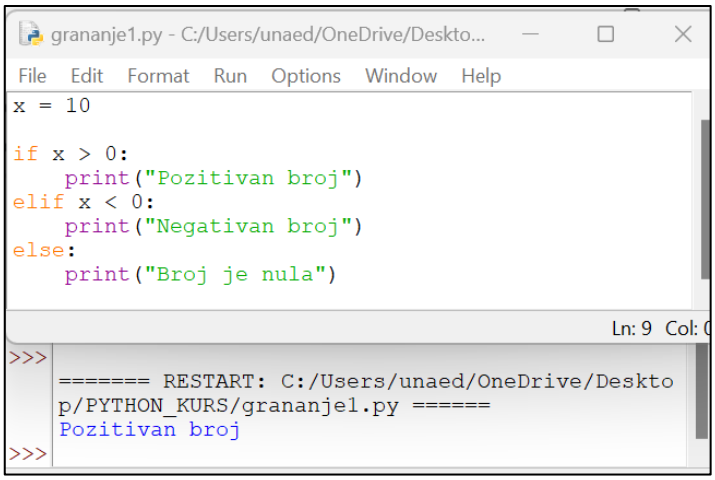
- Uslovne naredbe grananja (kao što su if, elif, else) omogućavaju donošenje odluka u programu na temelju različitih uvjeta. Ove naredbe omogućavaju fleksibilnost i kontrolu toka, jer omogućavaju različite grane izvršavanja ovisno o vrijednostima varijabli ili uvjetima.
- Petlje (for, while) omogućavaju ponavljanje određenih radnji više puta, što je posebno korisno kada treba obraditi više elemenata (npr. liste, stringove ili druge kolekcije podataka). Petlje čine kod kraćim i efikasnijim, a u kombinaciji sa naredbama poput break i continue omogućavaju precizno upravljanje tokom tih ponavljanja.
- Naredbe za prekid toka izvršavanja kao što su break, continue i pass omogućavaju da program kasnije donese odluke koje mogu preskočiti određene dijelove koda ili uopšte ne rade ništa u određenim situacijama. Break se koristi za prekidanje petlji, continue za preskakanje trenutne iteracije, dok pass omogućava da ostavimo prazne blokove koda bez sintaktičkih grešaka.

7.1. Uslovne naredbe grananja (if, elif, else)

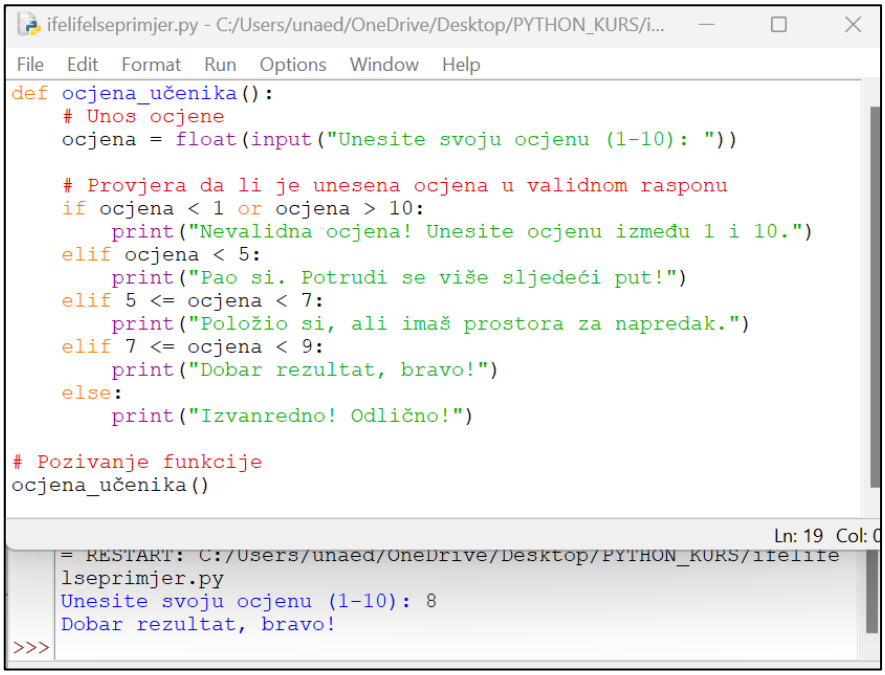
Uslovne naredbe grananja omogućavaju donošenje odluka na osnovu uslova. Sintaksa je:

```
if    uslov:
      Naredbe 1
elif uslov2:
      Naredba 2
else:
      Naredbe3
```

Primjer primjene uslovnih naredbi grananja je dat na slici 74 i 75.



Slika 74. Primjer primjene naredbi grananja



Slika 75. Primjer primjene naredbi grananja

7.2. Petlje (for, while)

Petlje omogućavaju ponavljanje dijela koda više puta.

Sintaksa za for petlju je:

```
for var in sekvenca:
    naredba
```

Sintaksa za while petlju je:

```
while uslov:
        naredba
```

Primjer primjene naredbi grananja je dat na slici 76 i 77.

forprimjer.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/forprimjer.py (3.10.11)

File Edit Format Run Options Window Help

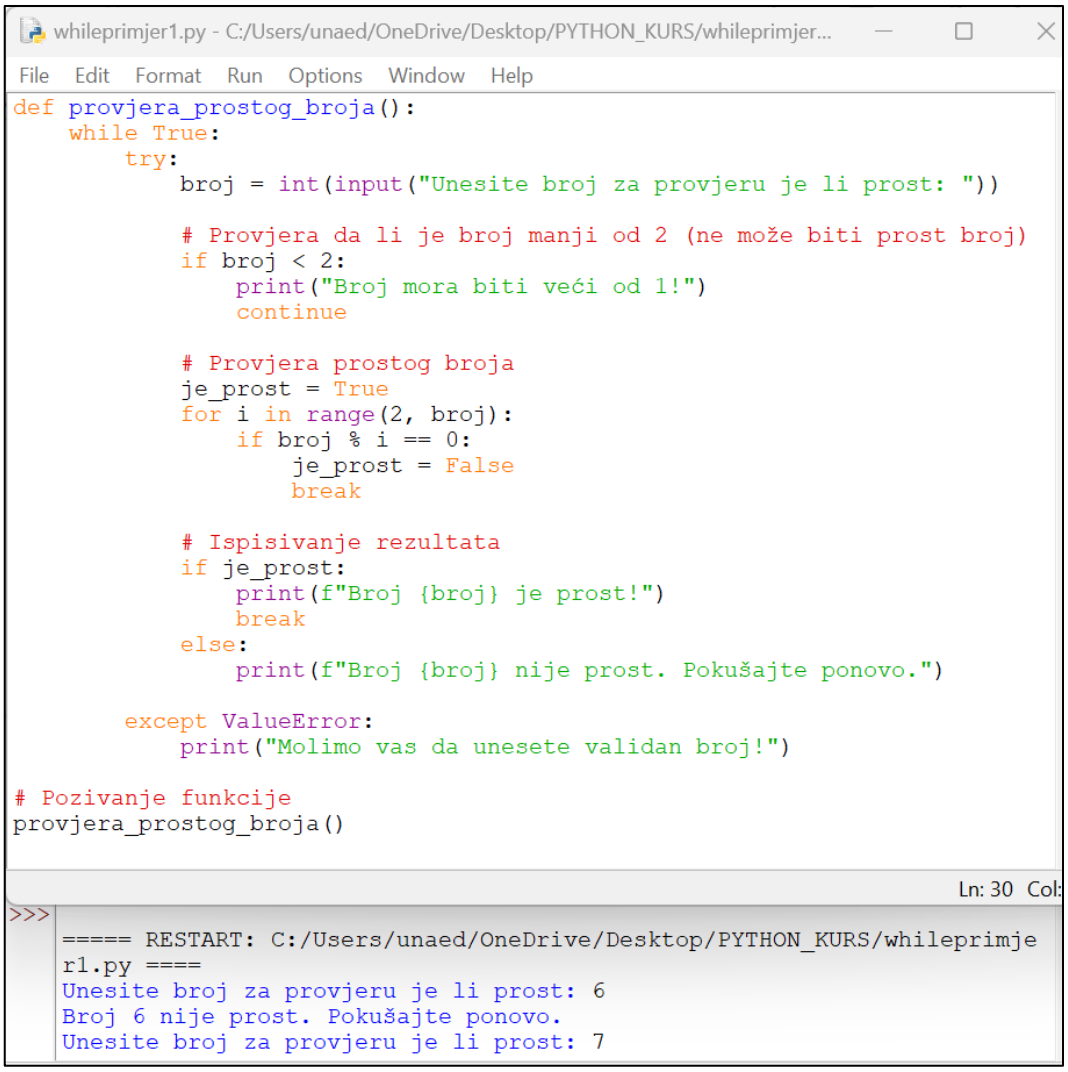
```
def prosti_brojevi_opseg():
    # Unos početnog i krajnjeg broja
    pocetak = int(input("Unesite početni broj opsega: "))
    kraj = int(input("Unesite krajnji broj opsega: "))

    # Lista za čuvanje prostih brojeva
    prosti_brojevi = []
    suma = 0
    broj_prostih = 0

    # Provjera svih brojeva u opsegu
    for broj in range(pocetak, kraj + 1):
        # Provjera da li je broj prost
        if broj > 1:
            prost = True
            for i in range(2, broj):
                if broj % i == 0:
                    prost = False
                    break
            if prost:
                prosti_brojevi.append(broj)
                suma += broj
                broj_prostih += 1

    # Ispis rezultata
    print(f"Prosti brojevi u opsegu od {pocetak} do {kraj} su: {prosti_brojevi}")
    print(f"Suma prostih brojeva je: {suma}")
    print(f"Broj prostih brojeva u opsegu je: {broj_prostih}")

# Pozivanje funkcije
prosti_brojevi_opseg()
```

IDLE Shell 3.10.11

File Edit Shell Debug Options Window Help

```
Dobar rezultat, bravo!
>>>
= RESTART: C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/forprimjer.py
Unesite početni broj opsega: 5
Unesite krajnji broj opsega: 25
Prosti brojevi u opsegu od 5 do 25 su: [5, 7, 11, 13, 17, 19, 23]
Suma prostih brojeva je: 95
Broj prostih brojeva u opsegu je: 7
>>>
```

Slika 76. Primjer primjene for



Slika 77. Primjer primjene while

7.3. Izlazne i upravljačke naredbe (break, continue, pass)

Ove naredbe omogućavaju dodatnu kontrolu nad tokom izvršavanja petlji. Naredba break se koristi za prekidanje petlje (bilo for ili while) prije nego što je dosegla svoj normalni kraj. Kada se break izvrši, petlja odmah prestaje s radom, a program prelazi na naredbu koja slijedi nakon petlje.

Sintaksa za break je:

for var in sekvenca:
  if uslov:
  break
	while uslov:
	  if uslov2:
	  break

Naredba continue koristi se za preskakanje trenutne iteracije petlje i prelazak na sljedeću. Ova naredba ne prekida cijelu petlju, već samo preskače trenutnu iteraciju i nastavlja s radom od sljedeće.

Sintaksa za continue je:

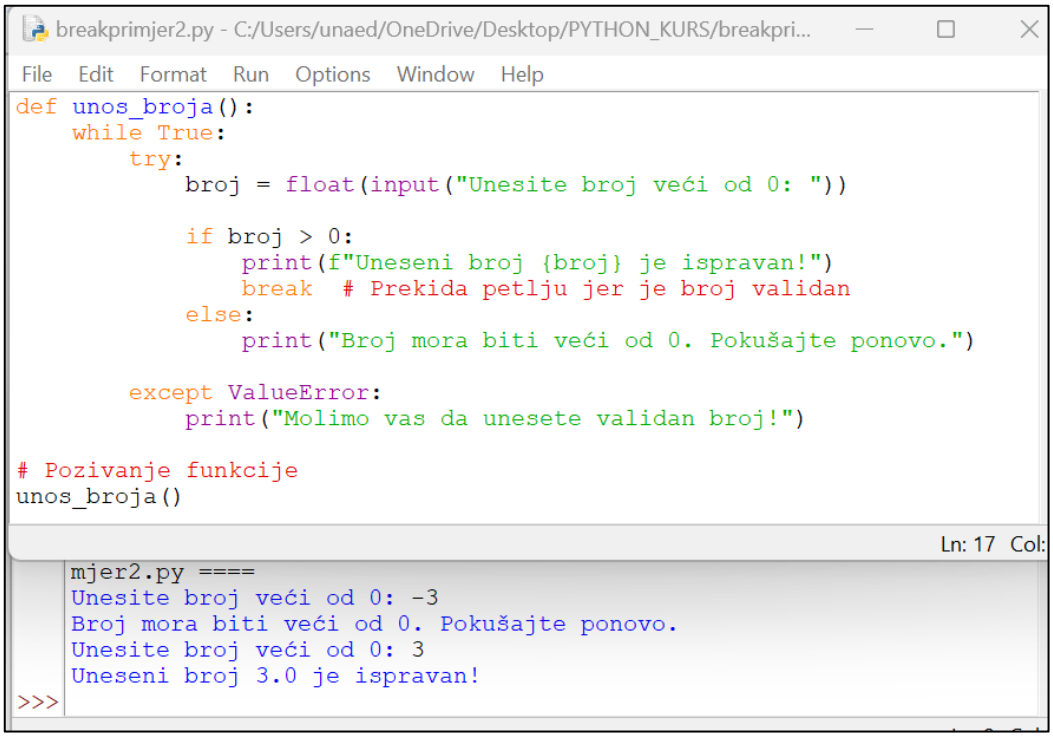
for var in sekvenca:
  if uslov:
  continue
	while uslov:
	  if uslov2:
	  continue

Naredba pass je prazna naredba koja se koristi kao placeholder u situacijama gdje sintaktički Python očekuje neki kod, ali ništa se ne želi izvršiti. Često se koristi u definicijama funkcija, petlji ili uvjetnim granama koje još nisu implementirane.

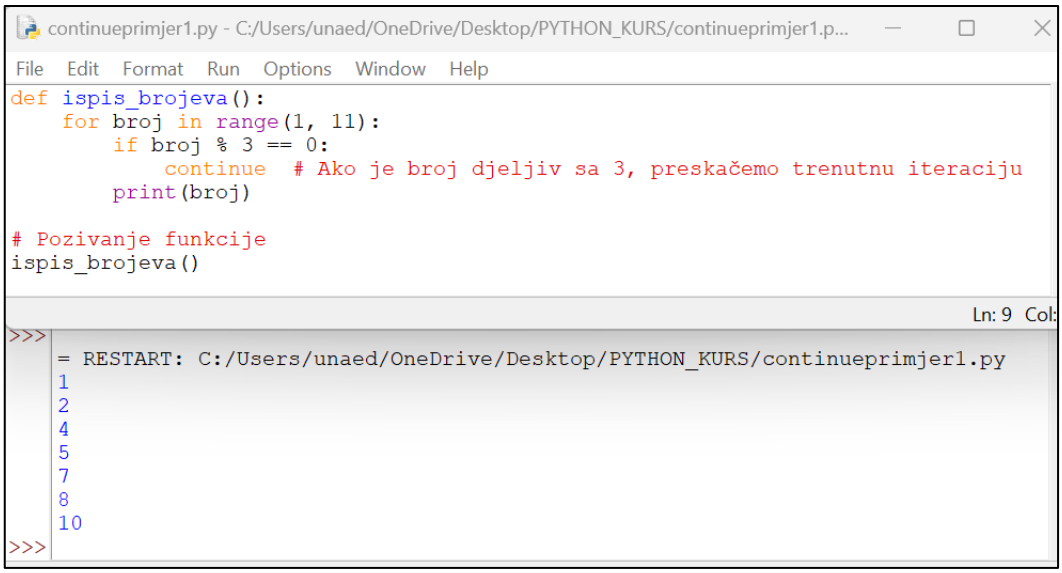
Sintaksa za pass je:

if uslov:
  pass
	for var in sekvenca:
	  pass

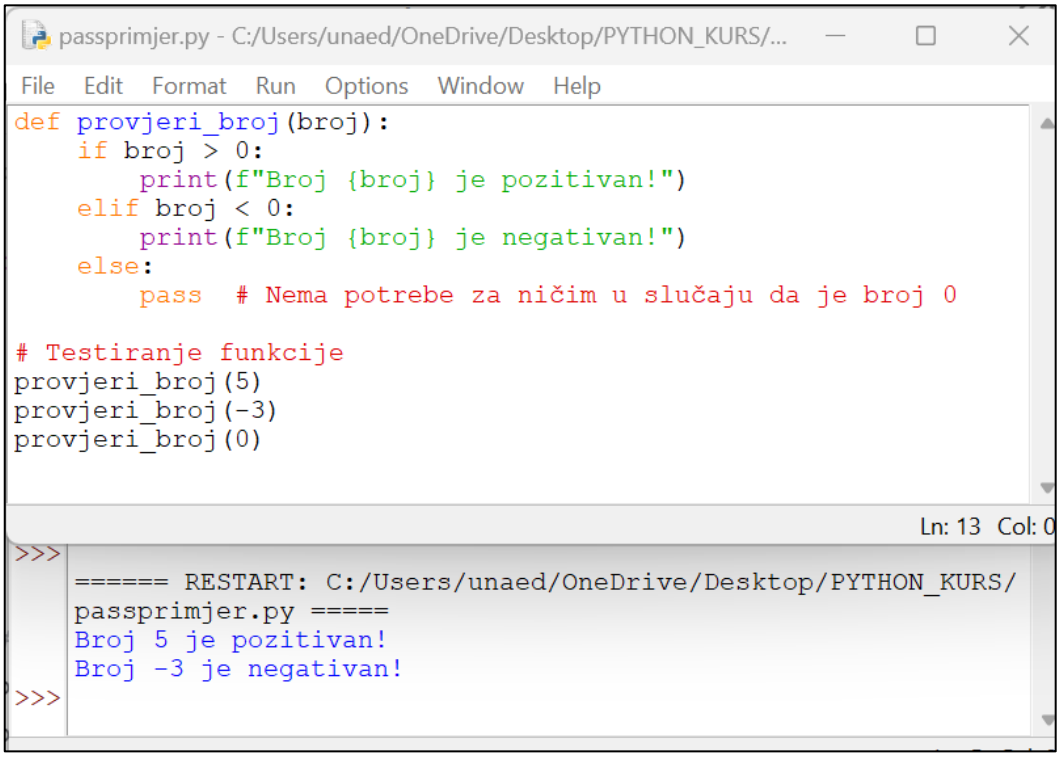
Primjer primjene izlaznih i upravljačkih naredbi je dat na slikama 78, 79 i 80.



Slika 78. Primjer primjene break



Slika 79. Primjer primjene continue



Slika 80. Primjer primjene pass

Uz pomoć ovih upravljačkih naredbi, programeri mogu efikasno upravljati logikom programa i donositi dinamične odluke na temelju podataka. Bez njih, programi bi bili linearni i nedovoljno fleksibilni. Upravo kroz ove naredbe Python omogućava kreiranje sofisticiranih aplikacija koje mogu reagirati na različite situacije i korisničke ulaze.

8 FUNKCIJE ZA GRAFIČKI PRIKAZ

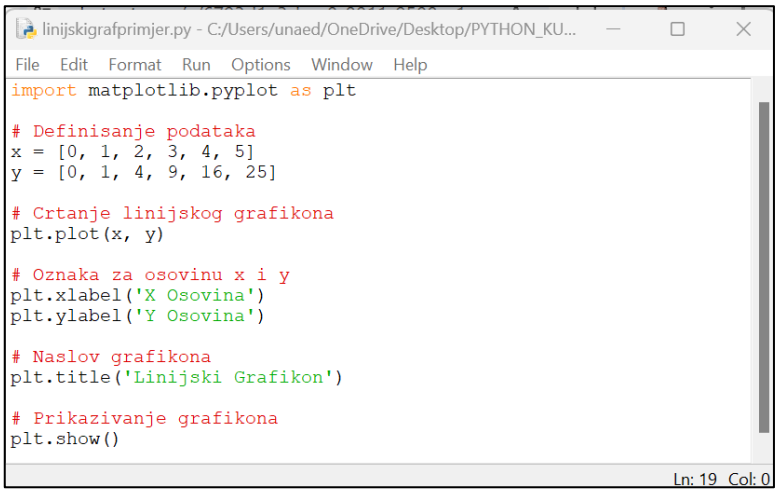
U Pythonu, za vizualizaciju podataka u 2D i 3D prostoru, najčešće se koristi paket Matplotlib. Matplotlib omogućava jednostavan i fleksibilan način za generiranje širokog spektra grafova i dijagrama, uključujući 2D i 3D prikaze podataka. Osim toga, za napredniji grafički prikaz i interaktivne 3D grafike, možemo koristiti mpl_toolkits.mplot3d, koji je ekstenzija Matplotlib-a koja omogućava rad sa 3D podacima.

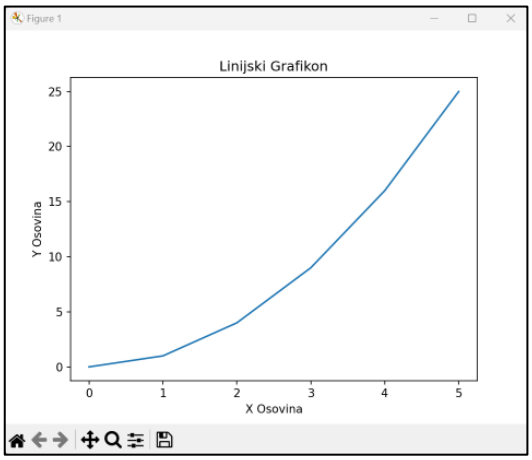
8.1. 2D grafički prikaz

2D grafički prikazi su najčešći i koriste se za vizualizaciju podataka u dvodimenzionalnom prostoru. Matplotlib pruža bogat skup funkcionalnosti za crtanje linijskih grafova, scatter grafova, histogram grafova, bar grafova i drugih.

8.1.1. Linijski graf

Linijski graf se koriste za prikazivanje promjena podataka tokom vremena ili u odnosu na neku varijablu, slika 81.





Slika 81. Primjer linijskog grafa

8.1.2. Scatter graf

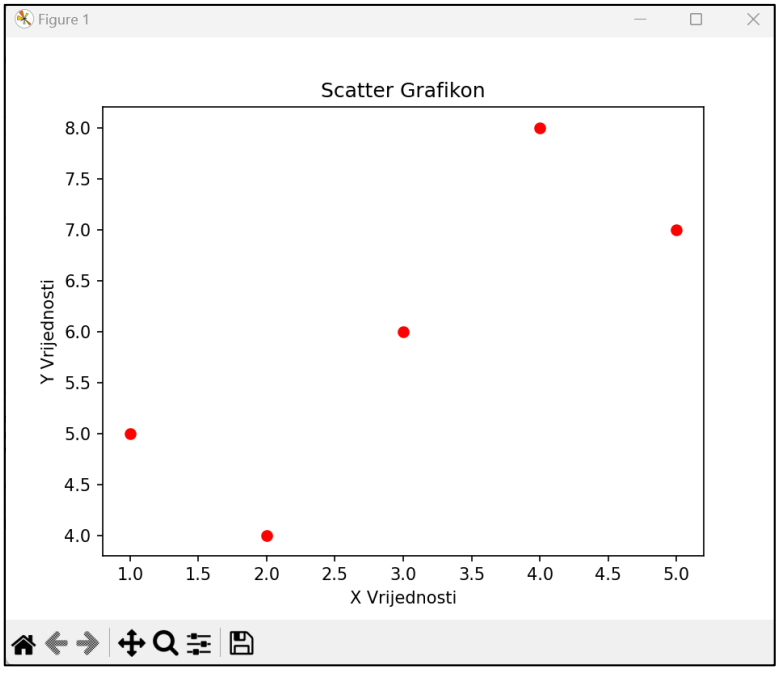
Scatter grafikon se koristi za prikazivanje podataka u obliku tačaka na ravni. Često se koristi za analizu odnosa između dvije varijable, slika 82.

scatterprimjer.py - C:/Users/unaed/OneDrive/Desktop/PYTHO...

File Edit Format Run Options Window Help

```
import matplotlib.pyplot as plt

# Podaci za scatter grafikon
x = [1, 2, 3, 4, 5]
y = [5, 4, 6, 8, 7]

# Crtanje scatter grafikona
plt.scatter(x, y, color='red')

# Oznake
plt.xlabel('X Vrijednosti')
plt.ylabel('Y Vrijednosti')

# Naslov
plt.title('Scatter Grafikon')

# Prikazivanje grafikona
plt.show()
```

Ln: 12 Col: 27



Slika 82. Primjer scatter grafa

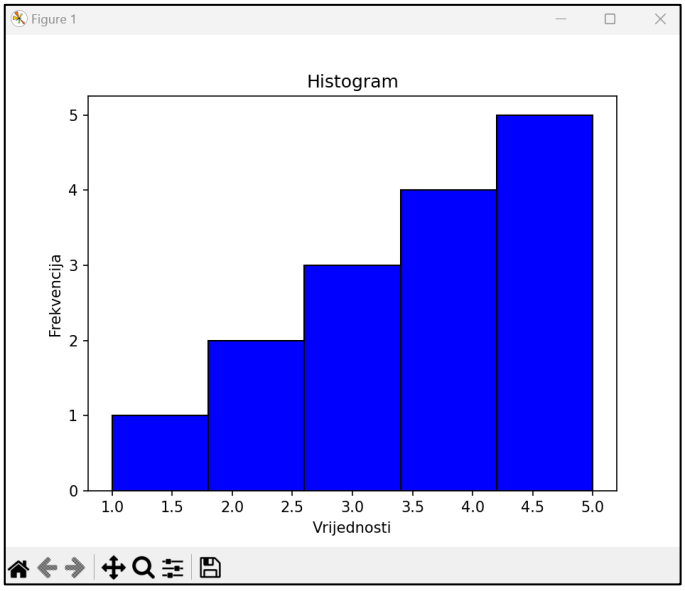
8.2.3. Histogram

Histogram prikazuje distribuciju podataka, tj. broj ponavljanja vrijednosti u određenim intervalima, slika 83.

histogramprimjer.py - C:/Users/unaed/OneDrive/Desktop/PYTH...

File Edit Format Run Options Window Help

```
import matplotlib.pyplot as plt

# Podaci za histogram
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]

# Crtanje histograma
plt.hist(data, bins=5, color='blue', edgecolor='black')

# Oznake
plt.xlabel('Vrijednosti')
plt.ylabel('Frekvencija')

# Naslov
plt.title('Histogram')

# Prikazivanje grafikona
plt.show()
```



Slika 83. Primjer histograma

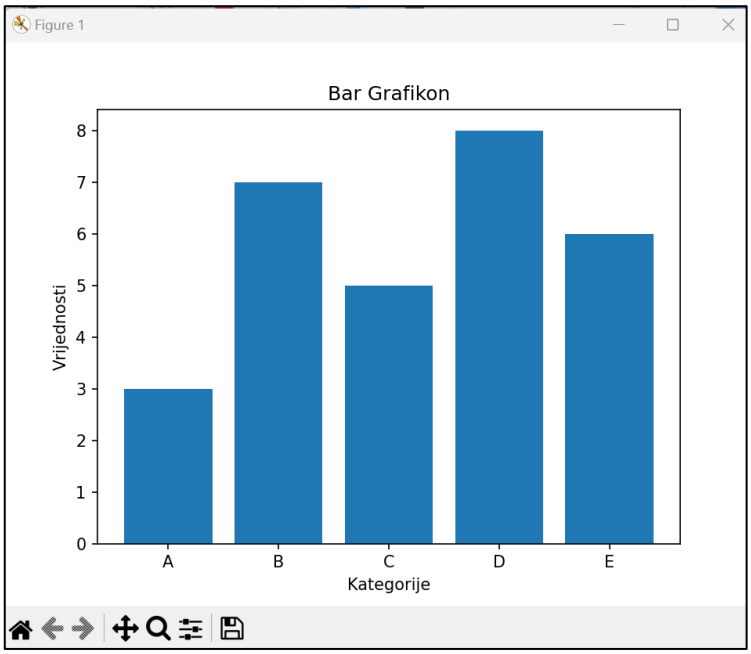
8.2.4. Bar graf

Bar grafikon je koristan za prikazivanje uspoređenja između različitih kategorija , slika 84.

bargrafprimjer.py - C:/Users/unaed/OneDrive/Desktop/PYT...

File Edit Format Run Options Window Help

```
import matplotlib.pyplot as plt

# Kategorije i vrednosti
categories = ['A', 'B', 'C', 'D', 'E']
values = [3, 7, 5, 8, 6]

# Crtanje bar grafikona
plt.bar(categories, values)

# Oznake
plt.xlabel('Kategorije')
plt.ylabel('Vrijednosti')

# Naslov
plt.title('Bar Grafikon')

# Prikazivanje grafikona
plt.show()
```



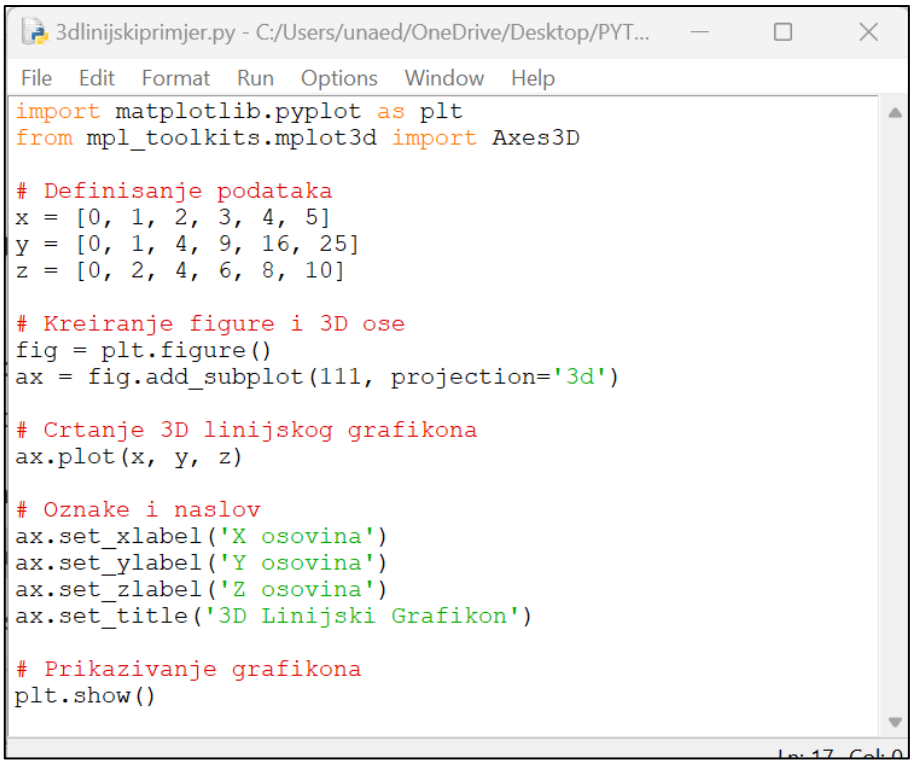
Slika 84. Primjer bar grafa

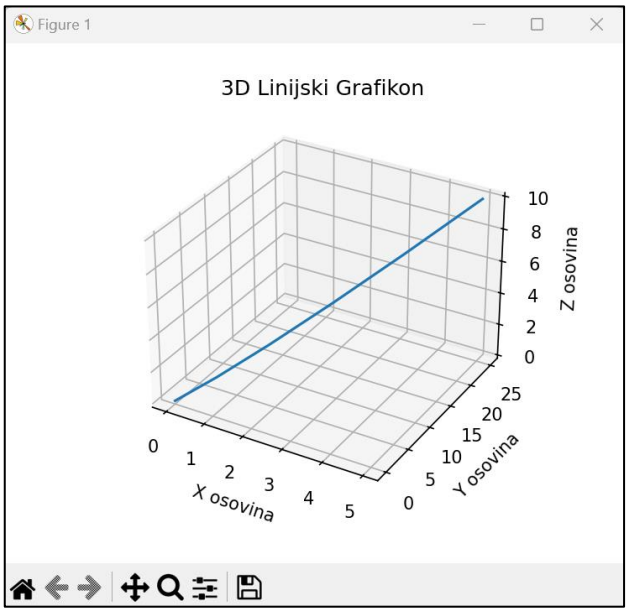
8.3. 3D grafički prikaz

Za rad sa 3D grafovima u Pythonu koristi se ekstenzija **mpl_toolkits.mplot3d**, koja omogućava crtanje trodimenzionalnih objekata kao što su 3D linijski grafovi, scatter grafovi, površinski grafovi, itd.

8.3.1. 3D linijski graf

Za crtanje 3D linijskog grafikona, koristimo **Axes3D** iz **mpl_toolkits.mplot3d**, slika 85.





Slika 85. Primjer 3D linijskog grafa

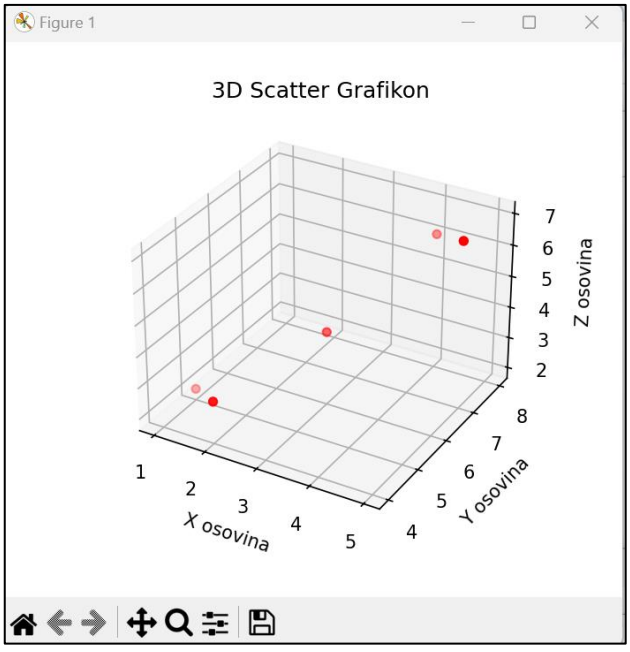
8.3.2. 3D scatter graf

Za prikazivanje tačaka u 3D prostoru, koristi se funkcija **scatter()** iz **Axes3D** objekta, primjer slika 86.

3dscatterprimjer.py - C:/Users/unaed/OneDrive/Desktop/PYTHO...

File Edit Format Run Options Window Help

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Definisanje podataka
x = [1, 2, 3, 4, 5]
y = [5, 4, 6, 8, 7]
z = [2, 3, 4, 6, 7]

# Kreiranje figure i 3D ose
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Crtanje 3D scatter grafikona
ax.scatter(x, y, z, color='red')

# Oznake i naslov
ax.set_xlabel('X osovina')
ax.set_ylabel('Y osovina')
ax.set_zlabel('Z osovina')
ax.set_title('3D Scatter Grafikon')

# Prikazivanje grafikona
plt.show()
```



Slika 86. Primjer 3D scatter grafa

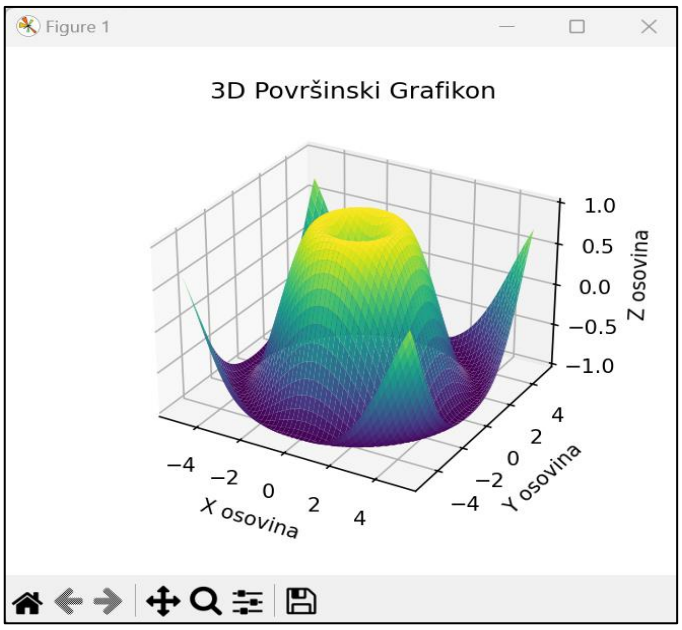
8.3.4. 3D površinski graf (surface plot)

Za prikazivanje 3D površinskog grafa koristi se funkcija **plot_surface()**. Ovaj graf je koristan za vizualizaciju funkcija sa tri promjenljive, slika 87.

3dpovrsinskigrafprimjer.py - C:/Users/unaed/OneDrive/Desktop/PYTH...

File Edit Format Run Options Window Help

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# Definisanje mreže podataka
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
x, y = np.meshgrid(x, y)
z = np.sin(np.sqrt(x**2 + y**2))

# Kreiranje figure i 3D ose
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Crtanje 3D površinskog grafikona
ax.plot_surface(x, y, z, cmap='viridis')

# Oznake i naslov
ax.set_xlabel('X osovina')
ax.set_ylabel('Y osovina')
ax.set_zlabel('Z osovina')
ax.set_title('3D Površinski Grafikon')

# Prikazivanje grafikona
plt.show()
```



Slika 87. Primjer 3D površinskog grafa

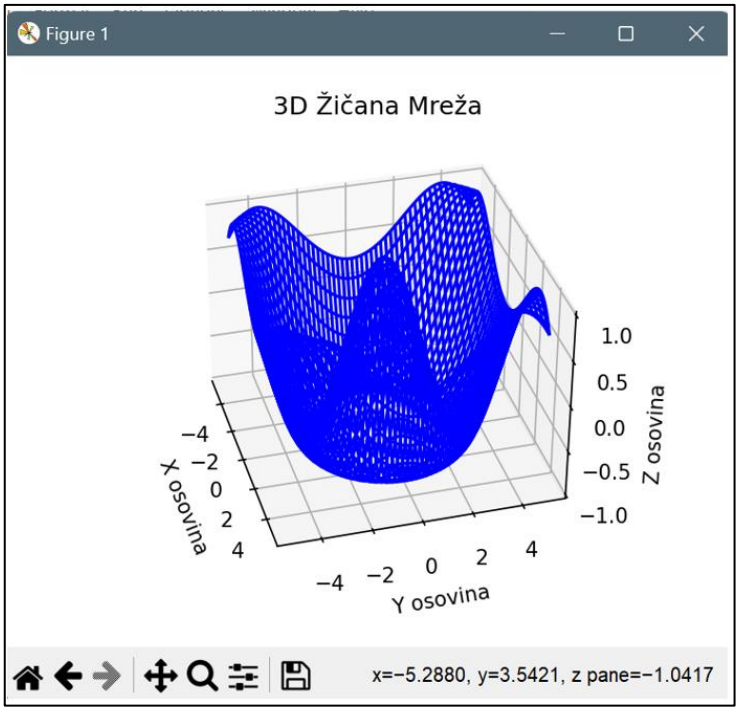
8.3.5. Rotacija 3D grafa

Rotacija 3D grafa omogućava interaktivno rotiranje 3D grafa kako bi se promijenio ugao gledanja i omogućilo bolje analiziranje podataka, slika 88.

rotacijagrafaprimjer.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/rotacijagrafaprimjer.py (3....

File Edit Format Run Options Window Help

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Generisanje podataka
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
x, y = np.meshgrid(x, y)
z = np.cos(np.sqrt(x**2 + y**2))

# Kreiranje figure i 3D ose
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Crtanje 3D žičane mreže
ax.plot_wireframe(x, y, z, color='blue')

# Oznake i naslov
ax.set_xlabel('X osovina')
ax.set_ylabel('Y osovina')
ax.set_zlabel('Z osovina')
ax.set_title('3D Žičana Mreža')

# Rotacija grafikona
ax.view_init(elev=45, azim=45)  # Promenjeni uglovi rotacije: elevacija 45, azimut 45

# Prikazivanje grafikona
plt.show()
```



Slika 88. Primjer rotacije 3D grafa

9 GRAFIČKI INTERFEJS

Grafički korisnički interfejs (GUI) u Pythonu omogućava razvoj aplikacija s interaktivnim prozorima, gumbima, unosima i drugim grafičkim komponentama. U Pythonu postoje različiti paketi i biblioteke koje omogućavaju kreiranje GUI aplikacija, a najpoznatiji i najčešće korišteni su:

1. **Tkinter** – Osnovna biblioteka za GUI u Pythonu, koja je standardno uključena u Python distribuciju.
2. **PyQt** – Moćna biblioteka koja se temelji na Qt frameworku, koja pruža veće mogućnosti i fleksibilnost.
3. **wxPython** – Također popularan GUI framework, zasnovan na wxWidgets biblioteci.
4. **Kivy** – Biblioteka za izradu GUI aplikacija s naglaskom na mobilne uređaje i touch-based aplikacije.

9.1. Tkinter

Tkinter je standardna biblioteka u Pythonu za izradu desktop aplikacija. Pruža jednostavan način za kreiranje prozora, dugmadi, tekstualnih polja, i drugih GUI elemenata.

Osnovni elementi Tkinter-a:

- **Tk()**: Glavni prozor aplikacije.
- **Label()**: Prikazuje tekst.
- **Button()**: Kreira dugme.
- **Entry()**: Unosni okvir za tekst.
- **Canvas()**: Površina za crtanje.
- **MessageBox()**: Dijaloški okvir za prikazivanje poruka.

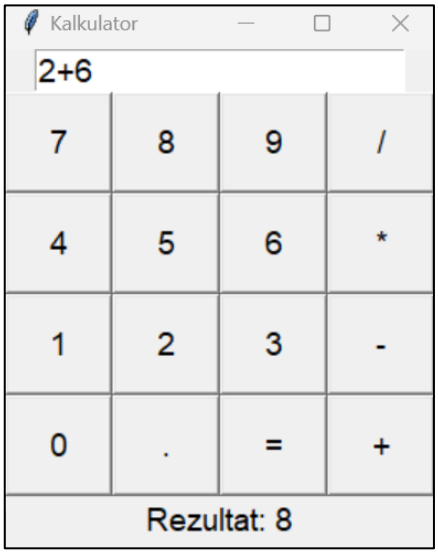
Za primjer je kreiran GUI kalkulator, slika 89.

tkinterprimjer.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/tkinterprimjer.py (3.10.11)

File Edit Format Run Options Window Help

```
import tkinter as tk

# Funkcija za računanje
def izracunaj():
    try:
        rezultat = eval(entry.get())  # Funkcija eval omogućava evaluaciju izraza u stringu
        label_rezultat.config(text="Rezultat: " + str(rezultat))
    except:
        label_rezultat.config(text="Greška u unosu!")

# Kreiranje glavnog prozora
prozor = tk.Tk()
prozor.title("Kalkulator")  # Postavljanje naslova prozora

# Unosni okvir za unos izraza
entry = tk.Entry(prozor, width=20, font=('Arial', 14))
entry.grid(row=0, column=0, columnspan=4)

# Dugmadi za brojeve i operacije
dugmad = [
    ('7', 1, 0), ('8', 1, 1), ('9', 1, 2), ('/', 1, 3),
    ('4', 2, 0), ('5', 2, 1), ('6', 2, 2), ('*', 2, 3),
    ('1', 3, 0), ('2', 3, 1), ('3', 3, 2), ('-', 3, 3),
    ('0', 4, 0), ('.', 4, 1), ('=', 4, 2), ('+', 4, 3)
]

# Kreiranje dugmadi i dodavanje u grid
for (text, row, col) in dugmad:
    tk.Button(prozor, text=text, width=5, height=2, font=('Arial', 14),
              command=lambda t=text: entry.insert(tk.END, t) if t != "=" else izracunaj()).grid(row=row, column=col)

# Label za rezultat
label_rezultat = tk.Label(prozor, text="Rezultat: ", font=('Arial', 14))
label_rezultat.grid(row=5, column=0, columnspan=4)

# Glavna petlja koja pokreće GUI
prozor.mainloop()
```

Pritisni "Prnt Scrn" na svojoj tip



Slika 89. Primjer primjene Tkinter biblioteke

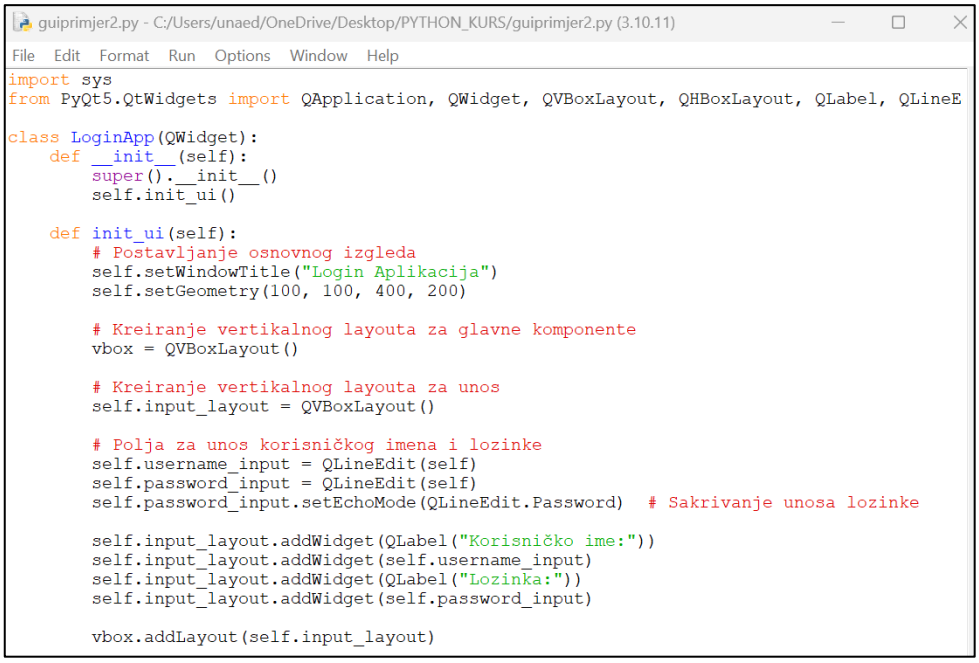
Objašnjenje:

1. **Glavni prozor (Tk())**: prozor = tk.Tk() stvara glavni prozor aplikacije.
2. **Unosni okvir (Entry)**: entry = tk.Entry(prozor) omogućava korisnicima da unesu matematički izraz.
3. **Dugmadi (Button)**: Kreiramo dugmadi za brojeve i operacije. Svako dugme ima funkciju koja ili dodaje broj u unosni okvir ili pokreće računanje kada je dugme za "=" pritisnuto.
4. **Funkcija izracunaj()**: Koristi Pythonovu funkciju eval() za evaluaciju matematičkog izraza unesenog u unosni okvir. Ako postoji greška u unosu, prikazuje poruku o grešci.

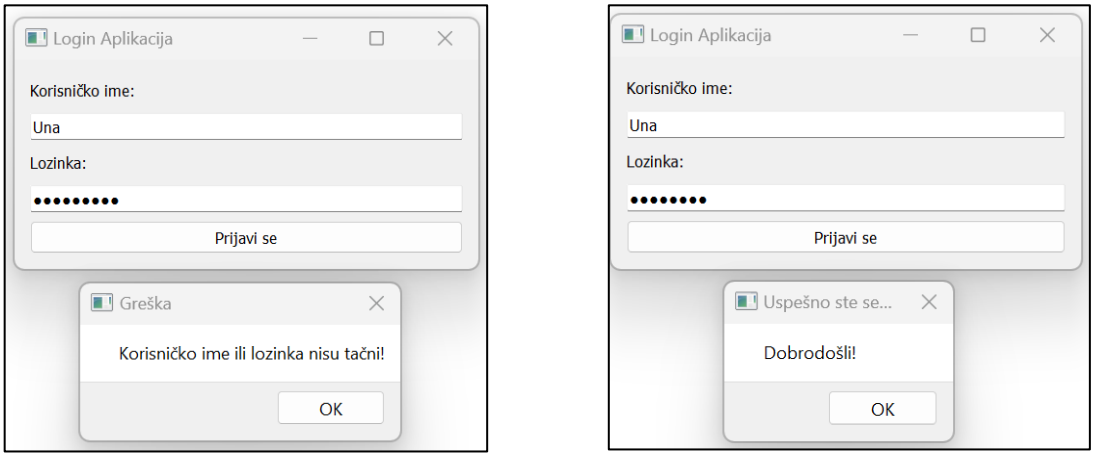
5. **Rezultat (Label)**: label_rezultat prikazuje rezultat izraza ili poruku o grešci.
6. **Glavna petlja (mainloop())**: Petlja koja omogućava da GUI ostane otvoren dok korisnik ne zatvori prozor.

9.2. PyQt5

Moćna biblioteka koja se temelji na Qt frameworku, koja pruža veće mogućnosti i fleksibilnost, primjer slika 90.




```
        # Dugme za prijavu
        self.login_button = QPushButton("Prijavi se", self)
        self.login_button.clicked.connect(self.login)
        vbox.addWidget(self.login_button)

        # Postavljanje layouta na glavni prozor
        self.setLayout(vbox)

    def login(self):
        # Unapred definisani korisnik i lozinka
        correct_username = "Una"
        correct_password = "12345una"

        # Uzimanje unetih podataka
        username = self.username_input.text()
        password = self.password_input.text()

        # Provera unetih podataka
        if username == correct_username and password == correct_password:
            # Ako su podaci tačni
            self.show_message("Uspešno ste se prijavili!", "Dobrodošli!")
        else:
            # Ako podaci nisu tačni
            self.show_message("Greška", "Korisničko ime ili lozinka nisu tačni!")

    def show_message(self, title, message):
        # Prikazivanje poruke sa naslovom i tekstom
        msg = QMessageBox(self)
        msg.setWindowTitle(title)
        msg.setText(message)
        msg.exec_()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    login_app = LoginApp()
    login_app.show()
    sys.exit(app.exec_())
```



Slika 90. Primjer primjene PyQt5 biblioteke (Aplikacija za prijavu korisnika)

Objašnjenje**:**

1. **QWidget** – Osnovna klasa za GUI aplikacije u PyQt5, kao što je prethodno objašnjeno.
2. **QVBoxLayout** i **QHBoxLayout** – Koristimo ove layout-ove za raspored komponenata unutar prozora. U ovom primeru koristimo vertikalni raspored za unos podataka i dugme za prijavu.
3. **QLineEdit** – Koristi se za unos korisničkog imena i lozinke. Lozainka je sakrivena pomoću setEchoMode(QLineEdit.Password), što sprečava prikazivanje unetih karaktera u tekstualnom formatu.
4. **QPushButton** – Dugme za prijavu koje poziva funkciju login() kada se pritisne.
5. **QMessageBox** – Ova komponenta se koristi za prikazivanje poruka sa informacijama. U ovom primeru prikazujemo poruku kada su podaci tačni ili kada postoji greška u unosu.
6. **login()** – Funkcija koja proverava unete podatke sa unapred definisanim korisničkim imenom i lozinkom. Ako su podaci tačni, prikazuje se poruka dobrodošlice, a ako nisu tačni, prikazuje se poruka o grešci.
7. **show_message()** – Funkcija koja koristi QMessageBox za prikazivanje poruke sa željenim naslovom i tekstom.

9.3. wxPython

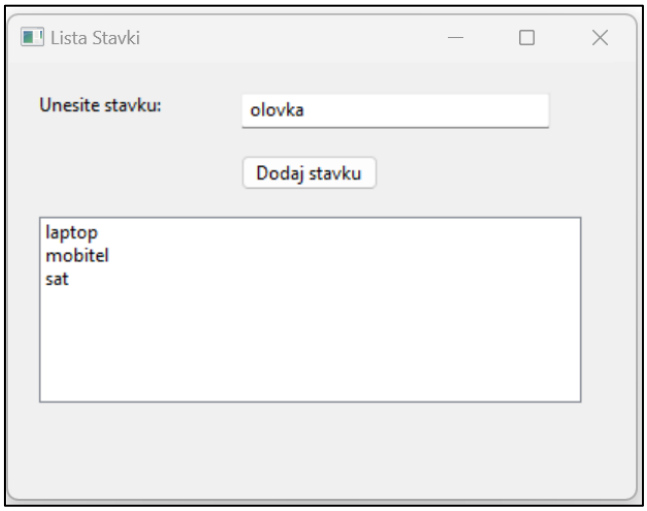
wxPython je biblioteka za razvoj grafičkog korisničkog interfejsa (GUI) u Pythonu. To je Python wrapper za **wxWidgets**, koji je C++ biblioteka koja omogućava kreiranje aplikacija sa grafičkim interfejsom. wxPython omogućava programerima da kreiraju aplikacije sa desktop interfejsom koje izgledaju nativno na različitim operativnim sistemima, uključujući Windows, macOS i Linux. Primjer je dat na slici 91.

wxPythonprimjer.py - C:\Users\unaed\OneDrive\Desktop\PYTHON_KURS\wxPythonprimjer.py (3.10.11)

File Edit Format Run Options Window Help

```
import wx

class ItemListFrame(wx.Frame):
    def __init__(self, parent, title):
        super().__init__(parent, title=title, size=(400, 300))

        self.panel = wx.Panel(self)

        # Kreiranje unosa za novu stavku
        self.item_label = wx.StaticText(self.panel, label="Unesite stavku:", pos=(20, 20))
        self.item_input = wx.TextCtrl(self.panel, pos=(150, 20), size=(200, -1))

        # Kreiranje dugmeta za dodavanje stavke
        self.add_button = wx.Button(self.panel, label="Dodaj stavku", pos=(150, 60))
        self.add_button.Bind(wx.EVT_BUTTON, self.add_item)  # Veza za događaj dodavanja stavke

        # Kreiranje liste za prikaz stavki
        self.item_listbox = wx.ListBox(self.panel, pos=(20, 100), size=(350, 120))

        # Prikazivanje prozora
        self.Centre()
        self.Show()

    def add_item(self, event):
        # Uzimanje unetih podataka
        item = self.item_input.GetValue()

        # Ako je unos prazan, prikazujemo poruku
        if not item:
            self.show_message("Greška", "Molimo unesite stavku!")
        else:
            # Dodavanje stavke u listu
            self.item_listbox.Append(item)
            self.item_input.Clear()  # Čišćenje unosa nakon dodavanja stavke
```

```
    def show_message(self, title, message):
        # Kreiranje dijaloga sa porukom
        dialog = wx.MessageDialog(self.panel, message, title, wx.OK | wx.ICON_INFORMATION)
        dialog.ShowModal()  # Prikazivanje dijaloga
        dialog.Destroy()  # Uništavanje dijaloga nakon što je zatvoren


# Glavna funkcija za pokretanje aplikacije
def main():
    app = wx.App(False)
    ItemListFrame(None, "Lista Stavki")
    app.MainLoop()

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



Slika 91. Primjer primjene wxPython biblioteke

Objašnjenje:

1. **wx.Frame** – Glavni prozor aplikacije.
2. **wx.Panel** – Ploča na kojoj se nalaze svi elementi.
3. **wx.StaticText** – Komponenta koja prikazuje tekstualnu oznaku "Unesite stavku:".
4. **wx.TextCtrl** – Komponenta koja omogućava korisniku unos podataka (stavke).
5. **wx.Button** – Dugme koje korisnik pritisne da doda stavku u listu.
6. **wx.ListBox** – Komponenta koja prikazuje listu stavki koje je korisnik uneo. Svaka nova stavka se dodaje na kraj liste.
7. **wx.MessageDialog** – Dijalog koji se koristi za prikazivanje poruka. U ovom slučaju, koristi se da obavesti korisnika ako je polje za unos prazno.

9.4. Kivy

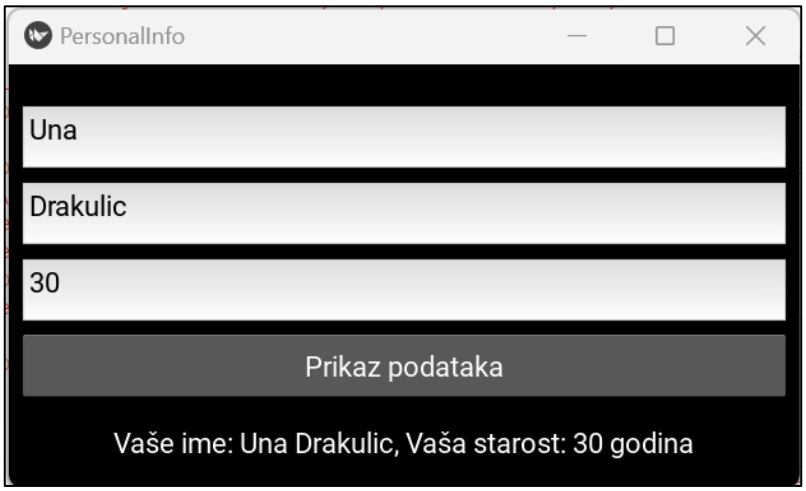
Kivy je biblioteka za izradu GUI aplikacija s naglaskom na mobilne uređaje i touch-based aplikacije, primjer slika 92.

kivyprimjer.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/kivyprimjer.py (3.10.11)

File Edit Format Run Options Window Help

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.textinput import TextInput
from kivy.uix.button import Button
from kivy.uix.label import Label

class PersonalInfoApp(App):

    def build(self):
        # Glavni Layout - vertikalno raspoređivanje
        main_layout = BoxLayout(orientation="vertical", padding=10, spacing=10)

        # Kreiranje polja za unos imena
        self.name_input = TextInput(hint_text="Unesite vaše ime", size_hint=(1, None), height=50)

        # Kreiranje polja za unos prezimena
        self.surname_input = TextInput(hint_text="Unesite vaše prezime", size_hint=(1, None), height=50)

        # Kreiranje polja za unos starosti
        self.age_input = TextInput(hint_text="Unesite vašu starost", size_hint=(1, None), height=50)

        # Dugme za prikazivanje unetih podataka
        display_button = Button(text="Prikaz podataka", size_hint=(1, None), height=50)
        display_button.bind(on_press=self.display_data)

        # Labela za prikazivanje rezultata
        self.result_label = Label(text="Podaci će biti prikazani ovde", size_hint=(1, None), height=50)
```

```
        # Dodavanje komponenti u glavni layout
        main_layout.add_widget(self.name_input)
        main_layout.add_widget(self.surname_input)
        main_layout.add_widget(self.age_input)
        main_layout.add_widget(display_button)
        main_layout.add_widget(self.result_label)

        return main_layout

    def display_data(self, instance):
        # Dobijanje podataka sa input polja
        name = self.name_input.text
        surname = self.surname_input.text
        age = self.age_input.text

        # Kreiranje formata za prikaz
        if name and surname and age:
            self.result_label.text = f"Vaše ime: {name} {surname}, Vaša starost: {age} godina"
        else:
            self.result_label.text = "Molimo vas da unesete sve podatke!"

if __name__ == "__main__":
    PersonalInfoApp().run()
```



Slika 92. Primjer primjene Kivy biblioteke

Objašnjenje koda:

1. **BoxLayout**: Glavni raspored aplikacije je vertikalni BoxLayout. Kroz njega se postavljaju svi widgeti (polja za unos, dugme, labela).
2. **TextInput**: Ovaj widget omogućava korisnicima da unesu tekst. Koristi se za unos imena, prezimena i starosti. Svako od ovih polja ima hint_text koji korisnicima pokazuje šta treba da unesu.
3. **Button**: Dugme koje pokreće akciju prikaza unesenih podataka. Klikom na dugme poziva se metoda display_data, koja uzima unesene vrijednosti i prikazuje ih na ekranu.
4. **Label**: Ovaj widget prikazuje tekst na ekranu. Kada korisnik pritisne dugme, rezultat (ime, prezime, starost) se prikazuje u ovoj labeli. Ako nisu uneseni svi podaci, korisniku se prikazuje poruka koja to označava.
5. **display_data**: Metoda koja se poziva kada korisnik pritisne dugme. Ona uzima vrijednosti sa svih input polja, formira odgovarajući tekst i prikazuje ga u labeli.

Primjer kreiranja GUI aplikacije koja omogućava korisnicima da:

a. Izaberu oblik koji žele da nacrtaju (krug ili pravougaonik).
b. Kliknu na platnu da nacrtaju izabrani oblik.
c. Promjene boju oblika.
d. Isprazne platno ako žele da počnu ispočetka

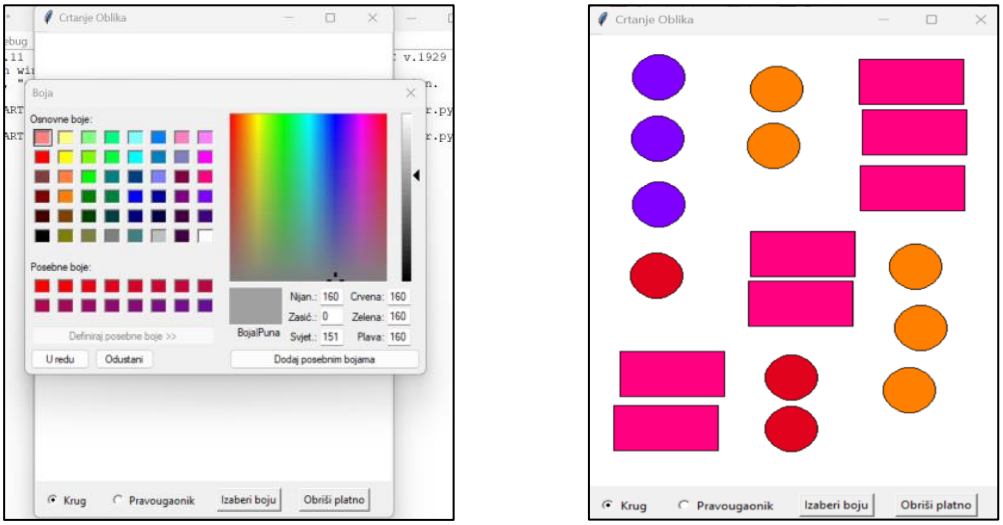
je dat na slici 93.

gui primjer.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/gui primjer.py (3.10.11)

File Edit Format Run Options Window Help

```
import tkinter as tk
from tkinter import colorchooser

# Funkcija za crtanje pravougaonika
def draw_rectangle(event):
    if shape_var.get() == "Rectangle":
        canvas.create_rectangle(event.x - 50, event.y - 25, event.x + 50, event.y + 25, fill=color_var.get())

# Funkcija za crtanje kruga
def draw_circle(event):
    if shape_var.get() == "Circle":
        canvas.create_oval(event.x - 25, event.y - 25, event.x + 25, event.y + 25, fill=color_var.get())

# Funkcija za promenu boje oblika
def change_color():
    color_code = colorchooser.askcolor()[1]
    color_var.set(color_code)

# Funkcija za brisanje platna
def clear_canvas():
    canvas.delete("all")

# Kreiranje glavnog prozora
root = tk.Tk()
root.title("Crtanje Oblika")
```

```
# Varijable za oblik i boju
shape_var = tk.StringVar(value="Circle")  # Podrazumevani oblik je krug
color_var = tk.StringVar(value="black")   # Podrazumevana boja je crna

# Kreiranje widgeta
canvas = tk.Canvas(root, width=500, height=500, bg="white")
canvas.pack()

# Dugmadi za biranje oblika
circle_button = tk.Radiobutton(root, text="Krug", variable=shape_var, value="Circle")
circle_button.pack(side=tk.LEFT, padx=10)

rectangle_button = tk.Radiobutton(root, text="Pravougaonik", variable=shape_var, value="Rectangle")
rectangle_button.pack(side=tk.LEFT, padx=10)

# Dugme za promenu boje
color_button = tk.Button(root, text="Izaberi boju", command=change_color)
color_button.pack(side=tk.LEFT, padx=10)

# Dugme za brisanje platna
clear_button = tk.Button(root, text="Obriši platno", command=clear_canvas)
clear_button.pack(side=tk.LEFT, padx=10)

# Veza između platna i funkcija za crtanje
canvas.bind("<Button-1>", lambda event: draw_circle(event) if shape_var.get() == "Circle" else draw_rectangle(event))

# Pokretanje glavne petlje
root.mainloop()
```



Slika 93. Primjer GUI aplikacije

Objašnjenje koda:

- **Tkinter**: Koristimo Tkinter biblioteku za kreiranje GUI aplikacije. Tkinter omogućava pravljenje prozora, dugmadi, platna i drugih interaktivnih elemenata.
- **Canvas (platno)**: canvas = tk.Canvas(root, width=500, height=500, bg="white") kreira platno na kojem korisnici mogu crtati oblike. Koristi se za crtanje i manipulaciju 2D objektima, kao što su linije, krugovi, pravougaonici, itd.
- **Radiobutton**: Koristimo Radiobutton za omogućavanje korisnicima da izaberu koji oblik žele da crtaju. Ovaj widget se koristi kada postoji nekoliko opcija, a korisnik može izabrati samo jednu.
    - circle_button = tk.Radiobutton(root, text="Krug", variable=shape_var, value="Circle")
    - rectangle_button = tk.Radiobutton(root, text="Pravougaonik", variable=shape_var, value="Rectangle")
- **Colorchooser**: Koristimo colorchooser.askcolor() za omogućavanje korisnicima da izaberu boju koja će se koristiti za oblike. Funkcija colorchooser.askcolor() prikazuje dijalog za odabir boje i vraća RGB vrednosti koje pretvaramo u heksadecimalnu boju koja se koristi za ispunjavanje oblika.
    - color_button = tk.Button(root, text="Izaberi boju", command=change_color)
- **Event Binding**: Koristimo canvas.bind("<Button-1>", lambda event: ...) za povezivanje događaja klika mišem sa funkcijama za crtanje. Kada korisnik klikne na platnu, poziva se funkcija koja crta oblik na osnovu izabranog oblika (krug ili pravougaonik).
- **Brisanje platna**: Dugme **Obriši platno** omogućava korisnicima da očiste platno i počnu iznova. To se postiže pomoću canvas.delete("all"), što briše sve objekte na platnu.

10 RAD SA DATOTEKAMA

Rad sa datotekama u Pythonu je vrlo važan za mnoge aplikacije, od jednostavnih programa koji čitaju i pišu podatke do složenih aplikacija za analizu podataka. Python pruža jednostavne funkcije za otvaranje, čitanje, pisanje i zatvaranje datoteka.

10.1. Otvaranje i zatvaranje datoteka

Za rad sa datotekama u Pythonu koristi se funkcija *open*(). Ona otvara datoteku i omogućava rad sa njom. Sintaksa je:

```
file = open ('ime_datoteke', 'mod')
```

Gdje su:

- 'ime_datoteke': Naziv datoteke koja se želi otvoriti (može biti apsolutni ili relativni put).
- 'mod': Mod koji određuje kako će se datoteka otvoriti.

Najčešći modovi su:

- 'r': Čitanje (read).
- 'w': Pisanje (write).
- 'a': Dodavanje (append).
- 'rb': Čitanje u binarnom formatu.
- 'wb': Pisanje u binarnom formatu.
-

Kada se završi radom sa datotekom, uvijek je preporučljivo zatvoriti je pomoću metode *close*().

```
file.close()
```

10.2. Čitanje datoteka

Postoji nekoliko načina za čitanje sadržaja datoteke u Pythonu, sintaksa:

```
file = open('ime_datoteke.txt', 'r')
content = file.read()
print(content)
file.close()
```

Metoda readline(): Čita jednu liniju iz datoteke.

```
file = open('ime_datoteke.txt', 'r')
line = file.readline()
 while line:
        print(line, end='')      # end='' je da ne bismo dodavali dodatne praznine
```

```
        line = file.readline()
file.close()
```

10.3. Pisanje u datoteku

Za pisanje u datoteku koristimo režim 'w' ili 'a'. Režim 'w' briše sadržaj datoteke ako datoteka već postoji, dok režim 'a' dodaje nove podatke na kraj datoteke.

```
file = open('ime_datoteke.txt', 'w')
file.write('Ovo je neki tekst.\n')
file.write('Ovo je još jedan red.')
 file.close()
```

Dodavanje sadržaja u datoteku:

```
file = open('ime_datoteke.txt', 'a')
file.write('Dodajemo još jedan red u datoteku.\n')
file.close()
```

10.4. Rad sa binarnim datotekama

Ako se radi sa binarnim datotekama, može se koristiti 'rb' i 'wb' modovi za čitanje i pisanje u binarnom formatu.

```
file = open('image.jpg', 'rb')
binary_data = file.read()
print(binary_data)    # Ispisivanje binarnih podataka
file.close()
```

Pisanje u binarnu datoteku:

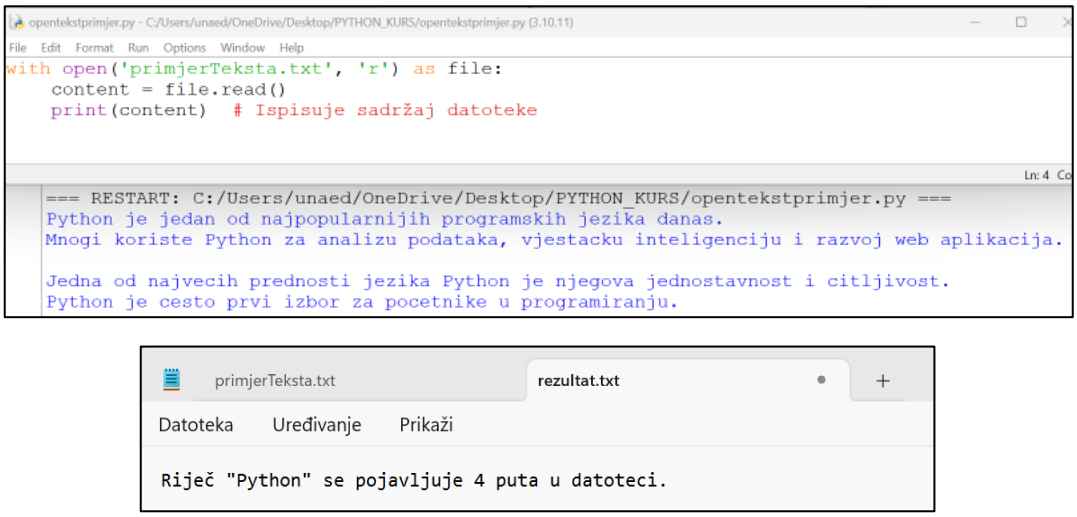
```
file = open('new_image.jpg', 'wb')
file.write(binary_data)
file.close()
```

10.5. Upotreba with izjave za automatsko zatvaranje datoteka

Umjesto da pozivate file.close() eksplicitno, možete koristiti *with* izjavu koja automatski zatvara datoteku kada se završi rad sa njom. Ovo je bolji način jer osigurava da će datoteka biti zatvorena, čak iako dođe do greške u toku obrade.

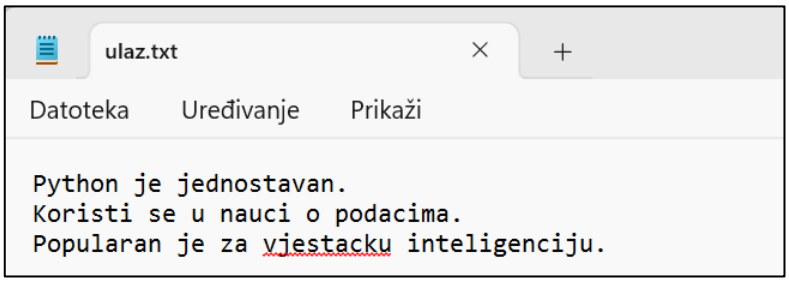
```
with open('ime_datoteke.txt', 'r') as file:
      content = file.read()
      print(content)
```

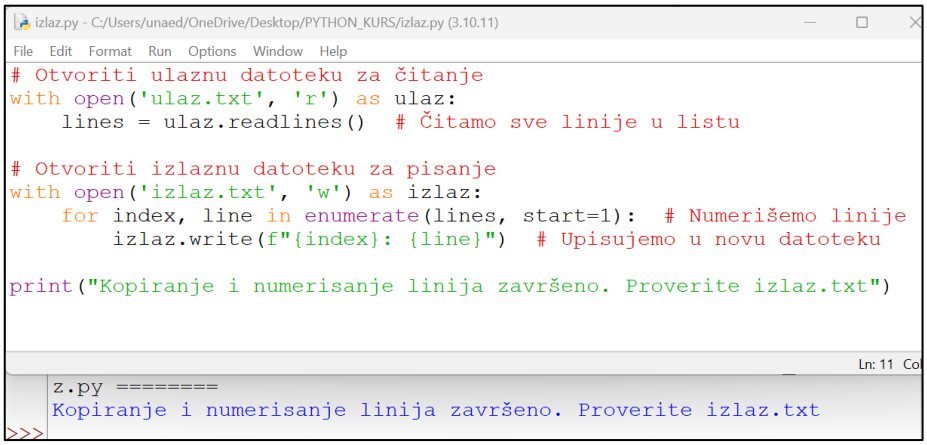
Primjer rada sa datotekama je dat na slici 94. Kreiran je program koji čita sadržaj tekstualne datoteke, broji koliko puta se određena riječ pojavljuje u datoteci, a zatim upisuje broj u novu datoteku.

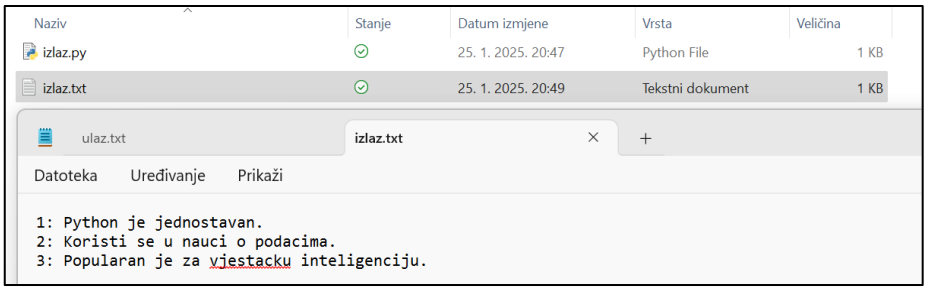


Slika 94. Primjer rada sa datotekom

Drugi primjer, kopiranje sadržaja iz jedne datoteke u drugu i numeriranje linija, je prikazan na slici 95.







Slika 95. Primjer rada sa datotekom

PRILOG

Zadatak 1

Napisati program koji predstavlja jednostavan sistem za upravljanje bibliotekom, omogućavajući korisniku da:

1) Dodaje knjige – Unosi naslov i autora knjige u bazu podataka
2) Pregleda knjige – Prikazuje sve knjige u bazi koristeći tablicu
3) Pretražuje knjige – Omogućava korisniku da pretraži knjige prema naslovu ili autoru.
4) Ažurira podatke knjige – Omogućava korisniku da promijeni naslov ili autora odabrane knjige
5) Briše knjige – Omogućava korisniku da izbriše knjigu iz baze prema njenom ID-u
6) Izlazi iz programa – Zatvara aplikaciju

zadatak_bookmenagment.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/zadatak_bookmenagment.py (3.10.11)

File Edit Format Run Options Window Help

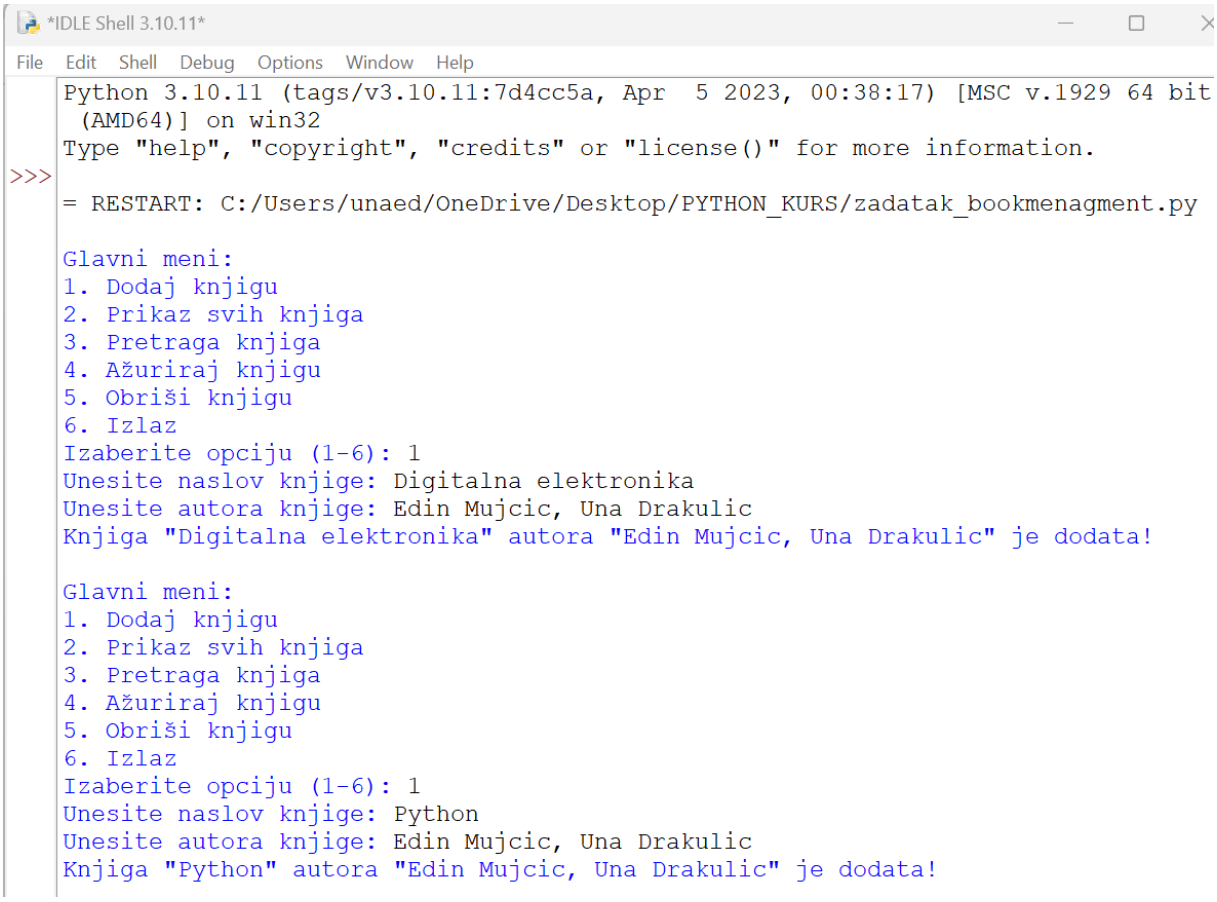
```
import sqlite3
from tabulate import tabulate

def connect_db():
    return sqlite3.connect('books.db')

def create_table():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS books (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            title TEXT NOT NULL,
            author TEXT NOT NULL
        )
    ''')
    conn.commit()
    conn.close()

def add_book():
    title = input("Unesite naslov knjige: ").strip()
    author = input("Unesite autora knjige: ").strip()
    if not title or not author:
        print("Naslov i autor ne mogu biti prazni!")
        return
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('INSERT INTO books (title, author) VALUES (?, ?)', (title, author))
    conn.commit()
    conn.close()
    print(f'Knjiga "{title}" autora "{author}" je dodata!')
```

```
def view_books():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM books')
    books = cursor.fetchall()
    conn.close()
    if books:
        print(tabulate(books, headers=["ID", "Naslov", "Autor"], tablefmt="grid"))
    else:
        print("Nema knjiga u biblioteci.")
```

```
def search_books():
    keyword = input("Unesite naslov ili autora za pretragu: ").strip()
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM books WHERE title LIKE ? OR author LIKE ?", (f'%{keyword}%', f'%{keyword}%'))
    books = cursor.fetchall()
    conn.close()
    if books:
        print(tabulate(books, headers=["ID", "Naslov", "Autor"], tablefmt="grid"))
    else:
        print("Nema rezultata za pretragu.")

def update_book():
    try:
        book_id = int(input("Unesite ID knjige koju želite ažurirati: "))
    except ValueError:
        print("ID mora biti broj!")
        return
    title = input("Unesite novi naslov (ostavite prazno za bez promjene): ").strip()
    author = input("Unesite novog autora (ostavite prazno za bez promjene): ").strip()
    conn = connect_db()
    cursor = conn.cursor()
    if title:
        cursor.execute("UPDATE books SET title = ? WHERE id = ?", (title, book_id))
    if author:
        cursor.execute("UPDATE books SET author = ? WHERE id = ?", (author, book_id))
    conn.commit()
    conn.close()
    print("Podaci su ažurirani!")
```

```
def delete_book():
    try:
        book_id = int(input("Unesite ID knjige koju želite obrisati: "))
    except ValueError:
        print("ID mora biti broj!")
        return
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('DELETE FROM books WHERE id = ?', (book_id,))
    conn.commit()
    conn.close()
    print(f'Knjiga sa ID: {book_id} je obrisana!')

def main_menu():
    create_table()
    while True:
        print("\nGlavni meni:")
        print("1. Dodaj knjigu")
        print("2. Prikaz svih knjiga")
        print("3. Pretraga knjiga")
        print("4. Ažuriraj knjigu")
        print("5. Obriši knjigu")
        print("6. Izlaz")
```

```
        choice = input("Izaberite opciju (1-6): ")
        if choice == '1':
            add_book()
        elif choice == '2':
            view_books()
        elif choice == '3':
            search_books()
        elif choice == '4':
            update_book()
        elif choice == '5':
            delete_book()
        elif choice == '6':
            print("Izlaz iz programa...")
            break
        else:
            print("Nepoznata opcija. Pokušajte ponovo.")

if __name__ == "__main__":
    main_menu()
```




```
Glavni meni:
1. Dodaj knjigu
2. Prikaz svih knjiga
3. Pretraga knjiga
4. Ažuriraj knjigu
5. Obriši knjigu
6. Izlaz
Izaberite opciju (1-6): 2
+------+-----------------------+---------------------------+
|   ID | Naslov                | Autor                     |
+======+=======================+===========================+
|    1 | Digitalna elektronika | Edin Mujcic, Una Drakulic |
+------+-----------------------+---------------------------+
|    2 | Python                | Edin Mujcic, Una Drakulic |
+------+-----------------------+---------------------------+

Glavni meni:
1. Dodaj knjigu
2. Prikaz svih knjiga
3. Pretraga knjiga
4. Ažuriraj knjigu
5. Obriši knjigu
6. Izlaz
Izaberite opciju (1-6): 3
Unesite naslov ili autora za pretragu: Python
+------+----------+---------------------------+
|   ID | Naslov   | Autor                     |
+======+==========+===========================+
|    2 | Python   | Edin Mujcic, Una Drakulic |
+------+----------+---------------------------+
```

```
Glavni meni:
1. Dodaj knjigu
2. Prikaz svih knjiga
3. Pretraga knjiga
4. Ažuriraj knjigu
5. Obriši knjigu
6. Izlaz
Izaberite opciju (1-6): 4
Unesite ID knjige koju želite ažurirati: 2
Unesite novi naslov (ostavite prazno za bez promjene): Osnove programiranja-Python
Unesite novog autora (ostavite prazno za bez promjene):
Podaci su ažurirani!

Glavni meni:
1. Dodaj knjigu
2. Prikaz svih knjiga
3. Pretraga knjiga
4. Ažuriraj knjigu
5. Obriši knjigu
6. Izlaz
Izaberite opciju (1-6): 2
+------+-----------------------------+---------------------------+
|   ID | Naslov                      | Autor                     |
+======+=============================+===========================+
|    1 | Digitalna elektronika       | Edin Mujcic, Una Drakulic |
+------+-----------------------------+---------------------------+
|    2 | Osnove programiranja-Python | Edin Mujcic, Una Drakulic |
+------+-----------------------------+---------------------------+
```

```
Glavni meni:
1. Dodaj knjigu
2. Prikaz svih knjiga
3. Pretraga knjiga
4. Ažuriraj knjigu
5. Obriši knjigu
6. Izlaz
Izaberite opciju (1-6): 5
Unesite ID knjige koju želite obrisati: 2
Knjiga sa ID: 2 je obrisana!

Glavni meni:
1. Dodaj knjigu
2. Prikaz svih knjiga
3. Pretraga knjiga
4. Ažuriraj knjigu
5. Obriši knjigu
6. Izlaz
Izaberite opciju (1-6): 2
+------+-----------------------+---------------------------+
|   ID | Naslov                | Autor                     |
+======+=======================+===========================+
|    1 | Digitalna elektronika | Edin Mujcic, Una Drakulic |
+------+-----------------------+---------------------------+
```

```
Glavni meni:
1. Dodaj knjigu
2. Prikaz svih knjiga
3. Pretraga knjiga
4. Ažuriraj knjigu
5. Obriši knjigu
6. Izlaz
Izaberite opciju (1-6): 6
Izlaz iz programa...
```

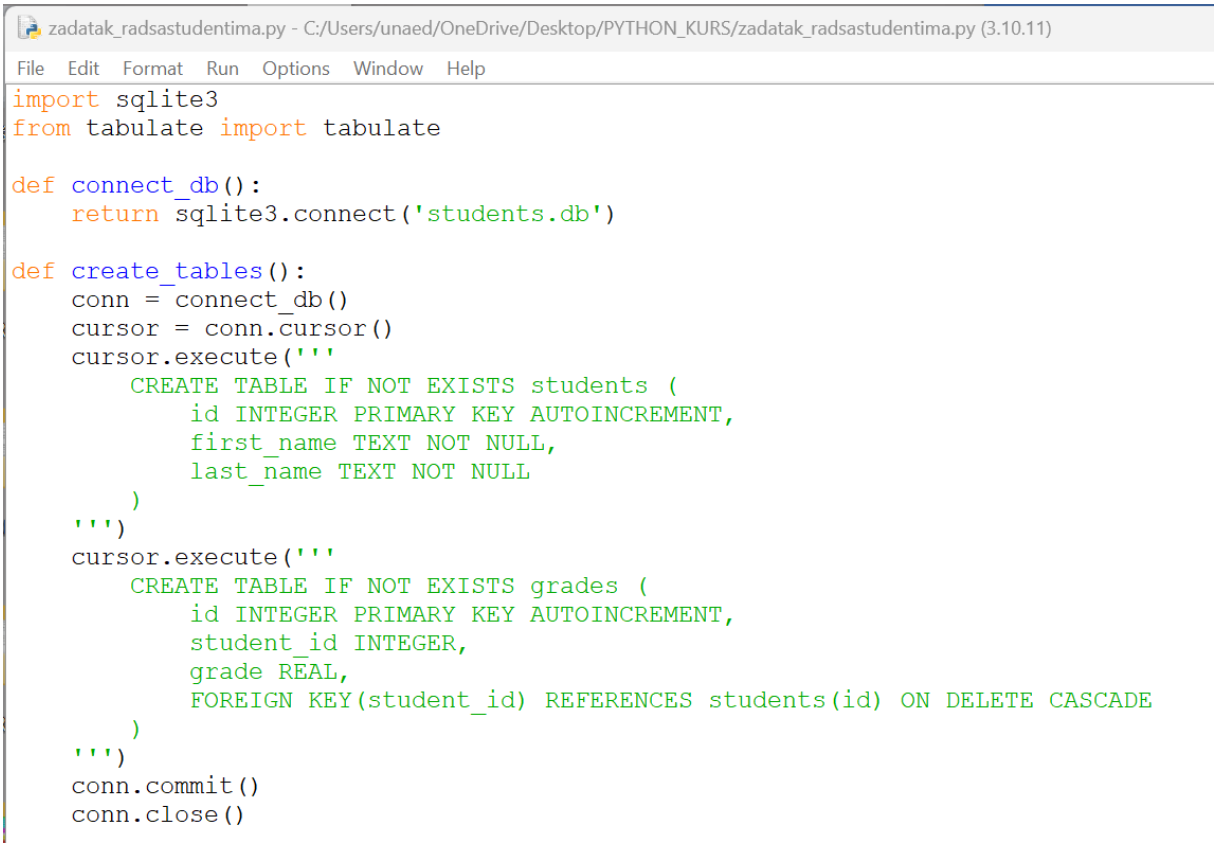
Baza podataka – Kreira SQLite bazu podataka (books.db) ako ne postoji.
Tabela "books" – Kreira tabelu sa ID-em, naslovom i autorom.
Korisnički interfejs – Omogućava korisniku da bira opcije kroz glavni meni.
Validacija unosa – Sprečava unos praznih podataka i neispravnih ID-eva.
Sigurnost i efikasnost – Koristi pripremljene upite (? umjesto formatiranog stringa) radi zaštite od SQL injekcija.

Zadatak 2

Napraviti aplikaciju koja omogućava dodavanje, pregled, ažuriranje i brisanje studenata i njihovih ocjena. Omogućiti računanje prosječne ocjene i sortiranje studenata po prosjeku.

Funkcionalnosti aplikacije su:

- Dodavanje studenta (ime, prezime)
- Dodavanje ocjena studentu
- Prikaz svih studenata sa prosjecima
- Ažuriranje ocjena
- Brisanje studenta
- Sortiranje po prosjeku

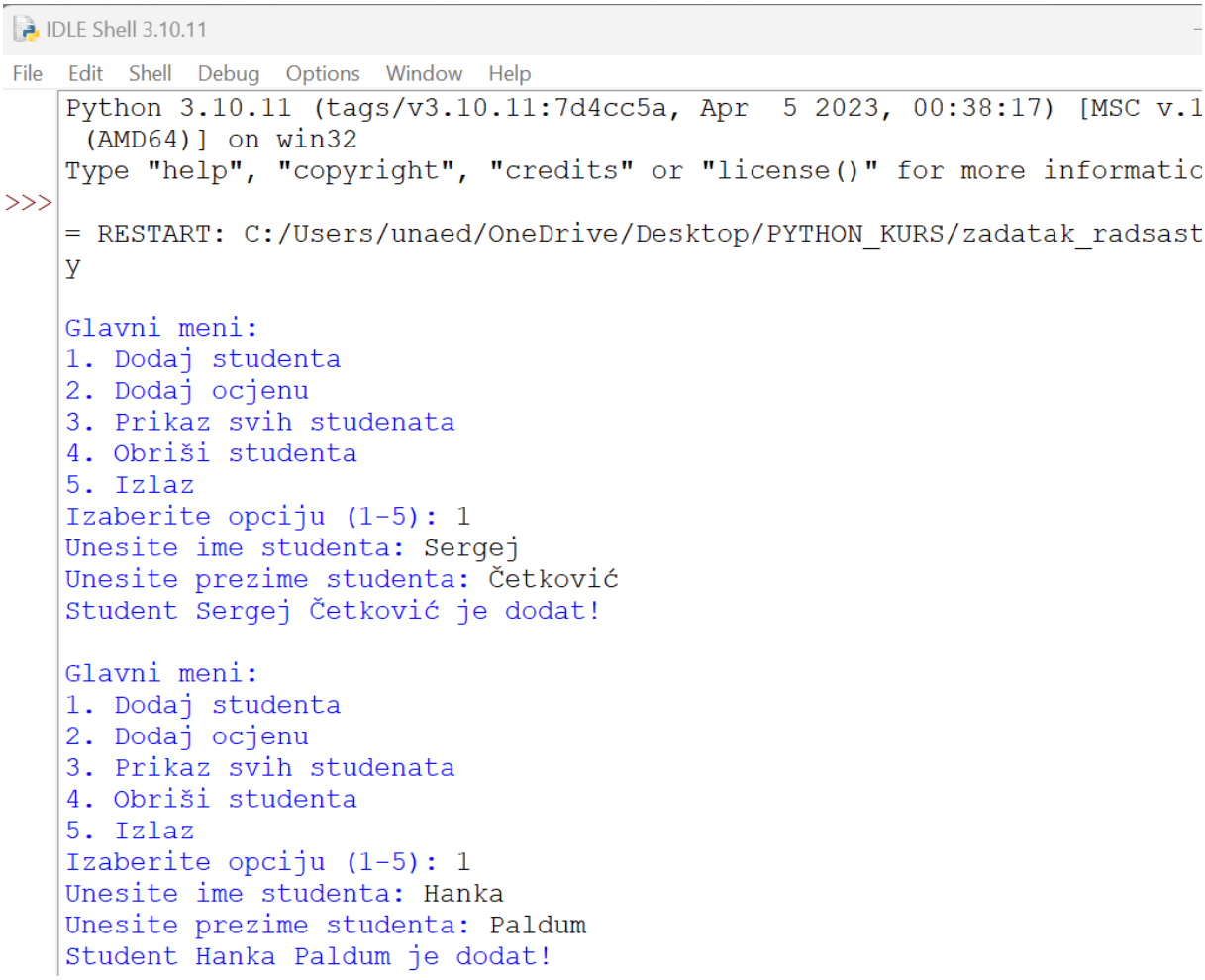


```
def add_student():
    first_name = input("Unesite ime studenta: ").strip()
    last_name = input("Unesite prezime studenta: ").strip()
    if not first_name or not last_name:
        print("Ime i prezime ne mogu biti prazni!")
        return
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('INSERT INTO students (first_name, last_name) VALUES (?, ?)', (first_name, last_name))
    conn.commit()
    conn.close()
    print(f'Student {first_name} {last_name} je dodat!')

def add_grade():
    try:
        student_id = int(input("Unesite ID studenta: "))
        grade = float(input("Unesite ocjenu (0-10): "))
        if grade < 0 or grade > 10:
            print("Ocjena mora biti između 0 i 10!")
            return
    except ValueError:
        print("Neispravan unos!")
        return
```

```
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('INSERT INTO grades (student_id, grade) VALUES (?, ?)', (student_id, grade))
    conn.commit()
    conn.close()
    print("Ocjena je dodana!")

def view_students():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('''
        SELECT s.id, s.first_name, s.last_name, IFNULL(AVG(g.grade), 0) AS avg_grade
        FROM students s
        LEFT JOIN grades g ON s.id = g.student_id
        GROUP BY s.id
        ORDER BY avg_grade DESC
    ''')
    students = cursor.fetchall()
    conn.close()
    if students:
        print(tabulate(students, headers=["ID", "Ime", "Prezime", "Prosječna ocjena"], tablefmt="grid"))
    else:
        print("Nema studenata u sistemu.")
```

```
def delete_student():
    try:
        student_id = int(input("Unesite ID studenta kojeg želite obrisati: "))
    except ValueError:
        print("ID mora biti broj!")
        return
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('DELETE FROM students WHERE id = ?', (student_id,))
    conn.commit()
    conn.close()
    print("Student je obrisan!")

def main_menu():
    create_tables()
    while True:
        print("\nGlavni meni:")
        print("1. Dodaj studenta")
        print("2. Dodaj ocjenu")
        print("3. Prikaz svih studenata")
        print("4. Obriši studenta")
        print("5. Izlaz")
```

```
        choice = input("Izaberite opciju (1-5): ")
        if choice == '1':
            add_student()
        elif choice == '2':
            add_grade()
        elif choice == '3':
            view_students()
        elif choice == '4':
            delete_student()
        elif choice == '5':
            print("Izlaz iz programa...")
            break
        else:
            print("Nepoznata opcija. Pokušajte ponovo.")

if __name__ == "__main__":
    main_menu()
```



```
Glavni meni:
1. Dodaj studenta
2. Dodaj ocjenu
3. Prikaz svih studenata
4. Obriši studenta
5. Izlaz
Izaberite opciju (1-5): 2
Unesite ID studenta: 2
Unesite ocjenu (0-10): 9
Ocjena je dodana!

Glavni meni:
1. Dodaj studenta
2. Dodaj ocjenu
3. Prikaz svih studenata
4. Obriši studenta
5. Izlaz
Izaberite opciju (1-5): 2
Unesite ID studenta: 1
Unesite ocjenu (0-10): 8
Ocjena je dodana!
```

Korištene tehnologije:

- Python + SQLite za backend
- Flask za API (ako želiš web verziju)
- HTML + JavaScript za frontend (opciono)

Zadatak 3

Napisati program koji omogućava korisniku izračunavanje površine i obima za različite geometrijske oblike. Program koristi Python i matematičke funkcije iz biblioteke math za računanje površina i obima oblika kao što su:

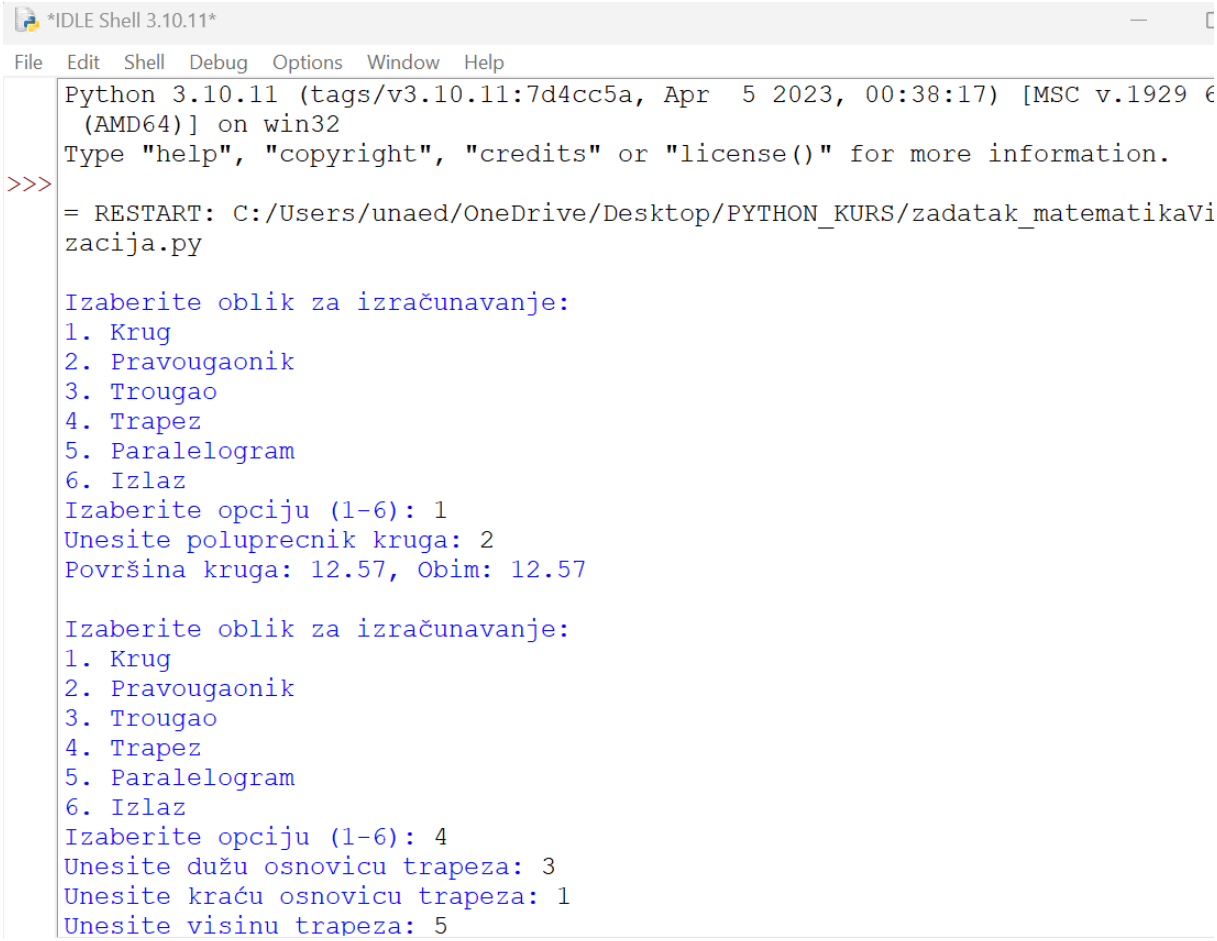
Krug – računa površinu i obim na osnovu unetog poluprečnika.
Pravougaonik – računa površinu i obim koristeći širinu i visinu.
Trougao – računa površinu pomoću Heronove formule na osnovu tri stranice, a obim je zbir svih stranica.
Trapez – računa samo površinu koristeći formule sa osnovicama i visinom.
Paralelogram – računa površinu koristeći osnovicu i visinu.

zadatak_matematikaVizualizacija.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/zadatak_matematikaVi...

File Edit Format Run Options Window Help

```
import math

def calculate_circle():
    radius = float(input("Unesite poluprecnik kruga: "))
    area = math.pi * radius ** 2
    perimeter = 2 * math.pi * radius
    print(f"Površina kruga: {area:.2f}, Obim: {perimeter:.2f}")

def calculate_rectangle():
    width = float(input("Unesite širinu pravougaonika: "))
    height = float(input("Unesite visinu pravougaonika: "))
    area = width * height
    perimeter = 2 * (width + height)
    print(f"Površina pravougaonika: {area:.2f}, Obim: {perimeter:.2f}")

def calculate_triangle():
    a = float(input("Unesite stranicu a: "))
    b = float(input("Unesite stranicu b: "))
    c = float(input("Unesite stranicu c: "))
    s = (a + b + c) / 2
    area = math.sqrt(s * (s - a) * (s - b) * (s - c))
    perimeter = a + b + c
    print(f"Površina trougla: {area:.2f}, Obim: {perimeter:.2f}")

def calculate_trapezoid():
    a = float(input("Unesite dužu osnovicu trapeza: "))
    b = float(input("Unesite kraću osnovicu trapeza: "))
    h = float(input("Unesite visinu trapeza: "))
    area = ((a + b) / 2) * h
    perimeter = None  # Perimeter requires additional side lengths
    print(f"Površina trapeza: {area:.2f}")


def calculate_parallelogram():
    base = float(input("Unesite osnovicu paralelograma: "))
    height = float(input("Unesite visinu paralelograma: "))
    area = base * height
    perimeter = None  # Perimeter requires additional side length
    print(f"Površina paralelograma: {area:.2f}")

def main_menu():
    while True:
        print("\nIzaberite oblik za izračunavanje:")
        print("1. Krug")
        print("2. Pravougaonik")
        print("3. Trougao")
        print("4. Trapez")
        print("5. Paralelogram")
        print("6. Izlaz")

        choice = input("Izaberite opciju (1-6): ")
        if choice == '1':
            calculate_circle()
        elif choice == '2':
            calculate_rectangle()
        elif choice == '3':
            calculate_triangle()
        elif choice == '4':
            calculate_trapezoid()
```

```
        elif choice == '5':
            calculate_parallelogram()
        elif choice == '6':
            print("Izlaz iz programa...")
            break
        else:
            print("Nepoznata opcija. Pokušajte ponovo.")

if __name__ == "__main__":
    main_menu()
```



Zadatak 4

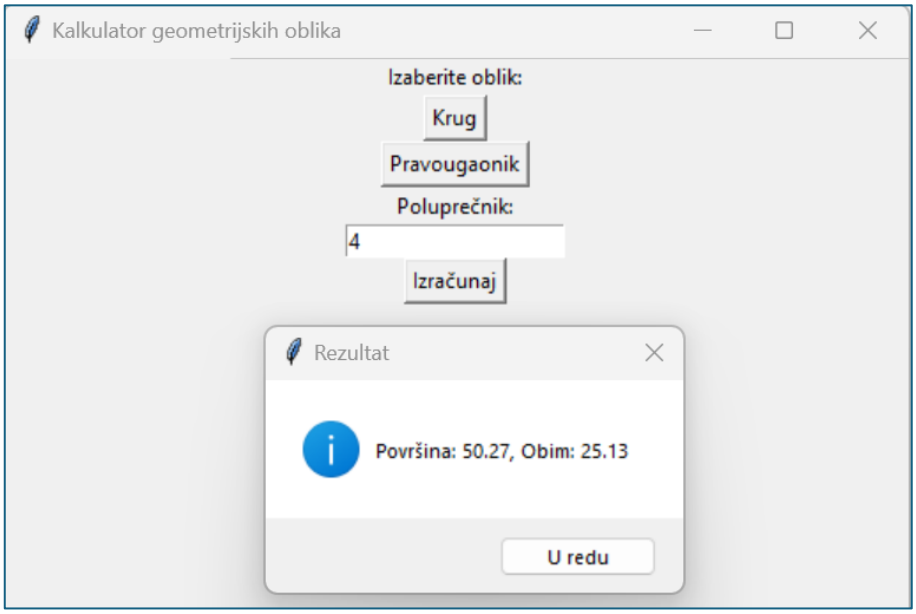
Napisati program koji omogućava korisniku da putem GUI grafičke aplikacije izračuna površinu i obim osnovnih geometrijskih oblika.

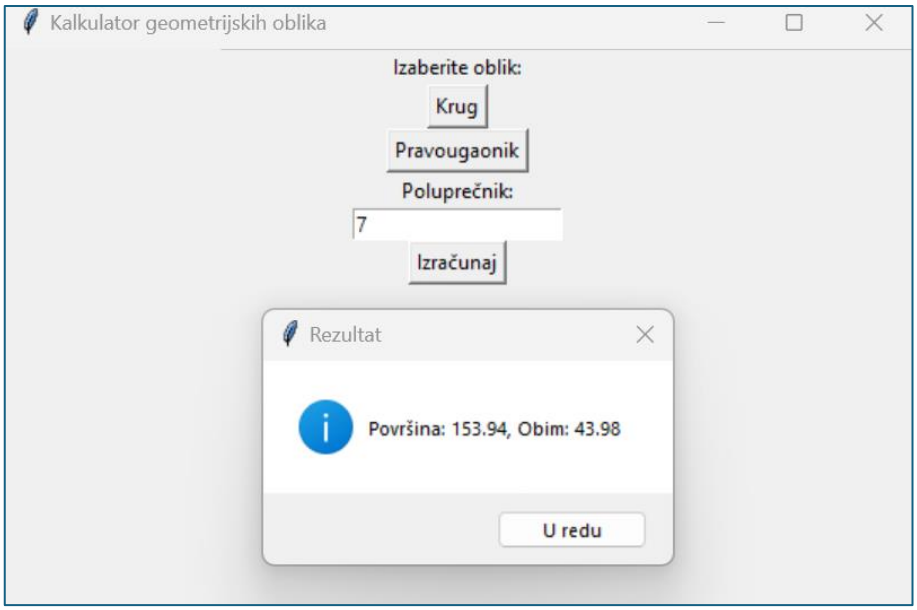
zadatak_GUImatoblici.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/zadatak_GUImatoblici.py (3.10.11)

File Edit Format Run Options Window Help

```
import tkinter as tk
from tkinter import messagebox
import math

def calculate_circle():
    try:
        radius = float(entry_radius.get())
        area = math.pi * radius ** 2
        perimeter = 2 * math.pi * radius
        messagebox.showinfo("Rezultat", f"Površina: {area:.2f}, Obim: {perimeter:.2f}")
    except ValueError:
        messagebox.showerror("Greška", "Unesite ispravan broj!")

def calculate_rectangle():
    try:
        width = float(entry_width.get())
        height = float(entry_height.get())
        area = width * height
        perimeter = 2 * (width + height)
        messagebox.showinfo("Rezultat", f"Površina: {area:.2f}, Obim: {perimeter:.2f}")
    except ValueError:
        messagebox.showerror("Greška", "Unesite ispravan broj!")


def show_circle():
    clear_frame()
    tk.Label(frame, text="Poluprečnik:").pack()
    global entry_radius
    entry_radius = tk.Entry(frame)
    entry_radius.pack()
    tk.Button(frame, text="Izračunaj", command=calculate_circle).pack()

def show_rectangle():
    clear_frame()
    tk.Label(frame, text="Širina:").pack()
    global entry_width
    entry_width = tk.Entry(frame)
    entry_width.pack()
    tk.Label(frame, text="Visina:").pack()
    global entry_height
    entry_height = tk.Entry(frame)
    entry_height.pack()
    tk.Button(frame, text="Izračunaj", command=calculate_rectangle).pack()
```

```
def clear_frame():
    for widget in frame.winfo_children():
        widget.destroy()

# Kreiranje glavnog prozora
root = tk.Tk()
root.title("Kalkulator geometrijskih oblika")

# Glavni meni
tk.Label(root, text="Izaberite oblik:").pack()
tk.Button(root, text="Krug", command=show_circle).pack()
tk.Button(root, text="Pravougaonik", command=show_rectangle).pack()

# Okvir za unos podataka
frame = tk.Frame(root)
frame.pack()

# Pokretanje aplikacije
root.mainloop()
```





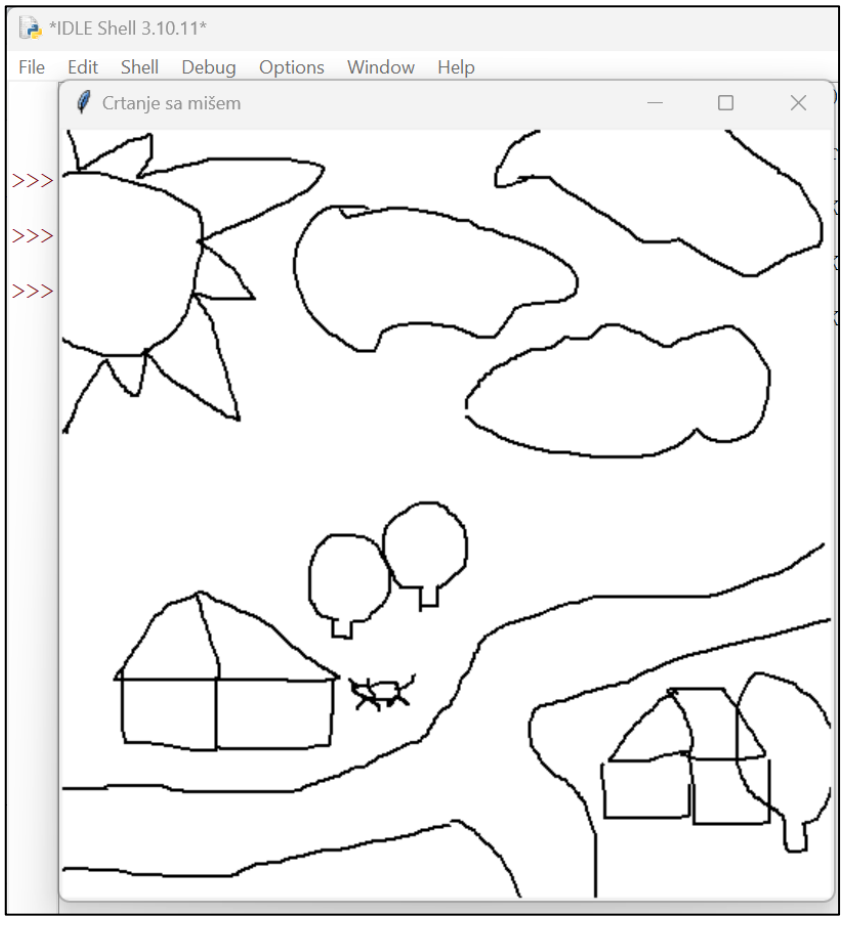
Zadatak 5

Napisati program koji omogućava korisnicima da crtaju slobodne linije na grafičkom platnu koristeći miš. Kada korisnik klikne na platno (lijevi klik miša), početne koordinate (gdje je kliknuo) se pamte. Kada korisnik povuče miš dok drži pritisnut lijevi klik, linija se povlači od tačke gdje je prethodno kliknuo do trenutne pozicije miša. Svaki povučeni segment linije treba da spaja prethodne i trenutne koordinate, stvarajući neprekidnu crtu.

zadatak_crtanje.py - C:/Users/unaed/OneDrive/Desktop/PYTHON_KURS/zadatak_crtanje.py (3.10.11)

File Edit Format Run Options Window Help

```
import tkinter as tk

class DrawingApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Crtanje sa mišem")

        self.canvas = tk.Canvas(self.root, width=500, height=500, bg="white")
        self.canvas.pack()

        self.last_x, self.last_y = None, None
        self.canvas.bind("<Button-1>", self.on_click)
        self.canvas.bind("<B1-Motion>", self.on_drag)

    def on_click(self, event):
        self.last_x, self.last_y = event.x, event.y

    def on_drag(self, event):
        if self.last_x and self.last_y:
            self.canvas.create_line(self.last_x, self.last_y, event.x, event.y,
        self.last_x, self.last_y = event.x, event.y

# Pokretanje aplikacije
root = tk.Tk()
app = DrawingApp(root)
root.mainloop()
```



Literatura

Dr. Edin Mujčić, Mr. Una Drakulić – Univerzitetski udžbenik PYTHON, ISBN 978-9926-508-00-5, Bihać 2022