Matematički fakultet
Univerzitet u Beogradu

Tipovi i strukture podataka u programskom jeziku Pajton

- master rad -

mentor:
doc. dr Miroslav Marić

kandidat:
Milan Josifović

Beograd 2014.

Sadržaj

1 Uvod

Programiranje je veština izrade računarskih aplikacija koja predstavlja jedno od najperspektivnijih zanimanja i grana privrede u modernom društvu. U srpskoj prosveti se puno pažnje posvećuje nastavi programiranja, pa se postavlja pitanje koji programski jezik je najjednostavniji za razumevanje učenicima, kao i početnicima koji se prvi put susreću sa konceptom programiranja. Ne može se sa sigurnošću reći koji programski jezik bi bio najbolje rešenje za prve korake u programiranju. U većini srpskih škola se Paskal uči kao prvi programski jezik. Međutim, u svetu je tendencija prelazak na skript jezike, koji sadrže interpreter i čije su promenljive dinamičke, a programski jezik Pajton je jedan od takvih programskih jezika.

Cilj ovog rada je da prikaže sve prednosti programskog jezika Pajton u efikasnijem savladavanju osnova programiranja, kao i razloge zašto Pajton treba učiti kao prvi programski jezik. U tu svrhu biće prikazani tipovi i strukture podataka koje se pojavljuju u ovom jeziku, kao i funkcije koje olakšavaju rad sa istim.

1.1 Istorijat programskog jezika Pajton

Pajton je programski jezik visokog nivoa čija je primena raznovrsna. Pajton je nastao 1991. godine, a razvio ga je holandski programer Gvido van Rosum. Zbog načina kompilacije, Pajton se koristi kao skript jezik, kao i u implementiranju veb aplikacija. Program se prevodi koristeći interpreter, koji proverava sintaksne greške koda programa kojeg je programer napisao i izvršava liniju po liniju ako nije došlo do greške. Pored toga, programski jezik Pajton je objektno orijentisan jezik, svi tipovi podataka su objekti pa omogućava rad sa postojećim objektima i lako implementiranje novih [1]. Jednostavan je i minimalistički jezik, lak za učenje, a čitanje dobrog izvornog koda Pajton programa izgleda kao čitanje engleskog jezika [2].

Još jedna velika prednost programskoj jezika Pajton je prenosivost i kompatibilnost na svim operativnim sistemima. Pajton se može koristiti na Windows-u, Mekintošu, kao i na svim distribucijama Linux-a [3].

U programski jezik Pajton je ugrađeno mnoštvo tipova podataka (npr. liste, rečnici i stringovi), funkcija, kao i nekoliko konstanti koji čine standradnu biblioteku. Pored standardne biblioteke, Pajton raspolaže i mnoštvom modula među kojima su moduli koji omogučavaju lakši rad sa regularnim izrazima, bazama podataka, nasumičnim brojevima, itd.

Pajton je ugradiv u druge programske jezike, što znači da možete programe napisane u programskom jeziku Pajton lako dodati u programe napisane u C/C++ programskim jezicima [4].

1.2 Prednosti i mane programskog jezika Pajton

Važan aspekt pri programiranju u Pajtonu su beline, koje zamenjuju blokove (zagrade { } koje se koriste u C-olikim jezicima). Ovom sintaksom se dosta smanjuje količina koda. Beline se koriste za označavanje gde blok počinje i gde se završava [5]. Na ovaj način se utvrđuje kojem bloku pripada linija koda.

Dosta polemike se vodi o tome da li je programski jezik Pajton pravi jezik za učenike koji se prvi put susreću sa programiranjem. U ovom delu rada biće prikazane neke od prednosti i mana programskog jezika Pajton. Pajton je besplatan program koji pruža jednostavan rad sa objektima, modulima, pritom je interaktivan, portabilan i ima jednostavnu integraciju sa ostalim programskim jezicima kao što su C i Java. Sintaksa čini programe napisane u Pajtonu čitljivim i za programere koji nisu radili na originalnom projektu. Takođe, programski jezik Pajton je čitljivi od programskog jezika C ili Jave, ali ipak moramo napomenuti da i Pajton ima svoje mane i zato Pajton nije na glavnoj sceni među programskim jezicima. Mali broj programera, u poređenju sa ostalim programskim jezicima, se bavi Pajtonom.

Štampanih izdanja koja se tiču programskog jezika Pajton je malo u poređenju sa programskim jezikom Java, iako je Java nastala kasnije. U razvoju aplikacija koje se tiču komunikacija, kalendara i instant poruka programski jezik Pajton je u zaostatku u odnosu na programski jezik PHP, jer PHP sadrži dosta biblioteka sa kojima je kreiranje ovakvih aplikacija dosta brže i jednostavnije. Rad sa datotekama u programskom jeziku Pajtonu je takvo da je dosta primitivniji u odnosu na konkurentske programske jezike [6]. Međutim, prikazaćemo jednostavan primer programa koji stampa poruku na ekranu zapisan u programskom jeziku C i Pajtonu:

```
#include <stdio.h>

int main(){
    print("Zdravo svete");
    return 0;
}
```

```
print("Zdravo svete")
```

Prethodnim programom prikazan je kod koji je neophodan da se ispiše poruka ,,Zdravo svete" u programskom jeziku C i Pajtonu. Program napisan u programskom jeziku C je dosta kompleskniji, dok se u Pajtonu ispisuje u jednom redu. U programskom jeziku Java je kod još duži i iz tog razloga neće biti prikazan.

Nakon svega navedenog i pored svih nedostataka koje programski jezik Pajton ima, ovaj programski jezik se čini kao pravi izbor u proučavanju načina

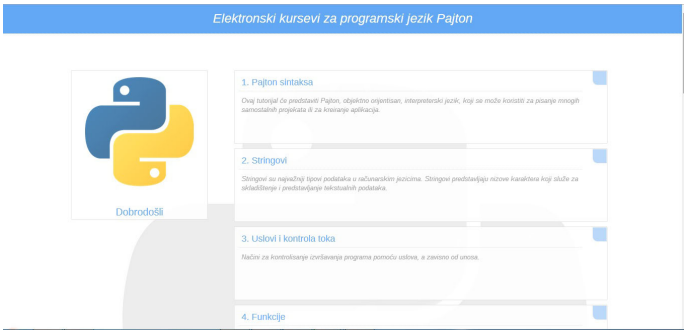
razmišaljanja u programiranju. Način pisanja koda, dinamičko deklarisanje promenljivih, kao i složeniji tipovi podataka koji se lako koriste u Pajtonu čine ga pravim programskim jezikom za učenje.

1.3 Elektronski kurs za programski jezik Pajton

Materijali prikazani u ovom radu se nalaze i u elektronskom kursu koji je javno dostupan na adresi

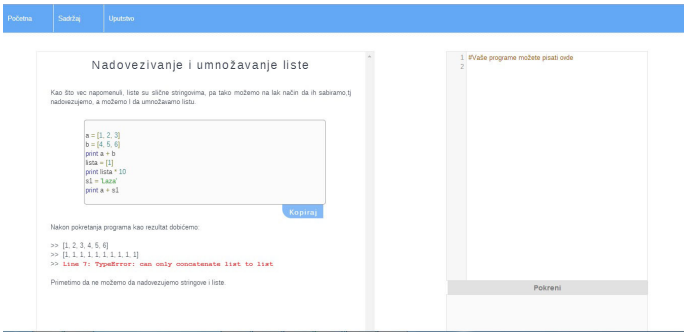
http://edusoft.math.rs/python/

Nakon pokretanja ove veb adrese biće prikazana naslovna strana ,,Elekronskih kurseva za programski jezik Pajton" i spisak tematskih celina, koje su obrađene uz mnoštvo slika i primera.



Slika 1. *Uvodna strana elektronskog kursa*

Ovakvi materijali su od velike koristi za učenike koji tek počinju da se bave programiranjem jer svojom sažetošću pokrivaju osnove programiranja. Materijali su podeljeni u trinaest tematskih celina, a svaka od njih sadrži veći broj podtema zbog lakše preglednosti i navigacije kroz sadržaj. Na Slici 1. prikazana je uvodna strana elektronskog kursa.



Slika 2. *Naslovna strana elektronskog kursa*

Klikom na određenu temu, prikazuje strana sa materijalima (Slika 2.). U gornjem levom uglu postoji dugme ,,Sadržaj" koje služi za bržu i lakšu navigaciju kroz materijale, kao i dugme ,,Uputstvo" koje prikazuje kako da preuzmete i instalirate Pajton i na svom računaru. Na Slici 3. je prikazano uputstvo za preuzimanje i instalaciju programskog jezika Pajton.

Slika 3. *Uputstvo za upotrebu elektronskog kursa*

Glavni deo strane je podeljen na dva dela tako što su tekstualni materijali smešteni sa leve strane dok je konzola za pisanje programa sa desne strane. U ovoj konzoli moguće je izvršiti većinu programa koji će biti prikazani u ovom radu što pokriva i osnove programiranja u programskom jeziku Pajton.

1.3.1 Sadržaj elektronskog kursa

Materijali koji se nalaze u ovom elektronskom kursu namenjeni su korisnicima koji se prvi put susreću sa konceptom programiranja i pokrivaju osnove programiranja. Sledeće teme su obrađene u ovom elektronskom kursu.

1. **Pajton sintaksa**: Sadrži kratak istorijat i funkcionalnosti programskog jezika, opis nekih primitivnih tipova (celi i realni brojevi, logički tip), kao i operacije, naredbe i neke funkcije nad njima.

2. **Stringovi**: U ovoj celini je dat opis stringova, kao i funkcije i metode za rad sa istim.

3. **Uslovi i kontrola toka**: Korisnik se u ovom delu upoznaje sa operatorima poređenja i logičkim operatorima, kao i sa naredbama grananja.

4. **Funkcije**: U ovom delu prikazane su neke ugrađene funkcije, a data je i konstrukcija za definisanje novih.

5. **Liste**: Korisnik se u ovom delu upoznaje sa listama kao strukturom podataka. Prikazane su i funkcije i metode koje olakšavaju rad sa listama.

6. **Torke i rečnici**: U ovoj celini, korisnik se upoznaje sa torkama, nepromenljivim tipom podataka, kao i sa rečnicima, koji podsećaju na asocijativne nizove.

7. **Petlje**: Uz mnošvo primera su prikazane petlje koje postoje u programskom jeziku Pajton.

8. **Liste i funkcije**: Detaljno je prikazano slanje listi kao argumenta funkcije, kao i još neke funkcije za rad sa listama.

9. **Napredne teme u Pajtonu**: Najviše pažnje je posvećeno naprednom radu sa listama, raznim vrstama isecanja listi, kao i skupovnim listama.

10. **Bitovski operatori**: Objašnjen je zapis brojeva zapisanih u binarnoj osnovi, kao i operatori koji rade sa bitovima.

11. **Klase**: Pajton, kao objektno orijentisan jezik. Kreiranje sopstvenih klasa. Nasleđivanje. Polimorfizam.

12. **Rad sa datotekama**: Primena datoteka za upisivanje i ispisavanje podataka iz iste.

13. **Rekurzija**: Opisan koncept rekurzije, u kojoj funkcija poziva samu sebe. Dobre i loše strane rekurzije. Primeri rekurzivnih funkcija.

1.4 Verzije programskog jezika Pajton

Pajton verzija 3.0 je objavljena 2008. godine. Finalna verzija 2.x Pajtona, verzija 2.7 je objavljena sredinom 2010. godine, pa je prirodno pitanje koju verziju koristiti. Verzija 2.x je završena i neće se izbacivati nove verzije [7]. Zato je predlog da se koristi verzija 3, i u ovom radu svi progami će biti pisani u verziji 3 programskog jezika Pajton [8].

Nekoliko bitnih stvari su promenjenje radi lakšeg korišćenja. U Tabeli 1. prikazane su neke razlike između verzija 2 i 3.

Pajton 2	**Pajton 3**
`print x`	`print(x)`
$4/3 = 1$	$4/3 = 1.3333$ $4//3 = 1$
`raw_input()`	`input()`
`file('dokument.txt')`	`open('dokument.txt')`

Tabela 1. *Razlike između verzija 2 i 3 programskog jezika Pajton.*

Velika razlika postoji i u kodiranju, verzija 3 koristi UNICODE kodiranje za predstavljanje svih tipova podataka [9].

1.5 Sadržaj rada

Najviše pažnje u ovom radu je posvećeno tipovima i strukturama u programskom jeziku Pajton. Pored toga, objašnjeni su neki operatori, funkcije i metode koje olakšavaju rad u ovom programskom jeziku. Svrha rada je da privuče pažnju čitalaca i da ih usmeri na programski jezik Pajton. Ovaj rad je podeljen na osam celina:

- **Promenljive i tipovi podataka**: U ovom delu rada su objašnjenje promenljive, tipovi promenljivih i imena promenljivih. Opis operatora u programskom jeziku Pajton, kao i funkcije `print()` i `input()`.
- **Brojevi**: Tipovi brojeva u programskom jeziku Pajton su prikazani u ovom delu, kao i zapis i ograničenja koja imaju. Detaljno su opisani celi, realni i kompleksni brojevi kao i operatori koji rade sa njima.
- **Stringovi**: U ovoj celini su opisani stringovi. Pored osnovnih osobina stringova prikazani su operatori nad stringovima, kao mnoge funkcije i metode za rad sa istim.

- **Liste**: Osnovne karakteristike i osobine lista su prikazane u ovom delu rada. Prikazana je upotreba listi kao stekova i ugnježdenih listi. Metode i funkcije za rad sa listama, kao i skupovne liste su takođe prikazane u ovom delu.

- **Torke**: Koncept torki kao nepromenljivog niz objekata je prikazan u ovom delu. Dodeljivanje vrednosti promenljivim korišćenjem torki i upotreba torki kao povratnih vrednosti funkcija su detaljno objašnjene u ovom delu.

- **Rečnici**: U ovom delu rada su prikazani rečnici, kolekcije slične listama koje se indeksiraju ključevima. Operacije nad rečnicima i načini iteracija kroz rečnike su prikazani u ovom delu.

- **Skupovi**: U ovom delu su prikazani skupovi kao neuređene kolekcije elemenata u kojima je svaki element jedinstven [10]. Razlike između skupova (`set`) i zamrznutih skupova (`frozenset`) su objašnjene u ovom delu rada, kao i operatori za rad sa skupovima.

- **Iteratori i Generatori**: Koncept i upotreba iteratora, kao i implementacija sopstvenih klasa koji su iterabilne su objašnjene u ovom delu rada. Upotreba generator funkcija korišćenjem ključne reči `yield` i metode za rad sa generatorima su prikazane u ovom delu.

2 Promenljive i tipovi podataka

Promenljive su mesta u memoriji u kojima podaci mogu da se čuvaju [11]. One imaju `tip, ime` i `vrednost`. Vrednost promenljive može da se menja za vreme izvršavanja programa. U Pajtonu, promeljiva ne mora da se deklariše, tj. ne mora da se striktno navede kog je tipa. Pajton kompajler sam zaključuje o kom tipu je reč. Imena promenljivih mogu da sadrže karaktere a-z, A-Z, 0-9, ne smeju sadržati blanko i moraju da počinju sa slovom ili donjom crtom (_). Pajton je ,,case-sensitive" što znači da pravi razliku između velikih i malih slova. Na primer, promenljive `ime` i `Ime` neće biti iste. Takođe postoji spisak ključnih reči koje se ne smeju koristiti za imena promenljivih. Pajton sadži 31 ključnu reč, spisak ključnih reči se nalazi u Tabeli 2.

and	del	global	not	while
as	elif	if	or	with
assert	else	import	pass	yield
break	except	in	print	
class	finally	is	raise	
continue	for	lambda	return	
def	from	nonlocal	try	

Tabela 2. *Spisak ključnih reči.*

2.1 Operatori

Operatori su specijalni simboli koji imaju značenje u programskom jeziku i izvršavaju specifične ranje nad podacima. U programskom jeziku Pajton postoje aritmetički, logički i bitovski operatori. Neki od aritmetičkih operatora koji se pojavljuju u programskom jeziku Pajton su:

+	−	*	**	/	//	%
<<	>>	&	|	^	~	
<	>	<=	>=	==	!=	

Tabela 3. *Operatori u programskom jeziku Pajton.*

U prvom redu su prikazani standardni matematički operatori sabiranja, oduzimanja, množenja i deljenja, operator stepenovanja, celobrojnog deljenja i ostatka pri deljenju. U drugom redu su prikazani bitovski operatori, bitovsko pomeranje u levo i desno, bitovsko `i`, bitovsko `ili`, ekskluzivno `ili` i negacija. U poslednjem redu su prikazani operatori za poredjenje: manje, veće, manje ili jednako, veće ili jednako, jednako i različito.

Postoje i logički operatori, za rad sa logičkim tipom.

and	or	is	not	in

Pored ovih, postoje i tačka operator (.), operator indeksiranja `[]`, i operator za poziv metoda i funkcija - zagrade ( ). U sledećoj tabeli dat je prikaz svih operatora u programskom jeziku Pajton po prioritetu, od najmanjeg do najvećeg prioriteta.

Operator	**Opis operatora**
`lambda`	Lambda izraz
`or`	Logičko `ili`
`and`	Logičko `i`
`not x`	Logičko ne
`in, not in`	Test pripadnosti
`is, is not`	Test identiteta
<,<=,>,>=,! =,==	Poređenje
—	Bitovsko `ili`
^	Bitovsko ekskluzivno `ili`
&	Bitovsko `i`
<<, >>	Bitovska pomeranja
+, −	Sabiranje i oduzimanje
∗, /, //, %	Množenje, deljenje, celobrojno deljenje, ostatak
`+x,-x`	Pozitivan, negativan
∗∗	Stepenovanje
`x._`	Referenciranje atributa
`x[index]`	Indeksiranje
`x[index:index]`	Seckanje
`f(argumenti)`	Poziv funkcije
`(izrazi ...)`	Torke
[`izrazi ...`]	Liste
{`ključ:vrednost ...`}	Rečnici
`'izrazi ...'`	Stringovi

Tabela 4. *Spisak svih operatora po prioritetu.*

2.2 Funkcija `print()`

Funkcija `print()` se koristi za štampanje. Sve do verzije 3 programskog jezika Pajton, `print()` je bio iskaz [12]. Svi argumenti koji nisu ključne reči se konvertuju u stringove i ispisuju se na izlaz. Moguće je štampati na standardni izlaz (`sys.stdout`), standardni izlaz za greške (`sys.stderr`) i pisati u datoteku (`sys.stdin`). Separator (`sep`) služi za razdvajanje argumenata određenim simbolom. Na primer, ukoliko je potrebno ispisati tri imena i razdvojiti ih crticom, onda bi `sep` bio postavljen na simbol `-`. Argumentom `end` se definiše čime se završava ispis nakon funkcije `print()`, ukoliko se ovaj argument izostavi nakon funkcije `print()` će se preći u novi red. Argumenti `sep` i `end` moraju biti tipa string.

Primer 1 *Primer prikazuje upotrebu funkcije* ***`print()`*** *za stampanje:*

```
>>> print("Vas odgovor je :" , 5)
Vas odgovor je : 5
>>> print(5, end=", ")
5, # Dodaje zarez i blanko pored broja 5
>>> print("Fatal error",file=sys.stderr)
Fatal error
>>> print("Luiz","Marcelo","Silva", sep='-')
Luiz-Marcelo-Silva
```

2.3 Funkcija `input()`

Većina programa zahteva komunikaciju sa korisnikom, odnosno da korisnik unese neke podatke, a nakon toga program radi sa unešenim podacima. Kao kod funkcije `print()`, koja je radila sa standardnim izlazima, funkcija `input()` radi sa standardnim ulazom (`sys.stdin`), koji je može biti tastatura ili neki tekstualni dokument. Ova funkcija zahteva od korisnika da unese podatak - uglavnom sa tastature. Neophodno je promenljivoj dodeliti prosleđeni podatak. U zavisnosti od toga koji podatak je prosleđen, programski jezik Pajton će odlučiti da li je u pitanju broj, string, itd. Kao argument funkcije moguće je proslediti string koji će biti prikazan pre zahteva za unos podataka.

Primer 2 *Primer prikazuje upotrebu funkcije* ***`input()`*** *za unos podataka:*

```
>>> sifra = input('Unesite svoju sifru')
>>> print(sifra)
```

3 Brojevi

Programski jezik Pajton podržava rad sa tri tipa brojeva, celi brojevi (`int`), realni brojevi (`float`) i kompleksni brojevi (`complex`). Pri deklaraciji promenljivih ne mora da se vodi računa o tipu promenljive, jer to Pajton radi automatski.

3.1 Celi brojevi

U programskom jeziku Pajton celi brojevi su prikazani kao stringovi decimalnih brojeva, tako da ne postoji ograničenje za minimalan i maksimalan broj [10]. Broj ne sme da sadrži interpunkcijske znake i ne sme početi cifrom nula (0). Nula je rezervisana za prikaz binarnih, oktalnih i heksadekadnih brojeva.

Binarni brojevi u programskom jeziku Pajton počinju sa ,,0b" ili ,,0B", nakon čega slede cifre 0 i 1. Oktalni brojevi u verziji 3 programskog jezika Pajton počinju cifrom 0 i slovom ,,o", tj. ,,0o" nakon čega slede cifre od 0 do 7. Heksadekadne cifre imaju osnovu 16, i koriste cifre od 0 do 9 plus mala ili velika slova {a, A, b, B, c, C, d, D, e, E, f, F}.

3.2 Realni brojevi

Brojevi sa pokretnim zarezom u programskom jeziku Pajtonu su ekvivalentni tipu `double` u programskom jeziku C. Za prikaz broja obično koriste 64 bita. Postoje dva načina zapisa realnih brojeva. Prvi način predstavlja niz cifara odvojenih zarezom, odnosno tačkom, i drugi, kompleksniji koji sadrži eksponent (E).

```
.0625
0.0625
6.25E-2
625E-4
```

Poslednja dva predstavljaju broj zapisan u osnovi i eksponentu. Karakter ,,E" predstavlja broj 10 posle kojeg sledi stepen, pa zapravo poslednja dva broja predstavljaju: 6.25×10^{-2} i 625×10^{-4}. Generalno, osnova bi trebala da bude broj između 0 i 10, tako da poslednji broj nije pravilno zapisan.

3.3 Komplekski brojevi

Pored celih i realnih brojeva, u programskom jeziku Pajtonu moguće je raditi i sa kompleksnim brojevima. Za kompleksnu jedinicu koristi se slovo

,,j" ili ,,J". Kompleksni broj se sastoji od realne i imaginarne jedinice $a + jb$ gde su brojevi a i b realni brojevi.

3.4 Konverzija brojeva

Postoji nekoliko ugrađenih funkcija koje konvertuju broj iz jednog tipa u drugi. Treba obratiti pažnju kako konvertovati brojeve jer lako može doći do greške prilikom konvertovanja zbog smanjenja ili povećanja broja bitova za prikaz broja.

3.4.1 Funkcija `int()`

Funkcija `int()` konvertuje prosleđeni argument u ceo broj. Ukoliko je prosleđeni broj realni, cifre posle zareza će biti uklonjene. Kompleksne brojeve nije moguće konvertovati u cele korišćenjem ove funkcije. Takođe, ako je prosleđeni argument string koji sadrži samo brojeve i opciono počinje znakom `+` ili `-`, funkcija `int()` će konvertovati ovakav string.

Primer 3 *Primer prikazuje konvertovanje u cele brojeve korišćenjem funkcije* ***`int()`****:*

```
>>> int(5.4521)
5
>>> int('1234')
1234
>>> int(1 + 2j)
TypeError: can't convert complex to int
```

3.4.2 Funkcija `float()`

Funkcija `float()` konvertuje prosleđeni argument u realni broj. Ukoliko je funkciji `float()` prosleđen ceo broj, funkcija će kreirati decimalni zarez i dodati cifru 0 iza zareza, tako da se dobije zapis realnog broja. Kao i funkcija koja konvertuje u cele brojeve, ni ova funkcija ne može konvertovati kompleksne brojeve direktno u realne. Stringovi će biti konvertovani ukoliko sadrže samo cifre, opciono počinju sa znakom `+` ili `-`. Decimalna tačka i eksponent e takođe mogu da se nađu u zapisu broja.

Primer 4 *Primer prikazuje konvertovanje u realne brojeve korišćenjem funkcije* ***`float()`****:*

```
>>> float(23)
23.0
```

```
>>> float("6.02E24")
6.02e+24
```

3.4.3 Funkcija `complex()`

Funkciji `complex()` se prosleđuju dva argumenta. Prvi argument predstavlja realni deo, a drugi imaginarni deo kompleksnog broja. Moguće je izostaviti drugi argument, u tom slučaju imaginarni deo se postavlja na 0.0. Takođe, moguće je proslediti i string.

Primer 5 *Primer prikazuje konvertovanje u kompleksne brojeve korišćenjem funkcije* `complex()`*:*

```
>>> complex(4,3)
(4+3j)
>>> complex(4)
(4+0j)
>>> complex("3+4j")
(3+4j)
```

3.5 Moduli `math` i `cmath`

Kao i u drugim programskim jezicima i u programskom jeziku Pajton postoji mnogo ugrađenih funkcija koje olašavaju pisanje programa. U ovom delu biće opisane neke funkcije koje se nalaze u modulima `math` i `cmath` i olakšavaju rad sa matematičkim sadržajem. Modul `math` sadrži iste matematičke funkcije kao i istoimeni modul u progamskom jeziku C [13]. Ove funkcije ne mogu biti korišćene na kompleksnim brojevima, za rad sa funkcijama koje koriste kompleksne brojeve neophodno je uključiti modul `cmath`. Neke od funkcija koje se nalaze u ovom modulu će ovde biti objašnjene.

- `fabs()`: Prima jedan broj i vraća apsolutnu vrednost.
- `ceil()`: Prima jedan broj i vraća najmanji ceo broj veći ili jednak od proslednjenog argumenta.
- `exp()`: Prima jedan broj i računa e^x.
- `log()`: Prima dve vrednosti i računa logaritam od x za osnovu koja je prosleđena kao drugi argument. Ukoliko drugi argument nije prosleđen, osnova je broj e. Takođe postoje i funkcije `log1p()`,`log10()` i `log2()`.
- `sin(), cos(), tan(), asin(), acos(), atan()`: Trigonometrijske i inverzne trigonometrijske funkcije.

- `degrees() i radians()`: Konverzija ugla iz radijana u stepene i obrnuto.

- `sinh(), cosh(), tanh(), asinh(), acosh(), atanh()`: Hiperboličke i inverzne hiperboličke funkcije

Od konstanti koje se nalaze u ovom modulu postoje konstanta `pi` ekvivalenta matematičkoj konstanti π zaokružena na 15 decimala i broj `e` ekvivalentan matematičkoj konstanti takođe zaokružen na 15 decimala.

Modul `cmath` sadrži matematičke funkcije za rad sa kompleksnim brojevima [14]. Ove funkcije takođe prihvataju cele i realne brojeve.

Kompleksni broj z u Pajtonu se kao i u matematici predstavlja preko Dekartovih koordinata i određen je preko svog realnog dela `z.real` i svog imaginarnog dela `z.imag`. Drugim rečima:

```
z = z.real + z.imag*1j
```

Drugi način za prikaz kompleksnih brojeva je preko polarnih koordinata. Kompleksan broj z je definisan preko rastojanja od koordinatnog početka r i ugla ϕ koji se meri od pozitivnog dela x ose u pravcu suprotnom od kazaljke na satu a izražen je u radijanima. Sledeće funkcije konvertuju zapis kompleksnog broja z u polarne koordinate i obrnuto.

- `polar()`: Prima kompleksan broj zapisan preko Dekartovih koordinata i vraća reprezentaciju ovog broja u polarnim koordinatama, tj. vraća par (r, ϕ).

  Primer 6 *Primer prikazuje funkciju* ***`polar()`*** *koja prikazuje zapis kompleksnog broja preko polarnih koordinata.*

  ```
  >>> z = 1j
  >>> polar(z)
  (1.0, 1.5707963267948966)
  ```

- `rect()`: Prima dva argumenta, poluprečnik i ugao i vraća kompleksan broj z koji je jednak $z = |r| \times (\cos\phi + j\sin\phi)$

  Primer 7 *Primer prikazuje prebacivanje kompleksnog broja iz polarnih koordinata u pravougaone koordinate:*

  ```
  >>> r = 2
  >>> phi = radians(60)
  >>> rect(r,phi)
  (1.0000000000000002+1.7320508075688772j)
  ```

Pored ovih, u modulu `cmath` postoji i veći deo funkcija koje postoje u modulu `math`, a koje se tiču eksponencijalnih, logaritamskih, trigonometrijskih i hiperboličkih funkcija.

4 Stringovi

Stringovi su niz karaktera koji služe za skladištenje i predstavljanje tekstualnih podataka. Stringovi su nepromenljivi tipovi podataka, što znači da kada promenljivoj tipa string dodelite neku vrednost, ta vrednost se više ne može promeniti. U situacijama kada je neophodno promeniti vrednost stringa pravi se kopija sa željenim promenama.

U programskom jeziku Pajton postoje tri načina za deklarisanje stringa. Dozvoljeno je koristiti jednostruke, dvostruke i trostruke navodnike. Ne postoji razlika između jednostrukih i dvostrukih navodnika. U slučaju trostrukih navodnika, string se može deklarisati u više redova.

Primer 8 *Primer prikazuje tri načina za deklarisanje stringa:*

```
>>> ime = 'Milan'
>>> grad = "Novi Sad"
>>> strofa = """Treci gol je dao Djole,
a cetvrti potom Sima,
samo sto je pogresio
dao nama a ne njima."""
```

Ponekad je potrebno u zapisu stringa zapisati jednostruke ili dvostruke navodnike. Prvi način je koristiti dvostuke navodnike za stringove u slučaju kada su u tekstu potrebni jednostruki i obrnuto. Drugi način za prikaz navodnika je korišćenjem obrnute kose crte (\), tj \' zapravo kreira jednostruke navodnike ', a \" kreira dvostruke navodnike.

Primer 9 *Primer prikazuje način za pisanje jednostrukih i dvostrukih navodnika:*

```
>>> print("I'm a little lion.")
I'm a little lion.
>>> print('He said:"Stop now!!!"')
He said:"Stop now!!!"
>>> print("He said:\"I\'m little lion.\"")
He said:"I'm little lion."
```

Operator \ moguće je upotrebiti i za kreiranje vertikalnog taba (4 blanko znaka), korišćenjem naredbe ,,\t", kao i za prelazak u novi red ,,\n".

4.1 Poređenje stringova

Dva stringa su jednaka ako i samo ako su istog sadržaja, što znači da moraju da budu iste dužine i na istim pozicijama sadrže iste karaktere. Operator koji proverava da li su dva stringa jednaka je `==`, a operator koji proverava da li su dva stringa različita je `!=`. Kao povratna vrednost izvršavanja ova dva operatora dobijamo logičke vrednosti `True` ili `False`. U mnogim programskim jezicima, operator `==` poredi da li dva objekta zauzimaju iste pozicije u memoriji. U programskom jeziku Pajton, operator `is` poredi da li dva objekta, u ovom slučaju stringa, zauzimaju istu lokaciju u memoriji.

Primer 10 *Primer prikazuje operatore za poređenje stringova:*

```
>>> sport1 = "fudbal"
>>> sport2 = "kosarka"
>>> sport1 == sport2
False
>>> sport1 != sport2
True
```

4.2 Pristup elementima stringa

Pristupanje individualnim elementima stringa vrši se navođenjem celog broja koji predstavlja njegovu poziciju u stringu između uglastih zagrada `[]`. Pozicija prvog karaktera u stringu je `0`, a pozicija n-tog karaktera u stringu je `n-1`. U progamskom jeziku ne postoji tip podataka ,,char", svi karakteri su tipa string. Takođe, pokušaj pristupanja elementu koji ne postoji dovodi do greske `IndexError: string index out of range`

Primer 11 *Primer prikazuje pristupanje elementima stringa:*

```
>>> klub = "Napredak"
>>> print(klub[0])
N
>>> print(klub[7])
k
>>> print(klub[8])
IndexError: string index out of range
```

U programskom jeziku Pajton, moguće je pristupati elementima niza unazad, korišćenjem negativnih brojeva. Indeks poslednjeg karaktera u stringu je `-1`, pretposlednjeg `-2` itd.

Primer 12 *Primer prikazuje pristupanje elementima stringa korišćenjem negativnih indeksa:*

```
>>> klub = "Crvena Zvezda"
>>> print(klub[-1])
a
>>> print(klub[-2])
d
```

4.3 Isecanje stringa

Kao što možemo da izdvojimo jedan karakter stringa, moguće je izdvojiti više karaktera. Ako je `s` string, onda naredbom `s[a:b]` izdvajamo podstring koji počinje sa `s[a]`, a završava se sa `s[b-1]`. Naravno, i ovde je moguće koristiti negativne brojeve za indekse.

Primer 13 *Primer isecanja stringa:*

```
>>> x = "NesCafe"
>>> print(x[1:4])
esC
>>> print(x[-4:-1])
Caf
```

Takođe, moguće je izostaviti neke od argumenata isecanja. Izostavljanjem indeksa `a` iz opšteg oblika `s[a:b]` dobićemo string od prvog karaktera. Slično se dešava ako izostavimo drugi argument.

Primer 14 *Primer prikazuje isecanje stringa izostavljajući neke argumente:*

```
>>> y = "CocaCola"
>>> print(y[:4])
Coca
>>> print(y[4:])
Cola
```

Postoji još jedan način za isecanje stringa. Moguće je zadati treći argument. Treći argument označava korak, odnosno koliko karaktera preskačemo.

Primer 15 *Primer prikazuje isecanje stringa korišćenjem koraka:*

```
>>> predmet = "matematika"
>>> print(predmet[1:7:2])
aea
>>> stih = 'Tatatatira'
>>> print(stih[::2])
tttr
```

4.4 Nadovezivanje i umnožavanje stringova

Generalno govoreći, ne mogu se primenjivati sve matematičke operacije na stringovima. Na primer, ne mogu se oduzimati ili deliti stringovi operatorima `-`, odnosno `/`. Međutim, stringovi se mogu sabirati, odnosno spajati. Operator `+` nadovezuje stringove, tako što drugi operand nalepi na kraj prvog.

Primer 16 *Primer prikazuje nadovezivanje stringa korišćenjem operatora `+`:*

```
>>> print('Ingrid ' + 'je ' + 'lepa!')
Ingrid je lepa!
```

Operator `*` takođe radi sa stringovima. Ovim operatorom se umnožava string. Naravno, ako je jedan operand string, drugi mora biti ceo broj koji predstavlja koliko puta želimo da umnožimo dati string.

Primer 17 *Primer prikazuje umnožavanje stringa korišćenjem operatora `*`:*

```
>>> r1 = "Potreban je samo "
>>> r2 = "rad "
>>> print(r1 + r2*3 + '.')
Potreban je samo rad rad rad.
```

4.5 Metode za rad sa stringovima

Postoji mnoštvo metoda i ugrađenih funkcija za rad sa stringovima. Programski jezik Pajton obezbeđuje dva stila za formatiranje ispisa, prvi je visoko fleksibilan, koristi metodu `format()`, a drugi baziran na C-ovskom `printf`-u. Kod oba tipa formatiranja koncept je isti, potrebno je vrednost koja nije string ubaciti u tekst na određenim pozicijama u stringu. Metoda `format()` se koristi tako što se u vitičastim zagradama {} navede redni broj, a zatim se kao argument metode navedu vrednosti koje će ih zameniti u tekstu. Na taj način će vrednost {1} u tekstu biti zamenjena prvim prosleđenim argumentom, vrednost {2} drugim prosleđenim argumentom itd.

Primer 18 *Primer prikazuje načine za formatiranje stringa korišćenjem metode `format()`:*

```
>>> 'Zovem se {0}, a prezivam se {1}.'.format('Stefan','Nemanja')
'Zovem se Stefan, a prezivam se Nemanja.'
>>> 'Zovem se {1}, a prezivam se {0}.'.format('Stefan','Nemanja')
'Zovem se Nemanja, a prezivam se Stefan.'
```

Ukoliko se argument u vitičastim zagradama izostavi, metod `format()` će redom koristiti argumente i ubacivati u string. Moguće je postaviti koji tip argumenta se očekuje, ako se u vitičastoj zagradi posle dvotačke (`:`) zapiše `d` za decimalni broj, `s` za string, `f` za realne brojeve. Pri ovakvom prikazu realnih brojeva moguće je i odrediti koliko cifara posle decimalne tačke će biti prikazano.

Primer 19 *Primer prikazuje napredniji način za formatiranje stringa:*

```
>>> cena = 52.95
>>> proizvod = 'Hleb'
>>> 'Cena za proizvod {0:s} je: {1:.1f}'.format(proizvod,cena)
'Cena za proizvod Hleb je: 53.0'
```

C-ovsko formatiranje je nešto složenije [15]. Zahteva oznaku `%` posle koje sledi slovo (`s` za string, `d` za ceo broj, `f` za realni broj) na poziciji gde želimo da ubacimo određenu vrednost. Očekivane vrednosti se navode posle oznake `%` u zagradi i to u istom redosledu kao i u stringu. I u ovom slučaju je moguće odrediti broj decimala realnog broja.

Primer 20 *Primer prikazuje formatiranje stringa na „C-ovski" način:*

```
>>> ime = 'LeBron'
>>> prezime = 'James'
>>> "Zovem se \%s, a prezivam se \%s. " \% (ime,prezime)
```

U sledećoj tabeli su navedene sve ugrađene funkcije i metode za rad sa stringovima.

`capitalize`	`format`	`isnumeric`	`lstrip`	`split`
`casefold`	`format_map`	`isprintable`	`maketrans`	`splitlines`
`center`	`index`	`isspace`	`partition`	`startswith`
`count`	`isalnum`	`istitle`	`replace`	`strip`
`encode`	`isalpha`	`isupper`	`rfind`	`swapcase`
`endswith`	`isdecimal`	`join`	`rindex`	`title`
`expandtabs`	`isidentifier`	`ljust`	`rjust`	`translate`
`find`	`islower`	`lower`	`rstrip`	`upper`
				`zfill`

U ovoj tabeli su nabrojane samo one metode i funkcije koje rade isključivo sa stringovi. Postoje i funkcije koje rade i sa drugim tipovima podataka, kao što je funkcija `len()` čija je povratna vrednost dužina objekta, u slučaju stringa, vraća broj karaktera koji sadrži prosleđeni string.

Primer 21 *Primer prikazuje funkciju* `len()` *koja meri dužinu stringa:*

```
>>> igrac = "Paul George"
>>> print(len(igrac))
11
```

4.5.1 Metode `is*`

Metode koje počinju prefiksom `is` (`isalnum()`, `isalpha()`, `isdigit()`, `isidentifier()`, `islower()`, `isupper()`, `isspace()` i `istitle()`), zahtevaju da dužina prosleđenog stringa bude najmanje 1, u suprotnom povratna vrednost ovakvih metoda je `False`.

- `isalnum()` vraća `True` ako se string u potpunosti sastoji od alfabetskih i/ili numeričkih karaktera. Tj. ne sadrži znake interpunkcije i razmake.

  Primer 22 *Primer prikazuje metod* `isalnum()`*:*

  ```
  >>> 'Cena CocaCole je 119 dinara!?'.isalnum()
  False
  >>> 'Tekstbezrazmaka'.isalnum()
  True
  ```

- `isalpha()` vraća True ukoliko su svi karakteri alfabetski i postoji najmanje jedan karakter. U suprotnom vraća False.

  Primer 23 *Primer prikazuje metod* `isalpha()`*:*

  ```
  >>> 'Frizider'.isalpha()
  True
  >>> 'Pun Frizider'.isalpha()
  False
  ```

- `isdigit()`,`isdecimal()`,`isnumeric()` Ove metode vraćaju `True` ukoliko su sve cifre u stringu cifre, ali se međusobno razliku. I one ovde neće biti detaljno objašnjene.

- `isidentifer()` vraća `True` ukoliko prosleđeni string može biti ime promeljive, u suprotnom vraća `False`.

- `isspace()` vraća `True` ako se string sastoji samo od belina.

- `islower()`,`isupper()` i `istitle()` vraća `True` ako je string u celosti napisan malim, velikim, odnosno zapisan kao ,,naslov"[1]. Numerički karakteri su dozvoljeni, ali je neophodno da string sadrzi makar jedan alfabetsksi karakter da bi metode vratile `True`.

  Primer 24 *Primer prikazuje metode* ***`islower()`*** *i* ***`isupper()`****:*

```
>>> 'kisobran'.islower(), 'kisobran'.isupper()
(True, False)
>>> 'ASCII'.islower(), 'ASCII'.isupper()
(False, True)
>>> 'Programski Jezik Pajton'.istitle()
True
```

- `isprintable()` Vraća `True` ukoliko je moguće sve karaktere odštampati. Karakteri koje nije moguće odštampati pripadaju UNICODE tabeli koja počinje sa ,,Other" [9].

4.5.2 Metode za formatiranje stringova

Sledeća grupa metoda konvertuje sting u drugačiji zapis. Ovoj grupi metoda pripadaju `title()`, `upper()`, `lower()`, `swapcase()`, `capitalize()`.

- `title()` metod postavlja prvo slovo svake reči u stringu u veliko slovo, a ostala postavlja u mala slova. Reči su indentifikovane kao podstingovi alfabetskih karaktera odvojenih nealfabetskim karakterima, kao što su brojevi ili beline.

  Primer 25 *Primer prikazuje metod* ***`title()`***

```
>>> naslov = "zagrljaj PITONA"
>>> print(naslov.title())
Zagrljaj Pitona
```

- `upper()`, `lower()` metode konvertuju sve karaktere stringa u velika, odnosno u mala slova.

  Primer 26 *Primer prikazuje metode* ***`lower()`*** *i* ***`upper()`****:*

[1] String je zapisan kao naslov ako je početno slovo svake reč u stringu zapisano velikim slovima, a ostatak reči malim slovima

```
>>> klub = "Miami Heat"
>>> print(klub.lower())
miami heat
>>> print(klub.upper())
MIAMI HEAT
```

- `swapcase()` metod prebacuje sva velika slova u mala i obrnuto, sva mala slova prebacuje u velika.
- `capitalize()` metod postavlja početno slovo stringa u veliko, a sva ostala postavlja u mala slova.

4.5.3 Metod `count()`

Metod `count()` broji broj pojavljivanja podstinga u stringu. Ovaj metod razlikuje velika i mala slova.

Primer 27 *Primer prikazuje metod* ***`count()`****:*

```
>>> s = """I'm going back to 505,
If it's a 7 hour flight or a 45 minute drive, In my imagination
you're waiting lying on your side, With your hands between your
thighs"""
>>> print(s.count('i'))
14
>>> print(s.count('I'))
3
```

4.5.4 Metode `strip()`, `rstrip()`, `lstrip()`

Metod `lstrip()` uklanja beline kojim počinje string, metod `rstrip()` uklanja beline kojim se završava string, dok metod `strip()` uklanja beline i sa početka i sa kraja. Povratna vrednost je string sa uklonjenim belinama. Dok originalni string ostaje nepromenjen.

Primer 28 *Primer prikazuje metode* ***`strip()`*** *i* ***`rstrip()`****:*

```
>>> text = "   Some text missing  "
>>> print(text.strip())
Some text missing
>>> print(text.lstrip())
Some text missing
```

Takođe, prethodnim metodama je moguće proslediti i string kao argument. U ovom slučaju, ove metode ukloniće pojavljivanja datog stringa sa početka i/ili kraja stringa.

Primer 29 *Primer prikazuje napredniji način za korišćenje metoda* ***strip()*** *i* ***rstrip():***

```
>>> web = "www.python.org"
>>> mail = "admin@python.org"
>>> print(web.lstrip('w'))
.python.org
>>> print(mail.strip(".org"))
admin@python
```

4.5.5 Metode `ljust(),rjust(),center()`

Sledeća grupa metoda vrši poravnanje reči u stringu. Odnosno postavlja string u polje zadate širine. Metod `ljust()` postavlja string levo i dodaje beline tako da zbir belina i dužine stringa bude jednak broju koji je prosleđen kao argument metodi. Slično metode `rjust()` i `center()` vrše poravnanje po desnoj strani, odnosno centriraju string u željenu veličinu.

Primer 30 *Primer prikazje načine za centriranje teksta korišćenjem metoda* ***ljust()****,****rjust()*** *i* ***center():***

```
>>> gric = 'kikiriki'
>>> print(gric.ljust(12),gric.center(10),gric.rjust(15),sep=",")
kikiriki    , kikiriki ,       kikiriki
```

4.5.6 Metode `find(), index(), rfind(), rindex()`

Oba metode, `find()` i `index()` su veoma slične. One pronalaze i vraćaju indeks prvog pojavljivanja prosleđenog podstringa. Ako traženi podstring nije pronadjen, metod `find()` vraća -1, dok metod `index()` vraća grešku. Verzije `rfind()` i `rindex()` su slične kao prethodne, `find()` i `index()`, ali one vrše pretragu od kraja ka početku.

Primer 31 *Primer prikazuje upotrebu metoda* ***find()*** *i* ***index():***

```
>>> x = 'doktorirati'
>>> print(x.find('k'), x.find('m'),sep=',')
2,-1
>>> print(x.find('k'))
```

```
2
>>> print(x.index('m'))
ValueError: substring not found
```

4.5.7 Metode `replace()`

Zamena podstringova datog stringa moguća je korišćenjem ugrađene metode `replace()`. Potrebno je proslediti dva argumenta, prvi argument predstavlja podstring koji treba zameniti, a drugi je novi podstring. Povratna vrednost ove metode je novi string, originalni string ostaje nepromenjen.

Primer 32 *Primer prikazuje upotrebu metode* ***`replace()`****:*

```
>>>  y = "Lubenica je voce."
>>> print(y.replace("je","nije."))
Lubenica nije voce.
```

4.5.8 Metode `split()` i `splitlines()`

Metod `split()` služi za podelu stringa na više delova po nekom parametru. Povratna vrednost ove metode je lista. Postoje dva opciona argumenta. Prvi argument je separator, odnosno karakter po kome treba vršiti separaciju, a drugi je maksimalni broj podela ,,maxsplit". U slučaju kada je maksimalan broj podela postavljen, lista će imati maksimalno ,,maxsplit+1" element. Takođe, postoji i metod `rsplit()` koji vrši podelu krećući se od poslednjeg elementa stringa ka prvom. Ukoliko nije prosleđen parametar po kom treba deliti, to je blanko karakter.

Primer 33 *Primer prikazuje metod* ***`split()`****:*

```
>>> str1 = "Dnevne novine Politika"
>>> print(str.split())
['Dnevne','novine','Politika']
>>> str2 = '1<>2<>3'.split('<>',1)
['1', '2<>3']
```

Primetimo da ni u jednom slučaju separator nije uključen u podeljen string.

Za kraj biće pomenut metod `splitlines()` koji višelinijske stringove deli u jednolinijske stringove. Sličan je prethodnom metodu uz dodati argument `split('\n')` ali prihvata i opcije ,,\r" i ,,\r\n".

Primer 34 *Primer prikazuje upotrebu metode* ***`splitlines()`****:*

```
>>> str = """ One line
Two lines
Three lines
Four lines"""
>>> str.splitlines()
[' One line', 'Two lines', 'Three lines', 'Four lines']
```

5 Liste

Lista je promenljiv uređeni niz objekata, gde je svaki član liste identifikovan svojim indeksom. Vrednosti koje čine listu nazivaju se elementima. Ovi elementi ne moraju biti istog tipa. Moguće je staviti brojeve, stringove pa čak i druge liste u listu. Smeštaju se unutar uglastih zagrada [ ], a odvajaju zarezom (,). Prazna lista se označava parom uglastih zagrada.

Postoje nekoliko načina za kreiranje liste. Najprostiji je navođenjem elemenata u uglastim zagradama.

Primer 35 *Primer nekih listi:*

```
>>> []
>>> [2, 4, 6, 8, 10]
>>> ["Milan" , "Marko", "Pera"]
```

Prvi primer predstavlja praznu listu. Drugi primer je lista parnih celih brojeva, a treći primer je lista stringova. Kao što je i naglašeno, članovi liste ne moraju biti istog tipa. Sledeći primer sadrži listu čiji su elementi string, ceo broj, realan broj i lista.

Primer 36 *Primer prikazuje listu različitih tipova elemenata:*

```
>>> ["Zdravo", 3.14, 10, [100, 5]]
```

Liste lako možemo da dodeljujemo promenljivim, a moguće je liste proslediti funkcijama.

Primer 37 *Primer deklarisanja liste:*

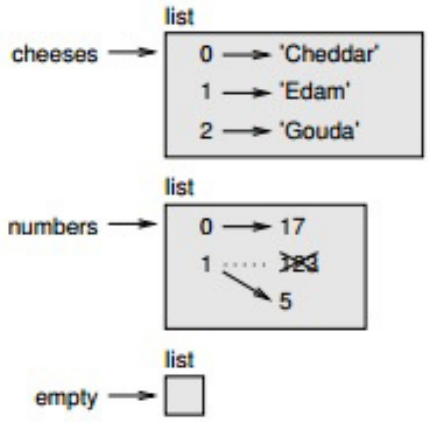
```
>>> empty = []
>>> numbers = [2, 4, 6, 8, 10]
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> rand = ["Zdravo", 3.14, 10, [100, 5]]
```

5.1 Pristup elementima liste

Svakom elementu liste možemo pristupiti preko njegove pozicije u listi, odnosno njegovom indeksu. Indeksiranje liste počinje od 0. Svaki ceo broj predstavlja poziciju u listi, a kao i kod stringova , mogu se koristiti i negativni indeksi.

Primer 38
```
>>> numbers = [2, 3, 7, -1, 0]
>>> print(numbers[0])
2
>>> print(numbers[4])
0
>>> print(numbers[-2])
-1
```

Za razliku od stringova, liste su promenljive, što znači da je moguće promeniti elemente liste. O listama može da se razmišlja kao o vezi indeksa i elemenata. Svaki indeks pokazuje na jedan element. Slika 4. prikazuje stanje liste.



Slika 4. *Stanje liste.*

Indeksiranje liste funkcioniše na isti način kao i kod stringova, tj. svaki ceo broj može biti indeks. Dolazi do greške ukoliko se pokuša pristupanje ili promena elementa niza koji ne postoji.

Primer 39 *Primer menjanja elemenata liste koji opisuje proces sa Slike 4.:*

```
>>> numbers = [17, 123, 0, 3, 5]
>>> numbers[1] = 5
```

```
>>> numbers[-1] = numbers[-1] + 5
>>> print(numbers)
[1, 87, 0, 3, 5]
```

5.2 Operatori nad listama

Provera da li element pripada listi dobija se korišćenjem operatora `in` koji je takođe radio i sa stringovima. Operator `in` vrši linearnu pretragu, tako što se kreće kroz celu listu i proverava da li je trenutni element jednak traženom elementu. Naravno, moguće je proveriti i da li element ne pripada listi korišćenjem operatora `not in`.

Primer 40 *Primer prikazuje operatore za proveru pripadnosti listi:*

```
>>> voce = ['ananas', 'breskva', 'jagoda', 'kruska']
>>> print('jabuka' in voce)
False
>>> print('jagoda' in voce)
True
```

Operatori `+` i `*` se takođe ponašaju isto kao i kod stringova. Operator `+` nadovezuje liste, a operator `*` umnožava listu. Naravno drugi operand je ceo broj koji predstavlja koliko puta želimo da umnožimo listu.

Primer 41 *Primer prikazuje upotrebu operatora `+` i `*` za nadovezivanje i umnožavanje liste:*

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> print(a+b)
[1, 2, 3, 4, 5, 6]
>>> lista = [1]
>>> print(lista*10)
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>> str = 'Eurocrem'
>>> print(a + str)
TypeError: Can't convert 'list' object to str implicitly
```

Primetimo da nije moguće koristiti operator `+` za nadovezivanje listi i stringova. Neophodno je konvertovati string u listu korišćenjem funkcije `list()`.

5.3 Isecanje liste

Kao što je i mnogo puta do sada navedeno, sličnosti sa stringovima su ogromne. Isecanje listi vrši se na potpuno isti način kao i kod stringova. Ipak, razlika postoji i ogleda se u tome što je podlistama moguće promeniti vrednost, pa čak i izbrisati izvučene elemente.

Primer 42 *Primer prikazuje napredniji način za korišćenje operatora za seckanje:*

```
>>> recenica = ['Matematika', 'je', 'moj', 'omiljeni', 'skolski',
'predmet']
>>> recenica[2:4] = []
>>> print(recenica)
['Matematika', 'je', 'skolski', 'predmet']
```

Takođe, na ovaj način je moguće dodati i nove elemente u listu. Koristeći se sledećom sintaksom prikazanoj u primeru.

Primer 43 *Primer za dodavanje preko elemenata u listu korišćenjem operatora za seckanje:*

```
>>> brojevi = [1, 2, 3, 7, 8, 9]
>>> brojevi[2:2] = [4, 5, 6]
>>> print(brojevi)
[1, 2, 4, 5, 6, 3, 7, 8, 9]
```

5.4 Metode i funkcije za rad sa listama

Programski jezik Pajton omugućava veliki fond funkcija i metoda koje olakšavaju rad sa listama. Spisak funkcija i metoda sa rad sa listama prikazan je u tabeli ().

`append`	`count`	`index`	`remove`
`clear`	`del`	`insert`	`reverse`
`copy`	`extend`	`pop`	`sort`

5.4.1 Liste kao stekovi

Veoma je lako koristiti liste kao stekove, tj postavljati element na kraj steka i skidati poslednji element sa steka. Za dodavanje elementa na stek koristi se metod `append()`. Željeni element se prosleđuje kao argument. Za skidanje poslednjeg elementa sa steka koristi se metod `pop()`. Povratna vrednost ovog metoda je element skinut sa steka.

Primer 44 *Primer korišćenja liste kao stekova:*

```
>>> lista = ["ovo", "je", "jedna"]
>>> lista.append("lista")
>>> print(lista)
['ovo', 'je', 'jedna', 'lista']
>>> lista.pop()
>>> print(lista)
['ovo', 'je', 'jedna']
```

Lako je liste u Pajtonu koristiti i kao lančane liste, odnosno strukturu u kojoj se uvek prvi element uklanja iz liste. Prosleđivanjem argumenta metodi `pop()` moguće je implementirati ovakvu strukturu. Ali se ovako nešto ne preporučuje zbog brzine izvršavanja programa. Za pametniji rad sa lančanim listama potrebno je uvesti funkciju `deque` iz modula `collections`.

Primer 45 *Primer prikazuje upotrebu listi kao lančanih listi:*

```
>>> from collections import deque
>>> queue = deque(["Abidal", "Mikel", "Torres"])
>>> queue.append("Terry")  # Dodajemo Terry-a
>>> queue.append("Matic") # Dodajemo Matica
>>> queue.popleft()        # Izbacamo prvog iz liste
'Abidal'
>>> queue.popleft()        # Izbacamo drugog iz liste
'Mikel'
>>> print(queue)                  # Stampamo listu
deque(['Torres', 'Terry', 'Matic'])
```

5.4.2 Metode `extend()` i `insert()`

Do sada smo videli kako da dodamo jedan element na kraj liste. To je urađeno koristeći metod `append()`. Ako hoćemo da na kraj liste dodamo više elemenata, koristimo metod `extend()`. Kao arugment prosleđujemo listu elemenata koje želimo da dodamo.

Primer 46 *Primer prikazuju razliku upotreba metoda* ***`append()`*** *i* ***`extend()`****:*

```
>>> lista1 = [1, 2, 3]
>>> lista1.append([4,5])
>>> print(lista1)
[1, 2, 3, [4, 5]]
>>> lista2 = [1, 2, 3]
```

```
>>> lista.extand([4,5])
>>> print(lista)
[1, 2, 3, 4, 5]
```

Primetimo da ako metodom `append()` želimo da dodamo više elemenata dobićemo rezultat koji ne želimo, [1, 2, 3, [4, 5]], pa zato koristimo metod `extend()`. Još jedan način koji čemo koristiti za dodavanje elemenata u listu je metod `insert()`. Za razliku od prethodne dve metode, metod `insert()` dodaje element na poziciju pre indeksa koji mu prosledimo kao prvi argument. A kao drugi argument metodi `insert()` prosleđujemo element koji želimo da dodamo.

Primer 47 *Primer za dodavanje elemenata u listu korišćenjem metode `insert()`:*

```
>>> lista = ["ovo", "je", "jedna", "lista"]
>>> lista.insert(2,"umetnuti clan")
>>> print(lista)
['ovo', 'je', 'umetnuti clan', 'jedna', 'lista']
```

5.4.3 Funkcija `len()` i metod `count()`

Ova funkcija i metod funkcionišu isto kao i sa stringovima. Funckija `len()` broji koliko elemenata sadrži data lista. Treba imati na umu da kako lista može sadržati drugu listu kao svoj element, da funkcija `len()` te elemente gleda kao i svake druge elemente, odnosno njihova dužina je 1. Metod `count()` broji broj pojavljivanja prosleđenog argumenta u listi.

Primer 48 *Primer za određivanje broja elemenata liste i metod koji broji broj pojavljivanja elementa u listi:*

```
>>> lista = ["zemlja", 0, [3,1,2], "dan", "dan"]
>>> print(len(lista))
5
>>> print(lista.count('dan'))
2
```

5.4.4 Funkcija `del()` i metod `remove()`

Za brisanje elementa iz liste koristimo funkciju `del()` tako što joj kao argument prosledimo element niza koji hoćemo da izbrišemo.

Primer 49 *Primer za brisanje elemenata iz liste korišćenjem funckije `del()`:*

```
>>> recenica = ['Prolece', 'je', 'najlepse', 'godisnje', 'doba']
>>> del(recenica[2])
>>> print(recenica)
['Prolece', 'je', 'godisnje', 'doba']
>>> del(recenica[1:3])
['Prolece', 'doba']
```

Drugi način za brisanje elemenata iz liste je metod `remove()` koji briše prvo pojavljivanje elementa iz liste. Kao argument mu se prosleđuje element koji želimo da izbrišemo.

Primer 50 *Primer prikazuje upotrebu metoda* `remove()` *za brisanje elemeneta liste:*

```
>>> reke = ['Dunav', 'Sava', 'Morava', 'Amazon', 'Ibar']
>>> reke.remove('Amazon')
>>> print(reke)
['Dunav', 'Sava', 'Morava', 'Ibar']
```

5.4.5 Sortiranje listi, metod `reverse()`

Metod koji sortira listu u rastućem poretku je metod `sort`. Brojeve poredi na način koji smo navikli, a slova poredi tako što poredi ASCII vrednosti karaktera. Više o ASCII karakterima možete pročitati u literaturi [16]

Primer 51 *Primer prikazuje upotrebu metoda* `sort()` *i* `reverse()`*:*

```
>>> brojevi = [33, 21, 5, 15, -3 , 0]
>>> brojevi.sort()
>>> print(brojevi)
[-3, 0, 5, 15, 21, 33]
>>> brojevi.reverse()
[33, 0, 5, 15, 21, 33]
>>> slova = ['A', 'a', 'b', 'E']
>>> slova.sort()
>>> print(slova)
['A', 'e', 'a', 'b']
```

5.4.6 Funkcije `min()`, `max()` i `sum()`

Na kraju, pomenućemo nekoliko funkcija koje se pretežno koriste za liste čiji su podaci brojevi. Metode `min()` i `max()` vraćaju najmanji odnosno najveći element liste. Ove metode moguće je koristiti i za stringove, jer vrše poređenje na identičan način kao i metoda `sort()`.

Primer 52 *Primer za nalaženje najmanje i najveće vrednosti liste:*

```
>>> rand = [33, 21, 5, 15, -3 , 0, 13, 42]
>>> maxi = max(rand)
42
>>> mini = min(rand)
-3
```

Metod koji određuje sumu liste je `sum()`. U slučaju nije da prosleđena lista sadrži objeske koji nisu brojevne vrednosti doći će do greške.

Primer 53 *Primer nalaženja sume liste i aritmetičke sredine:*

```
>>> poeni = [55, 76, 42, 56, 67, 90, 100]
>>> suma = sum(poeni)
>>> br = len(poeni)
>>> prosek = suma/br
>>> print(suma, prosek, sep=';')
486;69.42857142857143
```

5.4.7 Funkcija `range()`

Često se pojavljuje potreba za iteraciju kroz niz brojeva. Ugrađena funkcija `range()` generiše iterator brojeva. Može imati jedan, dva ili tri argumenta. U slučaju da se funkciji `range()` prosledi samo jedan argument ,,n", tada je njena povratna vrednost biti iterator brojeva od 0 pa do ,,n-1" [17]. Prosleđivanjem dva argumenta, a i b (a <b), kreiramo iterator koji sadrži elemente a, a+1, ... b-1. Prosleđivanjem trećeg argumenta funkciji određujemo korak, odnosno koliko će elemenata preskočiti. Naravno, treba voditi računa o tome da svi prosleđeni argumenti budi celi brojevi.

Primer 54 *Primer za funkciju* ***`range()`*** *sa jednim,dva i tri argumenta:*

```
>>> prvih10 = range(11)
>>> type(prvih10)
<class 'range'>
>>> for el in prvih10:
        print(el, end='; ')

0; 1; 2; 3; 4; 5; 6; 7; 8; 9; 10;
```

U prethodnim verzijama programskom jeziku Pajton je postojala i funkcija `xrange()` koja je preimenovana u `range()` u verziji 3, dok je originalni

`range()` iz verzije 2 programskog jezika Pajton izbačen. Funkcija `range()` iz verzije 2 je generisala listu brojeva.

O razlikama između iteratora i listi u ovom radu neće biti mnogo reči. Važno je samo napomenuti da nad iteratorima nije moguće koristiti seckanje.

5.5 Skupovne liste

U prethodnom poglavlju bilo je reči kako kreirati listu nabrajanjem elemenata, seckanjem i korišćenjem ugrađenih funkcija. U programskom jeziku Pajton postoji još jedan veoma brz i jednostavan način za kreiranje listi. Ovaj način za kreiranje listi dosta liči na opis matematičkih skupova [18].

$$S = \left\{x^2 : x \in \{0,\, 1,\, \ldots 9\}\right\}$$

$$V = \left(1, 2, 4, 8, \ldots, 2^9\right)$$

$$M = \{x : x \in S \cap 2|x\}$$

Ovakvi skupovi se slično zapisuju i u programskom jeziku Pajton. Skupovna lista se smešta u uglaste zagrade i sadrži izraz, praćen `for` petljom, nakon čega stoji 0 ili više uslova [19].

```
nova_lista = [izraz for elem in stara_lista if uslov(i)]
```

Sledi primer svih kubova brojeva od 1 do 10, deljivih sa 4.

Primer 55 *Primer skupovne liste:*

```
>>> kub = [x**3 for x in range(1,11) if (x**3)%4==0]
>>> print(kub)
[8, 64, 216, 512, 1000]
```

Takođe, skupovne liste je moguće kreirati korišćenjem većeg broja `for` petlji. Proći će kroz sve elemente objekata sekvencijalno i kretaće se kroz kraće objekte ukoliko postoje.

Primer 56 *Primer prikazuje upotrebu dve* ***for*** *petlje u kreiranju skupovnih listi:*

```
>>> item = [x+y for x in 'flo' for y in 'pot']
>>> print(item)
['fp', 'fo', 'ft', 'lp', 'lo', 'lt', 'op', 'oo', 'ot']
```

6 Torke

Torka je nepromenljivi niz članova. Članovi u torki su objekti istih ili različitih tipova. Torka se definiše nabrajanjem objekata odvojenih zarezom (,). Torka sa samo jednim članom mora imati zarez (,) na kraju jer inače gubi prirodu torke. Prazna torka označava se parom praznih malih zagrada ( ). Članovi torke mogu biti takođe torke. Iako su slične listama, torke su nepromenljive strukture. Indeksiraju se na isti način kao i liste. Torka se generiše nabrajanjem članova ili pozivom ugrađene funkcije `tuple()`. Ako je dat niz vrednosti lista, onda `tuple()` vraća torku sa članovima jednakim članovima date liste.

Primer 57 *Primer torki*

```
>>> torka1 = (1, 2, 3, 4, 5)
>>> torka2 = (3.14, 22, 'pi')
>>> tortka3 = ('a', )
>>> torka4 = ()
```

6.1 Pristup elementima torke

Pristupanje elementima torke je isto kao i kod listi. Indeksiranje počinje od nule. Seckanje i izdvajanja delova torki je takođe moguće.

Primer 58 *Primer pristup elementima torke isto je kao i kod listi:*

```
>>> torka = tuple('Matematika')
>>> print(torka)
('M', 'a', 't', 'e', 'm', 'a', 't', 'i', 'k', 'a')
>>> print(torka[0])
M
>>> print(torka[1:5])
('a', 't', 'e', 'm')
>>> print(torka[-2:])
('k', 'a')
>>> print(torka[::2])
('M', 't', 'm', 't', 'k')
```

Ali torke su nepromenljiv tip podataka, tako da je nemoguće promeniti vrednost elementu torke. Ipak, moguće je celu torku zameniti novom.

Primer 59 *Primer zamene torke novom tako da joj se promeni samo jedan element:*

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> t = ('A',) + t[1:]
>>> print(t)
('A', 'b', 'c', 'd', 'e')
```

6.2 Dodela vrednosti torkama

U mnogim programskim jezicima za zamenu vrednosti dve promenljive potrebno je deklarisati novu kako bi ona privremeno skladištila jednu vrednost. Na primer, za zamenu vrednosti promenljivih a i b bio bi neophodan sledeći kod:

```
>>> temp = a
>>> a = b
>>> b = temp
```

U programskom jeziku Pajton, ovakve probleme je moguće rešiti u jednoj liniji koda korišćenjem torki.

```
>>> a, b = b, a
```

Na levoj strani je torka promenljivih, a sa desne torka izraza. Svakoj promenljivoj se dodeljuje vrednost respektivno, odnosno `a` dobija vrednost promenljive `b`, `b` dobija vrednost promenljive `a`.

Takođe, desna strana izraza može biti i string ili lista. Sledeći primer prikazuje dvostruku dodelu korišćenjem metode `split()` na string.

Primer 60 *Primer prikazuje dvostruku dodelu:*

```
>>> addr = 'monty@python.org'
>>> name, domain = addr.split('@')
>>> print(name)
monty
>>> print(domain)
python.org
```

6.3 Torka kao povratna vrednost funkcije

Striktno govoreći, povratna vrednost funkcije može biti samo jedna vrednost, ali ako je povratna vrednost torka, onda se dobija isti efekat kao sa većim brojem povratnih vrednosti.

Na primer, za deljenje dva cela broja i računanje celog dela i ostatka bi bilo neefikasno računati odvojeno celi deo i ostatak, brže je izračunati oba i vratiti ih funkcijom kao torku. Ugrađena funkcija `divmod()` prima dva argumenta i vraća torku od dva elementa, celi deo i ostatak.

Primer 61 *Primer prikazuje korišćenje funkcije* `divmod()` *za računanje celog dela i ostatka pri deljenju dva broja.*

```
>>> t = divmod(7,3)
>>> print(t)
(2, 1)
>>> type(t)
<class 'tuple'>
```

Primetimo da je promenljiva `t` zapravo torka od dva elementa.

Primer 62 *Primer prikazuje funkciju koja računa minimum i maksimum objekta i vraća torku ovih vrednosti.*

```
>>> def min_max(t):
        return(min(t), max(t))

>>> lista = [2, 3, 4, 5, 1, 2]
>>> minMax = min_max(lista)
>>> print(minMax)
(1, 5)
```

6.4 Torka kao argument funkcije

Svaka funkcija može da primi različit broj argumenata. Parametar čije ime počinje se zvezdicom `*` skuplja argumente u torku. Tako je u sledećem primeru prikazana funkcija `printall()` koja prima bilo koji broj argumenata i štampa ih.

Primer 63 *Primer prikazuje korišćenje operatora* `*` *pri prosleđivanju nepoznatog broja argumenata funkciji:*

```
>>> def printall(*args):
        print(args)

>>> printall(1, 2, 3, 2.32, '3')
(1, 2, 3, 2.32, '3')
```

Sa druge strane, ako se funkciji koja prima `n` celih brojeva prosledi torka čija je dužina `n`, to će dovesti do greške.

Primer 64 *Primer prikazuje grešku koja dolazi ukoliko se funkciji prosledi pogrešan broj argumenata:*

```
>>> t = (7, 3)
>>> r = divmod(t)
TypeError: divmod expected 2 arguments, got 1
```

Ali ako se torka ,,podeli" zvezdicom, funkcija će raditi:

Primer 65 *Primer prikazuje način za prosleđivanje torke funkciji:*

```
>>> t = (7, 3)
>>> r = divmod(*t)
>>> print(r)
(2, 1)
```

6.5 Liste i torke

Ugrađena funkcija `zip()` prima dva ili više argumenata složenog tipa (liste, torki ili stringova) i vraća iterator čiji su elementi torke sastavljene od po jednog elementa argumenta.

Primer 66 *Primer prikazuje korišćenje funkcije* ***zip():***

```
>>> s = 'abc'
>>> t = [0, 1, 2]
>>> zipped = zip(s,t)
>>> type(zipped)
<type 'zip'>
>>> lista = list(zipped)
>>> print(lista)
[('a', 0), ('b', 1), ('c', 2)]
```

Rezultat je iterator čiji je svaki element torka sastavljena od jednog karaktera stringa i jednog elementa liste.

Ukoliko je dužina argumenata različita, rezultat će imati dužinu kraćeg argumenta.

Primer 67 *Primer prikazuje funkciju* ***zip()*** *kada su prosleđeni argumetni različite dužine:*

```
>>> zipped = zip ('Anna','Mac')
>>> zipList = list(zipped)
>>> print(zipList)
[('A', 'M'), ('n', 'a'), ('n', 'c')]
```

Najveće važnost i upotreba funkcije `zip()` se ogleda u lakom i brzom načinu za prolazak kroz složenije tipove podataka. Moguće je izbegavanje dvostrukih `for` petlji korišćenjem `zip` funkcije.

Primer 68 *Primer funkcije u kojoj se funkcija* ***`zip()`*** *koristi za kreiranje iteratora:*

```
def has_match(t1, t2):
    for x,y in zip(t1, t2):
        if x ==y:
            return True
        return False
```

7 Rečnici

Rečnici u programskom jeziku Pajton su kolekcije slične listama koji se ne indeksiraju brojevima već ključevima koji mogu biti tipa broj, string ili torka. Torka može biti ključ u rečniku ukoliko sadži samo stringove, brojeve ili druge torke. Lista ne može biti ključ, zbog svoje promenljivosti. Vrednosti mogu biti bilo kog tipa i mogu se ponavljati, ali rečnik ne može imati dva ili više ista ključa. Rečnik se definiše nabrajanjem parova ključ: vrednost, unutar vitičastih zagrada {}, a parovi se međusobno odvajaju zarezom (,). Prazan rečnik se označava parom praznih vitičastih zagrada.

Primer 69 *Primer praznog i nepraznog rečnika.*

```
>>> prazan_imenik = {}
>>> imenik = {'mika': '065333222', 'laza': '069555888',
'pera':'063123123'}
```

Pristupanje elementima rečnika se vrši kao kod listi, navođenjem ključa između uglastih zagrada. Dodavanje elemenata u listu se izvršava dodavanjem novog para ključ: vrednost, dok se elementi menjaju kao kod listi.

Primer 70 *Primer prikazuje dodavanje elementa u postojeći rečnik i menjanje vrednosti elementa rečnika.*

```
>>> imenik = {'mika': '065333222', 'laza': '069555888', 'pera':
'063123123'}
>>> imenik['milan'] = '066889977'
>>> imenik['mika'] = '066888222'
```

Napomena: redosled elemenata u rečniku ne ostaje isti kao pri deklarisanju rečnika. Zapravo, različiti računari će na različite načine urediti elemente u rečniku. Ali ovo ne predstavlja problem, jer se elementima rečnika pristupa isključivo korišćenjem ključeva.

Primer 71 *Primer prikazuje ispis sadržaja rečnika.*

```
>>> imenik = {'mika': '065333222', 'laza': '069555888', 'pera':
'063123123'}
>>> print(imenik)
{'laza': '069555888','mika': '065333222','pera': '063123123'}
```

7.1 Operacije nad rečnicima

Operator `in` drugačije radi za liste i za rečnike. O načinu na koji pretražuje liste je bilo reči. Kada su u pitanju rečnici, koriste se algoritam heš tabela. Operatoru `in` je potrebno približno isto vreme za pretragu rečnika bez obzira na veličinu rečnika.

7.2 Petlje i rečnici

Kretanje kroz rečnik korišćenjem `for/in` petlje se vrši po ključevima za razliku od lista. Sledeći primer će prikazati funkciju koja štampa ključ i vrednost elemenata rečnika.

Primer 72 *Primer prikazuje iteraciju kroz rečnik:*

```
>>> def print_hist(h):
        for key in h:
            print(key, h[key])
>>> imenik = {'mika': '065333222', 'laza': '069555888', 'pera':
'063123123'}
>>> print_hist(imenik)
mika 065333222 laza 069555888 pera 063123123
```

7.3 Funkcije i metode za rad sa rečnicima

U sledećoj tabeli su prikazane funkcije i metode koje omogućavaju lakši rad sa rečnicima. Neke od funkcija i metoda će biti detaljnije objašnjene.

`clear`	`items`	`keys`	`update`
`copy`	`iteritems`	`pop`	`values`
`get`	`iterkeys`	`popitem`	
`has_key`	`itervalues`	`setdefault`	

7.3.1 Funkcije `len()` i `del()`

Funkcija `len()` računa i vraća dužinu rečnika. Poziv je isti kao i za liste i torke.

Primer 73 *Primer meri duzinu rečnika korišćenjem funkcije* ***`len()`****:*

```
>>> kupovina = {'jabuka' : 213, 'kruska' : 435, 'banana':90}
>>> print(len(kupovina))
3
```

Za brisanje para ključ: vrednost iz rečnika koristi se ugrađena funkcija `del()`. Način korišćenja ove funkcije je isti kao i za liste.

7.3.2 Metode `keys()` i `values()`

Ugrađene metode `keys()` i `values()` izdvajaju iz rečnika ključeve, odnosno vrednosti rečnika. Ne prosleđuju im se dodatni argumenti.

Primer 74 *Primer upotrebe metoda* ***`keys()`*** *i* ***`values()`****:*

```
>>> imenik = {'ime': 'Milan', 'telefon' : '065222555',
 'adresa':'Skadarska 5'}
>>> print(imenik.keys())
dict_keys(['adresa', 'telefon', 'ime'])
>>> print(imenik.values())
dict_values(['Skadarska 5', '065222555', 'Milan'])
```

Ove dve funkcije vraćaju objekte tipa `dict_keys` i `dict_values`. Na ove objekte nije moguće izvršavati funkcije i operatore koji su radili sa listama. Neophodno je prebaciti ih u liste funkcijom `list()`. U verziji 2 programskog jezika Pajton, povratna vrednost ovih funkcija je bila lista.

7.3.3 Metod `items()`

Metod `items()` ispisuje parove ključ: vrednost. Poziv je isti kao i za prethodne dve metode. Zbog načina skladištenja rečnika, postoji verovatnoća da redosled ispisa ne bude identičan početnom rečniku.

Primer 75 *Primer korišćenja metode* ***`items()`****:*

```
>>> imenik = {'ime': 'Milan', 'telefon' : '065222555',
 'adresa':'Skadarska 5'}
>>> print(imenik.items())
dict_items([('adresa', 'Skadarska 5'), ('telefon', '065222555'),
('ime', 'Milan')])
```

7.3.4 Metode `get()` i `setdefault()`

Metod `get()` vrši pretragu po ključevima. Prima 2 argumenta, prvi je traženi ključ, a drugi argument je podrazumevana vrednost. Ukoliko rečnik sadrži ključ koji je tražen, onda metod `get` vraća vrednost koja odgovara tom ključu. Ukoliko ne pronađe ključ, vraća podrazumevanu vrednost koja je prosleđena kao drugi argument.

Primer 76 *Primer prikazuje upotrebu metode* ***`get()`****:*

```
>>> imenik = {'ime': 'Milan','prezime':'Petrovic',
'telefon' : '065222555', 'adresa':'Skadarska 5'}
>>> imenik.get('ime','nema ime')
'Milan'
>>> imenik.get('srednje_ime','nema srednje ime')
'nema srednje ime'
```

Metod sličan prethodnom metodu je metod `setdefault()`, koji ukoliko ne postoji zadati ključ pravi taj ključ, a za vrednost mu postavlja vrednost koja je prosleđena kao drugi argument.

Primer 77 *Primer prikazuje upotrebu metode* ***`setdefault()`****:*

```
>>> imenik = {'ime': 'Milan','prezime':'Petrovic',
'telefon' : '065222555', 'adresa':'Skadarska 5'}
>>> imenik.setdefault('ime','nema ime')
'Milan'
>>> imenik.setdefault('srednje_ime','nema srednje ime')
'nema srednje ime'
>>> print(imenik)
{'prezime': 'Petrovic', 'adresa': 'Skadarska 5', 'telefon':
'065222555', 'ime': 'Milan', 'srednje_ime': 'nema srednje ime'}
```

7.3.5 Metode `pop()` i `update()`

Ranije je pomenuta funkcija `del()` koja briše par ključ: vrednost iz rečnika. Drugi način za brisanje ovog para je metod `pop()`, a kao argument mu se prosleđuje vrednost ključa koji hoćemo da izbrišemo. Kao povratnu vrednost dobijamo par ključ: vrednost koju smo izbrisali.

Ponekad je poželjno proširiti rečnik drugim rečnikom. Metod `update()` se poziva na rečnik, a kao argument se prosleđuje rečnik kojim proširujemo. Ako postoji preklapanje u nekom od ključeva, onda će vrednosti iz prvog rečnika biti zamenjene vrednostima iz drugog rečnika.

Primer 78 *Primer prikazuje nadogradnju rečnika korišćenjem metode* `update()`*:*

```
>>> recnik1 = {'ime': 'Laza', 'telefon': '064224433'}
>>> recnik2 = {'ime_oca': 'Pera', 'prezime': 'Peric'}
>>> recnik1.update(recnik2)
>>> print(recnik1)
{'ime_oca': 'Pera', 'prezime': 'Peric', 'telefon': '064224433',
'ime': 'Laza'}
>>> recnik3 = {'ime_oca': 'Rajko', 'telefon' :
'062222111'}
>>> recnik1.update(recnik3)
>>> print(recnik1)
{'ime_oca': 'Rajko', 'prezime': 'Peric', 'telefon': '062222111',
'ime': 'Laza'}
```

8 Skupovi

Do sada smo videli kako se u Pajtonu generišu skupovne liste, ali treba praviti razliku između skupovnih listi i skupova. Skupovi, `set`, se za razliku od listi ne indeksiraju i nisu uređeni. Dupliranje elemenata u skupu nije moguće. Element se može najviše jednom naći u skupu. Svaki objekat koji se može koristiti kao ključ u rečniku može biti element skupa, ali celi skupovi ne mogu biti korišćeni kao ključevi rečnika. Ako želimo da skup bude ključ rečnika, potrebno je kreirati `frozenset` koji je nepromenljiva kopija skupa (`set`-a).

Koriste se kada je prisustvo elementa u objektu važnije od njegove pozicije. A mogu se korisitii kao način za izbacivanje duplikata iz listi.

Skup se kreira pozivom funkcija `set()` ili `frozenset()` na bilo koju listu, torku ili string. U slučaju rečnika, kreira se skup ključeva.

Primer 79 *Primer prikazuje kreiranje skupa:*

```
>>> imenik = {'mika':'066525152','laza':'0662002000',
'pera':'069123123'}
>>> skup = set(imenik)
>>> print(skup)
{'mika','laza','pera'}
>>> str = "matematika"
>>> s = set(str)
>>> print(s)
{'t', 'i', 'e', 'k', 'a', 'm'}
```

Funkcija `set()` prima samo jedan argument tipa sekvence, nije moguće proslediti listu argumenata. Ovaj poziv bi doveo do greške `TypeError`.

8.1 Operacije nad skupovima

Matematičke operacije nad skupovima su uključene u programski jezik Pajton, pa je tako omogućeno da se na lak način nađe unija, presek ili razlika skupova. O svakom od ovih operatora će detaljno biti reči u nastavku.

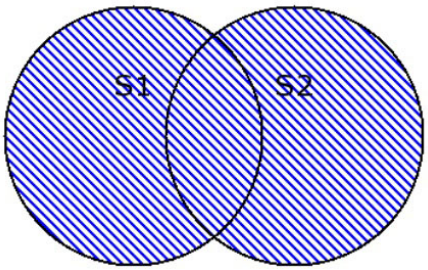
Kreiraćemo dva skupa, nad kojima će biti prikazane sve operacije.

```
>>> fib = set( (1,1,2,3,5,8,13) )
>>> prosti = set( (2,3,5,7,11,13) )
```

Prvi skup predstavlja niz Fibonačijevih brojeva, iz koga je izbačena jedna jedinica, a drugi skup predstavlja prvih šest prostih brojeva.

- **Unija** `|`: Rezultujući skup će sadržati elemente oba skupa.

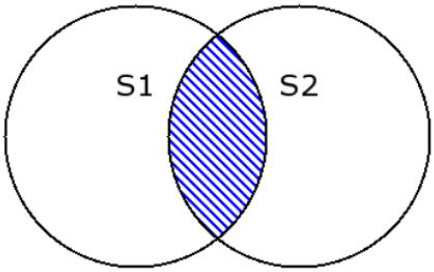
  ```
  >>> fib | prosti
  {1, 2, 3, 5, 7, 8, 11, 13}
  ```



Slika 5. *Unija dva skupa*

- **Presek** `&`: Rezultujući skup će sadržati samo elemente koji se nalaze u oba skupa.

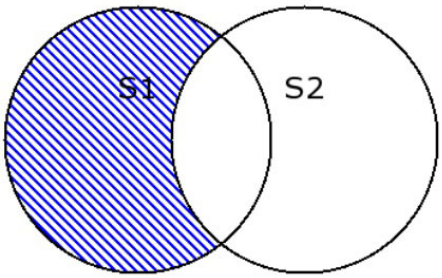
  ```
  >>> fib & prosti
  {2, 3, 5, 13}
  ```



Slika 6. *Presek dva skupa*

- **Razlika** -: Rezultujući skup sadrži elemente koji se nalaze u levom skupu, a ne nalaze se u desnom skupu.

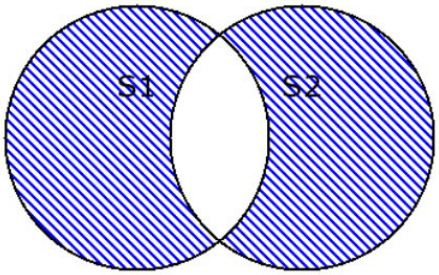
```
>>> fib - prosti
{8, 1}
>> prosti - fib
{11, 7}
```



Slika 7. *Razlika dva skupa*

- **Simetrična razlika** ^ Rezultujući skup će sadržati elemente oba skupa bez njihovog preseka.

```
>>> fib ^ prosti
{1, 7, 8, 11}
```



Slika 8. *Simetrična razlika dva skupa*

8.2 Poređenje skupova

Svi standardni operatori koje smo do sada videli(<,<=,>,>=,==,! =, `in`, `not in`) rade i sa skupovima, ali je njihova implementacija bazirana na teoriji skupova. Tako su na primer implementirane operacije podskupa i pravog podskupa.

Matematički operatori $\subset$,$\subseteq$, $\supset$ i $\supseteq$ su implementirani kao <, <=, > i >= redom.

Primer 80 *Primer prikazuje operator podskupa i pravog podskupa:*

```
>>> skup_ishoda = set([(1,1), (2,1), (1,2), (6,6)])
>>> izvuceno = set([(1,2), (2,1)])
>>> izvuceno < skup_ishoda
True
>>> izvuceno <= skup_ishoda
True
```

Operatori `in` i `not in` su implementirani kao matematički operatori $\in$ i $\notin$.

Primer 81 *Primer prikazuje korišćenje operatora* ***in*** *nad skupovima:*

```
>>> skup_ishoda = set([(1,1), (2,1), (1,2), (6,6)])
>>> (1,2) in skup_ishoda
True
>>> (3,4) in skup_ishoda
False
```

Za sve operatore koji su prikazani u ovom delu postoje i metode koje im odgovaraju, a o tome je više zapisano u [10].

8.3 Funkcije za rad sa skupovima

Postoji mnogo ugrađenih funkcija i metoda za rad sa skupovim. Takođe, moguća je iteracija nad elementima skupova iako oni nisu indeksirani, pa može doći do neočekivanih rezultata.

Primer 82 *Primer prikazuje iteraciju nad elementima skupa*

```
>>> s = set("plazma")
>>> for i in s:
        print(i, end=', ')

a, z, l, m, p,
```

8.3.1 Funkcija `frozenset()`

Do sada je prikazano kako kreirati promenljivi skup. Funkcija `frozenset()` kreira skup koji je nepromenljiv i koji bi mogao da se koristi kao ključ u rečniku. Ovakvom skupu nije moguće dodavati nove elemente, a ni izbacivati postojeće.

Primer 83 *Primer prikazuje kreiranje skupa korišćenjem funkcije* ***frozenset***

```
>>> skup = frozenset([1j, 2, 'i', '1j'])
>>> type(skup)
<class 'frozenset'>
```

8.3.2 Funkcije `len()`, `max()`, `min()` i `sum()`

Sledeći skup funkcija i njihova primena je već viđena u radu sa listama, ali će i ovom delu biti ukratno opisana.

- `len()`: Funkcija prima skup i vraća broj elemenata skupa. Kako skupovi ne sadrže duplikate, funkcija će prebrojati broj različitih elemenata skupa.
- `min()`, `max()`: Kao i kod listi, ove dve funkcije vraćaju najmanji, odnosno najveći element skupa.
- `sum()`: Funkcija računa i vraća sumu skupa. Naravno, elementi skupa u ovom slučaju moraju biti brojevi.

8.3.3 Funkcije `any()`, `all()`, `enumerate()` i `sorted()`

- `any()`: Funkcija vraća True ukoliko postoji element koji je True.

```
>>> s1 = set([1, True, None, False])
>>> s2 = set([0, False])
>>> print(s1, s2)
{None, 1, False} {0}
>>> any(s1)
True
>>> any(s2)
False
```

  Primetimo još da su vrednosti logičkog tipa `True` jednake 1, a `False` jednake 0.

- `all()`: Funkcija vraća `True` ukoliko su svi elementi skupa `True`.

```
>>> s = set([1, True, None])
>>> all(s)
False
```

- `enumerate()`: Funkcija se kreće kroz skup i vraća torke sačinjene od indeksa i elementa. Kako skupovi nisu indeksirani, redosled torki će biti proizvoljan.

```
>>> skup = set(['Heat','Pacers','Thunder','Spurs'])
>>> for ind,el in enumerate(skup):
        print(ind,el)

0 Thunder
1 Heat
2 Pacers
3 Spurs
```

- `sorted()`: Funkcija sortira skup. Ova funkcija može biti pozvana i na liste, rečnike, torke, stringove.

8.4 Metode za rad sa skupovima

Mnoge metode koje su radile sa listama će raditi i sa skupovima. U ovom delu biće prikazane metode koje rade sa skupovima. Većina ovih metoda nema povratnu vrednost, izuzetak je metod `pop()`.

8.4.1 Metode `clear()`, `pop()` i `remove()`

Metod `clear()` uklanja sve elemente iz skupa, metod `pop()` uklanja proizvoljan element iz skupa i taj element vraća kao povratnu vrednost. Ukoliko je skup prazan doći će do greške `KeyError`. Metodu `remove()` je neophodno proslediti element koji će biti izbačen iz skupa. U koliko se element ne nalazi u skupu, doći će do greške `KeyError`.

Primer 84 *Primer prikazuje korišćenje metoda* ***`pop()`****,****`remove()`*** *i* ***`clear()`****:*

```
>>> s = set(['Madrid','London', 'Paris','Moscow','Belgrade'])
>>> s.remove('Paris')
>>> print(s)
{'London', 'Madrid', 'Belgrade', 'Moscow'}
>>> s.pop()
'London'
>>> s.clear()
>>> print(s)
set()
>>> s.pop()
KeyError: 'pop from an empty set'
```

8.4.2 Metode za dodavanje elemeneta u skup

Sledećoj grupi pridapadaju metode za dodavanje elemenata u skup. Metod `add()` prima jedan argument i dodaje ga u skup. Ukoliko u skupu već postoji dodati element ništa se ne dešava.

Primer 85 *Primer prikazuje upotrebu metoda* ***`add()`****:*

```
>>> skup = set(['Madrid','London', 'Paris','Moscow','Belgrade'])
>>> skup.add('Berlin')
>>> print(skup)
{'Moscow', 'Berlin', 'Madrid', 'Paris', 'London', 'Belgrade'}
```

Metod `update()` spaja dva skupa tako što iz prosleđenog skupa dodaje one elemente koji se ne nalaze u originalnom skupu. Ovaj metod je ekvivalentan operatoru $|=$.

Primer 86 *Primer prikazuje metod* ***`update()`*** *i operator* $|=$*:*

```
>>> skup = set(['Madrid','London', 'Paris','Moscow','Berlin'])
>>> new1 = set(['New York', 'Berlin'])
>>> skup.update(new1)
```

```
>>> print(skup)
{'Berlin', 'Madrid', 'Moscow', 'Paris', 'London', 'New York'}
>>> new2 = set(['London','Barcelona'])
>>> skup |= new2
{'Berlin', 'Madrid', 'Moscow', 'Paris', 'London', 'Barcelona',
 'New York'}
```

Sledeći metod menja skup tako što će u skupu ostati samo oni elementi koji se nalaze u preseku originalnom i novog skupa koji je prosleđen kao arugment. Ovaj metod je `intersection_update()` i ekvivalentan je operatoru &=

Primer 87 *Primer prikazuje metod* ***`intersection_update()`*** *i operator &=:*

```
>>> skup = set(['Madrid','London', 'Paris','Moscow','Belgrade'])
>>> new = set(['Madrid', 'Moscow', 'New York', 'Bejing'])
>>> skup.intersection_update(new)
>>> print(skup)
{'Madrid','Moscow'}
```

Postoje i metode `difference_update()` i `symmetric_difference_update()`. Prva menja skup tako što postavlja skup da sadrži samo one elemente starog skupa koji se ne nalaze u novom, prosleđenom skupu. Ovaj metod je ekvivalentan operatoru −=. Drugi menja skup tako da sadrži sve elemente starog i novog, ali ne i one koji se nalaze u njihovom preseku. Operator ekvivalentan ovom metodu je $\hat{=}$.

8.4.3 Metod `copy()`

Za kraj će biti opisan metod koji kopira skup. Metod `copy()` pravi kopiju skupa, ali svi objekti u novom skupu pokazuju na iste elemente u originalnom skupu.

Primer 88 *Primer prikazuje upotrebu metode* ***`copy()`****:*

```
>>> skup = set(['football','basketball'])
>>> new = skup.copy()
>>> print(new)
{'football','basketball'}
>>> skup == new
True
```

9 Iteratori i generatori

9.1 Iteratori

Programski jezik Pajton podržava koncept iteratora. Iterator se može zamisliti kao pokazivač na element kontejnera. Na ovaj način je omogućen pristup svakom elementu. Baziran je na dve metode:

- `next()`: koji vraća sledeći element kontejnera.
- `__iter__`: koji vraća sam objekt iteratora.

Iterator se kreira korišćenjem ugrađene funkcije `iter()` kojoj se prosleđuje kontejner (string, lista, torka, rečnik ili skup). U slučaju rečnika, iterator se kreće kroz ključeve.

Primer 89 *Primer kreiranja iteratora nad objektom tipa string:*

```
>>> i = iter('abc')
>>> type(i)
<class 'str_iterator'>
>>> print(next(i))
a
>>> print(next(i))
b
>>> print(next(i))
c
```

Kada iterator prođe kroz sve elemente, dolazi do greške `StopIteration`. Na ovaj način iteratori su kompatibilni sa petljama, jer one očekuju ovu grešku za prestanak iteracije.

Kreiranje sopstvenih iteratora je takođe moguće. Postoje dva načina za definisanje istih. Moguće je kreirati objekat koji sadrži interfejs iterator. A drugi način je definisanje generator funkcije. U suštini, generator će sadržati interfejs iterator, ali neće biti neophodno kreirati klasu koja sadrži sve neophodna imena funkcija.

Primer 90 *Primer kreiranja klase* [2] *koja ima osobine iteratora.*

```
>>> class MojIterator(object):
        def __init__(self, korak):
            self.korak = korak:
```

[2] O klasama i objektno orijentisanom programiranju u progamskom jeziku Pajton možete više pročitati u radu [10].

```
    def next(self):
    """ Vraca sledeci element iteratora. """
        if self.korak == 0:
            raise StopIteration
        self.korak = -1
        return self.korak
    def __iter__(self):
    """ Vraca sam iterator"""
        return self
```

9.2 Generatori

Generatori su specijalne vrste funkcija koje omogućavaju implementiranje iteratora. Iako dosta liče na funkcije, konceptualno se dosta razlikuju od njih. Umesto `return` koristi se rezervisana reč `yield`. Lokalne promenljive se čuvaju između poziva. Ovo je veoma bitna stvar, jer se ponovni poziv generator funkcije nastavlja nakon poziva `yield` iskaza. Zapravo interpreter pronalazi `yield` iskaz unutar generatora, pamti njegovu poziciju i lokalne promenljive i izlazi iz iteratora. Sledeći put kada je iterator pozvan, nastaviće od `yield` pozicije [20]. Moguće je da generator sadrži više `yield` iskaza. Ukoliko postoji i `return` iskaz unutar generatora, izvršavanje će se zaustaviti uz grešku `StopIterator exception error`.

Primer 91 *Primer kreiranje Fibonačijevog niza korišćenjem generatora:*

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

>>> fib = fibonacci()
>>> next(fib)
1
>>> next(fib)
1
>>> next(fib)
2
>>> [next(fib) for i in range (10)]
[3, 5, 8, 13, 21, 34, 55, 89, 144, 233]
```

9.2.1 Metode za rad sa generatorima

Još jedna mogućnost koja je uvedena u programski jezik Pajton, a tiče se rada sa generatorima su metode koje olakšavaju rad sa generatorima i omogućavaju interakcije sa kodom.

Izvršavanje generator funkcije ili nastavljanje rada nakon `yield` izraza se izvršava funkcijom `next()`. Program se nakon toga izvršava do sledećeg `yield` izraza nakon čega se opet prekida.

Metod `send()` nastavlja izvršavanje i šalje vrednost u generator funkciju, pa je na ovaj način moguće promeniti ponašanje programa.

Primer 92 *Primer prikazuje upotrebu metoda* ***`send()`*** *za slanje vrednosti generator funkciji:*

```
>>> def psychologist():
        print('Please tell me your problems')
        while True:
            answer = (yield)
            if answer is not None:
                if answer.endswith('?'):
                    print("Don't ask yourself too much questions")
                elif 'good' in answer:
                    print("A that's good, go on")
                elif 'bad' in answer:
                    print("Don't be so negative")
>>> free = psychologist()
>>> next(free)
Please tell me your problems
>>> free.send('Why?')
Don't ask yourself too much questions
>>> free.send('I'm good')
A that's good, go on
```

Metod `throw()` omogućava da klijent generatoru prosledi bilo kakvu vrstu izuzetka. Metod će vratiti sledeću `yield` vrednost. Metod za zaustavljanje generatora je metod `close()` koji kreira izuzetak tipa `GeneratorExit` ili `StopIteration`. Stoga, dobra praksa je da se generator funkcije smeštaju u `try/except` blokove.

Primer 93 *Primer prikazuje upotrebu metoda* ***`throw()`*** *i* ***`close()`***

```
>>> def my_generator():
        try:
```

```
            yield 'something'
        except:
            yield 'dealing with the exception'
        finally:
            print("ok,it's clear")

>>> gen = my_generator()
>>> gen.next()
'something'
>>> gen.throw(ValueError('mean'))
'dealing with the exception'
>>> gen.close()
ok,it's clear
```

10 Zaključak

Kroz ovaj rad su detaljno i uz mnoštvo primera prikazani tipovi podataka, operatori, funkcije i metode u programskom jeziku Pajton. Rad predstavlja osnovne koncepte i načine razmišljanja pri savladavanju osnova u programiranju. Zamišljen je da privuče početnike koji se do sada nisu bavili programiranjem i da izaberu Pajton kao prvi programski jezik. U prvom delu rada su uvedene promenljive i tipovi podataka koji postoje u programskom jeziku Pajton i bez kojih je bezmisleno rešavati bilo koji zadatak. Bitno je da učenici i početnici koji se prvi put sreću sa programiranjem dobro upoznaju sa ovim tipovima, jer se one pojavljuju u svakom programu. Takođe su objašnjeni i najčešće korišćeni operatori i prioriteti operatora bez kojih ne bi bilo programiranja.

Pored samog upoznavanja sa celim, realnim i kompleksnim brojevima prikazani su i matematički moduli koji olakšavaju rad sa istim. Veoma je važno da početnici znaju koje su im sve matematičke funkcije na raspolaganju i kako mogu na lakši način raditi sa brojevima.

String tip je veoma bitan u savladavanju osnova programiranja, jer su informacije iz života uglavnom rečeničnog tipa. Čak se i mnogim numeričkim sadržajima lakše barata ako su zapisani u obliku stringa, a u radu su prikazani i ovakvi primeri.

Implementacija listi i njihova upotreba je nešto novo i drugačije za početnike u programiranju. U Pajtonu se liste koriste na jednostavan način, pa se o konceptu listi mnogo može naučiti u ovom programskom jeziku. Rad sa listama i savladavanje rada sa listama predstavlja dobru osnovu za naprednije programiranje.

Korišćenje rečnika za skladištenje podataka je takođe veoma bitno za početnike u programiranju. Ovaj način skladištenja podataka je veoma koristan i brz kada su u pitanju testovi pripadnosti. Sličnu primenu imaju i skupovi koji trebaju da podstaknu logičko razmišljanje kod početnika u programiranju.

Na kraju rada su prikazani iteratori i generatorske funkcije. Veoma korisne i naprednije teme u programiranju i omogućavaju početnicima u programiranju da shvate pojam iterabilnih objekata, kao i koncept neograničenih nizova.

Za sve tipove podataka su prikazane mnoge funkcije i metode koje omogućavaju izvršavanje složenijih problema koji se pojavljuju u programiranju.

Zbog manjka kvalitetne literature na srpskom jeziku o programskom jeziku Pajton, elektronski kurs po kome se ovaj rad i vodi je dobar početak za učenje programiranja u programskom jeziku Pajton. Elektronski kurs je urađen na moderan i interaktivan način. Interpreter koji se nalazi na sajtu umnogome olakšava savladavanje osnovnih pojmova o programskom jeziku Pajton. Slike i animacije su takođe prisutne na kursu i služe da učenika podstiču na razmišljanje o problemima i načinima za rešavanje problema. Razvoj interneta je omogućio da svaki učenik može da uči od kuće, na sopstvenom računaru, a činjenica je da su samostalno učenje i učenje otkrivanjem jedni od najboljih načina učenja.

Literatura

[1] W. Chun, *Core python programming*, Prentice Hall Professional, 2001.

[2] C. Swaroop, *A Byte of Python*, Enllaç web, 2003.

[3] M. Pilgrim, *Dive Into Python*, 2000.

[4] D. Kuhlmam, *A Python Book: Beginning Python, Advanced Python, and Python Exercises*, O'Reilly Media, Inc, 2010.

[5] G. Rossum, *Python tutorial*, CWI (Centre for Mathematics and Computer Science), 1995.

[6] *Python in the enterprise: Pros and cons*, `http://www.techrepublic.com/article/python-in-the-enterprise-pros-and-cons`, Pristupljeno Jun 2014.

[7] *Python2 or Python3*, `https://wiki.python.org/moin/Python2orPython3`, Pristupljeno Jun 2014.

[8] *Whats New In Python 3*, `https://docs.python.org/3/whatsnew/3.0.html`, Pristupljeno Jun 2014.

[9] *Lexical analysis*, `https://docs.python.org/3.1/reference/lexical_analysis.html`, Pristupljeno Jun 2014.

[10] S. F. Lott, *Building Skills in Python: A Programmers Introduction to Python*, 2000.

[11] A. Sweigart, *Invent your own computer games with python*, CreateSpace, 2009.

[12] A. Downey, *Think Python*, O'Reilly Media, Inc., 2012.

[13] *Math - Mathematical functions*, `https://docs.python.org/3/library/math.html`, Pristupljeno Jun 2014.

[14] *Cmath - Mathematical functions for complex numbers*, `https://docs.python.org/3/library/cmath.html#module-cmath`, Pristupljeno Jun 2014.

[15] *Printf-style String Formatting*,
https://docs.python.org/3/library/stdtypes.html#old-string-formatting,
Pristupljeno Jun 2014.

[16] *Sorting HOW TO*,
https://docs.python.org/3/howto/sorting.html,
Pristupljeno Jun 2014.

[17] *What is Python's range() Function?*,
http://www.pythoncentral.io/pythons-range-function-explained/,
Pristupljeno Jun 2014.

[18] T. Ziadé, *Expert Python Programming*, Packt Publishing Ltd, 2008.

[19] *Python: List Comprehensions*,
http://www.secnetix.de/olli/Python/list_comprehensions.hawk,
Pristupljeno Jun 2014.

[20] *Generators*,
http://www.python-course.eu/python3_generators.php,
Pristupljeno Jun 2014.