Univerzitet u Beogradu

Matematički fakultet

Elektronske lekcije o upotrebi biblioteke NumPy u programskom jeziku Pajton, za obradu podataka

MASTER RAD

Student:

Tanja Menković

Mentor:

dr Miroslav Marić

Beograd, 2021.

Sadržaj:

Uvod 5

1. Osnovni pojmovi o biblioteci NumPy 7
   - 1.1. Elektronske lekcije 7
   - 1.2. Istorija biblioteke NumPy 9
   - 1.3. Prednosti korišćenja NumPy niza u odnosu na listu 11
   - 1.4. Okruženje NumPy 12
2. Tipovi podataka u biblioteci NumPy 16
   - 2.1. Kreiranje n-dimenzionalnog niza 19
   - 2.2. Indeksiranje nizova 24
   - 2.3. Tipovi niza skalara 27
   - 2.4. Praktični primeri 34
3. Atributi i metode za rad sa nizovima 36
   - 3.1. Atributi nizova 36
   - 3.2. Metode za rad sa nizovima 40
   - 3.3. Kontrolisanje oblika 43
   - 3.4. Kontrolisanje elemenata 45
   - 3.5. Računske operacije primenom biblioteke NumPy 50
     - 3.5.1. Unarne operacije 54
     - 3.5.2. Operacija poređenja 56
     - 3.5.3. Aritmetika 59
     - 3.5.4. Množenje matrica 62
   - 3.6. Rešavanje sistema jednačina uz pomoć biblioteke NumPy . 67

3.7. Specijalne metod ……. 68

3.8. Praktični primeri ……. 78

4. Linearna regresija 81

4.1. Metoda najmanjih kvadrata 82

4.2. Višestruka linearna regresija 88

4.3. Praktični primeri 90

5. Rešenja praktičnih primera 91

5.1. Rešeni praktični primeri iz poglavlja 2 91

5.2. Rešeni praktični primeri iz poglavlja 3 99

5.3. Rešeni praktični primeri iz poglavlja 4 110

Zaključak 114

Literatura 115

Uvod

U današnje vreme teško je zamisliti da jedan dan prođe bez upotrebe interneta. Internet je znatno olakšao komunikaciju između ljudi širom sveta, kao i trgovinu, poslovanje i traženje bilo kakvog vida informacija. Nekada, kada bi želeli da dođu do neke nove informacije i da nauče nešto novo, ljudi bi pomoć tražili u bibliotekama, provodeći i po više sati samo da bi došli do tražene informacije. Danas, taj ceo proces traje i po samo nekoliko sekundi, jer internet omogućava pristup različitim tipovima kako knjiga, tako i konkretnih lekcija, širom sveta. Iz tog razloga, sve se više nailazi na elektronske lekcije na internetu, u vezi sa različitim oblastima, kojima može da se pristupi lako, brzo i često besplatno iz sopstvenog doma, preko računara, tableta ili mobilnih telefona.

U poslednje vreme, dolazi do sve veće potražnje za IT stručnjacima. Samim tim, nailazi se na internetu na velik broj video zapisa i elektronskih lekcija o različitim oblastima programiranja, koje omogućavaju da se lakše i brže nauči željeni programski jezik. Još uvek su lekcije u najvećem broju na engleskom jeziku, zbog čega se u okviru Matematičkog Univerziteta u Beogradu radi na stvaranju sve većeg broja lekcija na srpskom jeziku, da bi se olakšalo učenje budućim IT stručnjacima koji dolaze iz Srbije.

Ukoliko neko želi da se bavi analizom podataka[1], među najzastupljenijim programskim jezicima današnjice su R i Pajton. Pajton postaje sve traženiji, zbog jednostavne i čitljive sintakse i mnogobrojnih biblioteka[2] u okviru njega. Kada je reč o programskom jeziku Pajton, postoje mnogobrojne lekcije, knjige i video zapisi i na srpskom jeziku. Međutim, kada je potrebno da se pređe na sledeći nivo u učenju i radi sa nekom od biblioteka neophodnih za analizu podataka, pristup lekcijama na srpskom jeziku veoma je ograničen, šta više, za neke biblioteke ih još uvek nema. Iz tog razloga, za potrebe ovog rada kreirane su elektronske lekcije za jednu od biblioteka neophodnih za analizu podataka, odnosno, biblioteku NumPy, u programskom jeziku Pajton.

Biblioteka NumPy se koristi za lakše baratanje matematičkim operacijama i izrazima, čime predstavlja dobru zamenu za MatLab. Takođe, u kombinaciji sa bibliotekom Matplotlib, predstavlja odličan alat za statističke proračune i njihovo grafičko predstavljanje. Pored toga, biblioteka NumPy je korisna za backend programiranje u kombinaciji sa bibliotekom Pandas. U poslednje vreme, mašinsko učenje postaje sve popularnije i sve traženije. Da bi se isprogramirala takva aplikacija u programskom jeziku Pajton, neophodna je upotreba i biblioteke NumPy.

Elektronske lekcije su organizovane tako da postepeno uvode čitaoca u svaku temu koja je obrađena u radu. Svaka sledeća lekcija nadovezuje se na prethodnu i sa svakom novom lekcijom prelazi se na viši stepen razumevanja gradiva. Elektronske lekcije su kreirane tako da se

[1] Analiza podataka je proces pregleda, čišćenja, transformisanja i modelovanja podataka sa ciljem otkrivanja korisnih informacija, informisanja o donesenim zaključcima i podrške u donošenju odluka.

[2] Pored biblioteke Numpy, za bavljenje analizom podataka su neophodne i biblioteke poput: Pandas (za manipulaciju podataka i analizu), Matplotlib (za vizualizaciju podataka), Seaborn (za vizualizaciju, baziran na Matplotlib), SciPy (za resavanje matematičkih, naučnih, inženjerskih i tehničkih problema, baziran na NumPy) itd.

na samom početku rada objašnjavaju osnovni pojmovi, koji se nakon toga obrađuju kroz jednostavne primere. Nakon svake lekcije se nalaze praktični primeri za samostalno vežbanje naučenog gradiva, čija se rešenja detaljno objašnjena nalaze nakon svih obrađenih lekcija.

Na početku samoga rada, u prvom poglavlju, biće objašnjeno kako su napravljene elektronske lekcije i sa kojim ciljem. Zatim se korisnici postepeno uvode u istorijat biblioteke NumPy i koje su prednosti n-dimenzionalnih nizova u odnosu na liste. Pre početka samog rada sa bibliotekom NumPy, biće objašnjeno i kako da se izvrši instalacija ove biblioteke, kao i koja platforma se preporučuje za rad sa njom.

Nakon toga, u drugom poglavlju, korisnici će da se uvedu u osnovne pojmove, tako što će biti obrađena tri osnovna objekta u biblioteci NumPy, odnosno n-dimenzionalni nizovi, tipovi podataka i nizovi skalara.

Kada je korisnik u mogućnosti da razume osnovne pojmove, atribute i metode koje pruža ova biblioteka, biće obrađena linearna regresija. Time se čitalac uvodi u analizu podataka i ostavljena je mogućnost za dalje samostalno nadograđivanje stečenog znanja.

1. Osnovni pojmovi o biblioteci NumPy

U ovom poglavlju su obrađene elektronske lekcije koje su napravljene za potrebe ovoga rada, kako izgledaju, koji je cilj njihovog pravljenja, kao i način na koji su napravljene. Nakon čega će korisnici biti upućeni u kratku istoriju biblioteke NumPy. Takođe, u ovom poglavlju je objašnjeno koje su prednosti korišćenja biblioteke NumPy u odnosu na liste i iz kog razloga se programeri odlučuju na korišćenje ove biblioteke, pre nego na korišćenje listi. Na samom kraju poglavlja, je obrađeno samo NumPy okruženje, odnosno kako se instalira ova biblioteka. Pored toga je objašnjeno kako se koristi skup programa „Anaconda“, koji olakšava izvršavanje Pajton koda.

1.1. Elektronske lekcije

Cilj rada i elektronskih lekcija je da se početnicima omogući da nauče i savladaju osnovne segmente biblioteke NumPy i da dobiju dobru osnovu, na koju kasnije mogu da nadograde naprednije znanje. Elektronske lekcije su namenjene kako onima koji se prvi put susreću sa ovom bibliotekom, tako i onima koji žele da prošire svoje znanje. Organizovane su tako da su prilagođene početnicima, uz to da su dovoljno detaljne da mogu i da prošire znanje onom čitaocu koji već zna neke osnove vezane za ovu biblioteku. Za potpuno razumevanje sadržaja ovih lekcija, neophodno je osnovno znanje iz programskog jezika Pajton.

Elektronske lekcije su na srpskom jeziku, napisane na jednostavan i čitljiv način i javno su dostupne na adresi:

http://alas.matf.bg.ac.rs/~mv11141

Kada se pristupi stranici, ona izgleda kao što je prikazano na slici 1.1.1.

Slika 1.1.1: Prva strana koja se otvara pri pristupu elektronskim lekcijama

Kao što se može uočiti na slici 1.1.1, sa leve strane se nalazi sadržaj, gde se klikom na određenu celinu pristupa upravo njoj, da ne bi čitalac morao da lista sve lekcije, ukoliko ga zanima neki konkretan deo. Pored sadržaja se nalazi lekcija na koju je čitalac kliknuo. Na slici se mogu primetiti i strelice iznad naslova lekcije, koje omogućavaju vraćanje na prethodnu lekciju ili odlazak na sledeću.

Za izradu ove html prezentacije korišćen je css framework poznat pod imenom „*skeleton*“, koji se može pronaći na adresi:

http://getskeleton.com/

Za dinamičko učitavanje sadržaja, kao i obrađivanje svih događaja na stranici, korišćena je biblioteka *jQuery*, koja se može pronaći na adresi:

https://jquery.com/

JavaScript kod koji obrađuje ove događaje, može se pronaći na adresi:

http://alas.matf.bg.ac.rs/~mv11141/dist/main.js

Pri izradi rada, najviše vremena uloženo je na sadržaj lekcija, ali i na to da stranica ne bude preokupirana nepotrebnim podacima, već da bude jednostavno izrađena, u neupadljivim bojama, da bi se time dalo na značaju samom sadržaju i da se ne bi pažnja skretala sa njega.

Sam sadržaj elektronskih lekcija je organizovan tako da se čitalac prvo uputi u to šta je ustvari biblioteka NumPy i zbog čega je ona značajna, kao i da se uputi u kratak istorijat. Nakon toga je objašnjeno kako se vrši instalacija biblioteke, tek nakon čega slede konkretne lekcije. Tok izrade lekcija teče postepeno, krenuvši od osnova i nadograđujući nove segmente na njih, pri čemu se kroz svaku lekciju protežu jednostavni primeri koji demonstriraju upotrebu pojmova iz date lekcije, da bi na samom kraju bili konkretni primeri iz analize podataka, gde se može uočiti jedna od upotreba ove biblioteke.

Unutar lekcija, primeri su posebno naglašeni (slika 1.1.2), da bi bili uočljiviji u odnosu na teorijski deo, iz razloga što su primeri od velikog značaja korisnicima elektronskih lekcija radi boljeg razumevanja sadržaja.

Lekcije su organizovane tako da se u svakoj lekciji prvo teorijski objasni čitaocu gradivo koje se obrađuje, da bi se nakon toga ono praktično demonstriralo kroz neke od jednostavnijih primera. Svaki primer je detaljno objašnjen kroz komentare vezane za kod kao na slici 1.1.2.

```
# ukoliko ne želimo da pišemo numpy, već skraćenicu,
# definišemo to u prvoj liniji koda pomoću as
# ovde ćemo umesto numpy koristiti skraćenicu np:
import numpy as np

#kreiranje objekta n-dimenzionog niza zadatog u primeru:
f = np.array([[1,2,3],[1,2,3]])

#ispisivanje tipa objekta f:
print("Niz je tipa: ", type(f))

#ispisivanje dimenzije (axes):
print("Niz je dimenzije: ", f.ndim)

#ispisivanje oblika niza:
print("Niz je oblika: ", f.shape)

#ispisivanje veličine niza:
print("Niz je velicine: ", f.size)

#ispisivanje tipa elemenata niza:
print("Tip elemenata niza je: ", f.dtype)
```

Slika 1.1.2: Izgled dizajna gde se nalaze kodovi u okviru elektronskih lekcija

1.2. Istorija biblioteke NumPy

Naziv Pajton (engl. *Python*) potiče od naziva britanske komedijaške grupe „Monty Python“[3], a ne od naziva istoimene zmije, kako se često pretpostavlja. Nastao je 1991. godine, zbog čega se smatra relativno novim programskim jezikom. Od samog početka, za Pajton se smatralo da popunjava prazninu, da je to način pisanja skriptova koji „automatizuju dosadne stvari“ ili za brzu izradu prototipa aplikacije koje će se implementirati na jednom ili više drugih jezika. Međutim, tokom poslednjih nekoliko godina, Pajton važi za jedan od glavnih programskih jezika u domenu razvoja softvera, upravljanju infrastrukturom i analizi podataka. Logo za programski jezik Pajton prikazan je na slici 1.2.1.

Slika 1.2.1: Logo za programski jezik Pajton

[3] Komedijaška grupa „Monty Python“ je nastala u maju 1969. godine, u restoranu u blizini Londona, gde su pet britanaca (Grejam Čepmen, Džon Kliz, Erik Ajdl, Teri Džouns i Majkl Pejlin) i amerikanac (animator Teri Gilijam) prvi put seli da diskutuju o zajedničkom radu na novoj BBC komedijskoj seriji. Više o ovome se može pročitati na stranici: http://www.montypython.com/

Uspeh Pajtona se oslanja na nekoliko prednosti koje pruža kako početnicima, tako i stručnjacima:

- Lako se uči;
- Broj funkcija u samom jeziku je skroman, zbog čega ne zahteva mnogo uloženog vremena ili napora da se naprave prvi programi;
- Sintaksa je dizajnirana da bude čitljiva i jednostavna.

Radi boljeg razumevanja programa, svi savremeni programski jezici dizajnirani su tako da se dele na manje celine. Moduli predstavljaju deo softvera koji ima specifičnu funkciju. Primera radi, kada se pravi neka igrica, jedan modul bi bio odgovoran za logiku igre, a drugi modul za prikaz igrice na ekranu. Svaki modul je u zasebnom dokumentu, koji može da bude promenjen nezavisno od ostatka programa. Pri podeli programa na module popravlja se njegova čitljivost i omogućava se programeru i čitaocu programa da se usredsredi na bitna pitanja jednog modula. Ukoliko je kod kvalitetno podeljen na celine, moguće ih je upotrebiti i u nekom drugom kontekstu.

Biblioteke predstavljaju direktorijum koji se sastoji od više biblioteka ili modula. Svaka biblioteka u Pajtonu mora da sadrži poseban dokument koji se naziva *_init_.py*. Ovaj dokument može da bude prazan, i to implicira da direktorijum koji on sadrži je Pajton biblioteka, zbog čega može da se importuje na isti način kao što se moduli importuju.

NumPy[4] predstavlja jednu od mnogobrojnih biblioteka u programskom jeziku Pajton. NumPy, skraćeno od „Numerical Python“ se sastoji od mnoštva numeričkih funkcija, koje služe za obradu višedimenzionalnih nizovnih objekata.

Koristeći NumPy, programer može da izvede sledeće operacije:

- Matematičke i logičke operacije nad nizovima;
- Furijeove transformacije i funkcije za manipulaciju oblicima;
- Operacije povezane sa linearnom algebrom, poput operacija nad vektorima, matricama, rešavanje sistema jednačina i tome slično.

NumPy ima ugrađene funkcije za linearnu algebru i nasumičnu generaciju brojeva.

NumPy se često koristi zajedno sa bibliotekama kao što su *SciPy* (skraćeno od „Scientific Python“) i *Mat-plotlib* (biblioteka za crtanje i podešavanje grafika funkcija), u kombinaciji sa kojima omogućava moderniju i kompletniju zamenu za MatLab.

[4] Prvi put se matrice pojavljuju u okviru Pajtona 1994. godine, kada ih je uveo Džim Fulton (*Jim Fulton*) stvorivši biblioteku Matrice u programskom jeziku Pajton (engl. *Matrix Object in Python*). Nakon toga je Džim Haganin (*Jim Hugunin*) 1995. godine proširio, stvorivši biblioteku pod nazivom Numerička biblioteka (engl. *Numeric*). Posle šest godina Peri Grinfild (*Perry Greenfield*), Rik Vajt (*Rick White*) i Tod Miler (*Todd Miller*) su stvorili biblioteku Numerički nizovi (engl. *Numarray*). Sa bibliotekom NumPy se prvi put susrećemo 2005. godine, kada je stvorio Trevis Olifent (*Travis Oliphant*).

1.3. Prednosti korišćenja NumPy niza u odnosu na liste

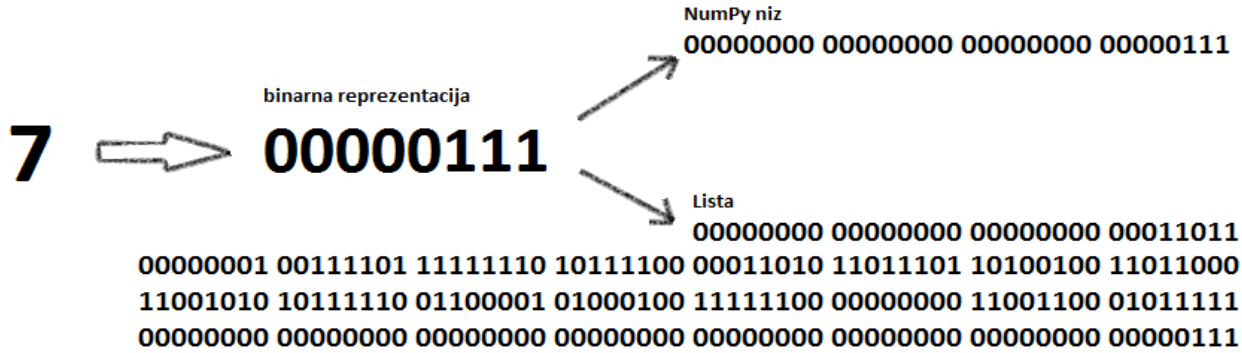
Pri učenju programskog jezika Pajton i biblioteke NumPy, dolazi se do nedoumice kada je potrebno da se odluči između korišćenja nizova kreiranih pomoću biblioteke NumPy ili listi[5]. Glavna razlika između njih proizilazi upravo iz brzine koja se koristi kada se barata sa NumPy nizovnim objektom i sa listom. Liste su jako spore, dok je NumPy niz veoma brz.

Neka se posmatra niz prikazan na slici 1.3.1.

1	24	53	9	0
5	7	5	3	2
7	24	0	53	4
23	5	12	76	21

Slika 1.3.1: Prikaz dvodimenzionog niza

Ukoliko se izdvoji iz niza broj 7, računar njega ne vidi kao broj 7, već kao binarnu reprezentaciju broja 7, odnosno kao binarni broj 00000111. Ovakav broj predstavlja 8 bitova, što čini 1 bajt. Kada se koristi NumPy, ovaj broj će se automatski sačuvati kao int32, odnosno zauzeće 4 bajta: 00000000 00000000 00000000 00000111. Ukoliko nije potrebno toliko memorije da zauzima, može da se specifikuje da bude tipa int16, ili čak i int8. Sa druge strane, sa listama je potrebno mnogo više informacija da se sačuva da bi bile tipa int, a to su: veličina niza, referentni broj, tip objekta i vrednost objekta. Svaka od ovih stavki zauzima određen broj bajtova, preciznije: veličina niza zauzima 4 bajta, dok preostale tri stavke zauzimaju po 8 bajtova (sačuvane su kao tip long). Šematski prikaz broja bitova koje zauzima broj 7 se može uočiti na slici 1.3.2.

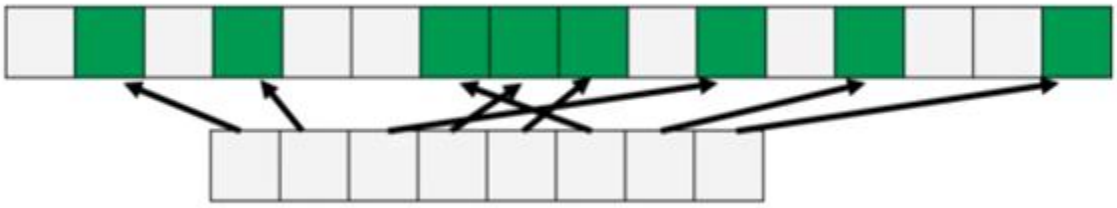


Slika 1.3.2: Šematski prikaz broja bitova koje zauzima podatak u NumPy nizu i u listi

[5] Liste se koriste radi čuvanja više stavki u jednoj promenljivoj. One su jedne od četiri ugrađenih tipova podataka u Pajtonu, koje se koriste radi skladištenja podataka. Preostale tri su torke, rečnici i setovi. Liste se kreiraju korišćenjem četvrtastih zagrada "[]". Više o listama, kao i ostalim osnovnim pojmovima programskog jezika Pajton, može se pronaći u knjizi *Uvod u Python, automatizovanje dosadnih poslova*, izdavač Al Sweigart.

To je jedan od razloga zašto su NumPy nizovi brži za korišćenje od listi, jer koriste manje memorije. Takođe u NumPy nizu nije potrebno svaki put kada se prolazi kroz njega, da se proverava kog je tipa određeni element, dok u listi jeste.

Drugi problem na koji se nailazi sa listama predstavlja raspoređenost elemenata u memorijskom prostoru. Ukoliko se posmatra jedna lista, njeni elementi nisu obavezno u memoriji raspoređeni jedan pored drugog. Dakle ukoliko se posmatra lista od 8 elemenata, oni su razbacani u memoriji kao što je na primer prikazano na slici 1.3.3.



Slika 1.3.3: Prikaz elemenata liste u memorijskom prostoru

Za razliku od listi, elementi NumPy niza se čuvaju u kontinuitetu, jedan pored drugog i nije potrebno da se pamti referentni broj za svaki element, odnosno, putanja gde je on smešten u memoriji, već je dovoljno da se zapamti na primer pozicija prvog elementa, kao što je prikazano na slici 1.3.4.

Slika 1.3.4: Prikaz elemenata NumPy niza u memorijskom prostoru

Dakle, osim što su NumPy nizovi brži za korišćenje od listi, glavna razlika između njih je što sa NumPy nizovima može da se uradi sve isto što i sa listama, ali i mnogo više, o čemu će biti reči u narednim poglavljima.

1.4. Okruženje NumPy

Standardna Pajton instalacija[6] ne dolazi u paketu sa bibliotekom NumPy. Jednostavna alternativa je da se instalira NumPy koristeći popularni program za instalaciju Pajton paketa, pip. Proces instalacije se izvršava veoma jednostavno u terminalu pomoću komande:

[6] Standardna instaliracija programskog jezika Pajton može da se preuzme sa sledeće adrese: https://www.python.org/downloads/

pip install numpy

- ***pip*** – softver za rad sa bibliotekama, koji omogućava lakšu instalaciju, nadograđivanje i lakše prenošenje biblioteka napisanih u Pajtonu; pip predstavlja skraćenicu od „*Preferred Installer Program*“;
- ***install*** – komanda kojom se dodaje željena biblioteka; umesto komande ***install*** može da se koristi i komanda ***uninstall*** ukoliko je potrebno da se ukloni neka biblioteka iz programa;
- ***numpy*** – naziv biblioteke koja se instalira; za potrebe ovoga rada to je biblioteka NumPy, ali na isti način bi se instalirala i bilo koja druga Pajton biblioteka.

Najbolji način da se omogući NumPy je da se koristi binarni paket specifičan za određeni operativni sistem. Ovi binarni paketi sadrže celo SciPy okruženje (uključujući NumPy, SciPy, matplotlib, Ipython, SymPy i još paketa zajedno sa jezgrom Pajtona).

Međutim, moguće je instalirati i NumPy zasebno. Za Windows okruženje koje je korišćeno za potrebe ovih elektronskih lekcija i ovoga rada, dovoljno je preuzeti Pajton instalaciju sa adrese:

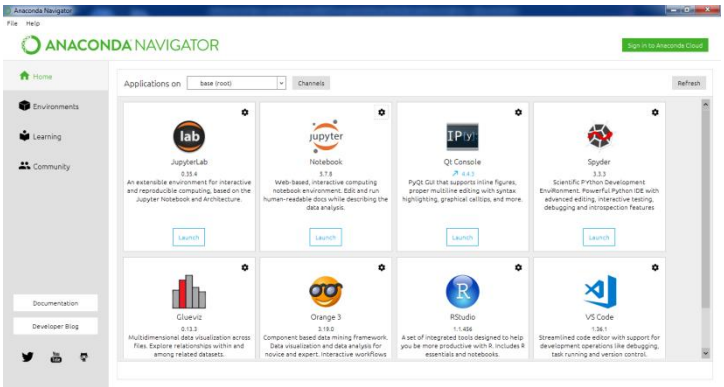
python.org.

Prilikom instalacije potrebno je odabrati da ista uključi pip paket menadžer i nakon instalacije u komandnoj liniji (cmd) izvršiti „pip install numpy“ da bi se dodatno instalirala biblioteka NumPy.

Jedan od načina za izvršavanje Pajton koda je pomoću skupa programa „*Anakonda*“, koji mogu da se preuzmu na sledećoj adresi:

https://www.anaconda.com/distribution/.

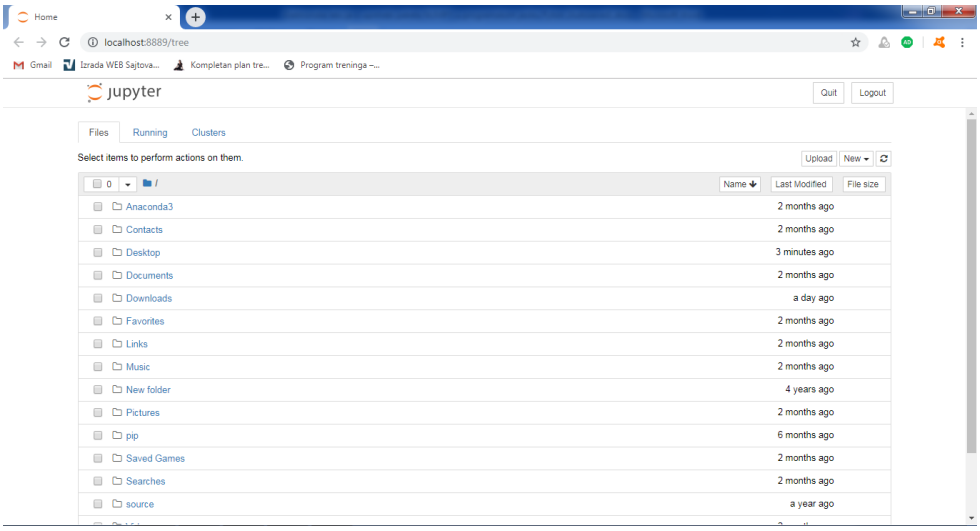
Anakonda distribucija, sa otvorenim kodom, sadrži mnoge alate za rad sa veštačkom inteligencijom, analizom podataka, vizualizacijom rezultata (različite vrste grafika) itd. Dostupna je na svim operativnim sistemima.



Slika 1.4.1: Skup programa koje nudi Anaconda Navigator

Kada se instalira i zatim pokrene „*Anaconda Navigator*“, otvoriće se prozor prikazan na slici 1.4.1.

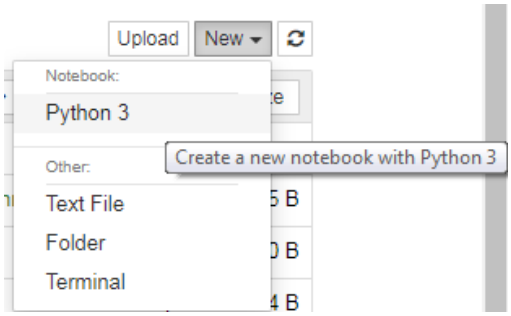
Za potrebe zadataka koji su obrađeni u ovome radu, potrebno je da se pokrene „*jupyter*“. Nakon čega će se otvoriti novi pregledač kao na slici 1.4.2.



Slika 1.4.2: Pregled sistema dokumenata na računaru

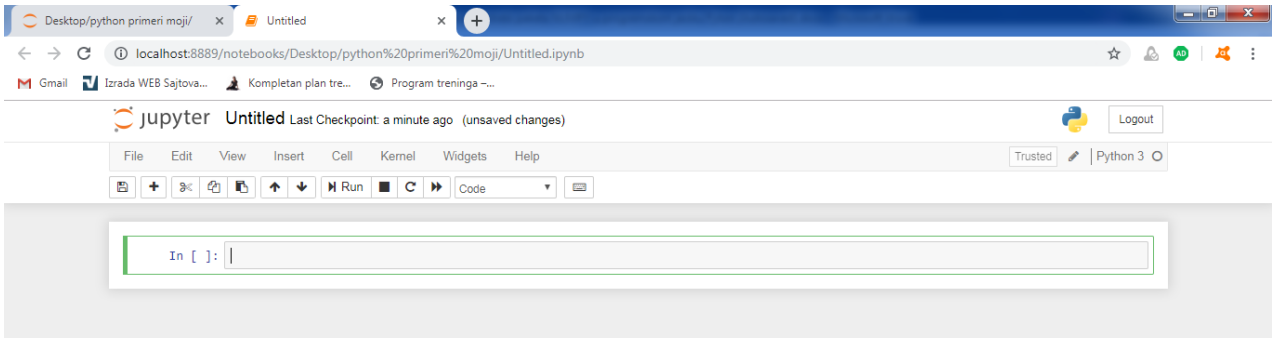
Na slici 1.4.2. može da se uoči da su prikazani svi dokumenti sa računara na kome se radi, od čega je potrebno da se odabere gde će se čuvati primeri koji se kreiraju.

Pri kreiranju novog primera potrebno je da se klikne na opciju *New*, nakon čega se odabere opcija *Python3,* kao što je prikazano na slici 1.4.3.



Slika 1.4.3: Pravljenje novog Pajton primera

Nakon toga će se otvoriti novi prozor sa praznom skriptom, prikazano na slici 1.4.4, u kome se programira željeni zadatak.

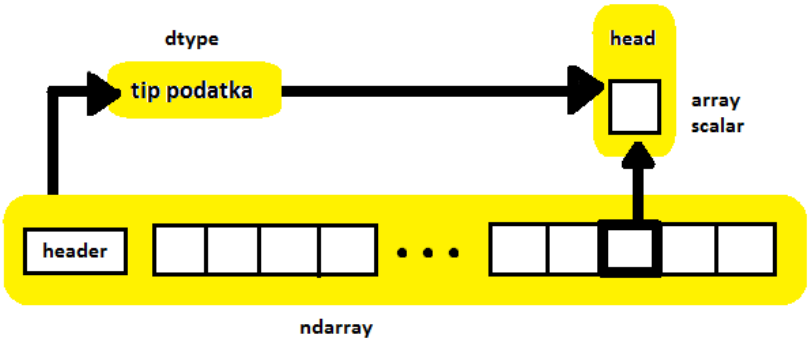


Slika 1.4.4: Prazna jupyter skripta u koju se unosi kod

2. Tipovi podataka u biblioteci NumPy

U NumPy biblioteci, najvažniji objekat, *ndarray*[7], predstavlja n-dimezionalni niz elemenata gde svaki element zauzima neki fiksirani broj bajtova. Obično taj broj bajtova reprezentuje neki broj, celi, kompleksni, realni itd. Međutim taj broj bajtova može da predstavlja zapis sačinjen od bilo kog drugog tipa podatka (karakter, logički tip – boolean, string itd). Svaki NumPy niz ima svoj *dtype* objekat koji opisuje tip podataka u tom nizu.

Ndarray predstavlja kolekciju podataka istoga tipa. Pristupa im se koristeći indekse počevši od 0 do n-1 (gde je n ukupan broj podataka). Ndarray objekat se koristi i za matrice i za vektore. Da bi se konstruisala matrica u biblioteci NumPy, ispisuju se redovi matrice u listi i prosleđuju se konstruktoru NumPy niza.



Slika 2.1: Dijagram veze između ndarray, dtype i array scalar

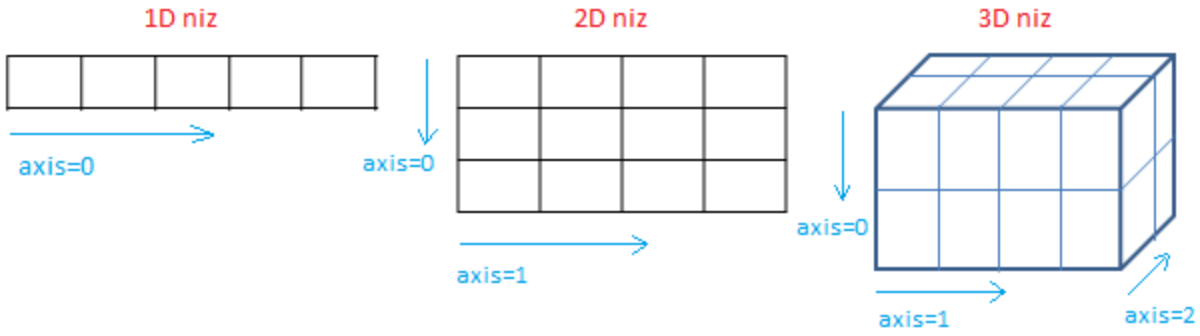
Dijagram na slici 2.1. pokazuje vezu između tri osnovna objekta korišćena da opišu podatak u nizu:

1) ***Ndarray*** predstavlja n-dimenzionalni niz elemenata;
2) ***Dtype*** predstavlja tip podataka objekta koji opisuje plan jednog elementa niza fiksirane veličine;
3) ***array scalar*** predstavlja objekat (niz) koji se vraća kada se pristupi nekom elementu niza.

U biblioteci NumPy dimenzija niza se naziva *axes* (na primer, niz može da bude jednodimenzionalni, dvodimenzionalni itd), a broj dimenzija predstavlja *rank*. Ukoliko je *axes=0*, posmatra se prva dimenzija, ukoliko je *axes=1*, posmatra se druga dimenzija itd. Dakle, za n-tu dimenziju piše se *axes=n-1*.

Prikaz nizova različitih dimenzija mogu da se vide na slici 2.2.

[7] Više o objektu ndarray se može pronaći u knjizi: *Guide to NumPy*, izdavač Travis E. Oliphant



Slika 2.2: Prikaz višedimenzionih nizova

Postoji niz već definisanih dtype objekata. Ovi već ugrađeni tipovi dozvoljavaju brojne numeričke operacije nad celim, realnim i kompleksnim tipovima podataka. Da bi se odredio tip podataka nekog elementa u n-dimenzionalnom nizu (ndarray) koristi se ugrađena funkcija **type** programskog jezika Pajton.

Primer 2.1. *Prikaz tri osnovna objekta NumPy biblioteke***:**

Posmatra se sledeća matrica:

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

NumPy je multidimenziona nizovna biblioteka, što znači da može da se koristi da se sačuvaju različite vrste podataka u jednodimenzionalnom nizu, dvodimenzionalnom nizu, trodimenzionalnom nizu itd.

Za kreiranje niza je korišćena metoda **numpy.array()**, o čemu će detaljnije biti reči u okviru poglavlja 2.1.

Funkcija **type()** omogućava da se pristupi samom tipu niza. U okviru ovog primera, tip je višedimenzionalni niz, odnosno ndarray. Sa druge strane, ukoliko je potrebno pristupiti tipu elemenata toga niza, to se vrši pomoću atributa **dtype**. Ovo je izuzetno važno da se naglasi i da se napravi razlika između ova dva pojma. U ovom primeru su elementi celobrojne vrednosti, odnosno tipa su int.

Ukoliko je potrebno da se odredi dimenzija, tačnije rank niza, to se vrši pomoću atributa **ndim.** U okviru ove matrice, rank iznosi 2, jer je u pitanju dvodimenzionalni niz.

Oblik ovog višedimenzionalnog niza se može prikazati u formatu (2, 3). Da bi se pristupilo obliku niza, koristi se atribut **shape**. Oblik niza znači da je dužina prve dimenzije 2, a druge dimenzije 3.

Veličina niza predstavlja ukupan broj elemenata u nizu. Navedena matrica ima ukupno 6 elemenata, pa je samim tim i njena veličina 6. Veličini se pristupa pomoću atributa **size**.

Radi kraćeg i preglednijeg zapisa, pri uključivanju određene biblioteke unutar programa, koriste se skraćenice. Skraćenica se definiše u prvoj liniji koda pomoću ključne reči **as**. U ovom primeru se umesto naziva „numpy“, uvodi skraćenica „np“.

U nastavku se može uočiti demonstracija primera 2.1. u programskom jeziku Pajton.

```
import numpy as np
# kreiranje objekta n-dimenzionalnog niza zadatog u primeru:
f = np.array([[1,2,3], [1,2,3]])
# ispisivanje tipa objekta f:
print("Niz je tipa: ", type(f))
# ispisivanje dimenzije (axes):
print("Niz je dimenzije: ", f.ndim)
# ispisivanje oblika niza:
print("Niz je oblika: ", f.shape)
# ispisivanje veličine niza:
print("Niz je velicine: ", f.size)
# ispisivanje tipa elemenata niza:
print("Tip elemenata niza je: ", f.dtype)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Niz je tipa: <class ‘numpy.ndarray’>
Niz je dimenzije: 2
Niz je oblika: (2, 3)
Niz je veličine: 6
Tip elemenata niza je: int32
```

2.1. Kreiranje n-dimenzionalnog niza elemenata

Instanca klase ndarray može da se kreira na različite načine. Osnovni način je korišćenjem funkcije za nizove u biblioteci NumPy **numpy.array()**.

Na ovaj način se kreira n-dimenzionalni niz od bilo kog objekta izloženog nizu interfejsa, ili od bilo koje metode koja vraća niz.

numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

- ***object*** – niz ili bilo koji tip koji implementira interfejs niza.
- ***dtype*** – opciono, predstavlja željeni tip podataka niza.
- ***copy*** – opciono, podrazumevano je *True*. Predstavlja kopiranje originalnog objekta.
- ***order*** – ukoliko se izabere opcija **C** onda je značajan red ili ukoliko se izabere **F** onda je značajna kolona ili ukoliko se izabere **A** onda je bilo šta. Podrazumevano je izabrana opcija A.
- ***subok*** – ako je *True* vraćeni niz biće bazni tip podataka (ndarray), inače moguće je dobiti instancu nekih potklasa bazne klase (npr: matrix).
- ***ndmin*** – minimalne dimenzije vraćenog niza.

Pomoću ove funkcije može da se kreira niz od regularnih Pajton listi i torki.

Često, elementi niza su nepoznati, ali je dimenzija poznata. NumPy nudi u tim situacijama nekoliko funkcija za kreiranje nizova. Ovo smanjuje potrebu za povećanjem nizova, što je skupa operacija.

Funkcija **arange()** vraća ravnomerno raspoređene vrednosti u sklopu datog intervala. Vrednosti su generisane sa polu-otvorenim intervalom, odnosno, interval uključuje početnu vrednost (*start*), ali ne uključuje krajnju vrednost (*stop*). Za celobrojne argumente funkcija je ekvivalentna ugrađenoj funkciji **range()** programskog jezika Pajton, ali vraća n-dimenzionalni niz umesto liste.

numpy.arange(start, stop, step, dtype=None)

- ***start*** – opciono, predstavlja broj, početak intervala. Interval uključuje vrednost. Podrazumevana početna vrednost je 0.
- ***stop*** – broj, predstavlja kraj intervala. Interval ne uključuje stop vrednost, osim u nekim slučajevima kada korak nije celobrojna vrednost i tačka zaokruživanja utiče na dužinu intervala.
- ***step*** – opciono, predstavlja broj, korak između dva broja u intervalu.
- ***dtype*** – tip niza koji se vraća. Ako dtype nije dat, uzima se tip podatka od drugih ulaznih argumenata.

Funkcija **linspace** vraća ravnomerno raspoređene vrednosti u sklopu datog intervala. Poslednja vrednost intervala može da bude i isključena.

numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

- ***start*** – opciono, predstavlja broj, početak intervala. Podrazumevana početna vrednost je 0.
- ***stop*** – broj, predstavlja kraj intervala, osim u slučaju kada je *endpoint=False*. U tom slučaju, niz se sastoji od svih elemenata, osim poslednjeg broja uvećanog za 1, ravnomerno raspoređenih, tako da je zaustavljanje isključeno. Obratiti pažnju da se dužina koraka menja kada je *endpoint=False*.
- ***endpoint*** – opciono. Ako je *True,* stop je poslednja vrednost u uzorku koji se vraća, u suprotnom, nije uključena. Podrazumevana vrednost je *True*.
- ***restep*** – opciono. Ako je *True*, vraća uzorak, korak između brojeva u intervalu. Podrazumevano je *False*.
- ***num*** – opciono, predstavlja broj uzoraka koji se generišu, celobrojna vrednost. Podrazumevano je 50. Mora da bude pozitivan broj.
- ***dtype*** – opciono, predstavlja tip niza koji se vraća. Ako dtype nije dat, uzima se tip podatka od drugih ulaznih argumenata.
- ***axis*** – opciono, celobrojna vrednost, dimenzija niza, odnosno osa vektora. Predstavlja rezultat za skladištenje uzoraka. Relevantno je samo ako su start ili stop slični nizu. Na osnovu podrazumevane vrednosti (0), uzorci će biti uz novu osu umetnutu na početak. Koristi se -1 kada je potrebno da osa bude na kraju. Ovo je novo dodato unutar verzije 1.16.0.

Moguće je da se promeni dimenzija niza pomoću funkcije **reshape.** Posmatra se niz oblika $(a_1, a_2, \dots, a_N)$. Moguće je da mu se promeni dimenzija i da se niz konvertuje u drugi niz oblika $(b_1, b_2, \dots, b_M)$. Jedini neophodan uslov je: $a_1 \cdot a_2 \cdot \dots \cdot a_N = b_1 \cdot b_2 \cdot \dots \cdot b_M$, odnosno, originalna veličina ne sme da se promeni.

numpy.reshape(a, newshape, order=`C`)

- ***a*** – niz kome je potrebno da se promeni dimenzija.
- ***newshape*** – celobrojna vrednost ili torka celobrojnih vrednosti. Novi oblik mora da bude kompatibilan sa originalnim oblikom. Ako je celobrojna vrednost, rezultat će biti jednodimenzionalni niz te dužine. Jedna dimenzija može biti oblika -1. U tom slučaju, vrednost se izvodi iz dužine niza i preostalih dimenzija.
- ***order*** – opciono. Čita elemente koristeći ovaj poredak indeksa i stavlja elemente u preoblikovani niz koristeći ovaj isti poredak. Opcija **C** znači čitanje, odnosno upisivanje, elemenata, pri čemu se indeks poslednje ose menja najbrže, nazad do indeksa prve ose koji se najsporije menja. Opcija **F** znači čitanje, osnosno upisivanje, elemenata, pri čemu se prvi indeks najbrže menja, a poslednji najsporije. Potrebno je naglasiti da opcije **C** i **F** ne uzimaju u obzir raspored memorije donjeg niza i odnose se samo na redosled indeksiranja. Opcija **A** znači čitanje, osnosno upisivanje, elemenata, na **F** način, ukoliko je Fortran neprekidan u memoriji, u suprotnom na **C** način.

Korišćenjem funkcije **flatten** dobija se kopija niza „spljoštenog“ u jednu dimenziju. Ova funkcija prihvata samo ***order*** argument. Podrazumevana vrednost je **C** da bi glavni poredak bio u odnosu na red. Koristi se takođe **F** da bi glavni poredak bio u odnosu na kolone.

Primer 2.1.1. *Različiti načini za kreiranje nizova*:

Pomoću metode **numpy.array()** kreiran je dvodimenzionalni niz (matrica):

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix},$$

na osnovu nekadašnje liste oblika:

[[1, 2, 3], [4, 5, 6]].

Tip elemenata ovog niza može eksplicitno da se definiše pri kreiranju niza, tako što se dodeli argumentu **dtype** željeni tip. Podrazumevani tip elemenata ovoga niza je int, isto kao u primeru 5.1, zbog čega je ovde promenjen u float. Ukoliko bi se sada pristupilo tipu elemenata navedene matrice, više ne bi povratna vrednost bila int32, već float. Pri ispisivanju formirane matrice, može da se primeti da su njeni elementi oblika, na primer, *1.* umesto samo *1*, zbog promene tipa.

Takođe pomoću ove metode je moguće kreirati niz na osnovu torke[8]. Na osnovu torke:

(9, 8, 7),

kreiran je niz:

$$b = [9 \quad 8 \quad 7].$$

Često je neophodno kreirati matricu koja kao elemente ima samo nule. To je moguće učiniti slično kao za prethodne dve matrice, pomoću metode **numpy.array()**, međutim, postoji jednostavniji i brži način da se to uradi uz pomoć metode **numpy.zeros((m, n))**, pri čemu *m* i *n* predstavlju dimenzije matrice. Koristeći ovu metodu kreirana je matrica:

$$c = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Pored toga što je moguće kreirati matricu koja se sastoji samo od nula, moguće je kreirati i matricu koja kao elemente ima bilo koji broj, ne samo nulu. To se postiže pomoću metode **numpy.full((m, n), a)**, pri čemu *m* i *n* predstavljaju dimenzije matrice, a *a* predstavlja broj kojim je potrebno da se popuni. U okviru ovog primera je kreirana matrica:

$$d = \begin{bmatrix} 6 & 6 & 6 \\ 6 & 6 & 6 \\ 6 & 6 & 6 \end{bmatrix}.$$

[8] Torka predstavlja nepromenljivu listu i piše se unutar malih zagrada "()".

Ukoliko je potrebno da se definiše i tip za elemente, to se može izvršiti dodavanjem argumenta **dtype**, slično kao za matricu *a*. Tip je postavljen da bude kompleksan broj, iz kog razloga elementi matrice više neće biti *6*, nego *6.+0.j*, pri čemu je *6.* realan deo kompleksnog broja, a *0.* imaginaran zbog čega se uz njega nalazi *j*.

Da bi se kreirao niz sa random elementima iz intervala [0, 1), poziva se funkcija **random()** modula random.

Ukoliko se može uočiti da je razlika između svaka dva uzastopna elementa u nizu konstantna, kao u sledećem nizu:

$$f = [0 \quad 5 \quad 10 \quad 15 \quad 20 \quad 25],$$

može da se koristi metoda **numpy.arange()** za njegovo lakše definisanje.

Sa druge strane, ukoliko već postoji zadati interval i potrebno je da se podeli na jednake segmente, koristi se metoda **numpy.linspace()**.

Moguće je i od postojećeg višedimenzionalnog niza napraviti drugi niz promenom njegovog oblika, pozivom metode **reshape()** nad njim i zadavanjem novih dimenzija unutar zagrada.

Ako je potrebno od višedimenzionalnog niza napraviti jednodimenzionalni niz, to se najlakše postiže pomoću metode **flatten()** koji se poziva nad tim nizom i nema nikakve argumente.

U nastavku se može uočiti demonstracija primera 2.1.1. u programskom jeziku Pajton.

```
import numpy as np
# kreiranje niza od liste sa realnim tipom:
a = np.array([[1, 2, 3], [4, 5, 6]], dtype = 'float')
print ("Niz kreiran iz nekadasnje liste:\n", a)
# kreiranje niza od torke:
b = np.array((9, 8, 7))
print ("\nNiz kreiran iz nekadasnje torke:\n", b)
# kreiranje 3x4 niza sa svim nulama:
c = np.zeros((3, 4))
print ("\nNiz inicijalizovan sa nulama:\n", c)
# kreiranje konstantne vrednosti niza kompleksnog tipa:
d = np.full((3, 3), 6, dtype = 'complex')
print ("\nNiz inicijalizovan sa svim 6s. Tip niza je kompleksan:\n", d)
# kreiranje niza sa random vrednostima:
```

```
e = np.random.random((2, 2))
print ("\nRandom niz:\n", e)
# kreiranje niza celih brojeva od 0 do 30 sa korakom 5:
f = np.arange(0, 30, 5)
print ("\nNiz celih brojeva sa korakom 5:\n", f)
# kreiranje niza od 10 vrednosti u rangu od 0 do 5:
g = np.linspace(0, 5, 10)
print ("\nNiz od 10 vrednosti izmedju 0 i 5:\n", g)
# Promena oblika 3x4 niza u 2x2x3 niz:
h = np.array([[1, 2, 3, 4], [5, 2, 4, 2], [1, 2, 0, 1]])
h1 = h.reshape(2, 2, 3)
print ("\nOriginalni niz:\n", h)
print ("Niz posle promenjenog oblika:\n", h1)
# spljosten niz:
i = np.array([[1, 2, 3], [4, 5, 6]])
i1 = i.flatten()
print ("\nOriginalni niz:\n", i)
print ("Spljosten niz:\n", i1)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Niz kreiran iz nekadasnje liste:
 [[1. 2. 3.]
[4. 5. 6.]]
Niz kreiran iz nekadasnje torke:
[9 8 7]
Niz inicijalizovan sa nulama:
[[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]]
Niz inicijalizovan sa svim 6s. Tip niza je kompleksan:
```

```
[[6.+0.j 6.+0.j 6.+0.j]
[6.+0.j 6.+0.j 6.+0.j]
[6.+0.j 6.+0.j 6.+0.j]]
Random niz:
[[0.90989476 0.84011767]
[0.72367953 0.24145321]]
Niz celih brojeva sa korakom 5:
[0 5 10 15 20 25]
Niz od 10 vrednosti izmedju 0 i 5:
[0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
3.33333333 3.88888889 4.44444444 5.        ]
Originalni niz:
[[1 2 3 4]
[5 2 4 2]
[1 2 0 1]]
Niz posle promenjenog oblika:
[[[1 2 3 ]
[4 5 2]]
[[4 2 1 ]
[2 0 1]]]
Originalni niz:
[[1 2 3]
[4 5 6]]
Spljosteni niz:
[1 2 3 4 5 6]
```

2.2. Indeksiranje nizova

Poznavanje indeksiranja niza je bitno zbog analize i manipulacije nizovnim objektima. Biblioteka NumPy nudi mnogo načina za indeksiranje nizova.

***Srezanje nizova*:**

Isto kao i liste u Pajtonu, NumPy nizovi mogu biti srezani, odnosno moguće je dobiti podniz od već postojećeg niza izdvajanjem samo određenih elemenata. S obzirom da nizovi mogu biti višedimenzionalni, potrebno je da se specifikuje parče za svaku dimenziju niza.

***Celobrojno indeksiranje nizova*:**

U ovoj funkciji, prosleđuje se onoliko listi koliko niz ima dimenzija, pri čemu su elementi lista indeksi određenih elemenata koji su potrebni da se izdvoje iz niza. Svaka navedena lista predstavlja dimenziju u nizu. Jedan na jedan preslikavanje odgovarajućih elemenata izvršava se konstruisanjem novog proizvoljnog niza.

***Boolean indeksiranje nizova*:**

Ova funkcija se koristi kada je potrebno da se izabere element iz niza koji zadovoljava neke uslove.

Primer 2.2.1. *Načini indeksiranja nizova u Pajtonu***:**

Posmatra se matrica:

$$f = \begin{bmatrix} -1 & 2 & 0 & 4 \\ 4 & -0.5 & 6 & 0 \\ 2.6 & 0 & 7 & 8 \\ 3 & -7 & 4 & 2.0 \end{bmatrix}.$$

Ukoliko je potrebno da se kreira novi niz nastao od nekih određenih elemenata ovoga niza (matrice), može da se upotrebi srezanje nizova. Ono se vrši na sledeći način:

$$s = f[p, q],$$

pri čemu *p* označava kriterijum po kome se izdvajaju određeni redovi, a *q* kriterijum po kome se izdvajaju kolone. Kriterijum na osnovu koga se vrši određeno izdvajanje ima oblik *a:b:k*, gde je *a* indeks početnog reda (kolone), *b* indeks krajnjeg reda (kolone), dok je *k* korak i sva tri su opciona. Ukoliko se srezanje vrši od početnoga reda (kolone), indeks *a* može da se izostavi ili ukoliko se srezanje vrši do krajnjeg reda (kolone), indeks *b* može da se izostavi. Ukoliko se posmetraju svi redovi (kolone) sa određenim korakom, mogu da se izostave i indeks *a* i indeks *b*. Iz toga razloga, da bi se izdvojila prva dva reda, može da se kao prvi argument *p* zapiše *:2*, a da bi se izdvojila svaka druga kolona, može da se kao drugi argument *q* zapiše *::2*.

Pri celobrojnom indeksiranju se kreira novi niz koji se sastoji od konkretnih elemenata drugog niza. Vrši se na sledeći način:

$$c = f[p, q],$$

pri čemu *p* označava spisak indeksa redova elemenata koji se izdvajaju, dok je *q* spisak indeksa kolona odgovarajućih elemenata koji se izdvajaju i bitno je da budu navedeni istim redosledom.

U okviru ovoga primera su izdvojeni elementi sa indeksima (0, 3), (1, 2), (2, 1) i (3, 0), tako što je argument *p* postao *0, 1, 2, 3*, dok je argument *q* postao *3, 2, 1, 0*.

Ukoliko je potrebno da se kreira novi niz nastao od elemenata nekog drugog niza izdvojenih po određenom kriterijumu, koristi se Boolean indeksiranje nizova. Vrši se na sledeći način:

$$b = f[p],$$

pri čemu *p* predstavlja kriterijum na osnovu koga se vrši izdvajanje elemenata.

U nastavku se može uočiti demonstracija primera 2.2.1. u programskom jeziku Pajton.

```
import numpy as np
f = np.array([[-1, 2, 0, 4], [4, -0.5, 6, 0], [2.6, 0, 7, 8], [3, -7, 4, 2.0]])
# srezan niz:
s = f[:2, ::2]
print ("Niz od prva dva reda i kolonama 0 i 2:\n", s)
# celobrojno indeksiranje niza:
c = f[[0, 1, 2, 3], [3, 2, 1, 0]]
print ("\nElementi indeksa (0, 3), (1, 2), (2, 1), (3, 0):\n", c)
# boolean indeksiranje niza:
pom = f > 0 # pom je boolean tipa
b = f[pom]
print ("\nElementi veci od 0:\n", b)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Niz od prva dva reda i kolonama 0 i 2:
[[-1. 0.]
[4. 6.]]
Elementi indeksa (0, 3), (1, 2), (2, 1), (3, 0):
[4. 6. 0. 3.]
Elementi veci od 0:
[2. 4. 4. 6. 2.6 7. 8. 3. 4. 2.]
```

2.3. Tipovi niza skalara

U biblioteci NumPy, skalar je bilo koji objekat koji može da se stavi u niz. Svi skalari u nizu imaju isti tip. Nije moguće da jedan skalar ima tip int32, a drugi da ima tip int64. NumPy skalar predstavlja objekat klase **np.generic**, odnosno instancu generičkog tipa skalara.

Primer2.3.1. *Provera da li je prosleđena vrednost skalar*:

Pomoću funkcije **isscalar()** se ispituje da li je prosleđena vrednost skalar. Ova funkcija vraća vrednost *True* ukoliko jeste skalar ili *False* ukoliko nije.

Definisana je funkcija *skalar(x)* koja za vrednost x proverava da li je skalar i nakon toga ispisuje „Vrednost *x* jeste skalar“ ukoliko prosleđena vrednost jeste skalar ili „Vrednost *x* nije skalar“ ukoliko prosleđena vrednost nije skalar.

U nastavku se može uočiti demonstracija primera 2.3.1. u programskom jeziku Pajton.

```
import numpy as np
# definicija funkcije koja proverava da li je nesto skalar
def skalar(x):
    if np.isscalar(x) == True:
        print("Vrednost "+str(x)+ " jeste skalar.")
    else:
        print("Vrednost "+str(x)+ " nije skalar.")
print("Da li su sledece vrednosti skalari?")
skalar(np.int64(2.0))
skalar(3.7)
skalar(16)
skalar(np.float(12))
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Da li su sledece vrednosti skalari?
Vrednost 2 jeste skalar.
Vrednost 3.7 jeste skalar.
```

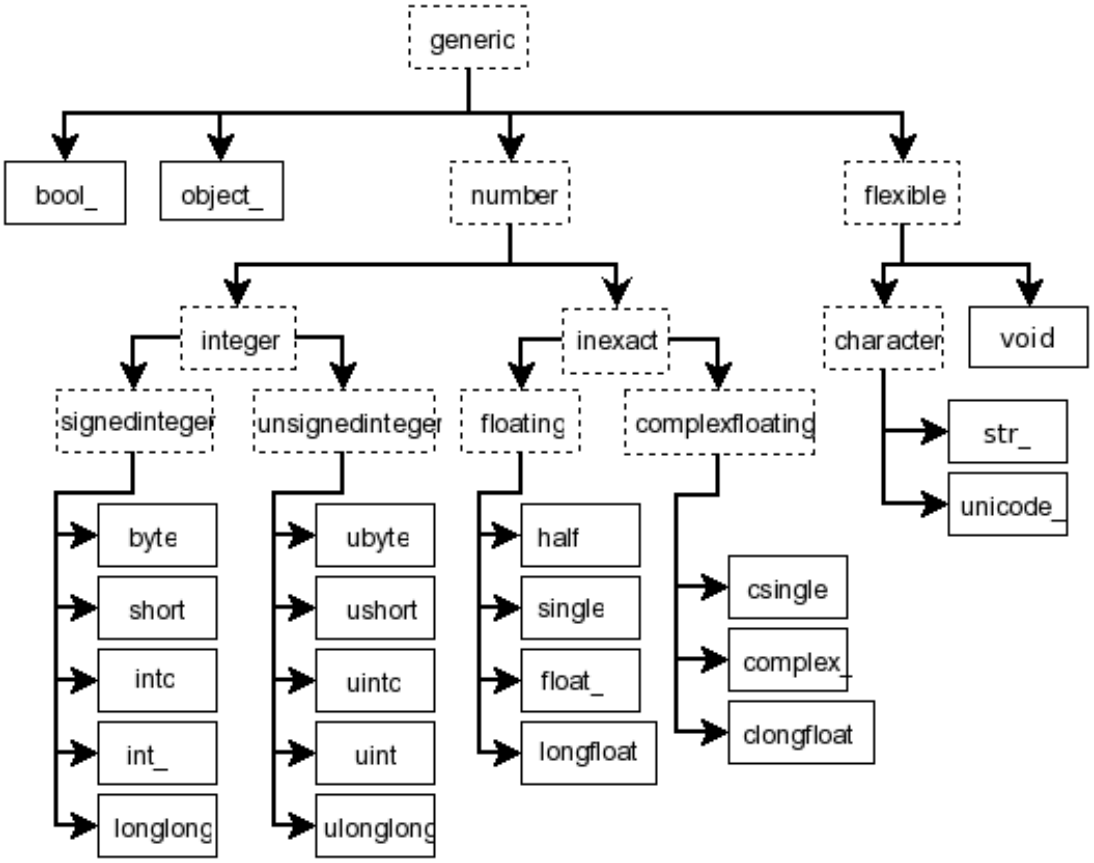
Vrednost 16 jeste skalar.

Vrednost 12.0 jeste skalar.

Biblioteka NumPy podržava mnogo veći izbor numeričkih tipova podataka, nego sam programski jezik Pajton. Pajton podržava samo po jedan tip od određene klase podataka (jedan celobrojni tip, jedan realni tip i td). U okviru biblioteke NumPy postoje 24 nova tipa niza skalara[9]. Jedan od njih (int32) je upotrebljen u primeru 2.1.

Svi skalari imaju iste atribute i metode kao ndarray nizovi.

Postoji određena hijerarhija u kojoj se nalaze tipovi niza skalara, koja je prikazana na slici 2.3.1[10]. Na slici nisu predstavljena dva celobrojna tipa intp i uintp, jer se ne koriste često.



Slika 2.3.1: Hijerarhija tipova niza skalara

[9] Ovi novi tipovi su većinom bazirani na tipove podataka dostupne u programskom jeziku C, na kojem je napisan Cpython, sa nekoliko dodatnih tipova kompatibilnih sa Pajtonovim tipovima. Na primer, int_ je isto što i int64, dok je intc isto što i int32 (ovo zavisi od računara).

[10] Slika je preuzeta sa stranice http://omz-software.com/pythonista/numpy/reference/arrays.scalars.html.

Na osnovu ove hijerarhije tipova podataka, nizovi skalara mogu da se pronađu i moguće je proveriti da li je neka vrednost instanca tipa neke od navedenih grupa unutar hijerarhije.

Primer2.3.2. *Provera da li je vrednost kompleksnog tipa niza skalara*:

Pronalaženje tipova niza skalara je moguće pomoću funkcije isinstance(). Primera radi, **isinstance(x, np.generic)** vraća vrednost *True* ukoliko je x objekat niza skalara i *False* u suprotnom.

U ovom primeru je ispitivano da li je neki broj kompleksnog tipa skalara, odnosno da li spada u grupu *complexfloating*. To se proverava na sledeći način: **isinstance(x, np.complexfloating).** Ukoliko vrednost x spada u tu grupu, vratiće se *True*, ukoliko ne spada *False*.

Definisana je funkcija *kompleksni_tip*, koja ukoliko vrednost *x* jeste skalar kompleksnog tipa, ispisuje „Vrednost *x* jeste kompleksnog tipa. “, dok u suprotnom ispisuje kog je tipa prosleđena vrednost.

U nastavku se može uočiti demonstracija primera 2.3.2. u programskom jeziku Pajton.

```
import numpy as np
# definicija funkcije koja proverava da li je skalar kompleksnog tipa
def kompleksni_tip(x):
    if isinstance(x, np.complexfloating):
        print("Vrednost "+str(x)+ " jeste kompleksnog tipa.")
    else:
        print("Vrednost "+str(x)+ " je tipa: "+str(type(x)))
print("Da li je skalar kompleksnog tipa?")

kompleksni_tip(np.int64(2.0))
kompleksni_tip(3.7)
kompleksni_tip(16)
kompleksni_tip(np.float_(12))
kompleksni_tip(np.complex_(13))
kompleksni_tip(1.5+2j)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Da li je skalar kompleksnog tipa?
Vrednost 2 je tipa: <class 'numpy.int64'>
Vrednost 3.7 je tipa: <class 'float'>
Vrednost 16 je tipa: <class 'int'>
Vrednost 12.0 je tipa: <class 'numpy.float64'>
Vrednost (13+0j) jeste kompelksnog tipa.
Vrednost (1.5+2j) je tipa: <class 'complex' >
```

Ugrađeni tipovi[11] niza skalara su: *int_*, *float_*, *bool_*, *complex_*, *str_*, *unicode_* i tako dalje. Podrazumevani tip podataka u biblioteci NumPy je float_.

Tip bool_ je jako sličan Pajtonovom tipu BooleanType, ali ne nastaje od njega. Tip bool_ ne predstavlja potklasu tipa int_ i ne predstavlja čak ni numerički tip podataka. To je razlika u odnosu na Pajtonov podrazumevani tip bool, koji je potklasa tipa int (True može da se posmatra kao 1, a False može da se posmatra kao 0).

U nastavku teksta nalaze se tabele sa svim tipovima niza skalara, napomenama i kodom. Napomena predstavlja objašnjenje za određene tipove, dok kod predstavlja kratku reprezentaciju tog tipa. Svaki ugrađeni tip ima svoj karakter kod, primera radi, neoznačeni ceo broj (Intc) je predstavljen kao „i“.

Logički tipovi elemenata su predstavljeni u tabeli 2.3.1, zajedno sa napomenom o njima i kodom.

Tabela 2.3.1. *Logički tipovi elemenata*:

Tip	**Napomena**	**Kod**
bool_	Kompatibilno sa ugrađenim tipom bool u programskom jeziku Pajton.	'?'
bool8	8 bita.	

Celobrojni tipovi elemenata su predstavljeni u tabeli 2.3.2, zajedno sa napomenom o njima i kodom.

[11] Ugrađeni tipovi su osnovni tipovi podataka koji dolaze sa osnovnom instalacijom programskog jezika Pajton.

Tabela 2.3.2. *Celobrojni tipovi elemenata*:

Tip	Napomena	Kod
byte	Kompatibilno sa tipom podataka char u programskom jeziku C.	'b'
short	Kompatibilno sa tipom podataka short u programskom jeziku C.	'h'
intc	Kompatibilno sa tipom podataka int u programskom jeziku C.	'i'
int_	Kompatibilno sa ugrađnim tipom int u programskom jeziku Pajton.	'l'
longlong	Kompatibilno sa tipom podataka long long u programskom jeziku C.	'q'
intp	Koristi se za čuvanje pokazivača.	'p'
int8	8 bita.	
int16	16 bita.	
int32	32 bita.	
int64	64 bita.	

Neoznačeni celobrojni tipovi elemenata su predstavljeni u tabeli 2.3.3, zajedno sa napomenom o njima i kodom.

Tabela 2.3.3. *Neoznačeni celobrojni tipovi elemenata*:

Tip	Napomena	Kod
ubyte	Kompatibilno sa tipom podataka unsigned char u programskom jeziku C.	'B'
ushort	Kompatibilno sa tipom podataka unsigned short u programskom jeziku C.	'H'
uintc	Kompatibilno sa tipom podataka unsigned int u programskom jeziku C.	'I'
uint	Kompatibilno sa ugrađenim tipom podataka int u programskom jeziku Pajton.	'L'
ulonglong	Kompatibilno sa tipom podataka long long u programskom jeziku C.	'Q'
uintp	Koristi se za čuvanje pokazivača.	'P'
uint8	8 bita.	

uint16	16 bita.	
uint32	32 bita.	
uint64	64 bita.	

Realni tipovi elemenata su predstavljeni u tabeli 2.3.4, zajedno sa napomenom o njima i kodom.

Tabela 2.3.4. *Realni tipovi elemenata*:

Tip	Napomena	Kod
half		'e'
single	Kompatibilno sa tipom podataka float programskog jezika C.	'f'
double	Kompatibilno sa tipom podataka double programskog jezika C.	
float_	Kompatibilno sa ugrađenim tipom podataka float programskog jezika Pajton.	'd'
longfloat	Kompatibilno sa tipom podataka long float programskog jezika C.	'g'
float16	16 bita.	
float32	32 bita.	
float64	64 bita.	
float96	96 bita, zavisi od platforme – os.	
float128	128 bita, zavisi od platforme – os.	

Kompleksni tipovi elemenata su predstavljeni u tabeli 2.3.5, zajedno sa napomenom o njima i kodom.

Tabela 2.3.5. *Kompleksni tipovi elemanata*:

Tip	Napomena	Kod
csingle		'F'
complex_	Kompatibilno sa ugrađenim tipom podataka complex programskog jezika Pajton.	'D'
clongfloat		'G'

complex64	2 puta po 32-bita.	
complex128	2 puta po 64-bita.	
complex192	2 puta po 96-bita, zavisi od platform.	
complex256	2 puta po 128bita, zavisi od platform.	

Tipovi elemenata bilo kog objekta su predstavljeni u tabeli 2.3.6, zajedno sa napomenom o njima i kodom.

Tabela 2.3.6. *Tipovi elemenata bilo kog objekta*:

Tip	Napomena	Kod
object_	Bilo koji Pajton objekat.	'O'

Fleksibilni tipovi elemenata su predstavljeni u tabeli 2.3.7, zajedno sa napomenom o njima i kodom.

Tabela 2.3.7. *Fleksibilni tipovi elemenata*:

Tip	Napomena	Kod
str_	Kompatibilno sa ugrađenim tipom str u programskom jeziku Pajton.	'S#'
unicode_	Kompatibilno sa ugrađenim tipom unicode u programskom jeziku Pajton.	'U#'
void		'V#'

Primer 2.3.3. *Izlistavanje svih tipova podataka biblioteke NumPy***:**

Funkcija **dir()** izlistava sve metode i atribute određenog objekta. Upotrebljena je *for* petlja kojom se prolazi kroz sve metode i atribute biblioteke NumPy.

Funkcija **isinstance()** vraća *True* ako je prosleđeni objekat navedenog tipa, u suprotnom *False*.

Funkcija **getattr()** vraća vrednost prosleđenog atributa određenog objekta.

Na osnovu ovoga kod *isinstance(getattr(np, i), type)* pristupa *i*-tom atributu NumPy objekta i proverava da li je u pitanju tip podataka. Ukoliko jeste, potrebno je da se ispiše.

U nastavku sledi celokupni kod neophodan za izlistavanje svih tipova podataka biblioteke NumPy.

```
import numpy as np
for i in dir(np):
    try:
        if isinstance(getattr(np, i), type):
            print(i)
    except:
        pass
```

Nakon pokretanja programa u primeru 2.3.3. kao rezultat će se izlistati svi tipovi podataka jedan ispod drugog.

2.4. Praktični primeri

U ovom poglavlju su prvo obrađena osnovna tri objekta biblioteke NumPy: ndarray (n-dimenzionalni niz), dtype (tip toga niza) i array scalar (niz skalara). Nakon toga je bilo reči o tome kako na različite načine kreirati n-dimenzionalni niz elemenata. Zatim su demonstrirana tri načina za indeksiranje nizova. Na samom kraju ove glave su detaljno objašnjeni tipovi niza skalara. Sve ove teme su obrađene kroz teorijski deo i primere.

U nastavku se nalaze primeri za samostalno vežbanje, koji obuhvataju teme obrađene u ovoj glavi.

1. Na osnovu date liste [3, 6, 3, 2, 5, 8, 1, 2, 6, 1, 8, 7] kreirati sledeći dvodimenzionalni niz:

$$\begin{bmatrix} 3 & 6 & 3 \\ 2 & 5 & 8 \\ 1 & 2 & 6 \\ 1 & 8 & 7 \end{bmatrix}.$$

   Zatim ispitati:

- kog je tipa kreirani objekat;
- kog su tipa elementi kreiranog niza;
- koje je dimenzije kreirani niz;
- kog je oblika niz;
- koje je veličine niz.

2. Kreirati sledeći dvodimenzionalni niz:

$$\begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 2 \\ 5 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix}.$$

Zatim od njega kreirati sledeći niz:

$$\begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix}$$

3. Kreirati sledeći dvodimenzionalni niz:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 5 \\ 6 & 2 & 2 & 1 & 2 \end{bmatrix}.$$

Zatim:

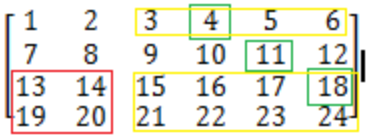
- napraviti od njega jednodimenzionalni niz;
- izdvojiti sve vrednosti niza različite od broja 2.

4. Kreirati dvodimenzionalni niz sa random vrednostima oblika 6x3 i ispisati ga.

5. Kreirati sledeću matricu:

$$\begin{bmatrix} 5 & 5 & 5 & 5 & 5 \\ 5 & 3 & 3 & 3 & 5 \\ 5 & 3 & 1 & 3 & 5 \\ 5 & 3 & 3 & 3 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{bmatrix}.$$

6. Na osnovu date matrice, kreirati tri nove matrice od elemenata označenih različitim bojama na slici:



7. Proveriti da li je broj 25 tipa niza skalara. Ukoliko nije, konvertovati ga u odgovarajući tip skalara.

3. Atributi i metode za rad sa nizovima

U ovom poglavlju će biti obrađeni atributi i metode za rad sa nizovima, koje omogućava biblioteka NumPy. I atributima i metodama sa pristupa pomoću operatora „**.**“ (rečima: „*tačka*“).

3.1. Atributi nizova

Atributi nizova reflektuju informaciju koja je specifična za taj niz. Generalno, pristupanje nizu pomoću njegovih atributa, omogućava da se pristupi, a nekada i promene svojstva niza bez kreiranja novog niza. Vidljivi atributi su glavni deo niza i samo neki od njih mogu da se upotrebe smisleno bez kreiranja novog niza.

U tabeli 3.1.1. su navedeni atributi koji sadrže informacije o memorijskom planu niza i njihova kratka objašnjenja.

Tabela 3.1.1. *Atributi koji sadrže informacije o memorijskom planu niza*:

Atribut	**Napomena**
ndarray.flags	Informacija o memorijskom planu niza.
ndarray.shape	Torka dimenzija niza.
ndarray.strides	Torka bajtova za korak u svakoj dimenziji kada se preklapaju nizovi.
ndarray.ndim	Broj koji označava dimenziju niza.
ndarray.data	Pajtonov bafer objekat koji pokazuje na početak niza podataka.
ndarray.size	Broj elemenata u nizu.
ndarray.itemsize	Dužina elemenata niza u bajtovima.
ndarray.nbytes	Ukupni korišćeni bajtovi od strane elemenata u nizu.
ndarray.base	Osnovni objekat ako je memorija iz nekog drugog objekta.

Primer 3.1.1. *Upotreba informacija o memorijskom planu niza***:**

Data je matrica:

$$f = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{bmatrix}.$$

Da bi se izlistao celokupni memorijski plan niza koristi se atribut **flags**.

Ovaj dvodimenzionalni niz (matrica) je oblika (3, 2) i pristupa mu se pomoću atributa **shape**.

Broj bajtova za korak u svakoj dimenziji se računa na sledeći način: 2^n, pri čemu *n* predstavlja rank određene dimenzije. Broju bajtova za korak u svakoj dimenziji se pristupa pomoću atributa **strides**.

Bafer objektu koji prikazuje na početak niza se pristupa pomoću atributa **data**.

Broj elemenata u datoj matrici je 6 i pristupa mu se pomoću atributa **size**, dok se dužini elemenata niza pristupa pomoću atributa **itemsize** i u navedenom primeru iznosi 4.

Ukupni iskorišćeni bajtovi od strane elemenata u datom nizu iznose $4 \cdot 6 = 24$, a moguće je pristupiti ovom broju, bez računanja, pomoću atributa **nbytes**.

U nastavku se može uočiti demonstracija primera 3.1.1. u programskom jeziku Pajton.

```
import numpy as np
f = np.array([[1,1], [2,2], [3,3]])
print("Memorijski plan niza:\n", f.flags)
print("Dimenzije niza:\n", f.shape)
print("Broj bajtova za korak u svakoj dimenziji:\n", f.strides)
print("Bafer objekat koji pokazuje na pocetak niza:\n", f.data)
print("Broj elemenata u nizu:\n", f.size)
print("Duzina elemenata niza:\n", f.itemsize)
print("Ukupni korisceni bajtovi od strane elemenata u nizu:\n", f.nbytes)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Memorijski plan niza:
        C_CONTIGUOUS : True
        F_CONTIGUOUS : False
        OWNDATA : True
        ALIGNED : True
        WRITEBACKIF COPY : False
        UPDATEIFCOPY : False
Dimenzije niza:
```

```
(3, 2)
Broj bajtova za korak u svakoj dimenziji:
(8, 4)
Bafer objekat koji pokazuje na pocetak niza:
<memory at 0x0000000003CB5828>
Broj elemenata u nizu:
6
Duzina elemenata niza:
4
Ukupni korisceni bajtovi od strane elemenata u nizu:
24
```

Tip podataka objekta povezanog sa nizom može da se nađe u **dtype** atributu.

Primer 3.1.2. *Pristup tipu podataka elemenata***:**

Način pristupanja tipu se izvršava na isti način kao što je izvršeno u okviru primera 2.1.

```
import numpy as np
f = np.array([[1,1], [2,2], [3,3]])
print(f.dtype)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Int32
```

U tabeli 3.1.2. su navedeni drugi atributi biblioteke NumPy i njihova kratka objašnjenja.

Tabela 3.1.2. *Još neki atributi biblioteke NumPy*:

Atributi	**Napomena**
ndarray.T	Isto kao self.transpose(), osim što self se vraća ako je self.ndim manja od 2.

ndarray.real	Izdvaja se realni deo niza.
ndarray.imag	Izdvaja se imaginarni deo niza.
ndarray.flat	1-D iterator preko niza.
ndarray.ctypes	Objekat koji pojednostavljuje interakciju niza sa ctypes modulom.

Primer 3.1.3. *Izdvajanje realnog i imaginarnog dela kompleksne matrice***:**

Neka je data matrica:

$$x = \begin{bmatrix} 1.2 + 5j & 2.1 - 3j & 3 \\ 1.5 & 2.3 + 2j & 4 \end{bmatrix}.$$

Ukoliko je potrebno pristupiti realnom delu ove matrice, koristi se atribut **real**, čija povratna vrednost je matrica koja se sastoji od realnih delova elemenata početne matrice. Na analogan način se pomoću atributa **imag** pristupa imaginarnom delu matrice *x*.

U nastavku se može uočiti demonstracija primera 3.1.3. u programskom jeziku Pajton.

```
import numpy as np
x = np.array([[1.2+5j, 2.1-3j, 3], [1.5,2.3+2j,4]])
print("Originalna matrica:\n", x)
print("Realni deo:\n", x.real)
print("Imaginarni deo:\n", x.imag)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Originalna matrica:
[[1.2+5.j 2.1-3.j 3.+0.j]
[1.5+0.j 2.3+2.j 4.+0.j]]
Realni deo:
[[1.2 2.1 3.]
[1.5 2.3 4.]]
Imaginarni deo:
[[5. -3. 0.]
```

```
[0. 2. 0.]]
```

3.2. Metode za rad sa nizovima

Objekat ndarray ima mnogo metoda[12] koje rade nad nizom ili sa nizom u nekom obliku, tipično vraćajući kao rezultat drugi niz. Za naredne metode tu su takođe i odgovarajuće funkcije u biblioteci NumPy: **all**, **any**, **argmax**, **argmin**, **argpartition**, **argsort**, **choose**, **slip**, **compress**, **copy**, **cumprod**, **cumsum**, **diagonal**, **imag**, **max**, **mean**, **min**, **nonzero**, **partition**, **prod**, **ptp**, **put**, **ravel**, **real**, **repeat**, **reshape**, **round**, **searchsorted**, **sort**, **squeeze**, **std**, **sum**, **swapaxes**, **take**, **trace**, **transpose**, **var**.

Metodama se pristupa takođe pomoću operatora „**.**“ (rečima: „*tačka*“).

U tabeli 3.2.1. u prikazane metode koje imaju za cilj različite konverzije tipova. Pored svake metode dat je njen kratak opis.

Tabela 3.2.1. *Metode koje služe za konverziju tipa podataka*:

Metode	**Napomena**
ndarray.item(*args)	Kopira se element niza u standardni Pajton scalar i vraća ga.
ndarray.tolist()	Vraća niz od *n* elemenata, gde je *n* dimenzija niza, a elementi niza su takođe nizovi čiji su elementi Pajton skalari.
ndarray.itemset(*args)	Unosi podatak skalarnog tipa u niz (skalar je pretvoren u dtype niza, ukoliko je to moguće).
ndarray.tostring([order])	Konstruiše Pajton bajtove koji uključuju čiste bajtove podataka u nizu.
ndarray.tobytes([order])	Konstruiše Pajton bajtove koji uključuju čiste bajtove podataka u nizu.
ndarray.tofile(fid[, sep, format])	Ispisuje niz u fajl kao tekst ili binarno (podrazumevano).
ndarray.dump(file)	Ispisuje serijalizovanu reprezentaciju niza u fajlu.
ndarray.dumps()	Vraća serijalizovanu reprezentaciju niza kao string.

[12] Metoda je isto što i funkcija, osim što se on odnosi na podatak na koji se poziva. Svaka vrsta podataka ima svoj set metoda. Na primer, liste imaju nekoliko korisnih metoda za pronalaženje, dodavanje i uklanjanje vrednosti, kao i manipulisanje njima na drugi način.

ndarray.astype(dtype[, order, casting, …])	Kopija niza, promenjena u specifični tip.
ndarray.byteswap([inplace])	Zameni bajtove elemenata niza.
ndarray.copy([order])	Vraća kopiju niza.
ndarray.view([dtype, type])	Novi izgled niza sa istim podacima.
ndarray.getfield(dtype[, offset])	Vraća polje datog niza kao određeni tip.
ndarray.setflags([write, align, uic])	Postavlja naznake niza WRITEABLE, ALIGNED, (WRITEBACKIFCOPY i UPDATEIFCOPY), redom.
ndarray.fill(value)	Popunjava niz sa skalarnom vrednošću.

Primer 3.2.1. *Upotreba metoda za konverziju tipa podataka***:**

Za kreiranje niza od random celobrojnih brojeva, koristi se metod **randint()** modula random[13]. Prvi argument u okviru datog primera je 9 i predstavlja do koje celobrojne vrednosti mogu da se generišu random brojevi. Drugi argument predstavlja koje veličine će da bude kreirani niz, u ovom primeru matrica dimenzije 3×3.

Kada je potrebno postaviti neki drugi element na određenu poziciju, to se vrši pomoću metode **itemset()**, dok kada se pristupa određenom elementu iz niza, koristi se metoda **item().**

Isto kao što je moguće od liste da se kreira niz, moguće je i od niza da se kreira lista pomoću metode **tolist()**, kome se kao argument prosleđuje željeni niz.

U nastavku se može uočiti demonstracija primera 3.2.1. u programskom jeziku Pajton.

```
import numpy as np
x = np.random.randint(9, size=(3, 3))
print("Niz:\n", x)
print("Postavljamo skalar 0 na poziciju (0, 0):")
x.itemset(0, 0)
print(x)
print("Postavljamo skalar 0 na poziciju (1, 1):")
x.itemset((1, 1), 0)
print(x)
```

[13] Da bi se naznačilo iz kog modula određene biblioteke je metod (ili atribut), pristupa mu se pomoću operatora "." na sledeći način: *ime_biblioteke.ime_modula.ime_metoda(argumenti).* Modul random se koristi radi generisanja random brojeva. Više o ovom modulu se može pronaći na adresi: https://docs.python.org/3/library/random.html

```
print("Postavljamo skalar 0 na poziciju (2, 2):")
x.itemset((2, 2), 0)
print(x)
print("Element na poziciji 1: ", x.item(1))
print("Element na poziciji 5: ", x.item(5))
print("Prebacivanje matrice u listu: ", x.tolist())
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Niz:
[[0 8 1]
[8 8 5]
[6 7 1]]
Postavljamo skalar 0 na poziciju (0,0):
[[0 8 1]
[8 8 5]
[6 7 1]]
Postavljamo skalar 0 na poziciju (1,1):
[[0 8 1]
[8 0 5]
[6 7 1]]
Postavljamo skalar 0 na poziciju (2,2):
[[0 8 1]
[8 0 5]
[6 7 0]]
Element na poziciji 1: 8
Element na poziciji 5: 5
Prebacivanje matrice u listu: [[0, 8, 1], [8, 0, 5], [6, 7, 0]]
```

3.3. Kontrolisanje oblika

Pod kontrolisanjem oblika smatra se menjanje oblika ili veličine nizova. Moguće je da se promeni poredak elemenata niza, kao i da se promeni veličina, da se zamene ose itd.

U tabeli 3.3.1 su prikazane metode koje se koriste za menjanje oblika i veličine niza, kao i njihov opis.

Tabela 3.3.1. *Metode koje se koriste za menjanje oblika i veličine niza*:

Metode	**Napomena**
ndarray.reshape(shape[, order])	Vraća niz koji ima iste podatke ali novi oblik.
ndarray.resize(new_shape[, refcheck])	Menja oblik i veličinu niza.
ndarray.transpose(*axes)	Vraća niz sa zamenjenim osama.
ndarray.swapaxes(axis1, axis2)	Vraća niz gde su ose axis1 i axis2 zamenjene.
ndarray.flatten([order])	Sabija niz u jednu dimenziju.
ndarray.ravel([order])	Vraća sabijen niz u jednu dimenziju.
ndarray.squeeze([axis])	Briše sve jednodimenzionalne elemente za zadatu osu.

Primer 3.3.1. *Upotreba metoda za kontrolisanje oblika***:**

Kreirana je matrica oblika:

$$a = \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}.$$

Potrebno je da se promeni njen oblik, nakon čega će izgledati ovako:

$$a = [0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5].$$

To može da se postigne upotrebom metode **resize()**. Isti efekat je mogao da se postigne pomoću metode **reshape()**. Razlika između ove dve metode je što pomoću metode **resize()** može da se promeni i veličina niza, a pomoću **reshape()** se vraćaju isti podaci, samo se menja oblik.

Još jedna od najčešće korišćenih metoda za manipulaciju oblicima je **transpose()** koja vraća transponovanu matricu, odnosno dolazi do zamene osa u okviru matrice.

U nastavku se može uočiti demonstracija primera 3.3.1. u programskom jeziku Pajton.

```
import numpy as np
# pomocu arange napravimo niz brojeva od 0 do 5,
# zatim pomocu reshape od njega pravimo matricu dimenzije 3x2:
a = np.arange(6).reshape((3, 2))
print("Niz:\n", a)
a.resize((1, 6))
print("Promenjena dimenzija niza u 1x6:\n", a)
a.resize((6, 1))
print("Promenjena dimenzija niza u 6x1:\n", a)
a= a.reshape(3, 2)
print("Vracanje na dimenziju 3x2:\n", a)
print("Transponovana matrica:\n", a.transpose())
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Niz:
[[0 1]
[2 3]
[4 5]]
Promenjena dimenzija niza u 1x6:
[[0 1 2 3 4 5]]
Promenjena dimenzija niza u 6x1:
[[0]
[1]
[2]
[3]
[4]
[5]]
Vracanje na dimenziju 3x2:
[[0 1]
```

```
[2 3]
[4 5]]
Transponovana matrica:
[[0 2 4]
[1 3 5]]
```

3.4. Kontrolisanje elemenata

Za metode nizova koje prihvataju axis ključnu reč, osnovna vrednost je *None*. Ako je osa None, onda se niz tretira kao jednodimenzionalni niz. Svaka druga vrednost za axis predstavlja dimenziju preko koje će se ta operacija izvršavati.

U tabeli 3.4.1. su metode koje služe za kontrolisanje elemenata niza i njihov kratak opis.

Tabela 3.4.1. *Metode za kontrolisanje elemenata niza*:

Metode	**Napomena**
ndarray.take(indices[, axis, out, mode])	Vraća niz formiran od elemenata koji su dati kao ključevi (*indices*), odnosno indeksi u nizu.
ndarray.put(indices, values[, mode])	Postavlja prosleđene vrednosti (*values*) na zadata mesta u nizu, koja su data kao niz ključeva.
ndarray.repeat(repeats[, axis])	Ponavlja elemente u nizu.
ndarray.choose(choices[, out, mode])	Koristi zadate ključeve da kreira novi niz sa elementima koji se nalaze na zadatim pozicijama.
ndarray.sort([axis, kind, order])	Sortira zadati niz.
ndarray.argsort([axis, kind, order])	Vraća niz ključeva na osnovu kojih bi se dobio sortiran niz.
ndarray.partition(kth[, axis, kind, order])	Particioniše, reorganizuje niz tako da elementi čija je vrednost veća od vrednosti na zadatom ključu su pomereni udesno, a preostali su pomereni ulevo.
ndarray.argpartition(kth[, axis, kind, order])	Vraća niz ključeva koji bi reorganizovali zadati niz.
ndarray.searchsorted(v[, side, sorter])	Pronalazi ključeve gde elementi iz *v* treba da budu dodati da bi se zadržao redosled u nizu.

ndarray.nonzero()	Vraća ključeve gde su elementi različiti od nule.
ndarray.compress(condition[, axis, out])	Vraća izabrane delove niza za zadatu osu.
ndarray.diagonal([offset, axis1, axis2])	Vraća dijagonalu za zadati višedimenzionalni niz.

Primer3.4.1. *Upotreba metoda za kontrolisanje elemenata niza***:**

Kontrolisanje elemenata će se vršiti nad sledećom matricom:

$$a = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 14 & 15 & 16 & 17 & 18 & 19 & 20 \end{bmatrix}.$$

Moguće je pristupiti konkretnim elementima pomoću metode **take()**, kojoj se prosleđuje torka indeksa elemenata kojima je potrebno da se pristupi. Ukoliko se prosledi negativan argument, pozicija se posmatra u nizu od kraja ka početku, odnosno sdesna na levo.

Kada je potrebno da se na određene pozicije postave neki drugi elementi, koristi se metoda **put()**.

Sortiranje od najmanjeg ka najvećem elementu u nizu se postiže metodom **sort()**, bez unošenja argumenata, koja se primenjuje nad željenim nizom.

Ukoliko je potrebno da se pristupi dijagonali neke matrice, koristi se metoda **diagonal()**, pri čemu ona može imati kao argument celobrojnu vrednost, koja označava krenuvši od koje pozicije u okviru prvog reda se posmatra dijagonala.

U nastavku se može uočiti demonstracija primera 3.4.1. u programskom jeziku Pajton.

```
import numpy as np
# pravimo niz brojeva od 0 do 20 dimenzije 3x7:
a = np.arange(21).reshape((3, 7))
print("Niz:\n", a)
print("Elementi na pozicijama 1, 5 i 6: ", a.take((1, 5, 6)))
# ako zelimo da pristupimo npr poslednjem elementu niza
# a ne znamo koja je njegova pozicija, trazimo element na poziciji -1
print("Elementi na pozicijama -1, -2 i -3: ", a.take((-1, -2, -3)))
# postavimo na pozicije 1, 5 i 6 elemente 0:
a.put((1, 5, 6),(0, 0, 0))
print("Nakon promene elemenata na pozicijama 1, 5 i 6 u nule:")
```

```
print("Niz:\n", a)
# sortiramo niz:
a.sort()
print("Sortiran niz:\n", a)
# u sortiranom nizu na pozicije 1, 5 i 6 postavljamo 0:
a.put((1, 5, 6), (0, 0, 0))
print("Nakon promene elemenata na pozicijama 1, 5 i 6 u nule:")
print("Niz:\n", a)
print("Dijagonala: \n", a.diagonal())
print("Dijagonala pocevsi od prve pozicije:\n", a.diagonal(1))
print("Dijagonala pocevsi od druge pozicije:\n", a.diagonal(2))
print("Niz:\n", a)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Niz:
[[0 1 2 3 4 5 6]
[7 8 9 10 11 12 13]
[14 15 16 17 18 19 20]]
Elementi na pozicijama 1, 5 i 6: [1 5 6]
Elementi na pozicijama -1, -2 i -3: [20 19 18]
Nakon promene elemenata na pozicijama 1, 5 i 6 u nule:
Niz:
[[0 0 2 3 4 0 0]
[7 8 9 10 11 12 13]
[14 15 16 17 18 19 20]]
Sortiran niz:
[[0 0 0 0 2 3 4]
[7 8 9 10 11 12 13]
[14 15 16 17 18 19 20]]
Nakon promene elemenata na pozicijama 1, 5 i 6 u nule:
```

```
Niz:
[[0 0 0 0 2 0 0]
[7 8 9 10 11 12 13]
[14 15 16 17 18 19 20]]
Dijagonala:
[0 8 16]
Dijagonala pocevsi od prve pozicije:
[ 0 9 17]
Dijagonala pocevsi od druge pozicije:
[0 10 18]
Niz:
[[0 0 0 0 2 0 0]
[7 8 9 10 11 12 13]
[14 15 16 17 18 19 20]]
```

Primer 3.4.2. *Upotreba metoda za sortiranje***:**

Ukoliko se metoda **sort()** primeni nad objektom biblioteke NumPy, kao argument joj se prosleđuje niz koji je potrebno da bude sortiran. Ukoliko je *axis=None*, sortira se ceo niz. Međutim, moguće je da se sortiranje vrši samo u okviru redova ili u okviru kolona, dodeljivanjem argumentu axis vrednost 1 ili 0. Pored toga, moguće je argumentu *kind* dodeliti precizno naglašen način na koji se vrši sortiranje.

U nastavku se može uočiti demonstracija primera 3.4.2. u programskom jeziku Pajton.

```
import numpy as np
a = np.array([[1, 4, 2], [3, 4, 6], [0, -1, 5]])
print ("Sortiran niz:\n", np.sort(a, axis = None))
print ("Sortirane vrednosti u redovima:\n", np.sort(a, axis = 1))
print ("Sortirane vrednosti u kolonama pomocu merge-sort:\n", np.sort(a, axis = 0, kind =
'mergesort'))
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Sortiran niz:
[-1 0 1 2 3 4 4 5 6]
Sortirane vrednosti u redovima:
[[1 2 4]
[3 4 6]
[-1 0 5]]
Sortirane vrednosti u kolonama pomocu merge-sort:
[[0 -1 2]
[1 4 5]
[3 4 6]]
```

Primer 3.4.3. *Sortiranje uz pomoć aliasa***:**

Prvo je potrebno da se definiše promenljiva u kojoj se čuvaju tipovi podataka koji će se pojavljivati u nizu. *S10* predstavlja string od 10 karaktera i njegov alias je *ime*, zatim je godina diplomiranja (*godina*) alias za ceo broj i poslednji, prosek ocena (*prosek*) je alias za realni broj.

U sledećoj promenljivoj (*vrednost*) se definišu vrednosti za svakog učenika (ime, godina diplomiranja, prosek ocena).

Kreiranje niza će biti izvršeno pomoću funkcije **np.array()**, gde se kao prvi argument prosleđuje niz sa studentima i njihovim podacima (*vrednost*), a tipovi i aliasi se prosleđuju pomoću opcionog argumenta dtype.

Sortiranje na osnovu aliasa se vrši tako što unutar funkcije **np.sort()** argumentu **order** dodelimo alias na osnovu koga se sortira. Ukoliko je potrebno da se sortira prvo na osnovu jednog aliasa, zatim na osnovu drugog itd, tada se argumentu order dodeljuje niz koji će sadržati aliase poređane po prioritetu za sortiranje.

U nastavku se može uočiti demonstracija primera 3.4.3. u programskom jeziku Pajton.

```
import numpy as np
# postavljanje aliasa za dtypes
dtypes = [('ime', 'S10'), ('godina', int), ('prosek', float)]
```

```
# vrednosti koje ce biti stavljene u niz:
vrednost = [(Marija, 2018, 8.5), ('Aleksandra', 2014, 8.7), ('Petar', 2017, 6.9), ('Teodora', 2018, 10.0)]
# kreiranje niza:
f = np.array(vrednost, dtype = dtypes)
print ("\nNiz sortiran na osnovu imena:\n", np.sort(f, order = 'ime'))
print ("Niz sortiran na osnovu godine diplomiranja i onda na osnovu proseka:\n", np.sort(f, order = ['godina', 'prosek']))
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Niz sortiran na osnovu imena:
[(b'Aleksandra', 2014, 8.7) (b'Marija', 2018, 8.5) (b'Petar', 2017, 6.9) (b'Teodora', 2018, 10.)]
Niz sortiran na osnovu godine diplomiranja i onda na osnovu proseka:
[(b'Aleksandra', 2014, 8.7) (b'Petar', 2017, 6.9) (b'Marija', 2018, 8.5) (b'Teodora', 2018, 10.)]
```

3.5. Računske operacije primenom biblioteke NumPy

U poglavlju 1.3. je bilo reči o prednostima korišćenja NumPy nizova u odnosu na liste. Jedna od prednosti se upravo može uočiti u računskim operacijama koje se mogu izvršavati nad NumPy nizovima, a ne mogu se izvršavati nad listama. Primera radi, ukoliko je potrebno da se pomnože članovi dve liste *a* i *b*, zapis *a*b* bi prijavljivao grešku, dok to nije slučaj sa NumPy nizom.

Uz pomoć biblioteke NumPy moguće je izvršiti različite računske operacije nad nizovima, kao što su: unarne operacije, operacije poređenja i aritmetičke operacije, o kojima će biti reči u nastavku.

Pored pomenutih operacija, bitno je izdvojiti da biblioteka NumPy omogućava korišćenje operacija koje se koriste u statistici, poput: izračunavanja maksimuma, minimum, srednje vrednosti, medijane, stadardne devijacije i td.

Mnoge od metoda koje se koriste za računske operacije prihvataju axis kao argument. U tim situacijama moguće su naredne dve opcije:

- Ako je axis argument *None* (podrazumevano stanje) onda se niz smatra jednodimenzionalni i operacija se izvršava nad celim nizom. Ovo važi takođe ako je u pitanju običan niz bez više dimenzija ili običan niz skalara gde su skalari instance tipova ili klasa kao što su *float64* ili *int32* itd.
- Ako je argument axis ceo broj onda operacija se izvršava nad tom osom – dimenzijom.

Parametar dtype predstavlja tip vrednosti nad kojom se obavlja neka operacija. Za neke od ovih metoda moguće je postaviti argument *out*, u tom slučaju krajnji rezultat biće smešten u tu vrednost koja mora da bude ndarray i da se poklapa broj elemenata. Ukoliko je tip podatka različit, dolazi do konverzije između tipova podataka.

U tabeli 3.5.1. su navedene metode koje služe za računske operacije koje se izvršavaju nad elementima niza i njihov kratak opis.

Tabela 3.5.1. *Metode koje služe za računske operacije*:

Metode	**Napomena**
ndarray.argmax([axis, out])	Vraća indekse gde se najveće vrednosti nalaze za zadatu osu.
ndarray.min([axis, out, keepdims, initial, …])	Vraća minimum na zadatoj osi.
ndarray.argmin([axis, out])	Vraća indekse gde se najmanje vrednosti nalaze za zadatu osu.
ndarray.ptp([axis, out, keepdims])	Ptp (engl. *Peak to peak*) vraća najveću i najmanju vrednost zajedno za zadatu osu.
ndarray.clip([min, max, out])	Vraća vrednosti između zadatih granica [min, max].
ndarray.conj()	Kompleksni konjugat svih elemenata.
ndarray.round([decimals, out])	Vraća niz gde je svaki element zaokružen na zadati broj decimal.
ndarray.trace([offset, axis1, axis2, dtype, out])	Vraća sumu dijagonala niza.
ndarray.sum([axis, dtype, out, keepdims, …])	Vraća sumu na zadatoj osi.
ndarray.cumsum([axis, dtype, out])	Vraća kumulativnu sumu za zadatu osu, odnosno svakom sledećem element se dodaju svi prethodni elementi niza.
ndarray.mean([axis, dtype, out, keepdims])	Vraća srednju vrednost svih elemenata za zadatu osu.
ndarray.var([axis, dtype, out, ddof, keepdims])	Vraća razliku između elemenata na zadatoj osi.

ndarray.std([axis, dtype, out, ddof, keepdims])	Vraća standardnu devijaciju elemenata niza na zadatoj osi.
ndarray.prod([axis, dtype, out, keepdims, …])	Vraća proizvod elemenata na zadatoj osi.
ndarray.cumprod([axis, dtype, out])	Vraća kumulativni proizvod na zadatoj osi.
ndarray.all([axis, out, keepdims])	Vraća tačno – True ako su svi elementi True.
ndarray.any([axis, out, keepdims])	Vraća tačno – True ako je bilo koji od elemenata tačan – True.

Primer 3.5.1. *Upotreba metoda za računske operacije***:**

Prvo se napravi matrica dimenzije 3 × 7 od random celih brojeva od 0 do 20. To se izvršava na sledeći način: a = np.random.randint(21, size=(3, 7)).

Ukoliko je potrebno da se pristupi najvećim vrednostima na svakoj od kolona, koristi se metoda **argmax()**, kojoj se dodeljuje argument *axis=0*, slično za redove, samo je *axis=1*. Minimalnoj i maksimalnoj vrednosti niza se pristupa pomoću metoda **min()** i **max()**.

Zbir se određuje pomoću metode **sum()**. Takođe je moguće odrediti zbir članova unutar redova ili kolona dodavanjem argumenta axis kome se dodeljuje vrednost 1 ili 0. Slično važi i za proizvod, koji se određuje pomoću metode **prod()**.

Nekada je potrebno da se proveri da li su sve vrednosti različite od nule. To je moguće da se uradi uz pomoć metode **all()**. Ukoliko vrednosti jesu različite od 0 vraća se *True*, u suprotnom *False*. Ova metoda takođe kao argument može da ima axis.

U nastavku se može uočiti demonstracija primera 3.5.1. u programskom jeziku Pajton.

```
import numpy as np
# pravimo niz dimenzije 3x7 od random brojeva od 0 do 20:
a = np.random.randint(21, size=(3, 7))
print("Niz:\n", a)
print("Indeksi gde se najvece vrednosti nalaze na osi 0:\n", a.argmax(axis=0))
print("Indeksi gde se najvece vrednosti nalaze na osi 1:\n", a.argmax(axis=1))
print("Minimalna vrednost: ", a.min())
print("Maksimalna vrednost:", a.max())
print("Zbir: ", a.sum())
print("Zbir na osi 0: ", a.sum(axis=0))
```

```
print("Proizvod na osi 0: ", a.prod(axis=0))
print("Vraca True ako su svi na osi 0 razliciti od 0:\n", a.all(axis=0))
print("Vraca True ako su svi na osi 1 razliciti od 0:\n", a.all(axis=1))
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Niz:
[[2 17 2 10 0 3 4]
[5 0 3 4 18 2 13]
[7 19 20 8 12 3 5]]
Indeksi gde se najvece vrednosti nalaze na osi 0:
[2 2 2 0 1 0 1]
Indeksi gde se najvece vrednosti nalaze na osi 1:
[1 4 2]
Minimalna vrednost: 0
Maksimalna vrednost: 20
Zbir: 157
Zbir na osi 0: [14 36 25 22 30 8 22]
Proizvod na osi 0: [70 0 120 320 0 18 260]
Vraca True ako su svi na osi 0 razliciti od 0:
[True False True True False True True]
Vraca True ako su svi na osi 1 razliciti od 0:
[False False True]
```

NumPy obezbeđuje poznate matematičke funkcije kao što su **sin**, **cos**, **exp**, itd. Ove funkcije se primenjuju nad elementima niza, čime se dobija novi niz kao rešenje.

Primer 3.5.2. *Upotreba matematičkih funkcija i atributa***:**

Kada je u okviru zadatka neophodna upotreba broja π, njemu se pristupa pomoću *np.pi*.

U nastavku se može uočiti demonstracija primera 3.5.2. u programskom jeziku Pajton.

```
import numpy as np
a = np.array([0, np.pi/2, np.pi])
print ("Sinus elemenata niza:", np.sin(a))
a = np.array([0, 1, 2, 3])
print ("Eksponencijalna vrednost elemenata niza:", np.exp(a))
print ("Kvadratni koren elemenata niza:", np.sqrt(a))
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Sinus elemenata niza: [0.0000000e+00 1.0000000e+00 1.2246468e-16]
Eksponencijalna vrednost elemenata niza: [1.          2.71828183 7.3890561
20.08553692]
Kvadratni koren elemenata niza: [0.          1.          1.41421356 1.73205081]
```

3.5.1. Unarne operacije

Mnoge unarne operacije su obezbeđene kao funkcije ndarray klase. Tu su uključene i **sum**, **max**, **min**, itd. Ove funkcije takođe mogu da budu primenjene na red ili kolonu n-dimenzionalnog niza.

Metode koje imaju oblik **ndarray.__ime__(argument)** su specijalne metode koje se pozivaju u specijalnim slučajevima. One se nazivaju *magične metode* i najčešće se koriste za definisanje preopterećenog ponašanja predefinisanih operatora u Pajtonu. One se ne pozivaju kao klasične metode. Na primer, aritmetički operatori podrazumevano rade na numeričkim operandima. To znači da se numerički objekti moraju koristiti zajedno sa operatorima kao što su +, -, *, / itd.

U tabeli 3.5.1.1. su navedene unarne metode koje se pozivaju upotrebom odgovarajućih unarnih operatora.

Tabela 3.5.1.1. *Unarne metode*:

Metode	**Napomena**
ndarray.__neg__($self, /)	Postavlja elemente na negativne vrednosti, odnosno menja im

	znak.
ndarray.__pos__($self, /)	Postavlja elemente na pozitivne vrednosti.
ndarray.__abs__(self)	Vraća apsolutnu vrednost elemenata.
ndarray.__invert__($self, /)	Vraća invertovanu matricu.

Primer 3.5.1.1. *Upotreba unarnih operacija***:**

Za postavljanje elemenata niza na negativne vrednosti koristi se unarni operator „-“ ispred niza nad čijim elementima se vrši promena, za postavljanje na pozitivne „+“, dok se za invertovanje matrice koristi operator „~“ ispred niza.

Ove unarne operacije su prikazane na matrici:

$$a = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}.$$

U nastavku se može uočiti demonstracija primera 3.5.1.1. u programskom jeziku Pajton.

```
import numpy as np
a = np.array([[1,0],[0,1],[1,-1]])
print("Originalna matrica:\n",a)
print("-a:\n",-a)
print("+a:\n",+a)
print("Apsolutna vrednost:\n",abs(a))
print("Invertovana matrica:\n",~a)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Originalna matrica:
[[1 0]
[0 1]
1 -1]]
-a:
```

```
[[-1 0]
[0 -1]
[-1 1]]
+a:
[[1 0]
[0 1]
[1 -1]]
Apsolutna vrednost:
[[1 0]
[0 1]
[1 1]]
Invertovana matrica:
[[-2 -1]
[-1 -2]
[-2 0]]
```

3.5.2. Operacije poređenja

Aritmetičke i operacije upoređivanja na n-dimenzionalnim nizovima definisane su kao operacije nad elementima tih nizova i generalno daju ndarray objekte kao rezultat. Svaka od aritmetičkih operacija (+, -, *, /, //, %, divmod(), ** ili pow(), <<, >>, &, ^, |, ~) i operacija poređenja (==, <, >, <=, >=, !=) ekvivalentna je odgovarajućoj univerzalnoj funkciji (ili *ufunc* skraćeno) u biblioteci NumPy.

U tabeli 3.5.2.1. su navedene magične metode koje se koriste za operacije poređenja i njihov opis.

Tabela 3.5.2.1. *Operacije poređenja*:

Metode	**Napomena**
ndarray.__lt__($self, value, /)	Vraća *self<value*, odnosno poredi svaki element niza da li je manji od zadate vrednosti i vraća *True* ako jeste i *False* ako nije, tako da je povratna vrednost niz koji se sastoji od True/False elemenata.
ndarray.__le__($self, value, /)	Vraća *self<=value*, odnosno poredi svaki element niza da li je manji ili jednak zadatoj vrednosti i vraća *True* ako jeste i *False*

	ako nije, tako da je povratna vrednost niz koji se sastoji od True/False elemenata.
ndarray.__gt__($self, value, /)	Vraća *self>value*, odnosno poredi svaki element niza da li je veći od zadate vrednosti i vraća *True* ako jeste i *False* ako nije, tako da je povratna vrednost niz koji se sastoji od True/False elemenata.
ndarray.__ge__($self, value, /)	Vraća *self>=value*, odnosno poredi svaki element niza da li je veći ili jednak zadatoj vrednosti i vraća *True* ako jeste i *False* ako nije, tako da je povratna vrednost niz koji se sastoji od True/False elemenata.
ndarray.__eq__($self, value, /)	Vraća *self==value*, odnosno poredi svaki element niza da li je jednak zadatoj vrednosti i vraća *True* ako jeste i *False* ako nije, tako da je povratna vrednost niz koji se sastoji od True/False elemenata.
ndarray.__ne__($self, value, /)	Vraća *self!=value*, odnosno poredi svaki element niza da li je različit od zadate vrednosti i vraća *True* ako jeste i *False* ako nije, tako da je povratna vrednost niz koji se sastoji od True/False elemenata.

U tabeli 3.5.2.2. je prikazan metod koji se koristi za proveru istinitosne vrednosti niza elemenata i njegov kratak opis.

Tabela 3.5.2.2. *Metod koji se koristi za provera istinitosne vrednosti*:

Metoda	**Napomena**
ndarray.__nonzero__	Vraća *True* ako je vrednost različita od nule i *False* ako nije.

U slučaju navedenih operacija poređenja, koriste se njima odgovarajući operatori.

Primer 3.5.2.1. *Upotreba operacija poređenja***:**

Neka su date su dve matrice:

$$a = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \text{ i}$$

$$b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

Operatori koji se koriste za poređenje neka dva niza (<, >, <=, >=, ==) kao rezultat vraćaju niz formata istog kao ta dva niza koje u sebi imaju vrednosti *True* ili *False* u zavisnosti da li je

ispunjen kriterijum po kome se porede. Preciznije, kada se porede dva niza, u stvari se poredi svaki element prvog niza sa odgovarajućim elementom drugog niza.

U nastavku se može uočiti demonstracija primera 3.5.2.1. u programskom jeziku Pajton.

```
import numpy as np
a = np.array([[0, 0], [0, 0], [0, 0]])
b = np.array([[1, 0], [0, 1], [1, 1]])
print("Nizovi a i b:")
print(a)
print(b)
print("a<b\n", a<b)
print("a>b\n", a>b)
print("a>=b\n", a>=b)
print("a<=b\n", a<=b)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Nizovi a i b:
[[0 0]
[0 0]
[0 0]]
[[1 0]
[0 1]
[1 1]]
a<b
[[True False]
[False True]
[True True]]
a>b
[[False False]
[False False]
```

```
[False False]]
a>=b
[[False True]
[True False]
[False False]]
a<=b
[[True True]
[True True]
[True True]]
```

3.5.3. Aritmetika

Moguće je koristiti aritmetičke operatore da bi se izvršile operacije nad elementima u nizu i kreirao novi niz. U slučaju operatora +=, -=, *=, modifikuje se postojeći niz.

U tabeli 3.5.3.1. su navedene metode koje se koriste za aritmetičke operacije i njihov kratak opis.

Tabela 3.5.3.1. *Metode koje se koriste za aritmetičke operacije*:

Metode	**Napomena**
ndarray.__add__($self, value, /)	Vraća *self+value*, odnosno kreiraće se novi niz sa svakim elementom uvećanim za vrednost koju smo prosledili kao argument (value).
ndarray.__sub__($self, value, /)	Vraća *self-value*, odnosno kreiraće se novi niz sa svakim elementom umanjenim za vrednost koju smo prosledili kao argument (value).
ndarray.__mul__($self, value, /)	Vraća *self*value*, odnosno kreiraće se novi niz sa svakim elementom pomnoženim sa vrednosti koju smo prosledili kao argument (value).
ndarray.__div__($self, value, /)	Vraća *self/value*, odnosno kreiraće se novi niz sa svakim elementom podeljenim sa vrednosti koju smo prosledili kao argument (value).
ndarray.__truediv__($self, value, /)	Analogno kao prethodno.
ndarray.__floordiv__($self, value, /)	Vraća *self//value*, odnosno kreiraće se novi niz sa svakim elementom podeljenim sa vrednosti koju smo prosledili kao argument (value), pri čemu je u pitanju

	celobrojno deljenje (vrednost je zaokružena na najbljiži ceo broj).
ndarray.__mod__($self, value, /)	Vraća *self%value*, odnosno kreiraće se novi niz sa određenim ostatkom pri deljenju svakog elementa niza sa vrednosti koju smo prosledili kao argument (value).
ndarray.__divmod__($self, value, /)	Vraća *divmod(self, value)*, odnosno kreiraće se novi niz sa dva argumenta, od kojih je prvi niz celih delova pri deljenju self/value, a drugi argument je niz ostataka pri tom deljenju
ndarray.__pow__($self, value[, mod])	Vraća *pow(self, value, mod)*, odnosno kreiraće se novi niz sa svakim elementom podignutim na prosleđeni stepen (value).
ndarray.__lshift__($self, value, /)	Šiftovanje ulevo, vraća *self<<value*, odnosno kreiraće se novi niz sa svakim elementom pomnoženim sa 2 na prosleđeni stepen (value).
ndarray.__rshift__($self, value, /)	Šiftovanje udesno, vraća *self>>value*, odnosno kreiraće se novi niz sa svakim elementom podeljenim sa 2 na prosleđeni stepen (value).
ndarray.__and__($self, value, /)	Logičko I, vraća *self&value*.
ndarray.__or__($self, value, /)	Logičko ILI, vraća *self|value*.
ndarray.__xor__($self, value, /)	Isključivo ILI, vraća *self^value*.
ndarray.__iadd__($self, value, /)	Vraća *self+=value*, odnosno svaki element niza će biti uvećan za prosleđeni argument.
ndarray.__isub__($self, value, /)	Vraća *self-=value*, odnosno svaki element niza će biti umanjen za prosleđeni argument.
ndarray.__imul__($self, value, /)	Vraća *self*=value*, odnosno svaki element niza će biti pomnožen sa prosleđenim argumentom.
ndarray.__idiv__($self, value, /)	Vraća *self/=value*, odnosno svaki element niza će biti podeljen sa prosleđenim argumentom.
ndarray.__itruediv__($self, value, /)	Analogno kao prethodno.
ndarray.__ifloordiv__($self, value, /)	Vraća *self//=value*, odnosno svaki element niza će biti podeljen sa vrednosti koju smo prosledili kao argument (value), pri čemu je u pitanju celobrojno deljenje (vrednost je zaokružena na najbljiži ceo broj).
ndarray.__imod__($self, value, /)	Vraća *self%=value*.
ndarray.__ipow__($self, value, /)	Vraća *self**=value*.
ndarray.__ilshift__($self, value, /)	Vraća *self<<=value*.
ndarray.__irshift__($self, value, /)	Vraća *self>>=value*.

ndarray.__iand__($self, value, /)	Vraća *self&=value*.
ndarray.__ior__($self, value, /)	Vraća *self|=value*.
ndarray.__ixor__($self, value, /)	Vraća *self^=value*.

Primer 3.5.3.1. *Upotreba aritmetičkih operacija***:**

Neka je dat niz:

$$a = [1 \quad 2 \quad 5 \quad 3].$$

Ukoliko je potrebno da se izvrši neka operacija nad svakim elementom niza (na primer da se svaki element uveća za vrednost 1), bez toga da se zauvek promeni početni niz, koriste se operatori za odgovarajuću operaciju. Na taj način ukoliko se zapiše *a+1* kreiraće se novi niz čiji je svaki element biti uvećan za 1.

S druge strane, ukoliko je potrebno da se trajno promeni početni niz, neophodno je da se njemu dodeli novokreirani niz nad kojim je izvršena odgovarajuća operacija. Na primer, ako se zapiše *a=a+1*, odnosno u kraćem zapisu *a+=1*, početni niz *a* će biti trajno izmenjen tako što će mu svaki element biti uvećan za 1.

Transponovanje niza osim sa metodom **transpose()** kao što je urađeno u primeru 8.1, može da se izvrši i primenom metode **T**.

U nastavku se može uočiti demonstracija primera 3.5.3.1. u programskom jeziku Pajton.

```
import numpy as np
a  = np.array([1, 2, 5, 3])
# dodavanje 1 svim elementima niza:
print("Dodavanje 1 svim elementima: ", a+1)
# oduzimanje 3 od svih elemenata niza:
print("Oduzimanje 3 od svih elemenata:", a-3)
# mnozenje svih elemenata sa 10 u nizu:
print("Mnozenje svih elemenata sa 10:", a*10)
# kvadriranje svih elemenata niza:
print("Kvadriranje svakog elementa:", a**2)
# modifikovanje postojeceg niza:
a*=2
print("Udvostrucavanje svakog elementa postojeceg niza:", a)
```

```
# transponovanje niza:
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])
print("\nOriginalni niz:\n", a)
print("Transponovani niz:\n", a.T)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Dodavanje 1 svim elementima: [2 3 6 4]
Oduzimanje 3 od svih elemenata: [-2 -1 2 0]
Mnozenje svih elemenata sa 10: [10 20 50 30]
Kvadriranje svakog elementa: [1 4 25 9]
Udvostrucavanje svakog elementa postojeceg niza: [2 4 10 6]
Originalni niz:
[[1 2 3]
[3 4 5]
[9 6 0]]
Transponovani niz:
[[1 3 9]
[2 4 6]
[3 5 0]]
```

3.5.4. Množenje matrica

Binarni operatori primenjeni na elemente niza kreiraju novi niz. Mogu da se koriste osnovni aritmetički operatori kao +, -, /, *… U slučaju +=, -=, = operatora, postojeći niz se modifikuje.

Kada se koristi operator * pri množenju dve matrice, množi se odgovarajući element prve matrice sa odgovarajućim elementom druge matrice. Međutim kada je potrebno da se pomnože matrice koristeći pravilo za množenje matrica definisano u algebri, koristi se funkcija **dot** ili funkcija **matmul**.

Funkcija **matmul** razlikuje se od funkcije **dot** na dva bitna načina:

- Množenje skalarom nije dozvoljeno;
- Nizovi matrica se emituju zajedno kao da su matrice elementi.

Primer 3.5.4.1. *Operacije nad matricama***:**

Date su tri matrice:

$$a = \begin{bmatrix} 1 & 2 \\ -1 & 1 \\ -2 & 2 \end{bmatrix},$$

$$b = \begin{bmatrix} 1 & -1 \\ 0 & 2 \\ 2 & 4 \end{bmatrix} \text{ i}$$

$$c = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}.$$

Kada se dve matrice sabiraju, oduzimaju, množe ili dele, pomoću odgovarajućih operatora koji služi za datu operaciju, tada se ustvari vrši željena operacija nad odgovarajućim elementima te dve matrice. Iz toga razloga ukoliko se zapiše $a+b$, kreira se nova matrica koja sadrži zbir svaka dva odgovarajuća elementa iz pomenutih matrica.

Kod deljenja matrica ukoliko dolazi do deljenja nekog broja brojem 0, pojavljuje se upozorenje „*RuntimeWarning: divide by zero encountered in true_divide*“ i rezultat toga deljenja je *inf*, što predstavlja oznaku za ∞ (rečima: *beskonačno*).

Kada je potrebno da se pomnože dve matrice na osnovu pravila definisanih u okviru algebre, moguće je koristiti metodu **dot()**. Pri ovakvoj vrsti množenja dve matrice, neophodno je voditi računa o dimenzijama te dve matrice.

U nastavku se može uočiti demonstracija primera 3.5.4.1. u programskom jeziku Pajton.

```
import numpy as np
a = np.array([[1, 2], [-1, 1], [-2, 2]])
b = np.array([[1, -1], [0, 2], [2, 4]])
c = np.array([[1, 2, 3], [4, 5, 6]])
print("a+b:\n", a+b)
print("a-b:\n", a-b)
print("Kada delimo matrice, izlazi nam upozorenje jer se dešava deljenje nulom:\n")
print("a/b:\n", a/b)
```

```
print("a%b:\n", a%b)
print("a*b:\n", a*b)
#pri mnozenju matrica mora da se pazi na dimenziju!
print("Mnozenje matrica a i c:\n", a.dot(c))
```

Nakon poketanja programa, dobija se rezultat prikazan u nastavku.

```
a+b:
[[2 1]
[-1 3]
[0 6]]
a-b:
[[0 3]
[-1 -1]
[-4 -2]]
Kada delimo matrice, izlazi nam upozorenje jer se dešava deljenje nulom:
RuntimeWarning: divide by zero encountered in true_divide
Print("a/b:\n",a/b)
a/b:
[[1. -2.]
[-inf 0.5]
[-1. 0.5]]
RuntimeWarning: divide by zero encountered in remainder
Print("a%b:\n",a%b)
a%b:
[[0 0]
[0 1]
[0 2]]
a*b:
[[1 -2]
[0 2]
```

[-4 8]]

Mnozenje matrica a i c:

[[9 12 15]

[3 3 3]

[6 6 6]]

Primer 3.5.4.2. *Računanje ugla između dva vektora***:**

U matematici skalarni proizvod predstavlja algebarsku operaciju koja uzima koordinate dva vektora iste veličine i vraća jedan broj. Definicija skalarnog proizvoda:

$$\vec{x} \cdot \vec{y} = |\vec{x}||\vec{y}|\cos\measuredangle(\vec{x}, \vec{y}).$$

Sa druge strane, skalarni proizvod može da se izračuna uz pomoć koordinata ta dva vektora. Neka se pretpostavi da su u pitanju vektori dimenzije 3. Skalarni proizvod se računa na sledeći način:

$$\vec{x} \cdot \vec{y} = x_1 y_1 + x_2 y_2 + x_3 y_3.$$

Iz navedenog može da se primeti da se za skalarni proizvod dva vektora koristi funkcija **dot()**.

Neka su dati vektori:

$$x = (5, 2, -3) \text{ i } y = (-1, 0, 9).$$

Njihov skalarni proizvod može lako da se izračuna, kao što je već navedeno.

Da bi se izračunao ugao između navedena dva vektora, prvo je neophodno da se izračunaju moduli ta dva vektora. Moduo vektora $\vec{x} = (x_1, x_2, x_3)$ se računa na sledeći način:

$$|\vec{x}| = \sqrt{x_1^2 + x_2^2 + x_3^2}.$$

Da bi se izračunao skalarni proizvod ovog vektora (niza), potrebno je da se pomnože elementi niza sa samim sobom pomoću odgovarajućeg operatora: *x*x*. Zatim je potrebno da se saberu sve vrednosti nastalog niza pomoću metode **sum()** na sledeći način: *(x*x).sum()*. Kada je to urađeno, izračunava se koren, na osnovu čega se dolazi do koda koji je potreban za računanje modula:

*mod_x = np.sqrt((x*x).sum()).*

Na sličan način se računa i moduo vektora *y*.

Kada su izračunati moduli, ugao se dobija na osnovu sledeće formule:

$$\measuredangle(\vec{x},\vec{y}) = \arccos(\frac{\vec{x}\cdot\vec{y}}{|\vec{x}||\vec{y}|}).$$

Izračunati ugao je dobijen u radijanima. Da bi se prebacio u stepene potrebno je da se dobijena vrednost pomnoži sa $\frac{180^\circ}{\pi}$.

U nastavku se može uočiti demonstracija primera 3.5.4.2. u programskom jeziku Pajton.

```
import numpy as np
x = np.array([5, 2, -3])
y = np.array([-1, 0, 9])
# na osnovu ova dva vektora se racuna njihov skalarni proizvod:
skal = np.dot(x, y)
# računanje modula vektora x i vektora y:
mod_x = np.sqrt((x*x).sum())
mod_y = np.sqrt((y*y).sum())
# racunanje cosinusa ugla između x i y vektora:
cos_xy = skal/mod_x/mod_y
# racunanje ugla između vektora x i y:
ugao = np.arccos(cos_xy)
print("Ugao izmedju vektora x i y u radijanima: ", ugao)
# prebacivanje ugla iz radijana u stepene:
ugao_stepeni = ugao*180/np.pi
print("Ugao izmedju vektora x i y u stepenima: ", ugao_stepeni)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Ugao izmedju vektora x i y u radijanima: 2.1812746986976554
Ugao izmedju vektora x i y u radijanima: 124.97783419404595
```

3.6. Rešavanje sistema jednačina uz pomoć biblioteke NumPy

Jedan od najčešćih problema u linearnoj algebri je rešavanje sistema jednačina. Svaki sistem jednačina može da se zapiše u matričnom obliku i da se rešenje traži u vidu matrice (vektora).

Primer 3.6.1. *Rešavanje sistema jednačina*:

Neka je zadat sistem jednačina:

$$2x_1 + x_2 - x_3 = -3$$

$$3x_1 + x_3 = 5$$

$$x_1 + x_2 - x_3 = -2.$$

Prvo je potrebno zapisati dati sistem u matričnom obliku:

$$\begin{bmatrix} \mathbf{2} & \mathbf{1} & \mathbf{-1} \\ \mathbf{3} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{-1} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \mathbf{-3} \\ \mathbf{5} \\ \mathbf{-2} \end{bmatrix}.$$

Nakon čega je potrebno da se odredi vektor x, koji zadovoljava sledeću jednačinu:

$$\boldsymbol{Ax = b},$$

pri čemu je:

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{2} & \mathbf{1} & \mathbf{-1} \\ \mathbf{3} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{-1} \end{bmatrix} \text{ i}$$

$$\boldsymbol{b} = \begin{bmatrix} \mathbf{-3} \\ \mathbf{5} \\ \mathbf{-2} \end{bmatrix}.$$

Za rešavanje ovog zadatka, neophodno je prvo objasniti metodu **numpy.linalg.solve(a, b)**[14] koja služi za rešavanje linearne jednačine zadate u matričnom obliku, ili sistema linearnih skalarnih jednačina. Povratna vrednost je rešenje toga sistema u obliku identičnom kao b.

numpy.linalg.solve(a, b)

- ***a*** – matrica koeficijenata
- ***b*** – ordinata ili vrednosti zavisnih promenljivih

[14] Ovde može da se uoči još jedan modul biblioteke NumPy, koji sadrži standardne algoritme koji se koriste u linearnoj algebri. Više o ovome modulu se može pronaći na adresi: https://numpy.org/doc/stable/reference/routines.linalg.html

Neophodno je voditi računa o dimenziji matrica, odnosno **a** je oblika $n \cdot n$, dok je **b** oblika $n \cdot 1$, zbog čega ako za b se kreira jednodimenzioni niz, neophodno je da se transponuje.

Ukoliko je potrebno da se izvrši provera da li je dobro rešen sisem, koristi se metody **allclose(a, b)**, koja poredi da li su nizovi a i b ekvivalentni.

U nastavku se može uočiti demonstracija primera 3.6.1. u programskom jeziku Pajton.

```
import numpy as np
# prvo formiramo matrice (nizove) A i b:
A = np.array([[2, 1, -1], [3, 0, 1], [1, 1, -1]])
b = np.array([-3, 5, -2])
# sistem resavamo na sledeci nacin:
x = np.linalg.solve(A, b.T)
print(x)
# provera da li je resenje ispravno:
np.allclose(np.dot(A, x), b)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
[1. -1. -2.]
True
```

3.7. Specijalne metode

U ovom poglavlju obrađene su neke od metoda za kopiranje, serijalizaciju i deserijalizaciju, kreiranje novog niza i podešavanje kontejnera.

U tabeli 3.7.1. su navedene funkcije koje se nasleđuju iz standardne biblioteke sa kratkim opisom.

Tabela 3.7.1. *Metode koje se nasleđuju iz standardne biblioteke*:

Metode	Napomena
ndarray.__copy__()	Koristi se ako je **copy.copy** pozvan na niz. Vraća se kopija niza.
ndarray.__deepcopy__(memo, /)	Koristi se ako je **copy.deepcopy** pozvan za niz. Dubinska kopija čini proces kopiranja rekurzivnim. To znači da se prvo konstruiše novi objekat za prikupljanje, a zatim se rekurzivno popunjava kopijama podređenih objekata koji se nalaze u original. Kopiranjem objekta na ovaj način obilazi se celo drvo objekta da bi se stvorio potpuno nezavisan klon originalnog objekta i njegove dece.
ndarray.__shallowcopy__(memo, /)	Koristi se ako je **copy.shallowcopy** pozvan za niz. Plitka kopija znači konstruisanje novog kolekcijskog objekta i zatim njegovo punjenje referencama na podređene objekte koji se nalaze u original. U suštini, plitka kopija je duboka samo na jednom nivou. Process kopiranja se ne ponavlja i zato neće sami stvoriti kopije podređenih objekata.
ndarray.__reduce__()	Za serijalizovanje. Serijalizacija je proces predstavljanja objekta nizom bajtova.
ndarray.__setstate__(state, /)	Za deserijalizovanje. Deserijalizacija je process kada se od niza bajtova dobijenih serijalizacijom ponovo dobija objekat.

U tabeli 3.7.2. su navedene metode koje služe za jednostavno podešavanje i njihov kratak opis.

Tabela 3.7.2. *Metode koje služe za jednostavno podešavanje*:

Metode	Napomena
ndarray.__new__($type, *args, **kwargs)	Kreira i vraća novi objekat.
ndarray.__array__(|dtype)	Vraća ili novi pokazivač na sebe ako dtype nije dat ili novi niz sa datim tipom podataka ako je dtype drugačiji od trenutnog dtype od niza.
ndarray.__array_wrap__(obj)	Vraća objekat istog tipa kao n-dimenzionalni objekat koji je prosleđen.

Kontejneri se već duže vreme upotrebljavaju u računarskim naukama. Pajton uključuje nekoliko ugrađenih tipova kontejnera: liste, rečnici, setovi i torke. Ndarray predstavlja višedimenzionalni kontejner elemenata istog tipa i veličine.

U tabeli 3.7.3. su navedene metode koje služe za podešavanje kontejnera, sa kratkim opisom.

Tabela 3.7.3. *Metode koje služe za podešavanje kontejnera*:

Metode	**Napomena**
ndarray.__len__($self, /)	Vraća *len(self),* odnosno dužinu niza.
ndarray.__getitem__($self, key, /)	Vraća *self[key]*, odnosno vraća element koji se nalazi u nizu na key poziciji.
ndarray.__setitem__($self, key, value, /)	Postavlja *self[key]* to value, odnosno postavlja element na key poziciji u nizu na value vrednost.
ndarray.__contains__($self, key, /)	Vraća *key in self*, odnosno proverava da li se key nalazi unutar našeg niza.

Primer 3.7.1. *Upotreba kontejnera***:**

Posmatra se sledeća matrica:

$$a = \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ -2 & 2 \end{bmatrix}.$$

Data matrica se tretira kao niz od tri elementa, čiji svaki element predstavlja niz od dva elementa i svaki od njih predstavlja jedan red unutar matrice *a*. Ukoliko je potrebno da se promeni neki od redova, njemu se pristupa pomoću indeksa koji označava njegovu poziciju unutar niza. Na primer, ukoliko se zapiše *a[1]=(1, 5),* vrednosti drugog reda su promenjene iz (0, 1) u (1, 5).

Nekada je potrebno da se proveri da li neka matrica sadrži određeni element, a da se ne prolazi kroz sve elemente matrice. To se postiže pomoću ključne reči **in**.

U nastavku se može uočiti demonstracija primera 3.7.1. u programskom jeziku Pajton.

```
import numpy as np
a = np.array([[1, 2], [0, 1], [-2, 2]])
print("Niz:\n", a)
print("a[1]=", a[1])
# mozemo da promenimo a[1] na sledeci nacin:
a[1]=(1, 5)
print("Niz:\n", a)
```

```
print("Da li je -21 u a[1]: ", -21 in a[1])
print("Da li je 5 u a[1]: ", 5 in a[1])
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Niz:
[[1 2]
[0 1]
[-2 2]]
a[1]=[0 1]
Niz:
[[1 2]
[1 5]
[-2 2]]
Da li je -21 u a[1]: False
Da li je 5 u a[1]: True
```

Konverzija se vrši samo na nizovima koji imaju jedan element u sebi i pri konverziji se vraća odgovarajući skalar.

U tabeli 3.7.4. su navedene metode koje se koriste za konverziju i njihov kratak opis.

Tabela 3.7.4. *Metode za konverziju tipova podataka*:

Metode	**Napomena**
ndarray.__int__(self)	Vraća se konvertovan element u tip int.
ndarray.__long__(self)	Vraća se konvertovan element u tip long.
ndarray.__float__(self)	Vraća se konvertovan element u tip float.
ndarray.__oct__(self)	Vraća se konvertovan element u oktalni tip.
ndarray.__hex__(self)	Vraća se konvertovan element u heksadekadni tip.

U tabeli 3.7.5. su navedene metode koje služe za konvertovanje niza u nisku karaktera (engl. *string*) i njihov kratak opis

Tabela 3.7.5. *Metode za konvertovanje niza u nisku karaktera*:

Metode	**Napomena**
ndarray.__str__($self, /)	Vraća str(self), odnosno konvertuje element u nisku karaktera.
ndarray.__repr__($self, /)	Vraća repr(self), odnosno konvertuje element u nisku karaktera, pri čemu pri reprezentaciji uključuje i tip objekta.

Primer 3.7.2. *Konvertovanje niza u nisku karaktera***:**

Konvertovanje može da se vrši pomoću funkcija **str()** i **repr()**, koje se razlikuju u reprezentaciji.

U nastavku se može uočiti demonstracija primera 3.7.2. u programskom jeziku Pajton.

```
import numpy as np
a = np.array([[1, 2], [1, 5], [-2, 2]])
print("Niz:\n", a)
print(str(a))
print(repr(a))
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Niz:
[[1 2]
[1 5]
[-2 2]]
[[ 1 2]
  [ 1 5]
  [-2 2]]
array([[ 1, 2],
          [ 1, 5],
          [-2, 2]])
```

Primer 3.7.3. *Upotreba atributa i funkcija objekta ndarray nad datotekama***:**

Primer prikazuje kreiranje matrice, čuvanje matrice sa N dimenzija na disku u datoteci i čitanje iste sa diska, korišćenje atributa matrice da bi se izmenio element, „zaključavanje“ matrice da bi se onemogućila izmena, čitanje elemenata u nizu.

Primer ima tri funkcije:

- Funkcija **ucitaj_niz** uzima za prvi argument ime datoteke gde je niz sačuvan. Prvo se proverava da li datoteka postoji. Ukoliko datoteka postoji iščitava se niz iz iste, inače se kreira nasumičan niz sa N dimenzija. N predstavlja nasumičan broj između 2 i 5 i svaka dimenzija ima između 5 i 15 elemenata da bi simulirali neke podatke.

- Funkcija **ispisi_atribute** ispisuje neke od atributa, veličinu, dimenzije, oblik niza, da li je niz moguće izmeniti i prvi element u nizu. Postoje dva načina za čitanje elemenata iz niza. Prvi je direktan pristup elementu, dok je drugi preko funkcije **item()**. Razlika između ova dva načina je što **item** vraća „zaokružen“ broj, dok direktan pristup vraća ceo broj bez zaokruživanja, što se može videti u ovom primeru.

- Funkcija **izbrisi_fajl** briše datoteku gde je niz sačuvan da bi se svaki sledeći put kad se pokrene program dobila novu matricu i da bi se radilo sa novim podacima.

Kada je potrebno da se definiše nova funkcija, to se vrši na sledeći način:

def ime_funkcije(argumenti):

telo funkcije

Path predstavlja modul u biblioteci os, koja sadrži metode za rad sa putanjama. Jedna od tih metoda je **isfile()**, koja proverava da li putanja postoji i da li je u pitanju datoteka.

Metoda **unlink()** služi za brisanje datoteka.

Kod *fajl_sa_matricom = '%s.npy' % fajl_sa_matricom* označava da se imenu datoteke dodaje ekstenzija „*.npy*“.

Funkcija **load()** je funkcija iz biblioteke NumPy i ona učitava sačuvan niz iz neke datoteke.

Funkcija **save()** je funkcija iz biblioteke NumPy, koja prima dva argumenta:

- Ime datoteke, gde se niz čuva;
- Niz koji čuvamo.

Kod *random_dimensions = [randint(5, 15) for i in range(randint(2, 5))]* će kreirati niz sa nasumičnim brojem elemenata između 2 i 5, tako što će taj niz da počinje od 0 do izabrani broj - 1, što se postiže pomoću *range(randint(2, 5))*, nakon čega će se za svaki element u nizu izabrati nasumičan broj između 5 i 15, što se postiže pomoću *randint(5, 15)* i promenljivoj *random_dimensions* se dodeljuje dobijeni niz nasumičnih brojeva između 5 i 15. Ti brojevi predstavlju broj elemenata u dimenziji.

Funkcija **random.rand()** je iz biblioteke NumPy i kao argumente prima dimenzije sa brojem elemenata, od kojih kreira nasumičan niz.

Kod *random_ndarray = np.random.rand(*random_dimensions)* će u promenljivoj *random_ndarray* sačuvati nasumičan niz. **random_dimensions* prosleđuje neki niz kao listu argmenata funkciji **random.rand().**

Primera radi, ukoliko postoji niz $niz = [1, 2, 3]$ i potrebno je da se pozove neka funkcija f kojoj će elementi toga niza biti prosleđeni kao argumenti, to može da se izvrši na dva načina: $f(1, 2, 3)$ ili mnogo efikasniji način kod dinamičkih nizova $f(*niz)$.

Na kraju promenljiva *random_ndarray* će biti sačuvana u datoteci *fajl_sa_matricom*, pomoću koda *np.save(fajl_sa_matricom[:-4], random_ndarray)*, bez ekstenzije („.npy“), jer je metoda **save()** automatski dodaje. Iz tog razloga se brišu poslednja četiri karaktera iz imena datoteke gde se niz čuva (*fajl_sa_matricom[:-4])*.

Nakon definisanja ovih funkcija program će prvo pokušati da učita datoteku pod imenom „*niz.npy*“. Iz razloga što datoteka ne postoji, biće automatski kreirana sa nasumičnim podacima i program će zatim ispisati atribute. Ovaj niz je sačuvan u promenljivoj pod imenom *random_ndarray*.

U sledećem koraku program će pokušati da učita iz iste datoteke niz i iz razloga što je već kreirana datoteka, dobiće se isti podaci kao i u prvom koraku i ovaj niz će biti učitan u promenljivu pod imenom *random_ndarray1*. Promenljive *random_ndarray* i *random_ndarray1* imaju isti sadržaj, ali se čuvaju na različitim mestima u memoriji tako da ukoliko se promeni jedna, druga se neće automatski promeniti.

Nakon ovoga je potrebno da se promeni prvi element u matrici u 10. Koristeći **itemset()** funkciju ovaj poziv će biti uspešan, zato što je moguće menjati matricu i *ndarray.flags['WRITEABLE']* je *True*.

Sledeći **try/except** blok[15] prikazuje „zaključavanje“ matrice, pri čemu korišćenje iste funkcije **itemset()** da bi se promenila vrednost prvog element u 11 nije moguća, jer je niz zaključan i *ndarray.flags['WRITEABLE']* je *False*. U *catch* bloku se nudi opcija da se niz „otključa“ i postavi na 11. Ukoliko se odgovori sa „*da*“, niz se otključava i prvi element se postavlja na 11. Svaki drugi odgovor se ignoriše.

„Otkuljučavanje“ i „zaključavanje“ niza se vrši metodom **setflags()**, pri čemu se prosleđuje kao argument *writable=1* kada je potrebno da se piše po nizu (odnosno da se „otključa“), u suprotnom *writable=0*.

Na samom kraju se ispisuju informacije o nizu i briše se datoteka, da bi se svaki sledeći put kada se pokrene ovaj program, imali različiti podaci.

U nastavku se može uočiti demonstracija primera 3.7.3. u programskom jeziku Pajton.

[15] U Pajtonu greška može da bude sintaksna ili izuzetak. Izuzetak predstavlja grešku koja nastaje bez obzira što je sintaksa koda tačna. Poslednja linija poruke ukazuje na koji tip izuzetka se misli. Radi hvatanja i rešavanja grešaka, u Pajtonu se koristi try/except blok. Try blok služi za testiranje koda da li ima greške. Except blok služi za rešavanje greške. Finally blok dopušta da se izvrši kod, bez obzira na rezultat try i except blokova. Više o greškama i njihovom rešavanju se može pronaći na sledećoj adresi: https://realpython.com/python-exceptions/.

```
import numpy as np
from random import randint
from os import path, unlink
# definicija funkcije izbrisi_fajl:
def izbrisi_fajl(fajl):
  if path.isfile(fajl):
    print("Brisanje fajla %s" % fajl)
    unlink(fajl)
# definicija funkcije ucitaj_niz:
def ucitaj_niz(fajl_sa_matricom):
  fajl_sa_matricom = '%s.npy' % fajl_sa_matricom
  if path.isfile(fajl_sa_matricom):
    print("Ucitaj podatke iz fajla '%s'" % fajl_sa_matricom)
    return np.load(fajl_sa_matricom)
  print("Generisi nasumicnu matricu i sacuvaj je u '%s'" % fajl_sa_matricom)
  # naredni kod kreira izmedju dve i pet dimenzija
  # gde svaka dimenzija ima izmedju 5 i 15 elemenata
  random_dimensions = [randint(5, 15) for i in range(randint(2, 5))]
  # naredni kod kreira random niz sa random_dimensions dimenzija
  random_ndarray = np.random.rand(*random_dimensions)
  np.save(fajl_sa_matricom[:-4], random_ndarray)
  return random_ndarray
# definicija funkcije ispisi_atribute:
def ispisi_atribute(ndarray):
  print("\n-----------------------")
  print("Oblik (ndarray.shape) = %s" % str(ndarray.shape))
  print("Velicina (ndarray.size) = %s" % str(ndarray.size))
  print("Dimenzije (ndarray.ndim) = %s" % str(ndarray.ndim))
 # provera da li je moguće izmeniti podatke u nizu:
 if ndarray.flags['WRITEABLE']:
    print("Moguce je izmeniti podatke u ovom nizu")
```

```
  else:
    print("Nije moguce izmeniti podatke u ovom nizu")
  first_element = ndarray[0]
  element = "ndarray[0]"
  for i in range(ndarray[0].ndim):
    first_element = first_element[0]
    element += "[0]"
  print("Prvi element (%s) = %s" % (element, first_element))
  print("Prvi element (ndarray.item(0)) = %s" % ndarray.item(0))
  print("------------------------\n")
# kod koji će se izvršiti:
random_ndarray = ucitaj_niz("niz")
ispisi_atribute(random_ndarray)
random_ndarray1 = ucitaj_niz("niz")
print("Izmeni prvi element u 10")
random_ndarray1.itemset(0, 10)
ispisi_atribute(random_ndarray1)
try:
  # moguce je i sa
  # random_ndarray1.flags['WRITEABLE'] = False
  random_ndarray1.setflags(write=0)
  print("WRITEABLE = False")
  print("Izmeni prvi element u 11")
  random_ndarray1.itemset(0, 11)
except Exception as e:
  print ("Izmena nije moguca jer je niz 'read-only' i dobija se greska:")
  print (str(e))
  if raw_input("'Otkljucaj' niz i postavi prvi element na 11? (da/ne) ").lower() == 'da':
    random_ndarray1.setflags(write=1)
    random_ndarray1.itemset(0, 11)
ispisi_atribute(random_ndarray1)
izbrisi_fajl('niz.npy')
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
Generisi nasumicnu matricu i sacuvaj je u 'niz.npy'

-----------------------
Oblik (ndarray.shape) = (14, 15, 5, 7)
Velicina (ndarray.size) = 7350
Dimenzije (ndarray.ndim) = 4
Moguce je izmeniti podatke u ovom nizu
Prvi element (ndarray[0][0][0][0]) = 0.01950631902562794
Prvi element (ndarray.item(0)) = 0.0195063190256
-----------------------

Ucitaj podatke iz fajla 'niz.npy'
Izmeni prvi element u 10

-----------------------
Oblik (ndarray.shape) = (14, 15, 5, 7)
Velicina (ndarray.size) = 7350
Dimenzije (ndarray.ndim) = 4
Moguce je izmeniti podatke u ovom nizu
Prvi element (ndarray[0][0][0][0]) = 10.0
Prvi element (ndarray.item(0)) = 10.0
-----------------------

WRITEABLE = False
Izmeni prvi element u 11
Izmena nije moguca jer je niz 'read-only' i dobija se greska:
assignment destination is read-only
'Otkljucaj' niz i postavi prvi element na 11? (da/ne) da

-----------------------
```

```
Oblik (ndarray.shape) = (14, 15, 5, 7)
Velicina (ndarray.size) = 7350
Dimenzije (ndarray.ndim) = 4
Moguce je izmeniti podatke u ovom nizu
Prvi element (ndarray[0][0][0][0]) = 11.0
Prvi element (ndarray.item(0)) = 11.0
-----------------------

Brisanje fajla niz.npy
```

3.8. Praktični primeri

U ovom poglavlju su obrađeni atributi koji sadrže informacije o memorijskom planu niza, kao i mnogi drugi atributi. Takođe su obrađene metode za konverziju tipova, metode za menjanje oblika ili veličine niza, metode za kontrolisanje elemenata niza, metode za sortiranje i td. Objašnjene su operacije (unarne operacije, operacije poređenja i aritmetičke operacije), aritmetički operatori, statistike, pristup matematičkim funkcijama i matematičkim konstantama. Objašnjeni su i neki novi pojmovi, poput magičnih metoda. Prikazana je razlika između množenja matrica tako što se množi svaki element sa njemu odgovarajućim elementom i množenje matrica prema pravilu iz algebre. Prikazane su i neke konkretne primene, kako u analitičkoj geometriji (računanje ugla između dva vektora), tako i u algebri (rešavanje sistema jednačina). Takođe su obrađene specijalne metode. Na samom kraju je prikazana upotreba kontejnera i kako da se konvertuju elementi u drugi tip. Sve ove teme su obrađene kroz teorijski deo i primere.

U nastavku se nalaze primeri za samostalno vežbanje, koji obuhvataju teme obrađene u ovoj glavi.

1. Neka je data matrica:

$$\begin{bmatrix} 3 & 4 & 6 & 3 & 0 & 1 \\ 2 & 15 & 0 & 4 & 6 & 12 \\ 7 & 11 & 9 & 2 & 8 & 5 \\ 1 & 9 & 7 & 1 & 14 & 9 \\ 6 & 3 & 2 & 0 & 6 & 3 \\ 0 & 19 & 8 & 2 & 8 & 4 \end{bmatrix}.$$

Odrediti:

- minimum i maksimum matrice;
- minimum po svim redovima;
- maksimum po svim kolonama;
- zbir svih elemenata;
- srednju vrednost;
- medijanu;
- standardnu devijaciju.

2. Izračunati:

$$\begin{bmatrix} 0 & i \\ i & o \end{bmatrix}^n,$$

gde je $i^2 = -1$, nϵN.

3. Neka su date sledeće matrice:
$A = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 4 \\ 0 & 9 \end{bmatrix}$, $C = \begin{bmatrix} 7 & 9 \\ 4 & 5 \end{bmatrix}$ i $D = \begin{bmatrix} 1 & 3 & 7 \\ 4 & 9 & 0 \end{bmatrix}$
Izračunati sledeće izraze:
   - $A \times B + C$,
   - $(A - C + B) \times D$,
   - $\frac{A+C*B}{B-10}$,

   pri čemu × označava množenje matrica po pravilima u algebri, a * označava množenje elemenata datih matrica koji se nalaze na istim pozicijama.

4. Neka je data sledeća matrica:

$$\begin{bmatrix} 5 & 7 & 1 & 4 & 7 \\ 0 & 5 & 12 & 2 & 9 \\ 9 & 3 & 7 & 1 & 0 \\ 0 & 8 & 21 & 0 & 3 \\ 11 & 0 & 0 & 1 & 6 \end{bmatrix}.$$

Izračunati:

- Sumu svih parnih elemenata u matrici;
- Proizvod svih nenula elemenata van dijagonale.

Zatim:

- sortirati početnu matricu;
- kreirati matricu kojoj se svim parnim elementima promeni znak, svi neparni elementi se uvećaju dva puta i sve nule se zamene najvećim brojem iz kolone u kojoj se nalazi ta nula.

5. Kreirati simulaciju koja ispisuje koje se vrednosti dobiju ukoliko se novčić baca 10 puta.

6. Proveriti da li je tačka (0, -1, 5) rešenje sistema jednačina:

$$x - y + z = 13$$
$$2x + 5y - 3z = -13$$
$$4x - y - 6z = -4.$$

4. Linearna regresija

Biblioteka NumPy se neretko koristi prilikom statističkih izračunavanja i pri mašinskom učenju, iz razloga što je potrebno raditi sa velikom količinom podataka i zbog toga što biblioteka NumPy omogućava bržu obradu podataka od ugrađenih nizova u okviru programskog jezika Pajton.

Za analizu podataka je neophodna i biblioteka Matplotlib[16], koja se unutar ovoga rada neće detaljnije obrađivati, već će biti objašnjeni samo oni delovi koji se koriste.

Jedan od najjednostavnijih za razumevanje i najčešćih modela u statistici je upravo linearna regresija. Kada postoji linearna veza između karakteristika i ciljane promenljive, sve što je potrebno je da se nađe jednačina prave linije u višedimenzionalnom prostoru. Bazira se na predviđenju na osnovu preostalih vrednosti podataka.

Tipovi linearne regresije:

1. ***Prosta linearna regresija*:** karakteriše je jedna nezavisna promenljiva.
2. ***Višestruka linearna regresija*:** karakterišu je više nezavisnih promenljivih.

Primera radi, ukoliko su dati podaci o prodaji nekternina: veličina, sprat, udaljenost od škole i cilj je da se proceni cena imovine. Ako je otkriveno da je ovo linearni problem, onda je samo potrebno da se pronađu koeficijenti i presretanje, odnosno:

cena = [veličina stana] * a1 + [sprat] * a2 + [udaljenost od škole] * a3 + presretanje

U tabeli 4.1. su prikazani stvarni podaci za izradu ovoga modela.

Tabela 4.1. *Podaci sa informacijama o stanovima*:

Nekretnina	Veličina (m^2)	Sprat	Udaljenost od škole (km^2)	Cena
Stan	60	7	2	85000€
Kuća	100		4	170000€
Stan	94	3	1.5	120000€
	….	…	…	…

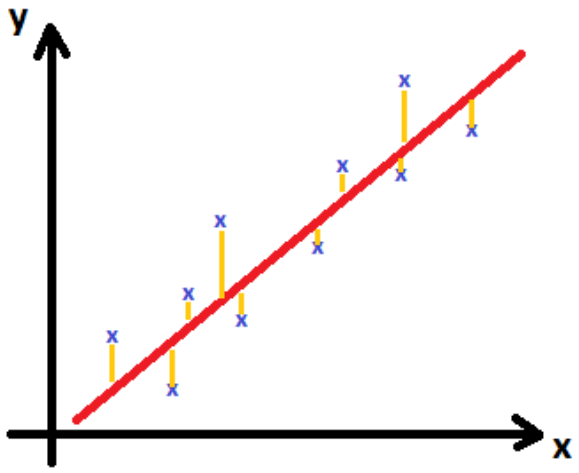
[16] Matplotlib je biblioteka za crtanje grafika, odnosno vizualizaciju podataka, u programskom jeziku Pajton. Više o ovoj biblioteci se može pronaći na adresi: https://matplotlib.org/.

Postoji mnogo načina da se pronađu koeficijenti i presretanje. Mogu da se koriste najmanji kvadrati ili jedna od metoda optimizacije kao što je *gradijent*.

U ovom zadatku korišćena je metoda najmanjih kvadrata.

4.1. Metoda najmanjih kvadrata

Najmanji kvadrat je metoda za pronalaženje linije koja najviše odgovara podacima, kao što je prikazano na slici 4.1.1. Koristi jednostavnu računicu i linearnu algebru da bi se pronašla najpogodnija linija i smanjila greška. Razmak između tačke i linije je uzet i svaki od njih je kvadriran, da bi nestale negativne vrednosti, nakon čega su vrednosti sumirane, što daje grešku, koja mora da se smanji. Pošto su i vrednosti grešaka kvadrirane, to rezultira samo pozitivnim vrednostima grešaka, koje mogu da se mere pravilno (iz razloga što se ne može vršiti merenje negativnih vrednosti). Metoda najmanjih kvadrata je najčešće korišćena jer pronalazi najmanju grešku u poređenju sa ostalim metodama.



Slika 4.1.1: metoda najmanjih kvadrata

Prvo će biti obrađen jednostavniji primer sa samo dve dimenzije. Potrebno je da se odrede m i b, da bi se pronašla jednačina:

$$\boldsymbol{Y = mX + b}\ ,$$

pri čemu su:

m – nagib linije;

b – konstanta koja označava pristrasnost (presretanje), odnosno razliku između očekivane vrednosti i stvarne vrednosti;

X – nezavisna promenljiva;

Y – zavisna promenljiva.

Dat je skup od (x, y) parova. U cilju pronalaženja m i b, potrebno je izračunati:

$$m = \frac{N \cdot \sum_{i=0}^{N-1}(x_i \cdot y_i) - \sum_{i=0}^{N-1} x_i \cdot \sum_{i=0}^{N-1} y_i}{N \cdot \sum_{i=0}^{N-1} x_i^2 - (\sum_{i=0}^{N-1} x_i)^2} \text{ i}$$

$$b = \frac{\sum_{i=0}^{N-1} x_i^2 - m \cdot \sum_{i=0}^{N-1} x_i}{N},$$

pri čemu N označava dužinu nizova $x = (x_1, x_2, \dots, x_N)$ i $y = (y_1, y_2, \dots, y_N)$.

Primer 4.1.1. *Primena linearne regresije pomoću biblioteke NumPy***:**

Radi lakšeg zapisa umesto punih imena paketa, koriste se skraćenice. Primera radi, umesto numpy.array(), pisaće **np.array()** ili umesto matplotlib.pylab.plot(x,y), pisaće **plt.plot(x,y)**, radi lakšeg kodiranja.

Na početku su kreirani jednostavni podaci za testiranje. Neka *x* predstavlja veličinu stana u kvadratnim metrima od 1 (10 m^2) do 8 (80 m^2), a *y* nam predstavlja cenu od 15 (15000 €) do 200 (200000 €):

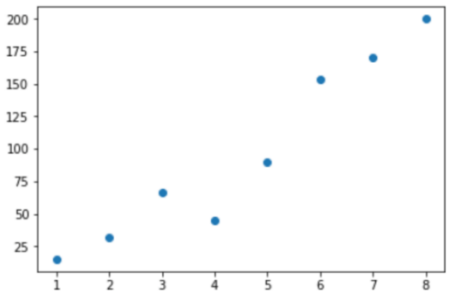
$$x = [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8] \text{ i}$$

$$y = [15 \quad 32 \quad 66 \quad 45 \quad 90 \quad 153 \quad 170 \quad 200].$$

Iz razloga što je u pitanju jednostavna linearna regresija, postoji samo jedan faktor (veličina stana) koji utiče na cenu stana. Potrebno je da se odredi linija koja najbolje odgovara zadatim podacima, kako bi moglo da se predvidi *y* (cena stana) za bilo koju vrednost *x* (veličinu stana).

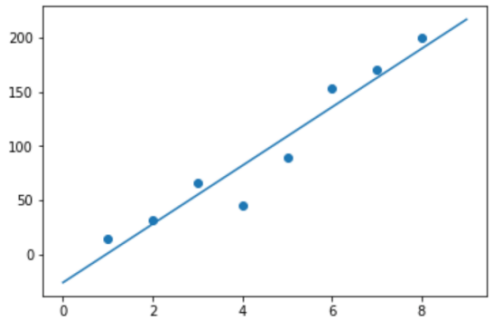
Da bi se odredila tražena linija, prvo je neophodno da se izračunaju nagib i pristrasnost na osnovu formula koje su definisane u okviru ove lekcije. Na osnovu izračunatih vrednosti je moguće definisati funkciju kojom se predviđaju vrednosti cene stana, za bilo koju veličinu stana.

Pri pozivu metode **plt.scatter(x,y)** na grafik se unose podaci u vidu tačaka. U ovom primeru se formira grafik kao na slici 4.1.2.



Slika 4.1.2: Grafik prikaza podataka pomoću metode scatter

Ukoliko se doda i poziv metode **plt.plot(vec,predict(vec))**, na prethodnu sliku će se docrtati linearna funkcija koja se dobija predviđanjem na osnovu datih parova vrednosti, kao što se može uočiti na slici 4.1.3.



Slika 4.1.3: Grafik prikaza linearne funkcije dobijene na osnovu predviđanja

U nastavku se može uočiti demonstracija primera 4.1.1. u programskom jeziku Pajton.

```
import numpy as np
import matplotlib.pylab as plt
x = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([15, 32, 66, 45, 90, 153, 170, 200])
# iscrtamo na grafiku u vidu tacaka unesene podatke
plt.scatter(x,y)
# prvo je potrebno da izracunamo nagib, što možemo da učinimo pomoću formule za m:
m = (len(x) * np.sum(x*y) - np.sum(x) * np.sum(y)) / (len(x)*np.sum(x*x) - np.sum(x) ** 2)
# zatim racunamo pristrasnost pomoću formule za b:
b = (np.sum(y) – m *np.sum(x)) / len(x)
# nakon čega ispisujemo rezultate
print(m)
print(b)
# sada kreiramo funkciju za predvidjanje za približne cele brojeve
# koji označavaju nagib i pristrasnost
def predict(x):
    return 27*x – 26
# uz pomoc te funkcije cemo da nacrtamo na grafiku linearnu funkciju
# koja najvise odgovara nasim podacima
```

```
vec = np.arange(10)
plt.scatter(x,y)
plt.plot(vec,predict(vec))
# sada je potrebno da napisemo funkciju za generisanje
# linearno regresionog modela na osnovu dva vektora
# koristi se zatvaranje radi dinamickog generisanja funkcije
def getlinear(x,y):
    def inner(x1):
        return m * x1 + b
   m = (len(x) * np.sum(x*y) - np.sum(x) * np.sum(y)) / (len(x)*np.sum(x*x) - np.sum(x)
   np.sum(x))
    b = (np.sum(y) - m *np.sum(x)) / len(x)
    return inner
# ukoliko imamo dva vektora X i Y i zelimo da kreiramo linearni model, koristimo:
predict = getlinear(x,y)
# da bi se vratila predvidjena vrednost uz pomoc funkcije:
y1 = predict(12)
print(y1)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
27.273809523809526
-26.35714285714286
300.9285714285714
```

Primer 4.1.2. *Primena linearne regresije uz pomoć ugrađenih funkcija***:**

Podaci iz prethodnog primera nisu morali „peške" da se računaju, već su mogle da se upotrebe već postojeće ugrađene funkcije za njih. Sada će biti odrađen isti primer, samo na drugi način.

Prva neophodna stavka koja mora da se uradi je da se importuje biblioteka NumPy i klasa *LinearRegression* iz *sklearn.linear_model*, nakon čega će biti dostupne sve funkcije koje su neophodne da bi linearnu regresija bila implementirana.

Klasa *sklearn.linear_model.LinearRegression* je neophodna da bi se izvela linearna regresija i predvidela vrednost.

Kada se formira niz x, potrebno je da mu se promeni oblik, jer se zahteva da bude u dve dimenzije, odnosno mora da ima jednu kolonu i onoliko redova koliko je neophodno. Upravo to radi argument (-1,1) funkcije **reshape()**. Kada se ispiše x, može da se uoči da postoje dve dimenzije, i da mu je oblik (8,1), dok y ima jednu dimenziju, odnosno oblika je (6,).

U ovom primeru za kreiranje linearno regresionog modela potrebno je kreirati instancu klase LinearRegression.

Pozivom:

model = LinearRegression(),

kreira se promenljiva model kao instaca te klase. Moguće je dodati nekoliko opcionih parametara:

- **fit_intercept** je tipa Boolean (podrazumevano *True*) i odlučuje da li da se računa pristrasnot (*True*) ili ga smatra jednakim 0 (*False*)
- **normalize** je tipa Boolean (podrazumevano *False*) i odlučuje da li da normira izlaznu promenljivu (*True*) ili ne (*False*)
- **copy_X** je tipa Boolean (podrazumevano *True*) i odlučuje da li da kopira (*True*) ili piše preko postojeće promenljive izlaznu promenljivu (*False*)
- **n_jobs** je celobrojna vrednost ili *None* (podrazumevano) i predstavlja broj posla korišćenog u paralelnom računanju. *None* obično znači jedan posao i -1 za korišćenje svih procesora.

U ovom primeru koriste se svi podrzumevani parametri.

Funkcija **fit()** računa optimalne vrednosti za dužinu pristrasnosti. Funkcija **score()** računa koeficijent determinacije, zatim atribut **intercept_** vraća pristrasnost, a atribut **coef_** nagib.

U nastavku se može uočiti demonstracija primera 4.1.2. u programskom jeziku Pajton.

```
import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([1,2,3,4,5,6,7,8])
y = np.array([15,32,66,45,90,153,170,200])
```

```
print(x)
print(y)
model = LinearRegression().fit(x,y)
r_sq = model.score(x,y)
print('koeficijent: ', r_sq)
print('pristrasnost: ', model.intercept_)
print('nagib: ', model.coef_)
y_pred = model.predict(x)
print('predvidjene vrednosti za y: ', y_pred, sep='\n')
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
[[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]]
[ 15 32 66 45 90 153 170 200]
koeficijent: 0.9261357851573176
pristrasnost: -26.357142857142847
nagib: [27.27380952]
predvidjene vrednosti za y:
[0.91666667 28.19047619 55.46428571 82.73809524 110.01190476
137.28571429 164.55952381 191.83333333]
```

4.2. Višestruka linearna regresija

Ukoliko se problem koji se posmatra može tretirati kao problem jedne zavisne i više nezavisnih promenljivih, radi se o pogodnoj situaciji za analizu podataka metodom višestruke regresije. Ako je veza između njih linearna, slučaj se svodi na *višestruki linearni model*. Svaki podatak je vektor $(x_1, x_2, \dots, x_m)$ sastavljen od dve ili više vrednosti podataka koje obuhvataju različite karakteristike ulaza. Za prikaz svih ulaznih podataka zajedno sa vektorom izlaznih vrednosti postavlja se ulazna matrica X i izlazni vektor y:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1m} \\ 1 & x_{21} & x_{22} & & x_{2m} \\ & \vdots & & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \text{i}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

Primer 4.2.1. *Višestruka linearne regresija***:**

Može da se implementira višestruka linearna regresija prateći iste korake kao u primeru 4.1.2, zbog čega naredni koraci neće biti detaljno objašnjeni.

```
import numpy as np
from sklearn.linear_model import LinearRegression
x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]
y = [4, 5, 20, 14, 32, 22, 38, 43]
x, y = np.array(x), np.array(y)
print(x)
print(y)
model = LinearRegression().fit(x, y)
r_sq = model.score(x, y)
print('koeficijent: ', r_sq)
print('pristrasnost: ', model.intercept_)
print('nagib: ', model.coef_)
y_pred = model.predict(x)
print('predvidjene vrednosti za y: ', y_pred, sep='\n')
# predvidjanje vrednosti y za druge vrednosti x-a, na osnovu postavljenog modela:
```

```
x_new = np.arange(10).reshape(-1, 2))
y_new = model.predict(x_new)
print(x)
print(y)
```

Nakon pokretanja programa, dobija se rezultat prikazan u nastavku.

```
[[ 0 1]
[ 5 1]
[15 2]
[25 5]
[35 11]
[45 15]
[55 34]
[60 35]]
[4 5 20 14 32 22 38 43]
koeficijent: 0.8615939258756776
pristrasnost: 5.522579275198183
nagib: [0.44706965 0.25502548]
predvidjene vrednosti za y:
[ 5.77760476 8.012953 12.73867497 17.9744479 23.97529728 29.4660957
38.78227633 41.27265006]
[[0 1]
[2 3]
[4 5]
[6 7]
[8 9]
[ 5.77760476 7.18179502 8.58598528 9.99017554 11.3943658 ]
```

4.3. Praktični primeri

U ovom poglavlju je objašnjena jedna od najbitnijih uloga biblioteke NumPy, odnosno kako može da se primeni ova biblioteka u analizi podataka. Obrađena je linearna regresija (prosta linearna regresija i višestruka linearna regresija), metodom najmanjih kvadrata. Sve to je obrađeno kroz teorijski deo i primere. Primeri su objašnjeni na dva različita načina, odnosno kako da se ispita linearnost funkcije „peške“ i kako da se ispita pomoću ugrađenih funkcija.

U nastavku se nalaze primeri za samostalno vežbanje, koji obuhvataju teme obrađene u ovoj glavi.

1. Vršeno je istraživanje koliko fizička aktivnost utiče na efikasnost na poslu. U tabeli se nalaze odgovori od 19 ispitanika koliko časova nedeljno imaju fizičke aktivnosti i koliko sati dnevno mogu efikasno da rade.

Fiz. akt.	2	6	8	3	5	6	4	4.5	3	3	6	8	10	9	7.5	5	2	3	12
Efik.	3	5.4	9	3.5	5.5	5	5	4	4	3	8	7	8	9.5	8	6.3	3	4.5	10

   Na osnovu datih podataka, predvideti kolika je efikasnost na poslu ukoliko neko vežba 5.5 sati nedeljno.

2. Vršeno je istraživanje da li starosna dob osobe utiče na broj godina radnog iskustva. U tabeli se nalaze odgovori od 8 ispitanika koliko imaju godina i koliko imaju godina radnog iskustva.

Godine	28	25	43	36	21	32	50	33
Radno iskustvo	4	3.5	7	10	1	3	25	0

   Na osnovu datih podataka, nacrtati grafik i predvideti koliko godina radnog iskustva treba da ima osoba stara 27 godina.

3. Vršeno je istraživanje da li godine i težina žene utiče na to koliko para mesečno troši na kozmetiku. U sledećoj tabeli se nalaze podaci za 10 ispitanika:

Godine	18	20	21	23	25	30	38	39	42	57
Težina (kg)	52	63	85	60.5	65.2	61	87.1	73	69	92.1
Potrošnja para	0.5	0.75	1.2	2	0.5	5	5.1	4	6.2	5.3

   Pri čemu se potrošnja para izražava u hiljadama, odnosno primera radi 4 predstavlja 4000 dinara. Na osnovu datih podataka predvideti koliko para mesečno troši na kozmetiku žena koja ima 28 godina i 89 kilograma.

5. Rešenja praktičnih primera

Ova glava sadrži odgovore na praktične primere, koji su postavljeni na kraju svakog poglavlja. Svaki odgovor je detaljno obrazložen, radi lakšeg razumevanja prethodno objašnjenog gradiva. Za rešavanje zadatih primera se koristi gradivo obrađeno u datom poglavlju.

Svi zadaci su rešavani u *Anaconda Navigator*, unutar *Jupyter Notbook*.

Uvek na početku svakog primera je neophodno uključiti biblioteku NumPy, radi korišćenja atributa, modula i funkcija koji su sadržani unutar nje. To se izvršava na sledeći način: *import numpy as np*.

5.1. Rešeni praktični primeri iz poglavlja 2

1. Prvo je potrebno da se kreira tražena lista:

$$l = [3, 6, 3, 2, 5, 8, 1, 2, 6, 1, 8, 7].$$

Potom se ispisuje njen tip pomoću funkcije type(), radi provere da li je stvarno kreirana lista. Kada se dokaže da je kreirani objekat lista, na osnovu nje se kreira NumPy jednodimenzionalni niz pomoću funkcije array().

```
In [4]: import numpy as np

        # neka je l zadata lista
        l = [3, 6, 3, 2, 5, 8, 1, 2, 6, 1 ,8 ,7]
        # proverava se da li je l tipa liste
        type(l)

Out[4]: list

In [5]: # kreira se NumPy niz na osnovu ove liste
        a = np.array(l)

In [6]: print("Sada niz izgleda ovako:\n", a)

        Sada niz izgleda ovako:
         [3 6 3 2 5 8 1 2 6 1 8 7]
```

Nakon toga se pomoću metoda reshape() menja oblik kreiranog jednodimenzionalnog niza u dvodimenzionalni niz oblika 4x3:

$$\begin{bmatrix} 3 & 6 & 3 \\ 2 & 5 & 8 \\ 1 & 2 & 6 \\ 1 & 8 & 7 \end{bmatrix}$$

```
In [9]: # menja se oblik niza pomocu metoda reshape()
        a1 = a.reshape(4, 3)
        print("Nakon menjanja oblika, niz izgleda ovako:\n", a1)

Nakon menjanja oblika, niz izgleda ovako:
 [[3 6 3]
 [2 5 8]
 [1 2 6]
 [1 8 7]]
```

Kada je kreiran traženi dvodimenzionalni niz, potrebno je proveriti kog je tipa pomoću funkcije type().Tip objekta je numpy.ndarray, jer je kreiran niz Numpy niz. Zatim se proverava kog su tipa elementi toga niza pomoću metoda dtype. Elementi niza su celobrojne vrednosti, odnosno tipa su int32.

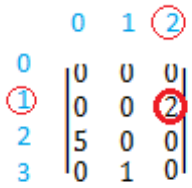
Nakon toga je potrebno proveriti koje je dimenzije kreirani niz pomoću metoda ndim. Dimenzija niza će biti 2, jer je u pitanju dvodimenzionalni niz. Nakon čega se proverava kog je oblika dati niz pomoću metoda shape. Oblik niza je (4, 3), odnosno ima 4 reda i 3 kolone. Na samom kraju se proverava koje je veličine niz pomoću metoda size. Niz je veličine 12, što znači da ima ukupno 12 elemenata u nizu.

```
In [10]: # kog je tipa kreirani objekat?
         print("Objekat je tipa: ", type(a1))

Objekat je tipa:  <class 'numpy.ndarray'>

In [11]: # kog su tipa elementi datog niza?
         print("Tip elemenata niza je: ", a1.dtype)

Tip elemenata niza je:  int32

In [12]: # koje je dimenzije niz?
         print("Niz je dimenzije: ", a1.ndim)

Niz je dimenzije:  2

In [13]: # kog je oblika niz?
         print("Niz je oblika: ", a1.shape)

Niz je oblika:  (4, 3)

In [14]: # koje je velicine niz?
         print("Niz je velicine: ", a1.size)

Niz je velicine:  12
```

2. Potrebno je kreirati sledeći dvodimenzionalni niz:

$$\begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 2 \\ 5 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix}.$$

To je moguće uraditi na više načina, ali ovde će se konkretno pokazati upotreba funkcije zeros() koja služi za kreiranje niza popunjenog samo vrednostima 0. Nakon kreiranja toga niza, potrebno je samo promeniti vrednosti na datim pozicijama, tako što se kao prvi argument prosleđuje u kom redu se nalazi željena vrednost, a kao drugi argument u kojoj koloni. Primera radi, a[1, 2] = 2 znači da će se promeniti sledeća vrednost u nizu:



Dakle, menja se vrednost u prvom redu i drugoj koloni, odnosno umesto vrednosti 0, na poziciji (1, 2) nalaziće se vrednost 2.

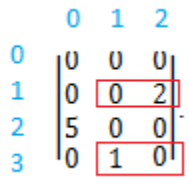
```
In [4]: import numpy as np

        # kreiranje niza oblika 4x3 popunjenog samo vrednositma 0:
        a = np.zeros((4, 3))
        print("Sada niz izgleda ovako:\n", a)
```

```
Sada niz izgleda ovako:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

```
In [6]: # kreiranje trazenog niza:
        a[1, 2] = 2
        a[2, 0] = 5
        a[3, 1] = 1
        print("Sada niz izgleda ovako:\n", a)
```

```
Sada niz izgleda ovako:
 [[0. 0. 0.]
 [0. 0. 2.]
 [5. 0. 0.]
 [0. 1. 0.]]
```

Nakon kreiranja traženog niza, potrebno je srezati ga na dimenziju 2x2, tako što se izvlače sledeći elementi iz njega:



To se vrši na sledeći način:

b = a[1::2, 1:],

pri čemu se kao prvi argument prosleđuje koji redovi se izdvajaju, a kao drugi argument koje kolone se izdvajaju.

„1::2“ znači da počevši od prvog reda izdvaja se svaki drugi red, odnosno biće izdvojeni redovi 1 i 3.

„1: “ znači da počevši od prve kolone izdvajaju se sve kolone do kraja.

```
In [8]: # Kreiranje niza oblika 2x2 na
        #osnovu dobijenog dvodimenzionalnog niza
        b = a[1::2, 1:]
        print("Novi niz izgleda ovako:\n", b)
```

```
Novi niz izgleda ovako:
 [[0. 2.]
 [1. 0.]]
```

3. Potrebno je kreirati sledeći dvodimenzionalni niz:

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 5 \\ 6 & 2 & 2 & 1 & 2 \end{bmatrix}.$$

Slično kao u prethodnom zadatku, kreiraće se prvo niz popunjen samo vrednostima 2 pomoću funkcije full(), nakon čega se promene samo potrebne vrednosti u nizu.

```
In [2]: import numpy as np

        # kreiranje niza oblika 2x5 cije su sve vrednosti 2:
        a = np.full((2, 5), 2)
        print("Niz izgleda ovako:\n", a)
```

```
Niz izgleda ovako:
 [[2 2 2 2 2]
 [2 2 2 2 2]]
```

```
In [3]: # kreiranje trazenog niza:
        a[0, 4] = 5
        a[1, 0] = 6
        a[1, 3] = 1
        print("Sada niz izgleda ovako:\n", a)

Sada niz izgleda ovako:
 [[2 2 2 2 5]
 [6 2 2 1 2]]
```

Nakon toga je potrebno da se od tog dvodimenzionalnog niza dobije jednodimenzionalni niz, odnosno da se spljošti niz. To se postiže pomoću metode flatten(). Kada se spljošti niz, potrebno je izdvojiti sve vrednosti niza koje su različite od 2, tako što se kao argument nizu prosleđuje uslov „b != 2“.

```
In [4]: # promena oblika u jednodimenzionalni niz:
        b = a.flatten()
        print("Jednodimenzionalni niz izgleda ovako:\n", b)

Jednodimenzionalni niz izgleda ovako:
 [2 2 2 2 5 6 2 2 1 2]
```

```
In [6]: # izdvajanje svih vrednosti iz niza koje su razlicite od 2:
        c = b[b != 2]
        print("Vrednosti niza koje su razlicite od 2:\n", c)

Vrednosti niza koje su razlicite od 2:
 [5 6 1]
```

4. U ovome zadatku je potrebno kreirati niz oblika 6x3 sa random vrednostima i ispisati ga.

Najlakši način za kreiranje toga niza je pomoću metode random.random(), kojoj se prosleđuje dimanzija niza koji je potrebno kreirati.

```
In [1]: import numpy as np

        # kreiranje niza oblika 6x3 sa random vrednostima:
        a = np.random.random((6, 3))
        print("Niz izgleda ovako:\n", a)

Niz izgleda ovako:
 [[0.77332381 0.09498371 0.66848126]
 [0.90052855 0.80357818 0.93504673]
 [0.77449452 0.30252921 0.69883606]
 [0.64223079 0.28432609 0.26984078]
 [0.45453597 0.66763246 0.51660526]
 [0.56243959 0.43894198 0.14000175]]
```

Međutim, nekada je potrebno kreirati niz samo sa celobrojnim vrednostima. To je moguće uraditi pomoću metoda random.randint(). Primera radi, neka se kreira niz celobrojnih vrednosti od 0 do 10. Da bi se to postiglo, kao prvi argument se u metodu prosleđuje do kog celog broja se kreiraju random vrednosti u nizu, dakle u ovom primeru će to biti broj 10, a kao drugi argument se prosleđuje oblik niza.

```
In [2]: # niz oblika 6x3 sa random celobrojnim vrednostima od 0 do 10
        a = np.random.randint(10, size=(6,3))
        print("Niz izgleda ovako:\n", a)
```

```
Niz izgleda ovako:
 [[7 2 3]
 [1 3 9]
 [0 7 8]
 [0 7 9]
 [7 4 5]
 [8 1 2]]
```

5. Da bi se kreirala sledeća matrica:

$$\begin{bmatrix} 5 & 5 & 5 & 5 & 5 \\ 5 & 3 & 3 & 3 & 5 \\ 5 & 3 & 1 & 3 & 5 \\ 5 & 3 & 3 & 3 & 5 \\ 5 & 5 & 5 & 5 & 5 \end{bmatrix},$$

potrebno je prvo kreirati matricu dimenzije 5x5 ispunjenu samo brojem 5 i jos jednu matricu dimenzije 3x3 ispunjenu samo brojem 3. Zatim se u drugoj matrici središnji član promeni da bude 1.

```
In [1]: import numpy as np

        # matrica ispunjena brojem 5:
        a = np.full((5,5), 5)
        print("Prva matrica:\n", a)
```

```
Prva matrica:
 [[5 5 5 5 5]
 [5 5 5 5 5]
 [5 5 5 5 5]
 [5 5 5 5 5]
 [5 5 5 5 5]]
```

```
In [2]: # matrica ispunjena brojm 3:
        b = np.full((3,3), 3)
        b[1,1] = 1
        print("Druga matrica:\n", b)
```

```
Druga matrica:
 [[3 3 3]
 [3 1 3]
 [3 3 3]]
```

Nakon kreiranja pomoćne matrice b, potrebno je da se centralni deo prve matrice a zameni sa tom pomoćnom matricom.

```
In [3]: # trazena matrica:
        a[1:4, 1:4] = b
        print("Trazena matrica je:\n", a)
```

```
Trazena matrica je:
 [[5 5 5 5 5]
 [5 3 3 3 5]
 [5 3 1 3 5]
 [5 3 3 3 5]
 [5 5 5 5 5]]
```

.

6. Potrebno je na osnovu date matrice, kreirati tri nove matrice od elemenata označenih različitim bojama na slici:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix}$$

Posmatra se prvo crvena boja. Elementi koji su označeni se nalaze u trećem i četvrom redu i nultoj i prvoj koloni. Dakle, njima se pristupa na sledeći način:

c = a1[2:, :2],

pri čemu za prvi argument uzimaju se redovi od drugog do kraja, a za drugi argument kolone od početka do druge.

```
In [5]: # kreiranje matrice od elemenata oznacenih crvenom bojom
        c = a1[2:, :2]
        print("Matrica od elemenata oznacenih crvenom bojom:\n", c)
```

```
Matrica od elemenata oznacenih crvenom bojom:
 [[13 14]
 [19 20]]
```

Posmatraju se sada elementi matrice označeni žutom bojom. Elementi se nalaze u nultom, drugom i trećem redu. Kada je potrebno izdvojiti redove ili kolone, koji se ne nalaze jedan do drugog ili nisu poredjani tako da je lako uočiti šablon (primera radi, svaki drugi red, svaka treća kolona i tome slično), tada je potrebno proslediti listu sa redovima koji se izdvajaju. Zbog toga će se ovim elementima pristupiti na sledeći način:

zu = a1[[0,2,3],2:].

```
In [7]: # kreiranje matrice od elemenata oznacenih zutom bojom
        zu = a1[[0,2,3],2:]
        print("Matrica od elemenata oznacenih zutom bojom:\n", zu)

        Matrica od elemenata oznacenih zutom bojom:
         [[ 3  4  5  6]
         [15 16 17 18]
         [21 22 23 24]]
```

Poslednji elementi koji su potrebni da se izdvoje su označeni zelenom bojom. Slično kao malo pre, uočava se da elementi nisu poređani po šablonu, jedan ispod drugog ili jedan pored drugog. Iz toga razloga da bi im se pristupilo, potrebno je proslediti liste sa redovima i kolonama u kojima se elementi nalaze, tako da prvi argumet u prvoj listi označava u kom redu se nalazi prvi broj, a prvi argument u drugoj listi označava u kojoj se koloni nalazi prvi broj. Slično se radi i za ostale brojeve.

```
In [8]: # kreiranje matrice od elemenata oznacenih zelenom bojom
        ze = a1[[0,1,2], [3,4,5]]
        print("Matrica od elemenata oznacenih zelenom bojom:\n", ze)

        Matrica od elemenata oznacenih zelenom bojom:
         [ 4 11 18]
```

7. Potrebno je proveriti da li je broj 25 tipa niza skalara i ukoliko nije, konvertovati ga u odgovarajući tip skalara.

Da li je neka vrednost tipa niza skalara, proverava se pomoću funkcije isinstance().

```
In [4]: import numpy as np

        isinstance(25, np.generic)

Out[4]: False
```

Može da se primeti da je funkcija isinstance() vratila False, što znači da vrednost nije tipa skalara. Pomoću funkcije type() može da se proveri kog je tipa broj 25.

```
In [6]: type(25)
Out[6]: int
```

Broj 25 je tipa int, koji je Pajton tip za celobrojne vrednosti. Potrebno je konvertovati broj 25 u neki od NumPy tipova. Ukoliko se broj 25 konvertuje u tip int_, to se izvršava na sledeći način: np.int_(25).

```
In [5]: isinstance(np.int_(25), np.generic)
Out[5]: True
```

Sada može da se primeti da ovako konvertovan broj jeste tipa niza skalara. Ukoliko je potrebno proveriti kog je konkretno tipa ovaj konvertovan broj, to se može ponovo izvršiti pomoću funkcije type().

```
In [7]: type(np.int_(25))
Out[7]: numpy.int32
```

Može da se primeti da je sada broj 25 tipa niza skalara int32. Odnosno, može da se primeti da su tipovi int_ i int32 ekvivalentni.

5.2. Rešeni praktični primeri iz poglavlja 3

1. Prvo je potrebno kreirati datu matrica:

$$\begin{bmatrix} 3 & 4 & 6 & 3 & 0 & 1 \\ 2 & 15 & 0 & 4 & 6 & 12 \\ 7 & 11 & 9 & 2 & 8 & 5 \\ 1 & 9 & 7 & 1 & 14 & 9 \\ 6 & 3 & 2 & 0 & 6 & 3 \\ 0 & 19 & 8 & 2 & 8 & 4 \end{bmatrix},$$

Tako što se funkciji array() prosledi lista sa željenim elementima.

```
In [2]: import numpy as np

        # kreiranje matrice
        a = np.array([[3,4,6,3,0,1], [2,15,0,4,6,12], [7,11,9,2,8,5], [1,9,7,1,14,9], [6
        print(a)
```

```
[[ 3  4  6  3  0  1]
 [ 2 15  0  4  6 12]
 [ 7 11  9  2  8  5]
 [ 1  9  7  1 14  9]
 [ 6  3  2  0  6  3]
 [ 0 19  8  2  8  4]]
```

Biblioteka NumPy sadrži veliki broj statističkih funkcija, koje su jako bitne u analizi podataka. U ovom primeru su odrađene neke od njih.

Minimumu i maksimumu matrice pristupa se pomoću funkcija min() i max(). Rezultat ovih funkcija je broj, koji označava minimalnu, odnosno maksimalnu, vrednost od svih brojeva unutar matrice.

```
In [3]: # minimum i maksimum matrice:
        print("Minimum matrice je: ", np.min(a))
        print("Maksimum matrice je: ", np.max(a))
```

```
Minimum matrice je:  0
Maksimum matrice je:  19
```

Kada je potrebno pristupiti minimalnim, odnosno maksimalnim, brojevima po svim redovima, unutar funkcije se doda i argument axis=1. Slično tome, ako je potrebno pristupiti po kolonama, dodaje se argument axis=0.

```
In [4]: # minimum po svim redovima:
        print("Minimum po redovima: ", np.min(a, axis=1))
```

```
Minimum po redovima:  [0 0 2 1 0 0]
```

```
In [5]: # maksimum po svim kolonama:
        print("Maksimum po kolonama: ", np.max(a, axis=0))
```

```
Maksimum po kolonama:  [ 7 19  9  4 14 12]
```

Zbir svih elemenata matrice se računa pomoću funkcije sum().

```
In [6]: # zbir svih elemenata:
        print("Zbir svih elemenata: ", np.sum(a))
```

```
Zbir svih elemenata:  200
```

Srednja vrednost svih elemenata u matrici se računa pomoću funkcije mean(). Medijana se računa pomoću funkcije median(). Standardna devijacija se računa pomoću funkcije std().

Kao i kod funkcija min() i max(), i kod ovih funkcija se može dodati argument axis.

```
In [7]: # srednja vrednost:
        print("Srednja vrednost matrice: ", np.mean(a))

        Srednja vrednost matrice:  5.555555555555555

In [8]: # medijana:
        print("Medijana: ", np.median(a))

        Medijana:  4.5

In [9]: # standardna devijacija:
        print("Standardna devijacija: ", np.std(a))

        Standardna devijacija:  4.536545702798578
```

2. Potrebno je izračunati:

$$\begin{bmatrix} 0 & i \\ i & o \end{bmatrix}^n,$$

gde je $i^2 = -1$, nϵN.

Prvo je potrebno kreirati matricu *a*, za n=1, pa zatim proveravati za stepene 2, 3 i tako dalje, dokle god se ne naiđe na neki šablon.

Može se uočiti da je u pitanju kompleksna matrica. Kompleksni broj *i* se označava sa „*j*“ u programskom jeziku Pajton.

```
In [5]: import numpy as np

        a = np.array([[0, 1j], [1j, 0]])
        print(a)

        [[0.+0.j 0.+1.j]
         [0.+1.j 0.+0.j]]
```

Nakon kreiranja početne matrice, računa se ta matrica sa stepenom 2 na sledeći način:

```
In [7]: a2 = a**2
        print("a^2 = ", a2)

        a^2 =  [[ 0.+0.j -1.+0.j]
         [-1.+0.j  0.+0.j]]
```

Može se uočiti da je $a^2 = -E$, pri čemu E označava jediničnu matricu.

Sada se računa data matrica sa stepenom 3:

```
In [8]: a3 = a**3
        print("a^3 = ", a3)

a^3 =  [[ 0.+0.j -0.-1.j]
 [-0.-1.j  0.+0.j]]
```

Ovde se može uočiti da je $a^3 = -a$.

Potom se računa data matrica sa stepenom 4:

```
In [9]: a4 = a**4
        print("a^4 = ", a4)

a^4 =  [[0.+0.j 1.+0.j]
 [1.+0.j 0.+0.j]]
```

Uočava se da je $a^4 = E$. Poznato je iz algebre da kada se neka matrica množi sa jediničnom matricom, ne menja se. Odnosno da važi:

$$A \times E = A,$$

gde je A proizvoljna matrica, a E je njoj odgovarajuća jedinična matrica.

Dakle, ako bi se računala sada matrica sa stepenom 5, bila bi jednaka početnoj matrici. Odatle može da se ustanovi šablon, odnosno da se donese zaključak:

$$a^{4k} = E,$$
$$a^{4k+1} = a,$$
$$a^{4k+2} = -E,$$
$$a^{4k+3} = -a,$$

gde je k=0,1,2...

3. Prvo je potrebno kreirati sledeće matrice:

$$A = \begin{bmatrix} 2 & 3 \\ 5 & 7 \end{bmatrix},$$
$$B = \begin{bmatrix} 1 & 4 \\ 0 & 9 \end{bmatrix},$$
$$C = \begin{bmatrix} 7 & 9 \\ 4 & 5 \end{bmatrix} \text{ i}$$
$$D = \begin{bmatrix} 1 & 3 & 7 \\ 4 & 9 & 0 \end{bmatrix}.$$

```
In [1]: import numpy as np

        a = np.array([[2,3],[5,7]])
        b = np.array([[1,4],[0,9]])
        c = np.array([[7,9],[4,5]])
        d = np.array([[1,3,7],[4,9,0]])
        print("A = ", a)
        print("B = ", b)
        print("C = ", c)
        print("D = ", d)
```

```
A =  [[2 3]
 [5 7]]
B =  [[1 4]
 [0 9]]
C =  [[7 9]
 [4 5]]
D =  [[1 3 7]
 [4 9 0]]
```

Izraz $A \times B + C^4$ se računa na sledeći način:

```
In [3]: x = a.dot(b) + c**4
        print("A x B + C^4 = ", x)
```

```
A x B + C^4 =  [[2403 6596]
 [ 261  708]]
```

Izraz $(A - C + B) \times D$ se računa na sledeći način:

```
In [4]: y = (a - c + b).dot(d)
        print("(A - C + b)/D = ", y)
```

```
(A - C + b)/D =  [[-12 -30 -28]
 [ 45 102   7]]
```

Izraz $\frac{A+C*B}{B-10}$ se računa na sledeći način:

```
In [5]: z = (a + c * b)/(b - 10)
        print("(A + C*B)/(B - 10) = ", z)
```

```
(A + C*B)/(B - 10) =  [[ -1.   -6.5]
 [ -0.5 -52. ]]
```

Za množenje elemenata matrica koji se nalaze na istim pozicijama, koristi se operator „*“, dok se za množenje matrica po algebarskim pravilima koristi metod dot(). Pri rešavanju ovakih zadataka, potrebno je obratiti pažnju da li su dimenzije matrica usklađene. Ukoliko nisu, dolazi do greške.

4. Prvo je potrebno kreirati početnu matricu:

$$\begin{bmatrix} 5 & 7 & 1 & 4 & 7 \\ 0 & 5 & 12 & 2 & 9 \\ 9 & 3 & 7 & 1 & 0 \\ 0 & 8 & 21 & 0 & 3 \\ 11 & 0 & 0 & 1 & 6 \end{bmatrix}.$$

```
In [32]: import numpy as np

         a = np.array([[5,7,1,4,7],[0,5,12,2,9],[9,3,7,1,0],[0,8,21,0,3],
                       [11,0,0,1,6]], dtype = np.int64)
         print(a)
         print(a.dtype)
```

```
[[ 5  7  1  4  7]
 [ 0  5 12  2  9]
 [ 9  3  7  1  0]
 [ 0  8 21  0  3]
 [11  0  0  1  6]]
int64
```

Može da se primeti da je tip elemenata u matrici postavljen da bude int64, jer ukuliko se to ne uradi, neće se dobiti tačno rešenje pri proizvodu, zbog toga što se dobija veliki broj, pa mu je potrebno više memorije.

Suma svih parnih elemenata u matrici se računa na sledeći način:

```
In [2]: # suma svih parnih elemenata u matrici
        s = np.sum(a[a%2 == 0])
        print("Suma svih parnih elemenata: ", s)
```

```
Suma svih parnih elemenata:  32
```

Proizvod svih nenula elemenata van dijagonale nije jednostavno da se izračuna kao prethodna suma. Da bi se izračunao taj proizvod, potrebno je da se koristi for petlja, pri čemu se u njoj pamti index (torka koordinata trenutne pozicije unutar matrice) i x (vrednost koja se nalazi na trenutnoj poziciji). Da bi se pamtile pozicija i vrednost unutar matrice, koristi se funkcija ndenumerate()[17].

Unutar for petlje je potrebno da se ispune određeni uslovi, da bi se pomnožile odgovarajuće vrednosti, koje se postavljaju unutar if uslova. Uslov da se množe nenula elementi se zadaje na sledeći način:

x != 0.

[17] Ovaj zadatak se mogao rešiti i korišćenjem dvostruke for petlje, bez upotrebe ndenumerate() funkcije.

Torka koordinata ima sledeći oblik: *index = (i, j)*, pri čemu *i* predstavlja koordinatu reda, a *j* predstavlja koordinatu kolone. Njima se pristupa na sledeći način: *i = index[0]*, dok je *j = index[1]*. Iz tog razloga se uslov da broj nije na dijagonali proverava na sledeći način:

index[0] != index[1].

Dakle, odgovarajući uslov ima sledeći oblik:

if x != 0 and index[0] != index[1].

Pre for petlje potrebno je proizvod postaviti na 1, da bi se kasnije unutar petlje množio taj broj sa odgovoarajućom vrednošću.

```
In [20]: # proizvod svih nenula elemenata van dijagonale
         p = 1
         for index, x in np.ndenumerate(a):
             if x != 0 and index[0] != index[1]:
                 p*=x
         print("Proizvod svih nenula elemenata van dijagonale: ", p)
```

```
Proizvod svih nenula elemenata van dijagonale:  6337191168
```

Matrica se sortira pomoću funkcije sort(). Sortiranje matrice se može vršiti na osnovu kolona i na osnovu redova. Podrazumevano je postavljeno na osnovu redova.

```
In [21]: # Sortiranje pocetne matrice:
         print("Sortirana matrica po redovima:")
         print(np.sort(a))
```

```
Sortirana matrica:
[[ 1  4  5  7  7]
 [ 0  2  5  9 12]
 [ 0  1  3  7  9]
 [ 0  0  3  8 21]
 [ 0  0  1  6 11]]
```

Ukoliko je potrebno da se sortiraju vrednosti na osnovu kolona, dodaje se argument axis=0.

```
In [23]: print("Sortirana matrica po kolonama:")
         print(np.sort(a, axis=0))
```

```
Sortirana matrica po kolonama:
[[ 0  0  0  0  0]
 [ 0  3  1  1  3]
 [ 5  5  7  1  6]
 [ 9  7 12  2  7]
 [11  8 21  4  9]]
```

Poslednji deo zadatka je da se kreira matrica kojoj se svim parnim elementima promeni znak, svi neparni elementi se uvećaju dva puta i sve nule se zamene najvećim brojem iz kolone u kojoj se nalazi ta nula. Ovo je moguće rešiti tako što se kreira nova matrica, pa se unutar for petlje njoj menjaju vrednosti u zavisnosti sa traženim uslovima, međutim u ovom rešenju su promenjeni elementi početne matrice, zbog jednostavnijeg koda.

Slično kao malo pre, koristi se for petlja koja čuva index i x.

Unutar for petlje se definišu promenljive *i* i *j* u kojima se čuvaju koordinate trenutnog reda i kolone.

Uslov da se svim parnim elementima promeni znak se postiže na sledeći način:

if x%2 == 0 and x !=0:

a[i,j] = -a[i,j].

Uslov da se svi neparni brojevi uvećaju dva puta se postiže na sledeći način:

elif x%2 != 0:

a[i,j] *= 2.

Uslov da se sve nule zamene najvećim brojem iz kolone u kojoj se nalazi ta nula, se postiže na sledeći način:

elif x == 0:

a[i,j] = max[j],

pri čemu se pre toga definiše matrica max u kojoj se čuvaju svi najveći brojevi iz kolona.

```
In [25]: max = np.max(a, axis=0)
         print(max)

         [11  8 21  4  9]
```

```
In [33]: for index, x in np.ndenumerate(a):
             i = index[0]
             j = index[1]
             if x%2 == 0 and x !=0:
                 a[i,j] = -a[i,j]
             elif x%2 != 0:
                 a[i,j] *= 2
             elif x == 0:
                 a[i,j] = max[j]
         print(a)
```

```
[[ 10  14   2  -4  14]
 [ 11  10 -12  -2  18]
 [ 18   6  14   2   9]
 [ 11  -8  42   4   6]
 [ 22   8  21   2  -6]]
```

5. Potrebno je kreirati simulaciju koja ispisuje koje se vrednosti dobiju ukoliko se novčić baca 10 puta.

Prvo je potrebno definisati praznu listu u kojoj će se čuvati vrednosti koje označavaju da li se dobija pri bacanju novčića pismo ili glava. Takodje je neophodno definisati promenljive pismo i glava, koji se koriste kao brojači koliko puta je dobijeno pismo, a koliko puta glava.

```
In [6]: import numpy as np

        a = []
        pismo = 0
        glava = 0
```

Radi kreiranja ove simulacije koristi se for petlja, koja omogućava da se kroz petlju prolazi 10 puta na sledeći način:

for x in range(10).

Da bi simulacija bila regularno kreirana, potrebno je da se omogući da se dobijaju pismo ili glava na slučajan način. To se može izvršiti pomoću metode random.randint(0,2), kojom se bira random broj u intervalu [0,2).

Pri svakom prolasku kroz for petlju se u listu koja je definisana na početku, pomoću metoda append(), dodaje string „glava“, ukoliko je dobijeni broj 0 ili string „pismo“, ukoliko je dobijeni broj 1. Pored toga se na brojač pismo dodaje 1 svaki put kada je dobijen broj 1, odnosno na brojač glava se dodaje 1 ukoliko je dobijen broj 0.

```
for x in range(10):
    novcic = np.random.randint(0,2)
    if novcic == 0:
        a.append("glava")
        glava+=1
    else:
        a.append("pismo")
        pismo+=1
print(a)
print("Dobija se ukupno glava: "+str(glava)+" i pisma: "+str(pismo))
```

```
['glava', 'pismo', 'pismo', 'glava', 'glava', 'pismo', 'glava', 'pismo', 'glav
a', 'glava']
Dobija se ukupno glava: 6 i pisma: 4
```

6. Potrebno je proveriti da li je tačka (0, -1, 5) rešenje sistema jednačina:

$$x - y + z = 13$$
$$2x + 5y - 3z = -13$$
$$4x - y - 6z = -4.$$

Da bi se to proverilo, potrebno je prvo sistem jednačina prebaciti u matrični oblik:

$$A \cdot X = B,$$

pri čemu su A, X i B:

$$A = \begin{bmatrix} 1 & -1 & 1 \\ 2 & 5 & -3 \\ 4 & -1 & -6 \end{bmatrix},$$
$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix},$$
$$B = \begin{bmatrix} 13 \\ -13 \\ -4 \end{bmatrix}.$$

Dakle, potrebno je pronaći matricu X, na osnovu matrice A i B.

Prvo je potrebno kreirati matrice A i B:

```
In [4]: import numpy as np

        a = np.array([[1,-1,1],[2,5,-3],[4,-1,-6]])
        print("Matrica A:")
        print(a)
```

```
Matrica A:
[[ 1 -1  1]
 [ 2  5 -3]
 [ 4 -1 -6]]
```

```
In [2]: b = np.array([[13,-13,-4]])
        print("Matrica B:")
        print(b)
```

```
Matrica B:
[[ 13 -13  -4]]
```

Matrica X se pronalazi pomoću metoda linalg.solve(). Njemu se prosleđuju kao argumenti matrica a i transponovana matrica b.

```
In [5]: # resavanje sistema jednacina:
        x = np.linalg.solve(a,b.T)
        print("Resenje jednacije je:")
        print(x)
```

```
Resenje jednacije je:
[[ 6.]
 [-2.]
 [ 5.]]
```

Kada se pronađe rešenje sistema, potrebno je proveriti da li je zadata tačka jednaka tome rešenju. Da bi se to uradilo, prvo je potrebno definisati niz resenje, koji sadrži zadatu tačku koja se proverava. Zatim je potrebno da se dobijeno rešenje sistema transponuje, da bi imalo isti oblik kao definisani niz. Nakon toga se porede ta dva niza pomoću naredbe if, pri čemu mora da se obrati pažnja da ukoliko bi se ta naredba definisala ovako:

if resenje == x_t,

došlo bi do greške, jer se na taj način poredi svaki element unutar ova dva niza i vraća se niz koji ima vrednosti True ukoliko su dva elementa jednaka i vrednosti False ukoliko nisu. If naredba očekuje da dobije samo jednu vrednost True ili False, zbog čega je potrebno da se doda i metod all(), koji proverava da li su svi elementi toga niza True i kao rezultat vraća samo True ili False. Dakle, if naredba treba da izgleda ovako:

if (resenje == x_t).all().

```
In [13]: # provera da li je tacka (0,-1,5) resenje datog sistema
resenje = np.array([[0,-1,5]])
x_t = x.T
if (resenje == x_t).all():
    print("Tacka (0,-1,5) jeste resenje datog sistema!")
else:
    print("Tacka (0,-1,5) nije resenje datog sistema!")
    print("Resenje je tacka: ", x_t)

Tacka (0,-1,5) nije resenje datog sistema!
Resenje je tacka:  [[ 6. -2.  5.]]
```

Iz ovoga rešenja može da se zaključi da data tačka nije rešenje sistema jednačina, već tačka (6,-2,5).

5.3. Rešeni praktični primeri iz poglavlja 4

1. U tabeli se nalaze podaci koliko časova nedeljno ispitanici imaju fizičke aktivnosti i koliko sati dnevno mogu efikasno da rade.

Fiz. akt.	2	6	8	3	5	6	4	4.5	3	3	6	8	10	9	7.5	5	2	3	12
Efik.	3	5.4	9	3.5	5.5	5	5	4	4	3	8	7	8	9.5	8	6.3	3	4.5	10

Na osnovu datih podataka, potrebno je predvideti kolika je efikasnost na poslu ukoliko neko vežba 5.5 sati nedeljno.

Prvo je potrebno da se importuje klasa *LinearRegression* iz *sklearn.linear_model*, nakon čega će biti dostupne sve funkcije koje su neophodne za implementaciju linearne regresije. U ovom zadatku se koristi metoda najmanjih kvadrata.

Kada se kreira niz x, potrebno je da mu se promeni oblik, jer se zahteva da bude u dve dimenzije, što se postiže pomoću argumenta (19,1) funkcije reshape().

Za kreiranje linearno regresionog modela koristi se instanca klase LinearRegression().

Pomoću funkcije fit() računaju se optimalne vrednosti za dužinu pristrasnosti. Funkcija score() računa koeficijent determinacije, atribut intercept_ vraća pristrasnost, a atribut coef_ nagib.

Za predviđanje vrednosti koristi se metod predict().

```
In [18]: import numpy as np
         from sklearn.linear_model import LinearRegression

         x = np.array([2,6,8,3,5,6,4,4.5,3,3,6,8,10,9,7.5,5,2,3,12]).reshape((19,1))
         y = np.array([3,5.4,9,3.5,5.5,5,5,4,4,3,8,7,8,9.5,8,6.3,3,4.5,10])

         model = LinearRegression().fit(x,y)
         print("Koeficijent: ", model.score(x,y))
         print("Pristrasnost: ", model.intercept_)
         print("Nagib: ", model.coef_)
         x_novo = np.array([5.5]).reshape((1,1))
         print("Predvidjena vrednost za x_novo: ", model.predict(x_novo))
```

```
Koeficijent:  0.850157987390134
Pristrasnost:  1.6489740330488463
Nagib:  [0.75111676]
Predvidjena vrednost za x_novo:  [5.78011622]
```

Na osnovu koeficijenta može se zaključiti da postoji jaka veza, što znači da su podaci pogodni za ovaj model.

Nakon predviđanja dolazi se do zaključka da ukoliko neko ima fizičku aktivnost 5.5 sati nedeljno, njegova efikasnost na poslu će biti približna 5.8 sati dnevno.

2. U tabeli su prikazani odgovori od 8 ispitanika koliko imaju godina i koliko imaju godina radnog iskustva.

Godine	28	25	43	36	21	32	50	33
Radno iskustvo	4	3.5	7	10	1	3	25	0

Na osnovu datih podataka, potrebno je nacrtati grafik i predvideti koliko godina radnog iskustva treba da ima osoba stara 27 godina.

Prvo je potrebno da se importuje klasa *LinearRegression* iz *sklearn.linear_model* i modul *matplotlib.pylab*.

Na sličan način kao u prethodnom zadataku se kreira model i ispisuje koeficijent.

```
In [9]: import numpy as np
        import matplotlib.pylab as plt
        from sklearn.linear_model import LinearRegression

        x = np.array([28,25,43,36,21,32,50,33]).reshape((-1,1))
        y = np.array([4,3.5,7,10,1,3,25,0])

        model = LinearRegression().fit(x,y)
        print("Koeficijent je: ", model.score(x,y))
```

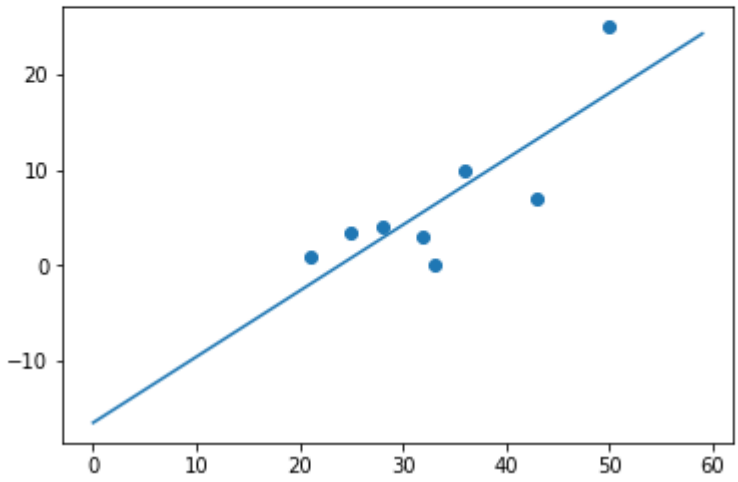
```
Koeficijent je:  0.6616571109866853
```

Na osnovu koeficijenta može da se uoči da je u pitanju slaba korelacija.

Zatim je potrebno iscrtati grafik pomoću funkcije scatter(). Na grafiku se takođe isctava regresiona linija pomoću funkcije plot(), tako sto joj se za argumente proslede tačke od 0 do 60 i njima odgovarajuće vrednosti koje se dobijaju predviđanjem.

```
In [11]: vec = np.arange(60).reshape((-1,1))
         plt.scatter(x,y)
         plt.plot(vec,model.predict(vec))
```

```
Out[11]: [<matplotlib.lines.Line2D at 0xfcbe9e8>]
```



Na grafiku takođe može da se uoči da veza nije jaka.

Nakon ovoga, potrebno je da se predvidi koliko radno iskustvo bi trebalo da ima osoba koja ima 27 godina. To se vrši na sledeći način:

```
In [13]: x_novo = np.array([[27]]).reshape((1,1))
         print("Predvidjena vrednost je: ", model.predict(x_novo))
```

```
Predvidjena vrednost je:  [2.1968254]
```

Dakle, osoba koja ima 27 godina, trebalo bi da ima malo više od 2 godine radnog iskustva.

3. U sledećoj tabeli se nalaze podaci od 10 žena, koliko imaju godina, koliko su teške (izraženo u kilogramima) i koliko para troše mesečno (izraženo u 1000 dinara):

Godine	18	20	21	23	25	30	38	39	42	57
Težina (kg)	52	63	85	60.5	65.2	61	87.1	73	69	92.1
Potrošnja para	0.5	0.75	1.2	2	0.5	5	5.1	4	6.2	5.3

Na osnovu datih podataka potrebno je predvideti koliko para mesečno troši na kozmetiku žena koja ima 28 godina i ima 89 kilograma. U ovom primeru je potrebno koristiti višestruku linearnu regresiju.

Predviđanje se vrši isto kao malo pre, s tim što sada niz x sadrži podatke i za godine i za težinu.

```
In [39]: import numpy as np
         from sklearn.linear_model import LinearRegression

         x = np.array([[18,52],[20,63],[21,85],[23,60.5],[25,65.2],
                       [30,61],[38,87.1],[39,73],[42,69],[57,92.1]])
         y = np.array([0.5,0.75,1.2,2,0.5,5,5.1,4,6.2,5.3])

         model = LinearRegression().fit(x,y)
         print("Koeficijent: ",model.score(x,y))
```

```
Koeficijent:  0.7153591829590137
```

Može se uočiti na osnovu koeficijenta da postoji linearna veza, pa ovaj model može da se koristi za predviđanje.

```
In [40]: print("Pristrasnost: ", model.intercept_)
         print("Nagib: ", model.coef_)
         x_novo = np.array([[28,89]])
         print("Ako neko ima 28 godina i 89 kilograma,")
         print("predvidja se da ce trositi: "+
               str(round(model.predict(x_novo)[0]*1000))+
               " dinara mesecno na sminku.")
```

```
Pristrasnost:  -0.2849014906613654
Nagib:  [ 0.17329535 -0.02944262]
Ako neko ima 28 godina i 89 kilograma,
predvidja se da ce trositi: 1947.0 dinara mesecno na sminku.
```

Iz razloga što su vrednosti izražene u hiljadama dinara, potrebno je da se pomnoži rezultat sa 1000 i time se dobije tačna vrednost koliko je dinara predviđeno. Zatim, pošto se dobija decimalna vrednost, koristi se funkcija round() da se zaokruži na celobrojnu vrednost, radi lepšeg rezultata.

Dakle, na osnovu predviđanja, može da se zaključi da ukoliko žena ima 28 godina i 89 kilograma, trošiće oko 1947 dinara mesečno na kozmetiku.

Zaključak

Elektronske lekcije o upotrebi biblioteke NumPy u programskom jeziku Pajton trebalo bi da olakšaju proces učenja svih bitnih koncepata vezanih za ovu biblioteku. Neke od bitnih stavki koje bi trebalo da se znaju pre početka pisanja samog programa uz pomoć ove biblioteke su: kada se ona koristi, kako da se instalira, koje okruženje se preporučuje za rad sa njom i koje su prednosti korišćenja n-dimenzionalnih nizova nad listama. Ove elektronske lekcije pružaju korisnicima odgovore na sva ta pitanja. Efikasnost u učenju uz pomoć ovih lekcija se dodatno postiže uz velik broj obrađenih praktičnih primera. Nakon pređenih svih lekcija, korisnici bi trebalo da budu u mogućnosti da samostalno napišu program.

Biblioteka NumPy omogućava korišćenje velikog broja operacija, pri čemu nije neophodno imati puno linija koda, što se upravo postiže velikim brojem metoda i funkcija koje ova biblioteka poseduje. U kombinaciji sa drugim bibliotekama, biblioteka NumPy predstavlja jako moćan alat za obradu statističkih podataka, zbog čega predstavlja jednu od najbitnijih biblioteka za analizu podataka. Mali deo analize podataka se upravo može uočiti u okviru ovih lekcija, kao i kombinacija biblioteke NumPy sa drugim modulima i bibliotekama.

Kao što se može uočiti u radu, elektronske lekcije su napisane na srpskom jeziku, što ih izdvaja od ostale literature, koja se najčešće može pronaći na engleskom jeziku. Sa posebnom pažnjom su lekcije obrađene metodički da budu prilagođene početnicima i da im omoguće osnovu za dalji razvoj. Pri izradi sajta su korišćene neupadljive i nežne boje i izrađen je na izuzetno jednostavan način, da ne bi korisniku bilo šta skretalo pažnju sa onoga što je bitno, a to je gradivo odrađeno u okviru lekcija.

Elektronske lekcije kreirane za potrebe ovog rada trebalo bi da korisniku omoguće da usvoji početna znanja i da stekne dobru osnovu o biblioteci NumPy. Sadržaj elektronskih lekcija pažljivo je odabran kako bi se njegovim savladavanjem omogućila dobra osnova i da bi se podstakla želja za daljim napredovanjem i učenjem biblioteka neophodnih za bavljenje analizom podataka. Nakon savladavanja ovih lekcija, sledeći korak za učenje analize podataka bi trebalo da bude učenje biblioteka Pandas, Matplotlib, Seaborn i Scipy.

Osim načina na koji su pisane elektronske lekcije, kao i jezika kojim su pisane, još jedna prednost ovih lekcija je što sadrže objašnjenje za instalaciju kako same biblioteke, tako i poznate platforme „Anakonda“.

Literatura

[1] Justin Johnson - *Python Numpy Tutorial*, http://cs231n.github.io/python-numpy-tutorial/#numpy-arrays;

[2] Zvanični sajt biblioteke NumPy programskog jezika Pajton, 2005-2020, http://www.numpy.org/;

[3] NumPy community - *NumPy User Guide*, 16. april 2018;

[4] Al Sweigart - *Uvod u Python, Automatizovanje dosadnih poslova, Praktično programiranje za početnike*, 2016;

[5] Računarski fakultet - *predavanje iz predmeta Programiranje*, 2003-2021, https://raf.edu.rs/citaliste/programiranje/4475-xa-sta-je-to-python-sve-sto-treba-znati-xa;

[6] Sajt platforme za interaktivno učenje biblioteke NumPy, 2021, www.tutorialspoint.com/numpy;

[7] Travis E. Oliphant - *Guide to NumPy*, Phd, 7. decembar 2006;

[8] Filip Schouwwnaars - sajt platforme za interaktivno učenje programskog jezika Pajton, https://www.learnpython.org/en/Modules_and_Packages;

[9] Sandeep J, Shikhar G, Dharmesh S, Shubham B. - sajt platforme za interaktivno učenje biblioteke NumPy, 2021. https://www.geeksforgeeks.org/numpy-in-python-set-1-introduction/;

[10] Liran B. H - *Linearna regresija sa bibliotekom NumPy*, 25. mart 2019, https://devarea.com/linear-regression-with-numpy/?fbclid=IwAR2U1n0STjydmlL3mXpa5-3XMFrQKvrt1bYZ_fSAe9sBK5ifEBvV3e4kFHk#.XO6XvBYzbIV;

[11] Cheathan Kumar - *Linearna regresija: tipovi, primeri, opadanje gradijenta*, 12. septembar 2018, https://towardsdatascience.com/linear-regression-part-1-types-examples-gradient-descent-example-2e8c22b05f61;

[12] Cheathan Kumar - *Linearna regresija: implementacija u Pajtonu*, 17. septembar 2018, https://towardsdatascience.com/implementation-linear-regression-in-python-in-5-minutes-from-scratch-f111c8cc5c99?fbclid=IwAR1cFrSaPaldqy7ejSP9JkGALx2INdPGYSI6G2j9TcPSjEEOB0CyBib7P3w;

[13] Zvanični sajt biblioteke SciPy, 2008-2021, https://docs.scipy.org/;

[14] Joe Gregg, Lawrence University - *predavanje iz predmeta Uvod u Pajton*, http://www2.lawrence.edu/fast/GREGGJ/Python/numpy/numpyLA.html?fbclid=IwAR32gRDcG_f1fv9sw2NJgg6xxGS70NFNiK9EJuvy6Bdz7U4dvPP4eV2UdTc;

[15] Mirko Stojiljković - *Linearna regresija u Pajtonu*, 2012-2021, https://realpython.com/linear-regression-in-python/;

[16] Zvanični sajt platforme sa grupom programa za analizu podataka, 2021, https://www.anaconda.com/distribution/;

[17] Travis E. Oliphant - *NumPy i ScuPy*, Tokio, 18. mart 2012. https://www.slideshare.net/shoheihido/sci-pyhistory;

[18] Krunal - *NumPy sortiranje niza, primeri*, 11. novembar 2020, https://appdividend.com/2019/01/31/numpy-arange-tutorial-with-example-python-numpy-functions/;

[19] Dan Bader - *Plitko i duboko kopiranje Pajton objekata*, 2012-2021, https://realpython.com/copying-python-objects/;

[20] Milan Popović, Fakultet Mef - *predavanje u okviru predmeta Osnove programiranja*, 2019, http://mef-lab.com;

[21] Bernd Klein – *Numerical Python Course*, 2011-2020, https://www.python-course.eu;

[22] Keith Galli – *Complete Python NumPy Tutorial*, 10. Jul 2019, https://www.youtube.com/watch?v=GB9ByFAIAH4, https://github.com/KeithGalli/NumPy;

[23] Okruženje za pisanje PAjton skripto na iPad-u ili iPhone-u , http://omz-software.com/pythonista/, http://omz-software.com/pythonista/numpy/reference/arrays.scalars.html;

[24] Sajt za interaktivno učenje principa za Developers/DevOps/Sysadmins, 2018, http://omz-software.com/pythonista/, https://enqueuezero.com/projects/numpy/scalar.html.